# Le PYTHON
# en bref
# ...

GALODÉ Alexandre

# Le PYTHON
# En Bref

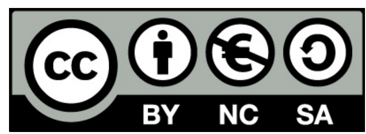
...

Ce livre a été rédigé sous Libre Office et écrit avec des polices d'écriture libres. Les exemples ont été conçus sous Geany sur une distribution Linux Ubuntu.

*" La connaissance appartient à tout le monde "*
*Film Antitrust*

Conçu et rédigé par Alexandre GALODÉ.

Ouvrage placé sous licence Creative Commons BY-NC-SA.
Texte complet de la licence depuis http://creativecommons.fr/licences/les-6-licences/



***Ce livre est dédié à ma fiancée, mes parents, ma famille, mon " frère " et mes amis, présents ou regrettés.***

# SOMMAIRE

Introduction........................................................................27
1 Présentation........................................................................29
1.1 Langages compilés et interprétés........................................30
1.2 Le langage PYTHON en bref................................................31
2 La Programmation Orientée Objet............................................33
2.1 Définition........................................................................34
2.2 Les objets par l'exemple....................................................35
2.2.1 Les classes....................................................................35
2.2.2 Les propriétés................................................................35
2.2.3 Les méthodes................................................................36
2.2.4 En bref..........................................................................36
2.3 L'encapsulation................................................................37
2.4 L'héritage........................................................................38
3 Le langage PYTHON................................................................41
3.1 Les types de données........................................................42
3.1.1 Booléens........................................................................42
3.1.2 Integer..........................................................................42
3.1.3 Long.............................................................................42
3.1.4 Float.............................................................................43
3.1.5 String............................................................................43
3.1.5.1 Méthodes....................................................................44
3.1.6 Liste..............................................................................46

3.1.6.1 Méthodes....................................................................................47
3.1.7 Dictionnaire.......................................................................47
3.1.7.1 Méthodes....................................................................................48
3.1.8 Tuple.....................................................................................48
3.2 Le transtypage.....................................................................49
3.2.1 Transtypage de type.............................................................49
3.2.2 Transtypage de format..........................................................49
4.7.3 Détection de la plateforme d'exécution.................................50
3.3 La portée des variables.......................................................51
3.4 Quelques fonctions prédéfinies..........................................52
3.4.1 PRINT.....................................................................................52
3.4.2 LEN.......................................................................................52
3.4.3 TYPE.....................................................................................52
3.4.4 INPUT....................................................................................53
3.4.5 GETPASS..............................................................................53
3.5 Le caractère de césure.......................................................54
3.6 Le caractère de commentaire.............................................55
3.7 Les opérateurs....................................................................56
3.8 Les tests conditionnels.......................................................57
3.8.1 IF, ELIF, ELSE......................................................................57
3.9 Les boucles........................................................................58
3.9.1 FOR.......................................................................................58
3.9.2 WHILE...................................................................................59
3.9.3 Break et continue..................................................................59
3.10 PYTHON et les fichiers.....................................................60

3.10.1 Chemin absolu et chemin relatif........................................60
3.10.2 Ouverture d'un fichier........................................60
3.10.3 Fermeture d'un fichier........................................61
3.10.4 Lecture........................................61
3.10.5 Écriture........................................63
3.11 La POO PYTHON........................................64
3.11.1 PYTHON et les principes de la POO........................................64
3.11.1.1 L'encapsulation........................................64
3.11.1.2 L'héritage........................................64
3.11.2 La modularité avec PYTHON........................................64
3.11.2.1 L'instruction import........................................64
3.11.2.2 L'instruction SELF........................................65
3.11.2.3 Les fonctions et les procédures........................................65
3.11.2.4 Les classes........................................66
3.11.2.4.1 Les attributs........................................67
3.11.2.4.2 Les accesseurs........................................68
3.11.2.4.3 Les mutateurs........................................68
3.11.2.4.4 Mise en situation........................................68
3.11.2.5 Les modules........................................70
3.11.2.6 Les packages........................................71
3.11.2.6.1 Composition d'un package........................................71
3.11.2.7 En résumé........................................71
3.11.3 Stockage d'objet dans des fichiers........................................73
3.11.3.1 Lecture........................................73
3.11.3.2 Enregistrement........................................73
3.12 Les expressions régulières........................................74
3.12.1 Le module re........................................75
3.12.1.1 Search........................................75

3.12.1.2 Sub.....76
3.13 Les exceptions.....77
3.14 Les mots réservés.....79
3.15 Convention de programmation.....80
3.15.1 La PEP20.....80
3.15.2 La PEP8.....81
3.15.3 Règles de codage.....81
3.15.3.1 Les variables.....82
3.15.3.2 Les fonctions/procédures.....82
3.15.3.3 Les modules et packages.....82
3.15.3.4 Les classes.....82
3.15.3.4.1 Le nom des classes.....82
3.15.3.4.2 Les propriétés et les méthodes.....82
3.15.3.5 Les exceptions.....82
3.15.3.6 Les DocStrings.....83
3.15.3.7 Début de code.....83
3.15.3.8 Sortie de code.....84
3.15.3.9 Autre.....84
3.15.3.10 En plus.....85
3.15.4 Bonne structure type d'un programme.....86
3.15.4.1 La mise en page.....87
3.15.4.2 Les règles.....87
4 Modules Complémentaires.....89
4.1 Pypi.....90
4.2 Le temps.....91
4.2.1 Le timestamp.....91
4.2.2 Date complète.....91

4.2.3 La mise en sommeil....................................................92
4.3 Les mathématiques....................................................93
4.3.1 Le module de base....................................................93
4.3.2 Le module NumPY....................................................93
4.3.2.1 Tableaux multidimensionnels....................................................94
4.3.2.2 Manipulation sur les tableaux....................................................94
4.3.2.2.1 Ajout....................................................94
4.3.2.2.2 Modification....................................................95
4.3.2.2.3 SUPPRESSION ....................................................96
4.3.2.2.4 COPIE ....................................................96
4.3.2.3 Transposition....................................................96
4.3.2.4 Méthodes associées aux tableaux....................................................97
4.3.2.5 Calcul sur tableau....................................................97
4.3.2.6 En plus....................................................98
4.4 Imagerie....................................................99
4.4.1 Ouverture d'une image....................................................99
4.4.2 Création d'une image....................................................99
4.4.3 Modification d'une image....................................................100
4.4.4 Sauvegarde d'une image....................................................100
4.4.5 Affichage d'une image....................................................100
4.4.6 Connaître les composantes d'un pixel....................................................101
4.4.7 Conversion en gris....................................................101
4.4.8 Exemple....................................................101
4.5 Les graphiques avec Matplotlib....................................................103
4.5.1 Création d'une courbe....................................................103
4.5.1.1 Le conteneur....................................................103
4.5.1.2 Ajout d'une courbe....................................................104

4.5.2 Paramétrage complémentaire....106
4.5.2.1 Axes....106
4.5.2.2 Légende de la courbe....106
4.5.2.3 Labels....106
4.5.2.4 Grille....107
4.5.2.5 Utilisation de date en X ou Y....107
4.5.2.6 Image de fond....108
4.5.2.7 Effacement de la courbe....108
4.5.2.8 Transformer un graphique en image....108
4.5.3 Exemple....109
4.6 Les bases de données....111
4.6.1 Présentation rapide....111
4.6.1.1 Composants....111
4.6.1.2 Fonctionnement générique....112
4.6.2 PYSQLITE....113
4.6.2.1 Connexion....113
4.6.2.2 Exécuter une requête....114
4.6.2.2.1 Curseur....114
4.6.2.2.2 Récupération des résultats....114
4.6.2.3 Sauvegarde....116
4.6.2.4 Déconnexion....116
4.6.3 PSYCOPG....116
4.6.3.1 Connexion....116
4.6.3.2 Exécuter une requête....117
4.6.3.2.1 Curseur....117
4.6.3.2.2 Récupération des résultats....117
4.6.3.2.3 Lecture de la dernière requête exécutée....118
4.6.3.2.4 Procédures....118
4.6.3.2.5 Gestion d'erreurs....118

4.6.3.3 Sauvegarde....................................................................................119
4.6.3.4 Déconnexion..................................................................................119
4.7 Le module OS.....................................................................120
4.7.1 Taille d'un fichier...........................................................120
4.7.2 Renommage d'un fichier..................................................120
4.8 Scripting...........................................................................121
4.8.1 Les  arguments...............................................................121
4.8.1.1 Passage d'argument......................................................................121
4.8.1.2 Tester le nombre d'argument.........................................................122
4.8.2 L'exécution de commande..............................................122
4.8.3 Exemple..........................................................................122
4.9 Les fichiers Zip..................................................................124
4.9.1 Création/modification d'un ZIP.......................................124
4.9.2 Ouverture d'un ZIP..........................................................125
4.9.2.1.1 Ouverture et accès aux données...............................................125
4.9.2.1.2 Décompression complète...........................................................126
4.10 Le XML............................................................................127
4.10.1 Codage..........................................................................127
4.10.1.1 L'entête......................................................................................127
4.10.1.2 Le corps.....................................................................................128
4.10.1.3 Exemple.....................................................................................128
4.10.2 Les principes de base....................................................129
4.10.2.1 Les commentaires......................................................................129
4.10.2.2 La racine et les nœuds...............................................................129
4.10.2.3 Les balises.................................................................................129
4.10.2.4 Chevauchement.........................................................................130
4.10.2.5 Les attributs...............................................................................130
4.10.2.6 Les instructions de traitement....................................................130

4.10.2.7 Le XPATH.....131
4.10.2.7.1 Sélection d'un nœud précis.....131
4.10.2.7.2 Sélection avec attributs.....132
4.10.2.7.3 Récupérer toutes les données d'un nœud.....132
4.10.2.7.4 Combinaison de XPATH.....132
4.10.2.7.5 En bref.....133
4.10.3 Le module LXML.....133
4.10.3.1 Utilisation de LXML.....133
4.10.3.2 Création d'un fichier XML.....134
4.10.3.3 Lecture d'un fichier XML.....135
4.10.3.4 Modification d'un fichier XML.....136
4.10.3.4.1 Serialisation et deserialisation.....136
4.10.3.4.2 Méthode générale.....136
4.11 L'Open Document Format.....138
4.11.1 Historique.....138
4.11.2 Les différents types de documents.....138
4.11.3 Les templates.....139
4.11.4 Description du format ODF.....139
4.11.5 Interaction avec PYTHON.....139
4.11.6 Modification de template.....139
4.11.6.1 Théorie.....139
4.11.7 Interaction complète.....141
4.11.7.1 Création et ouverture d'un document ODF.....141
4.11.7.2 ODT.....142
4.11.7.3 ODS.....143
4.11.8 Exemple.....144
4.12 Serveur FTP.....146
4.12.1 Connexion à un serveur FTP.....146

4.12.2 Dépôt de fichiers/dossiers....................................................147
4.12.3 Récupération de fichiers/dossiers.........................................147
4.12.4 Annulation d'un transfert en cours.........................................148
4.12.5 Liste des éléments du dossier...............................................148
4.12.6 Renommage d'un fichier/dossier............................................148
4.12.7 Création d'un dossier.............................................................148
4.12.8 Effacement d'un fichier/dossier..............................................148
4.12.9 Connaître la taille d'un fichier.................................................149
4.12.10 Savoir où l'on se trouve.......................................................149
4.12.11 Utilisation de ligne de commandes.....................................149
4.12.12 Déconnexion du serveur FTP..............................................149
4.12.13 Les messages d'erreurs possibles.......................................150
4.12.14 Exemple...............................................................................151
4.13 Les mails avec SMTP.......................................................153
4.13.1 Données nécessaires............................................................153
4.13.2 Exemple...............................................................................153
4.14 Le port série.....................................................................156
4.14.1 Paramétrage.........................................................................156
4.14.2 Exemple...............................................................................158
4.15 Le port parallèle...............................................................160
4.15.1 Paramétrage.........................................................................160
4.15.2 Exemple...............................................................................161
4.16 Le réseau: les bases........................................................162
4.16.1 Le module socket..................................................................162
4.16.2 Ouverture d'un port...............................................................162
4.16.3 Envoi de données..................................................................163

4.16.4 Réception de données....................................................163
4.16.5 Fermeture d'un port....................................................163
4.16.6 Exemple....................................................164
4.17 La webcam avec OpenCV....................................................165
4.17.1 Utilisation basique....................................................165
4.17.1.1 Connexion au flux vidéo....................................................165
4.17.1.2 Snapshot du flux....................................................166
4.17.1.3 Affichage des snapshots....................................................166
4.17.1.4 Exemple d'affichage d'un flux webcam....................................................167
4.17.2 Utilisation avancée....................................................167
4.17.2.1 Les images....................................................167
4.17.2.2 Les enregistreurs vidéos....................................................169
4.17.2.3 Les polices d'écriture....................................................170
4.17.2.4 Utilisation du clavier....................................................172
4.17.3 Paramétrage de la webcam....................................................173
4.17.3.1 SetCaptureProperty....................................................174
4.17.3.2 GetCaptureProperty....................................................174
4.17.4 Traitements sur les frames....................................................174
4.17.4.1 Les types de données OpenCV....................................................174
4.17.4.2 AbsDiff....................................................175
4.17.4.3 CvtColor....................................................176
4.17.4.4 Dilate....................................................177
4.17.4.5 Erode....................................................179
4.17.4.6 Combinaison: Ouverture....................................................180
4.17.4.7 Combinaison: Fermeture....................................................181
4.17.4.8 Combinaison: Gradient....................................................181
4.17.4.9 Combinaison: Top Hat....................................................182
4.17.4.10 Combinaison: Black Hat....................................................182
4.17.4.11 ContourArea....................................................182

4.17.4.12 DrawContours....183
4.17.4.13 EqualizeHist....183
4.17.4.14 FindContours....184
4.17.4.15 HaarDetectObject....186
4.17.4.16 InRangeS....187
4.17.4.17 Load....188
4.17.4.18 MatchTemplate....188
4.17.4.19 MemoryStorage....189
4.17.4.20 MinMaxLoc....190
4.17.4.21 RunningAvg....190
4.17.4.22 Smooth....191
4.17.4.23 Threshold....193
4.17.5 Ajout d'élément sur une image/frame....194
4.17.5.1 Rectangle....194
4.17.5.2 Cercle....195
4.17.5.3 Ligne....196
4.17.6 Détection de mouvement....196
4.17.6.1 Exemple....198
4.17.7 Reconnaissance des couleurs....199
4.17.7.1 Exemple....201
4.17.8 Détection de contours....201
4.17.8.1 Exemple basique....202
4.17.8.2 Exemple avancé....203
4.17.9 Détection d'une forme....204
4.17.9.1 Recherche d'une image dans l'image....204
4.17.10 Détection de visage....205
4.17.10.1 Exemple....206
4.17.11 Création d'un algorithme de détection....207
4.17.11.1 Création des positifs....208

4.17.11.2 Création des négatifs....208
4.17.11.3 Marquage des positifs....209
4.17.11.4 Création du fichier vecteur....210
4.17.11.5 Création du classificateur....210
4.18 Les Threads....212
4.18.1 Création....212
4.18.2 Lancement....213
4.18.3 Arrêt....214
4.18.4 Pause....214
4.18.5 Appel toutes les x secondes....214
4.18.6 Exemple....214
4.19 Les PDF: Reportlab....216
4.19.1 Principes de fonctionnement....216
4.19.2 Les bases....217
4.19.2.1 Canvas, format de page et unités....217
4.19.2.2 Texte....218
4.19.2.2.1 Polices d'écriture....218
4.19.2.2.2 Simple ligne texte....219
4.19.2.3 Paragraphe....220
4.19.2.3.1 Les styles....220
4.19.2.3.2 Création d'un paragraphe....221
4.19.2.3.3 Ajout d'un paragraphe à un canvas....221
4.19.2.4 Couleur....222
4.19.2.4.1 Principe....222
4.19.2.4.2 Couleur de ligne....222
4.19.2.4.3 Couleur de remplissage....222
4.19.2.4.4 Transparence....223
4.19.2.4.5 Couleur de texte....223

4.19.2.4.6 Définir une couleur....223
4.19.2.5 Rotation....224
4.19.2.6 Tableau....224
4.19.2.6.1 Tableau de base....224
4.19.2.6.2 Les styles de tableaux....224
4.19.2.6.3 Taille de cellules....226
4.19.2.6.4 Les bordures....226
4.19.2.6.5 Les images dans les tableaux....227
4.19.2.6.6 Insertion d'un tableau dans un canvas....228
4.19.2.7 Dessin....228
4.19.2.7.1 Ligne....229
4.19.2.7.2 Cercle....229
4.19.2.7.3 Rectangle....229
4.19.2.7.4 Épaisseur de lignes....230
4.19.2.7.5 Type de ligne....230
4.19.2.8 Image....230
4.19.2.9 graphique....231
4.19.2.9.1 Support de graphique....231
4.19.2.9.2 Histogramme....231
4.19.2.9.3 Linéaire....233
4.19.2.9.4 Camembert....235
4.19.2.9.5 Intégration du support dans un canvas....237
4.19.2.10 Changer de page....237
4.19.2.11 Métadonnées PDF....238
4.19.2.11.1 Auteur....238
4.19.2.11.2 Titre....238
4.19.2.11.3 Sujet....238
4.19.2.12 Sauvegarde....238
4.19.3 Exemple....239
5 Les interfaces graphiques....243

5.1 PYGTK....244
5.1.1 Les fenêtres....245
5.1.1.1 Main....245
5.1.1.2 About....247
5.1.1.3 Info / erreur / question / Attention....248
5.1.1.4 Print....251
5.1.1.5 File....253
5.1.2 Les conteneurs....255
5.1.2.1 Hbox / Vbox....255
5.1.2.2 Boîte à boutons....256
5.1.2.3 Tableau....257
5.1.2.4 Fixed....259
5.1.2.5 ScrolledWindow....259
5.1.2.5.1 Création....259
5.1.2.5.2 Paramétrage....259
5.1.2.5.3 Ajout de l'enfant....260
5.1.3 Les widgets....260
5.1.3.1 Boutons à cliquer....260
5.1.3.2 Boutons à commuter....262
5.1.3.3 Boutons à cocher....262
5.1.3.4 Boutons radios....263
5.1.3.5 Combobox....264
5.1.3.6 Les onglets....265
5.1.3.7 Les tableaux de données....267
5.1.3.7.1 Les TreeStores....268
5.1.3.7.2 Les TreeViews....271
5.1.3.7.3 Les TreeViewColumn....273
5.1.3.8 Label....276
5.1.3.9 Textbox....277

5.1.3.9.1 Monoligne....277
5.1.3.9.2 Multilignes....277
5.1.3.10 Compteur....280
5.1.3.11 Progressbar....282
5.1.3.12 Calendrier....283
5.1.3.13 Image....283
5.1.3.13.1 Image en fond de fenêtre....284
5.1.3.14 Cadre....286
5.1.3.15 Menu....287
5.1.3.15.1 Structure d'un menu....288
5.1.3.15.2 Menu texte....289
5.1.3.15.3 Sous menu....290
5.1.3.16 Barre d'outils....291
5.1.3.17 Barre de statut....292
5.1.3.18 Les curseurs....292
5.1.3.19 Bulles d'aide....294
5.1.3.20 Les stocks (icônes)....294
5.1.3.21 Exemple....301
6 La 3D....305
6.1 Le format STL....306
6.1.1 STL ASCII ....306
6.1.2 STL Binaire....308
6.2 Open GL et PYTHON: les bases....309
6.2.1 Le repère XYZ....309
6.2.2 Notions de base....310
6.2.2.1 Les vertex....310
6.2.2.2 Les polygones....311
6.2.2.3 Les formes de base....311
6.2.2.3.1 Les points....312

6.2.2.3.2 Les lignes....................................................................................312
6.2.2.3.3 Les triangles.............................................................................313
6.2.2.4 État de surface d'un polygone.......................................................313
6.2.2.4.1 Les couleurs..............................................................................313
6.2.2.4.2 Les textures...............................................................................313
6.2.2.5 La caméra.........................................................................................315
6.2.2.5.1 Les différents types de caméra.................................................315
6.2.2.5.2 Positionnement.........................................................................316
6.2.2.5.3 Zoom, translation, et rotation....................................................317
6.2.3 Fenêtre...................................................................................................318
6.2.3.1 Les indispensables...........................................................................318
6.2.3.1.1 Le constructeur de la classe.....................................................318
6.2.3.1.2 Le main.....................................................................................319
6.2.3.1.3 L'InitGL.....................................................................................320
6.2.3.1.4 Le ReSizeGLScene....................................................................321
6.2.3.1.5 Le DrawnGLScene.....................................................................322
6.2.3.2 Insertion dans une frame PYGTK....................................................324
6.2.3.2.1 Principe.....................................................................................324
6.2.3.2.2 Mise en œuvre...........................................................................325
6.2.4 Gestion de la souris.............................................................................328
6.2.4.1 Les boutons et la molette................................................................328
6.2.4.2 Le deplacement de la souris.............................................................329
6.2.5 Gestion du clavier.................................................................................330
6.2.5.1 Touches ALT, SHIFT, CTRL...........................................................330
6.2.5.2 Touches alphanumériques...............................................................331
6.2.5.3 Touches spéciales............................................................................331
6.2.6 Exemple..................................................................................................333
6.3 Concepts mathématiques 2D....................................................................337
6.3.1 Les plans 2D..........................................................................................337

6.3.2 La trigonométrie....338
6.3.2.1 Cercle trigonométrique....338
6.4 Concepts mathématiques 3D....340
6.4.1 Formules....340
6.4.1.1 Équation d'une droite en 3D....340
6.4.1.2 Équation d'un plan en 3D....340
6.4.1.2.1 Méthode 1....340
6.4.1.2.2 Méthode 2....342
6.4.1.2.3 Cas particuliers....342
6.4.1.3 Calcul de la normale au plan 3D....343
6.4.1.4 Calcul des coordonnées d'un point en 3D....343
6.4.1.5 Rotation d'un point en 3D....344
6.4.1.6 Un point appartient-il à un triangle ?....345
6.5 Quelques techniques 3D....346
6.5.1 Depthmaps....346
6.5.2 Stéréogramme....347
6.5.3 Scanner 3D laser....347
7 Gestion d'un projet....353
7.1 Organisation....354
7.1.1 Description du PJ....354
7.1.2 Structure du projet....354
7.1.3 Priorisation....355
7.1.4 Planification....356
7.1.4.1 Planner....356
7.1.4.2 Calligraplan....357
7.2 Viabilité & pérennité....359
7.2.1 Code....359

7.2.2 Tests....359
7.2.2.1 Test unitaire....360
7.2.2.2 Test d'intégration....360
7.2.2.3 Test de validation ou fonctionnel....361
7.2.2.4 Construction d'un test....361
7.2.2.5 Logiciel dédié: Unittest....362
7.2.2.5.1 Fonctionnement de base....362
7.2.2.5.2 Possibilités offertes....363
7.2.2.5.3 Test simple....364
7.2.2.5.4 Tests multiples....365
7.2.3 Documentation....366
7.2.3.1 Epydoc....366
7.2.4 Fichier de log....367
7.3 Versionning....369
7.4 Les licences libres....370
7.4.1 Licences Logicielles....370
7.4.1.1 GPL....371
7.4.2 Licences Documentaires....372
7.4.2.1 Creative Commons....372
8 Déploiement....377
8.1 Création d'un .deb Manuel....378
8.2 Création d'un .DEB automatique....382
8.2.1.1 Accueil - Information....383
8.2.1.2 Contrôle....384
8.2.1.3 Dépendances et conflits....385
8.2.1.4 Création de la structure d'install....386
8.2.1.5 Scripts d'install et de désinstall....387
8.2.1.6 Historique....388

8.2.1.7 COPYRIGHT....389
8.2.1.8 Menu de lancement....390
8.2.1.9 La création du .deb....391
8.3 Utilisation du logiciel Alien....392
9 Ressources....395
10 Annexes....403
10.1 Spécifications Fonctionnelles....404
10.2 Spécifications Techniques....406
10.3 Priorisation....408
10.4 Tests Unitaires....409
10.5 Tests Intégrations....410
10.6 Tests de Validation....411
10.7 Code ASCII....412
10.8 Nuancier RGB Hexa....413
11 Index....415

# Introduction

Que l'informaticien soit débutant ou averti, se pose à un moment la fatidique question du langage à adopter.

En effet, tantôt nous avons besoin d'un langage procédural, tantôt d'un langage objet ; tantôt d'un langage de script, tantôt d'un langage avec IHM...

C'est à ce niveau que **PYTHON** est intéressant. En effet, ce langage sait s'adapter à de nombreuses situations. Capable de créer des clients lourds ou léger, des applications standards ou mobiles, il sait tout faire.

De plus, langage **OPEN SOURCE** par excellence, ce langage bénéficie d'une très large communauté et d'appuis sérieux et solides tels la société GOOGLE qui s'en sert comme langage principal, la NASA ou encore de nombreux logiciels OPEN SOURCE reconnus comme BLENDER.

Prenant de plus en plus d'ampleur dans l'univers de la programmation, **PYTHON** remplace peu à peu ceux qui avaient la préférence dans de nombreux établissements informatiques, car contrairement à d'autres, **PYTHON** ne nécessite nullement de multiples mises à jour par mois. Chaque évolution est mûrement réfléchie avant d'être déployée.

Ce livre est conçu à la fois comme un condensé d'informations pour apprendre **PYTHON**, mais également comme un aide-mémoire toujours utile à avoir sous la main.

Ce livre a été conçu sous Linux, et par conséquent, certain exemples peuvent ne pas fonctionner sous d'autre OS.

Bonne lecture

# 1 Présentation

Wikimedia Commons,
Start_hand.svg

## 1.1 Langages compilés et interprétés

Lorsque vient le moment de faire un choix de langage, deux grandes catégories s'affrontent: les langages compilés et les **langages interprétés**.

Les langages compilés, comme l'indique leur nom, nécessite un compilateur afin de transformer le code source en langage machine. Cela rend le fichier résultant non modifiable, et lui confère une certaine vélocité d'exécution, la machine se contentant de lire des instructions et de les exécuter. C'est le cas du très connu langage C.

Les langages interprétés, eux, ne passent pas par cette phase de compilation. On utilise non pas un compilateur mais un interpréteur. Cet interpréteur va lire l'ensemble du code et le traduire en temps réel à la machine pour exécution.

Si cela peut paraître à certain moins rapide à l'exécution qu'un langage compilé, il n'y a en fait plus forcément beaucoup de différences entre ces deux types de langages avec nos machines récentes.

Le code du langage interprété reste lisible. Il y a aussi bien entendu la possibilité de créer un " **Byte Code** " qui est une pré-interprétation du code pour la machine (extension .pyc).

Deux avantages: une amélioration des performances, ce qui peut parfois s'avérer utile voir indispensable, et la possibilité de ne pas divulguer le code, ce que certain verront comme un point positif.

De fait, un langage interprété est très pratique en debuggage, puissant et réactif en exécution.

**PYTHON** est un de ces langages interprétés, avec toute la puissance que cela implique.

## 1.2 Le langage PYTHON en bref

Le langage **PYTHON** tire son nom du fait que le créateur de ce langage est un fan des comiques anglais les " Monthy PYTHON ".

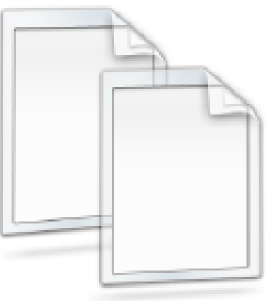
Créé en 1990 par **Guido VAN ROSSUM**, c'est un langage open source, multiplateformes et multifonctions. Les programmeurs s'en servent aussi bien pour remplacer du shell en effectuant du scripting que pour réaliser de la modélisation 3D avec interface graphique.

Ce langage est désormais géré par la **PYTHON** Software Foundation. Principalement utilisé en branche 2.x, la version 3.x est une version " nettoyée " dans le sens ou de nombreux doublons et de nombreuses redondances ont été supprimées afin d'épurer le code du langage.

De fait un code développé en 2.x peut se révéler partiellement incompatible avec la 3.x. Il est souvent recommandé de commencer à développer directement avec la 3.x.

Dans les faits, cette branche est actuellement encore peu utilisée, mais se tenir à ce conseil est garantir une certaine viabilité du code, ce qui est non négligeable.

# 2 La Programmation Orientée Objet



Wikimedia Commons,
1328102023 Copy.png

## 2.1 Définition

Il existe de nombreuses définitions de la **POO**. Pour faire simple, disons qu'il s'agit d'utiliser des copies de briques logicielles afin d'aboutir à notre fin.

Ces copies de briques de base, de références, sont ce qu'on appelle des **OBJETS**. Chaque objet va posséder des caractéristiques (appelées **PROPRIETES**) et des possibilités (appelées **METHODES**).

Cette façon de travailler, va permettre une plus grande rigueur dans la façon de coder, mais également une viabilité du travail effectué.

Chaque brique est ainsi réutilisable à l'infini dans n'importe quel projet, possède sa propre documentation et est maintenable facilement.

## 2.2 Les objets par l'exemple

L'**objet** est une brique de référence pour programmer en POO.

Afin de rentrer plus amplement dans le détail, nous allons prendre un exemple concret, beaucoup utilisé: la voiture.

Qu'est-ce qu'une voiture ? Quelle est la définition d'une voiture ?

Selon wikipedia, c'est un véhicule terrestre à roues équipé avec un moteur embarqué. En simplifié, une voiture possède des roues, un moteur et sert à se déplacer.

Retenons cette définition simpliste. Cette voiture peut avoir des formes, couleurs ou encore marques différentes. De même, elle peut réaliser différentes actions.

En POO, la voiture sera notre objet, sa forme ou sa couleur ses propriétés et les différentes actions qu'elle peut réaliser ses méthodes.

### 2.2.1 Les classes

Cet ensemble, ce qui constitue en quelque sorte la quintessence d'une voiture dans notre exemple est ce qu'on appelle une **classe**. On peut ainsi comparer une classe à un moule servant à créer des objets

### 2.2.2 Les propriétés

Les **propriétés** sont ce qui définit notre objet. Par exemple, sa couleur, ou encore sa marque pour notre exemple.

Ces propriétés peuvent être accessibles! uniquement en lecture, uniquement en écriture ou bien les deux. Elles vous permettront d'interagir avec votre objet afin de le paramétrer au mieux à vos besoins.

### 2.2.3 Les méthodes

Les **méthodes** sont les possibilités qu'offre notre objet. Dans notre exemple, une voiture peut avancer, reculer, allumer ses phares, …

Chacune de ces actions constitue une méthode différente de l'objet voiture.

### 2.2.4 En bref

Pour résumé, nous avons une classe voiture à partir de laquelle nous pouvons créer des objets:

```
1 ma_voiture = Voiture()
```

Notre objet possède des propriétés sur lesquelles nous pouvons agir pour le configurer:

```
1 ma_voiture.couleur = vert
```

Il possède également des méthodes pour nous permettre de lui dire quoi faire:

```
1 ma_voiture.phare(ON)
```

## 2.3 L'encapsulation

Le principe d'**encapsulation** est un des principes fort de la programmation objet. Pour simplifier, cela signifie que pour des raisons de sécurités ou de gestions, nous allons rendre certaines variables accessibles ou non depuis l’extérieur du code.

Si cela peut parfois sembler abstrait, dans les faits cela permet souvent d'éviter qu'un attribut soit changer en cours de calcul et ne provoque un crash du code.

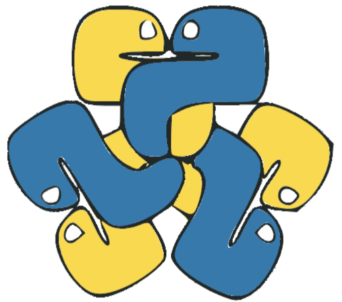
Cela peut également être un moyen d'encadrer précisément l'utilisation d'un module.

## 2.4 L'héritage

Le principe de l'**héritage**, autre principe fort de la programmation objet, est qu'une classe peut elle même hériter d'une autre classe.

Ainsi, pour reprendre l'exemple des voitures, une classe Twingo héritera d'une classe mère voiture. Cette classe Twingo possédera les mêmes caractéristiques que celles définies dans la classe voiture (un moteur, un châssis, un volant, …) mais en possédera des complémentaires (options spécifiques à la voiture).

# 3 Le langage PYTHON



Wikimedia Commons,
Rpyc3-logo-medium.png

## 3.1 Les types de données

**PYTHON** dispose d'un certain nombre de type de données. Nous allons ici voir les principales

### 3.1.1 Booléens

Les **booléens** comme dans tout langage peuvent prendre deux valeurs uniquement. En **PYTHON**, ces valeurs sont **True** et **False**.

```
1  >>> mon_booleen = True
2  >>> mon_booleen
3  True
4  >>> type(mon_booleen)
5  <type 'bool'>
```

La dernière commande type est expliquée plus loin en 3.4.3.

### 3.1.2 Integer

Le premier du trio des classiques. L'entier/**Integer** est un nombre sans virgule. Il est codé sur 4 octets et sa valeur peut s'étendre **de -2 147 483 648 à +2 147 483 647**.

```
1  >>> mon_integer = 56321
2  >>> mon_integer
3  56321
4  >>> type(mon_integer)
5  <type 'int'>
```

### 3.1.3 Long

Lorsque la précision d'un integer est insuffisante, ou peut potentiellement l'être, il faut utiliser un **long**. Ce type de valeur entière **n'a comme limite que la capacité mémoire de l'ordinateur.**

```
>>> mon_long = 2147483648
>>> mon_long
2147483648L
>>> type(mon_long)
<type 'long'>
```

### 3.1.4 Float

Le second du trio. On utilise le point pour indiquer qu'il s'agit d'un **float** (par exemple: a = 3. ou a = 3.0 ). Encodées sur 8 octets, leurs valeurs peut aller **de $10^{-308}$ à $10^{308}$, avec une précision de 12 chiffres significatifs après la virgule.**

***Attention: Le caractère de séparation est le point et non la virgule.***

```
>>> mon_float = 3.5632
>>> mon_float
3.5632
>>> type(mon_float)
<type 'float'>
```

### 3.1.5 String

Le troisième du trio de tête. Une chaîne de caractère est écrite entre simple ou double cote en **PYTHON**, au choix du programmeur.

```
>>> ma_string = "hello"
>>> ma_string
'hello'
>>> type(ma_string)
<type 'str'>
```

Le caractère d'échappement est l'antislash \ . Pour écrire un antislash, on saisit simplement \\ .

```
>>> ma_string2 = "texte avec antislash: \\"
>>> print ma_string2
texte avec antislash: \
```

Un **string** en **PYTHON** est comparable à un tableau de caractère. Ainsi, si ma_string = " Test ", alors ma_string[1] vaut " e ".

```
1 >>> ma_string3 = "Test"
2 >>> ma_string3[1]
3 'e'
```

Il est également possible de ne sélectionner qu'une partie de la chaîne avec le caractère ": "

```
1 >>> ma_string3 = "Test"
2 >>> ma_string3[0:2]
3 'Te'
4 >>> ma_string3[2:]
5 'st'
6 >>> ma_string3[:3]
7 'Tes'
```

***Remarque: Les index commencent à 0 en PYTHON***

Enfin, pour assembler deux chaînes de caractère, il suffit d'utiliser le " + "

```
1 >>> ma_string = "hello"
2 >>> ma_string2 = "world"
3 >>> ma_string3 = ma_string + ma_string2
4 >>> ma_string3
5 'helloworld'
```

#### 3.1.5.1 Méthodes

On peut manipuler une chaîne de caractères grâce à certaines de ces méthodes. Les plus usitées sont les suivantes:

>Changement de casse

```
1 >>> ma_chaine='Hello World'
2 >>> ma_chaine
3 'Hello World'
4 >>> ma_chaine.lower()
5 'hello world'
6 >>> ma_chaine.upper()
```

```
7 'HELLO WORLD'
```

>Mettre la première lettre en majuscule

```
1 >>> ma_chaine='hello world'
2 >>> ma_chaine.capitalize()
3 'Hello world'
```

>Séparation de caractère, avec un caractère prédéfini

```
1 >>> ma_chaine.split('l')
2 ['He', '', 'o Wor', 'd']
```

>Concaténation de chaîne, avec un caractère prédéfini

```
1 >>> ma_chaine.join('l')
2 'l'
3 >>> ma_chaine
4 'Hello World'
```

>Trouver la position d'une lettre

```
1 >>> ma_chaine = 'Hello world'
2 >>> ma_chaine.find('w')
3 6
```

>Compter le nombre d'occurrence d'un caractère

```
1 >>> ma_chaine = 'Hello world'
2 >>> ma_chaine.count('l')
3 3
```

>Supprimer les espaces en début et fin de chaîne

```
1 >>> ma_chaine = '    Ceci est un test    '
2 >>> ma_chaine
3 '    Ceci est un test    '
4 >>> ma_chaine.strip()
5 'Ceci est un test'
```

>Enfin, tester le type de donnée contenu dans la chaîne (**True** si vrai, **False** sinon)

```
1 >>>ma_chaine.isalpha()   #Teste s'il n'y a exclusivement que des lettres
2 >>>ma_chaine.isdigit()   #Teste s'il n'y a que des chiffres
3 >>>ma_chaine.isalnum()   #Teste s'il y a des caracteres alphanumeriques
4 >>>ma_chaine.isspace()   #Teste s'il n'y a que des espaces
```

### 3.1.6 Liste

Comme son nom l'indique une **liste** est un ensemble d'éléments divers: nombre, texte, ...

```
1 >>>#creation d'une liste pleine
2 >>>jour_ouvre = [" lundi ", " mardi ", " credi ", " jeudi ", " vendredi ",1,2,3,4,7]
3 >>>jour_semaine = [] #creation d'une liste vide
```

Ici, deux listes sont créées de manière différente.

Une liste une fois remplie, telle jour_ouvre, se comporte comme un tableau:

```
1 >>>jour_ouvre [1]
2 mardi
```

On peut également modifier un élément d'une liste:

```
1 >>>jour_ouvre[9] = jour_ouvre[9]-2
2 >>>print jour_ouvre[9]
3 5
4 >>>jour_ouvre[2] = " mercredi "
5 >>>print jour_ouvre[2]
6 mercredi
```

nous pouvons également utiliser la fonction prédéfinie **del** pour effacer un élément par son index:

```
1 >>>del(jour_ouvre[9])
2 >>>print jour_ouvre
3 [" lundi ", " mardi ", " credi ", " jeudi ", " vendredi ",1,2,3,4]
```

#### 3.1.6.1 Méthodes

Il existe également une méthode pour effacer un élément d'une liste, en passant comme paramètre sa valeur

```
1 >>>jour_ouvre.remove(4)
2 >>>print jour_ouvre
3 [" lundi ", " mardi ", " credi ", " jeudi ", " vendredi ",1,2,3]
```

Pour réaliser des ajouts, on utilise la méthode **append** des listes:

```
1 >>>jour_ouvre.append(4)
2 >>>jour_ouvre.append(5)
3 >>>print jour_ouvre
4 [" lundi ", " mardi ", " credi ", " jeudi ", " vendredi ",1,2,3,4,5]
```

***Attention: On peut insérer une liste dans une liste. Bien que cela ne soit pas la meilleure manière de procéder, on peut ainsi créer des matrices sommaires.***

Il est aussi possible d'insérer un élément dans une liste à une position donnée:

```
1 >>>jour_ouvre.insert(5, " samedi ")
2 >>>print jour_ouvre
3 [" lundi ", " mardi ", " credi ", " jeudi ", " vendredi ", "samedi ",1,2,3,4,5]
```

Nous pouvons concaténer deux listes avec la méthode **extend:**

```
1 >>>jour_semaine.extend(jour_ouvre)
2 >>>print jour_semaine
3 [" lundi ", " mardi ", " credi ", " jeudi ", " vendredi ", "samedi ",1,2,3,4,5]
```

### 3.1.7 Dictionnaire

Un **dictionnaire** peut, par certain points, être comparé à une liste. Cependant, dans un dictionnaire, les différents éléments ne possèdent pas de valeur mais une clé.

Pour vous représenter cela, imaginez que chaque valeur soit une définition (un string) et que chaque clé associée soit un mot. Pour obtenir la définition d'un mot vous faites appel au dictionnaire en précisant le mot clé.

La création et l'ajout d'élément sont très simples.

```
1 >>>mon_dico = {} #creation d'un dictionnaire vide
2 >>>mon_dico [" PYTHON "] = " langage informatique "
3 >>>mon_dico [" voiture "] = " vehicule automobile "
4 >>>mon_dico [" PYTHON "]
5 " langage informatique "
6 >>>mon_dico
7 {" voiture ": " vehicule automobile ", " PYTHON ": " langage informatique "}
```

Comme on peut le voir sur cet exemple pour créer une entrée il suffit simplement de la déclarer dans le dictionnaire, tel ligne 2 ou 3. Pour lire une valeur il suffit de préciser la clé au dictionnaire, sinon, il affichera la totalité de ses entrées.

#### 3.1.7.1 Méthodes

Pour supprimer une entrée, on utilise la méthode **pop** qui renvoie la valeur liée à la clé transmise:

```
1 >>>mon_dico.pop(" voiture ")
2 vehicule automobile
```

### 3.1.8 Tuple

Un **tuple** est comparable à une liste, à une exception: on utilise des parenthèses et non des crochets lors de la définition

```
1 >>>mon_tuple1 = (1,)       #tuple à un parametre (virgule obligatoire)
2 >>>mon_tuple2 = (2, 3)     #tuple à deux parametres
3 >>>mon_tuple3 = (4, 5, 6)  #tuple à trois parametres
```

## 3.2 Le transtypage

Le **transtypage** correspond à un changement de type de variable, comme par exemple, transformer une variable de int en string; ou encore de format (binaire, hexa, ...)

Le transtypage est très pratique, surtout quand l'on doit afficher un nombre dans une chaîne de caractères par exemple ou encore effectuer un travail sur un type donné de variable.

En langage anglophone, il s'agit d'une opération dite **cast**.

### 3.2.1 Transtypage de type

Pour effectuer un transtypage de type en **PYTHON**, il suffit de taper le type désiré puis la variable à transtyper entre parenthèses:

```
1  >>> a
2  2
3  >>> str(a)
4  '2'
5  >>> float(a)
6  2.0
7  >>> int(a)
8  2
```

### 3.2.2 Transtypage de format

Pour effectuer un transtypage de format en **PYTHON**, il suffit de taper le format désiré puis la variable à transtyper entre parenthèses:

```
1  >>> a
2  2
3  >>> bin(a)
4  '0b10'
5  >>> hex(a)
6  '0x2'
```

### 4.7.3 Détection de la plateforme d'exécution

Il peut parfois être pratique de connaître l'OS sur lequel tourne le script. Cela peut par exemple permettre au programme quelles commandes utilisées.

Pour cela il faut passer par une commande du module **sys**:

```
>>sys.platform
'linux2'
```

## 3.3 La portée des variables

La **portée des variables** est une notion importante dans la programmation.

En **PYTHON**, tout comme dans beaucoup d'autre langage, une variable peut être locale ou globale.

Dans le premier cas, la variable n'existe qu'à l'intérieur de la fonction(/procédure/…) où elle a été définie. Même si ailleurs dans le code une variable porte le même nom, il s'agira néanmoins de deux variables distinctes.

Cependant, il peut parfois être utile, même si cela n'est pas recommandé, d'avoir une variable globale, autrement dit, accessible depuis n'importe où dans le code.

Pour qu'une variable soit globale en **PYTHON**, il faut la définir au début du code en utilisant le mot clé **global**.

De même au début de chaque fonction(/procédure /...), il faudra redéfinir cette variable en globale pour que **PYTHON** comprenne que l'on veut faire référence à la variable globale et non à une variable locale portant le même nom.

```
1 global ma_variable
2 …
3 def ma_procedure():
4     global ma_variable #j'appelle ici la variable globale
5     …
6 …
7 def ma_procedure2():
8     ma_variable = 3 #ici, c'est une variable locale
```

## 3.4 Quelques fonctions prédéfinies

### 3.4.1 PRINT

La fonction **print** n'est utile qu'en mode ligne de commande. Cette fonction permet d'afficher du texte, ou des variables, sur l'écran

```
1 >>>mon_texte = " hello world !!! "
2 >>>print mon_texte
3 hello world !!!
4 >>>print " la variable mon_texte vaut: ", mon_texte
5 la variable mon_texte vaut: hello world !!!
```

Noter dans le dernier exemple la présence de la virgule. Elle permet de préciser à **PYTHON** qu'il faut afficher les données à la suite et non à la ligne.

### 3.4.2 LEN

La commande **len** permet de connaître le nombre de caractères dans un string ou encore le nombre d'éléments dans une liste

```
1 >>>ma_string = " bonjour "
2 >>>len(ma_string)
3 7
4 >>>ma_liste = [4,5," hello "]
5 >>>len(ma_liste)
6 3
```

### 3.4.3 TYPE

La commande **type** permet de connaître le type d'une variable. Cette fonction est souvent utilisée afin de déterminer quel traitement est le plus adapté à une variable donnée.

```
1 >>> mon_integer = 56321
2 >>> type(mon_integer)
3 <type 'int'>
```

### 3.4.4 INPUT

La fonction **input** permet, en ligne de commande, de demander à l'utilisateur de renseigner des paramètres ou informations.

Il existe deux façons d'utiliser cette fonction:

```
1 >>>#Methode 1
2 >>>print " Merci de renseigner votre nom: "
3 >>>nom = input()
4 >>>print " votre nom est: ", nom
5 >>>
6 >>>#Methode 2
7 >>>nom = input(" Merci de renseigner votre nom: ")
8 >>>print " votre nom est: ", nom
```

***Remarque: la fonction input renvoie ce que saisit l'utilisateur. C'est à dire que si l'utilisateur saisit un entier à la place de son nom, cela posera problème. Pour cette raison, il peut sembler utile d'utiliser la fonction raw_input qui renvoie systématiquement la saisie de l'utilisateur, convertie en string.***

### 3.4.5 GETPASS

Le rôle de la fonction **getpass** est identique à la fonction input à un détail près: la confidentialité.

En effet, cette fonction ne réalise pas d'écho de votre saisie.

```
1 >>>mot_passe = getpass (" Merci de saisir un mot de passe:")
2 Merci de saisir un mot de passe:
3 >>>print mot_passe
4 MotDePasse
```

## 3.5 Le caractère de césure

Le caractère de césure permet d’écrire sur plusieurs ligne une instruction qui serait trop longue à écrire sur une seule ligne. Il s'agit du caractère \

Lorsque l'on utilise le caractère de césure, on effectue la coupure après un opérateur, jamais avant, pour des questions de lisibilité.

De plus, l'utilisation de ce caractère de césure fait que les tabulations ne sont plus prises en compte. Aussi faut-il faire bien attention à faire en sorte que le code reste propre et lisible.

```
1  >>> ma_string = "texte \
2  ecrit sur \
3  trois lignes"
4  >>> ma_string
5  'texte ecrit sur trois lignes'
```

## 3.6 Le caractère de commentaire

En **PYTHON**, il n'existe qu'un seul caractère pour écrire un commentaire. Il s'agit du symbole **#**.

Aussi pour écrire vos commentaires sur plusieurs lignes, il faut faire commencer chaque ligne par ce symbole.

Cependant, les éditeurs de code modernes, tel **GEANY**, vous permettent d'écrire l'intégralité de votre commentaire comme un texte normal, avant de le sélectionner et de demander à commenter la sélection.

***Remarque: Pour éviter tout problème lors de l'exécution du code, je vous conseille de prendre l'habitude de ne jamais utiliser de caractères spéciaux (accents par exemple) dans vos commentaires.***

## 3.7 Les opérateurs

Il existe un certain nombre d'astuces pour le codage.

Ainsi au lieu d'écrire ma_variable = ma_variable + 1, nous pouvons écrire ma_variable += 1. De même existe -=, *=, /=.

Il est également possible d'écrire a = b = 1 au lieu de a = 1 et ensuite b = 1.

Plus subtil, au lieu de a = 2 puis b = 3, nous pouvons saisir a, b = 2, 3. Cela ouvre la porte aux permutations rapides comme a, b = b, a.

Pour le reste, les opérateurs sont classiques:

| Paramètre | Description |
|---|---|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| / | Division (avec chiffre après la virgule) |
| // | Partie entière d'une division |
| % | Reste d'une division (modulo) |
| ** | Puissance |
| == | Test d’égalité |
| <> | Test de différence |
| >= | Test supérieur ou égal |
| <= | Test inférieur ou égal |
| > | Test supérieur à |
| < | Test inférieur à |

Il existe en plus de cela les instructions **and**, **or** et **not** qui peuvent servir dans des tests.

## 3.8 Les tests conditionnels

### 3.8.1 IF, ELIF, ELSE

La forme complète d'une boucle conditionnelle **if** est la suivante:

```
if a > 0:
    #code 1
elif a < 0:
    #code 2
else: #a vaut 0
    #code 3
```

***Remarque: Les conditions de test placées entre if et ": " sont appelées prédicats***

## 3.9 Les boucles

### 3.9.1 FOR

La boucle **for** en **PYTHON** n'a pas le même fonctionnement que dans d'autre langage tel le C.

En effet, l'instruction for permet ici de parcourir une variable.

```
1  >>>ma_chaine = " hello "
2  >>>for letter in ma_chaine:
3          print letter
```

Ce code aura pour effet d'afficher sur l'écran les lettres de ma_chaîne les unes après les autres.

Concrètement, cela revient au fait que letter parcourt ma_chaîne depuis son index 0 jusqu'à la fin.

Cela peut être très utile comme par exemple avec des listes:

```
1  >>>for jour in jour_ouvre:
2          print jour
```

Cette façon de faire n'est pas cependant la plus adéquate. On lui préfère alors ceci:

```
1  >>>for index, jour in enumerate(jour_ouvre):
2          print index, jour
```

La fonction **enumerate** prend comme paramètre une liste et nous renvoie un tuple contenant son index et la valeur de l'index. Pour le constater, il suffit de n'indiquer qu'une variable à la place de deux après le for et de l'imprimer.

Pour les dictionnaires, c'est approximativement le même principe:

```
1  >>>for cle, valeur in mon_dico.items():
2          print cle, valeur
```

Cependant, à la place de **.items()**, vous pouvez utiliser **.keys()** pour les clés ou **.values()** pour les valeurs.

### 3.9.2 WHILE

La boucle **while**, comme dans tout autre langage permet d'effectuer des opérations tant que la condition de la boucle est remplie

```
1 >>>while a <> 0:
2         #code
```

### 3.9.3 Break et continue

Dans les **for** et **while**, il existe deux mots clés qui peuvent servir occasionnellement.

Le mot **break** permet d'interrompre une boucle quelle que soit sa condition.

Le mot clé **continue** revient à faire un saut directement de l'endroit ou il est codé au **for** ou **while**, et ce sans exécuter le code qui aurait pu rester en dessous.

## 3.10 PYTHON et les fichiers

La gestion d'un fichier avec **PYTHON** est extrêmement simple

### 3.10.1 Chemin absolu et chemin relatif

La distinction entre ces deux types est très importante. En effet, lors de la programmation **PYTHON**, vous serez amené à utiliser les deux.

Le **chemin absolu** est le chemin complet (par exemple: c:\windows\solitaire.exe). Ce type de chemin est à utiliser par exemple pour accéder en ouverture/écriture à un fichier utilisateur.

Le **chemin relatif**, lui, part de la position du code exécuté. " ./ " correspond alors au dossier en cours, et " ../ " au dossier parent.

Ce type de chemin est pratique pour accéder à des fichiers de configuration du logiciel qui seraient stockés dans le même dossier ou dans un sous-dossier.

***Attention: Sous windows on utilise des antislash dans les chemins, mais sous Linux/UNIX, on utilise des slashs. La notation des chemins relatifs utilisée ici par exemple est la notation Linux.***

### 3.10.2 Ouverture d'un fichier

Avant d'écrire ou de lire un fichier, il faut l'ouvrir. Pour cela, nous avons besoin d'un chemin d'accès contenant le nom du fichier, ainsi que de son mode d'accès:

| Paramètre | Description |
|---|---|
| **r** | Lecture seule |
| **w** | Écriture seule |
| **a** | Mode ajout (append). On complète le fichier |

On peut également écrire **rb**, **wb** ou **ab** pour signifier que nous n'accédons pas au fichier en mode ASCII mais en mode binaire.

Pour ouvrir un fichier, on exécute la commande suivante:

```
mon_fichier = open("CHEMIN ", " MODE ")
```

Ceci est la méthode la plus simple mais pas la plus sécurisée. En effet, si jamais votre logiciel crash pour une raison ou pour une autre, alors le fichier que vous aviez ouvert risquerait de devenir inutilisable. Pour éviter cela, nous utilisons le mot clé " **with** ":

```
with open(" ./config.txt ", w) as fichier_config:
    #Code
```

Cette dernière façon de faire est la meilleure d'un point de vue sécurité. Avec cette méthode, même en cas de crash, le fichier sera fermé proprement.

### 3.10.3 Fermeture d'un fichier

La fermeture d'un fichier est on ne peut plus simple. Il suffit simplement d'écrire:

```
mon_fichier.close()
```

Pour vérifier si le fichier est bien fermé, vous pouvez tester **mon_fichier.closed**. Il s'agit d'un **booléen**

### 3.10.4 Lecture

Il existe différentes façons de lire un fichier. Nous verrons ici les deux principales.

Tout d'abord lire l'intégralité d'un fichier:

```
contenu = mon_fichier.read()
```

La variable contenue sera ainsi de type **string**, et contiendra une chaîne de caractères qui sera en réalité l'intégralité du contenu de mon_fichier.

Cela peut être utile pour analyser des données par exemple ou pour rechercher une information précise. Cette fonction peut également prendre en paramètre un nombre qui correspond au nombre de caractères que l'on désire lire.

A noter l'instruction **readlines** produit sensiblement le même effet, à savoir la lecture intégrale du fichier. Cependant, readlines fait la distinction entre les différentes lignes. Ainsi, il est possible d'effectuer le code suivant:

```
1  f = open('myfile.txt','r')
2  for line in f.readlines():
3      print line
4  f.close()
```

L'autre façon de faire est de lire ligne par ligne avec l'instruction **readline** (remarquer l'absence de " s " final). Cette instruction est à utiliser dans une boucle **while**. Quand un **readline** a atteint la fin de fichier, line vaut alors "". C'est la condition de sortie de la boucle.

```
1  f = open('myfile','r')
2  f_line = f.readline()
3  while f_line <> "":
4      ...          #code
5  f_line = f.readline()
```

***Attention: Quand vous lisez un fichier, assurez vous de connaître sa taille précise afin de ne pas avoir un dépassement de mémoire. Ce dépassement de mémoire peut être évité de manière sûre avec la méthode readline, mais cette dernière est beaucoup plus lente que la méthode readlines. Pour connaître la taille d'un fichier, il faut utiliser la méthode os.path.getsize(" chemin_fichier ") du module os.***

### 3.10.5 Écriture

Pour écrire dans un fichier nous utilisons la fonction **write**.

```
1 mon_fichier.write(" HelloWorld \n ")
```

Attention toutefois car la fonction write n'accepte que des chaînes de caractères. N'oubliez donc pas de faire des transtypages si besoin.

## 3.11 La POO PYTHON

### 3.11.1 PYTHON et les principes de la POO

#### 3.11.1.1 L'encapsulation

L'encapsulation, côté **PYTHON** est un peu particulière. La notion de privé ou public de certain langages est inconnue.

En effet, **PYTHON** privilégie le bon sens à la répression. En clair, nous partons du principe que si l'auteur du code indique qu'une variable ne doit pas être accédée depuis l'extérieur du module, alors l'utilisateur ne devra pas chercher à y accéder.

Il est cependant possible de ruser pour faire en sorte que l'utilisateur soit automatiquement redirigé en cas de tentative d'accès non correcte. On utilisera pour cela les **accesseurs** et les **mutateurs** (cf 3.11.2.4.2)

#### 3.11.1.2 L'héritage

En **PYTHON**, voici comment se réalise un **héritage** de classe.

```
1  >>>class MaClasse1:
2      #code1
3  >>>class MaClasse2(Maclasse1):
4      #code2
```

**PYTHON** respecte l'ensemble des principes de l'héritage. MaClasse2 héritera donc de l'ensemble des caractéristiques de MaClasse1

### 3.11.2 La modularité avec PYTHON

#### 3.11.2.1 L'instruction import

La fonction **import** permet de préciser à **PYTHON** que vous allez utiliser du code externe et où le trouver. Cas échéant, vous pouvez

également préciser quelles fonctions vous désirez importer (**from...import...**)

```
1 import math #importe le module de mathematique
2 from math import sqrt #importe uniquement la fonction sqrt
```

La question qui peut se poser ici est l'utilité d'un code tel celui ligne 2. En réalité, il trouve toute son utilité dans des systèmes possédant peu de ressources. Cela permet de contrôler au mieux notre consommation des ressources hôte.

### 3.11.2.2 L'instruction SELF

L'instruction **self** est très importante en **PYTHON**. En effet, il indique au langage que vous faites référence à l'objet que vous utilisé et non à la classe mère.

Par exemple, prenons deux objets: une Twingo et une A6. Ce sont tous les deux des objets créés à partir de la classe mère voiture. Sans l'utilisation du self, le fait de changer une propriété sur l'objet Twingo n'affecterait pas cet objet mais directement la classe mère voiture.

Ainsi dans une classe, le premier et/ou seul paramètre des fonctions/procédures sera toujours self. De même, les variables dans une classe devront être précédées de self**.** On parle alors d'attributs de l'objet.

La notion de self sera précisée par des exemples lors de l'explication des classes ci-après.

### 3.11.2.3 Les fonctions et les procédures

Les **fonctions** et les **procédures** fonctionnent sur le même principe. La différence réside dans le fait qu'une fonction renvoie un résultat, une procédure ne renvoie rien.

Pour définir une fonction ou une procédure nous utilisons simplement le mot clé **def**:

```
1 >>>def ma_fonction():
2     print " Hello World !!! "
3     return True
```

***Remarque: Les parenthèses sont obligatoires, même si aucun argument n'est passé.***

Nous pouvons voir ici que nous retournons un booléen. On utilise pour cela le mot clé **return**. Si nous désirons renvoyer plusieurs variables, il suffit de les écrire à la suite et de les récupérer dans des variables adaptées:

```
1 >>> def ma_fonction(fnom, fprenom):
2     return "nom: " + fnom, "prenom: "+fprenom
3 >>> nom, prenom = ma_fonction("DUPONT", "Jean")
4 >>> print nom
5 nom: DUPONT
6 >>> print prenom
7 prenom: Jean
```

Il est également possible de définir une ou plusieurs valeurs par défaut:

```
1 >>> def ma_fonction(fnom, fprenom= 'Jean'):
2     return "nom: " + fnom, "prenom: "+fprenom
3 >>> nom, prenom = ma_fonction("DUPONT")
4 >>> print nom
5 nom: DUPONT
6 >>> print prenom
7 prenom: Jean
```

***Remarque: Quand certain paramètres n'ont pas de valeurs par défaut, ces paramètres doivent êtres placés avant ceux possédant une valeur par défaut.***

#### 3.11.2.4 Les classes

Lorsque nous allons créer un objet, nous nous référerons en fait à une **classe**. Pour reprendre la définition donnée précédemment, une classe est une brique logicielle de référence.

Pour déclarer une classe en **PYTHON**, on utilise le mot clé **class**. La définition d'une classe doit s'accompagner de sa méthode constructeur:

```
1 class MaClasse:
2     def __init__(self, param1, param2, …, paramn):
3         #code
```

Le rôle de ce constructeur est de créer un objet copie de la classe lorsque nous définissons un nouvel objet.

Pour créer un objet depuis une classe, il suffit juste de l'appeler:

```
1 >>>mon objet = Maclasse()
```

#### 3.11.2.4.1 Les attributs

Il est possible de déclarer des **attributs** propres non pas à l'objet (avec l'utilisation du self) mais à la classe.

```
1 class MaClasse:
2     mon_compteur = 0
3     def __init__(self, param1, param2, …, paramn):
4         MaClasse.mon_compteur += 1
5         #code
```

Les attributs ne sont ni plus ni moins que les variables de notre objet. Selon le principe d'encapsulation, nous ne devons pas avoir un accès direct à ces variables.

Bien que **PYTHON** autorise cet accès direct de par sa philosophie (rappel: en **PYTHON**, tout est public), il est recommandé, lors de la création d'une classe de créer des méthodes d'accès en lecture et en écriture sur chaque attribut de la classe.

Elles permettent entre autre, outre le fait de rajouter de la sécurité et ainsi de respecter au mieux le principe d'encapsulation, de pouvoir exécuter, de façon totalement transparente pour l'utilisateur, du code complémentaire.

Ces méthodes dédiées et spécifiques sont ce qu'on appelle des **accesseurs** (lecture) et des **mutateurs** (écriture).

#### 3.11.2.4.2 Les accesseurs

Les **accesseurs** se présentent toujours sous la même forme

```
1 _get_ATTRIBUT()
```

En **PYTHON**, déclarer un accesseur se fait de la façon, suivante:

```
1 def _get_ATTRIBUT(self):
2     """Accesseur de l'attribut"""
3     #Code, finissant souvent par " return self._ATTRIBUT "
```

***Remarque: Notez le underscore en sus dans le _get_ATTRIBUT_ et le self._ATTRIBUT. Il a son importance que nous verrons dans l'exemple final***

#### 3.11.2.4.3 Les mutateurs

Tout comme les accesseurs, les **mutateurs** présentent également toujours la même forme:

```
1 _set_ATTRIBUT()
```

En **PYTHON**, nous déclarons un mutateur de la façon suivante:

```
1 def _set_ATTRIBUT(self, paramètre):
2     """Mutateur de l'attribut"""
3     #code, finissant souvent par " self._ATTRIBUT = new_value "
```

***Remarque: Notez bien ici aussi la présence de l'underscore***

#### 3.11.2.4.4 Mise en situation

Dans cet exemple, nous allons mettre en situation une classe simplifiée " Twingo " et son attribut " phares ". Cet exemple va nous

permettre de présenter les derniers éléments nécessaires à l'implémentation des accesseurs et des mutateurs.

```
1   class Twingo:
2       """Classe Twingo simplifiée possédant un seul attribut: phares"""
3   
4       def __init__(self):
5           """Constructeur de la classe Twingo"""
6           self._phares = False #Phares éteints par défaut
7   
8   
9       def _get_phares(self):
10          """Accesseur de l'attribut phares, permet de récupérer l'état des phares"""
11          if self._phares:
12              return 'ON'
13          else:
14              return 'OFF'
15  
16  
17      def _set_phares(self, on_off):
18          """Mutateur de phares, permet de changer l'etat"""
19          if on_off = 'ON':
20              self._phares = True
21          else:
22              self._phares = False
23          phares = property(_get_phares, _set_phares)
```

Analysons cet exemple de plus près. Vous aurez reconnu maintenant les accesseurs et les mutateurs. Aussi allons nous nous attarder sur les underscores en sus, et sur la dernière ligne.

Les underscores en sus n'influent en rien sur le fonctionnement du programme. Ils servent juste à distinguer visuellement les éléments auxquels on peut, ou non, accéder depuis l'extérieur. Il s'agit uniquement d'une convention de codage.

Bien entendu, comme rappelé plus tôt, **PYTHON** n'effectuant aucune restriction ou contrôle à ce niveau, c'est à vous de ne pas chercher à passer outre les recommandations du programmeur.

Dans notre exemple, les attributs _phares et phares sont donc deux variables différentes.

_phares ne doit pas être accédé depuis l'extérieur. A la place nous passerons par l'attribut phares, ce qui nous amène à la dernière ligne.

Nous y utilisons le mot clé **property**. Ce mot clé permet d'indiquer à **PYTHON** pour l'attribut concerné l'accesseur (1er paramètre) et le mutateur (2nd paramètre) qui lui sont liés: ***property(accesseur, mutateur).***

Grâce à cette dernière ligne, et au principe d'accesseur et mutateur, alors que l'utilisateur pense accéder directement à l'attribut, il passe en réalité par un ensemble de procédures/fonctions de manière totalement transparente.

### 3.11.2.5 Les modules

En **PYTHON**, un **module** est un fichier contenant différents codes (fonctions, procédures, classes, …) ayant un lien entre eux.

Pour utiliser le code contenu dans un module, on doit l'importer avec le mot clé **import**. Cependant, lorsque nous développons un module, il est recommandé de créer la fonction de test du module.

Cette fonction de test a pour but de réaliser un ensemble d'instructions utilisant le code défini dans le module afin de le tester sans avoir besoin d'utiliser un logiciel dédié.

Cette fonction se déclare ainsi:

```
1 if __name__ == "__main__":
```

La variable **__name__** est une variable système initialisée au lancement de l'interpréteur **PYTHON**. Si cette dernière vaut **__main__**, cela signifie que le fichier appelé est le fichier exécuté.

### 3.11.2.6 Les packages

Les **packages** sont le niveau supérieur des modules. Ils permettent donc de regrouper plusieurs modules.

Dans la pratique, un package n'est rien d'autre qu'un dossier. Ces dossiers peuvent contenir d'autres dossiers (package) ou d'autres fichiers (module).

Tout comme pour les modules, pour les utiliser dans un code, il faut utiliser le mot clé import. Pour utiliser ensuite un sous package ou un module du package, on utilise le point " . " afin de modéliser le chemin menant à la fonction ou la procédure que nous désirons utiliser.

#### 3.11.2.6.1 Composition d'un package

Un package **PYTHON** n'est jamais vide. En effet, chaque package (et sous package, quel que soit le niveau) doit contenir outre les modules qui s'y trouvent, un fichier **__init__.py**.

Ce fichier **__init__.py** est la plupart du temps totalement vide. Il est surtout là pour indiquer à **PYTHON** qu'il ne s'agit pas d'un dossier classique, au sens répertoire d'un OS, mais d'un package **PYTHON**.

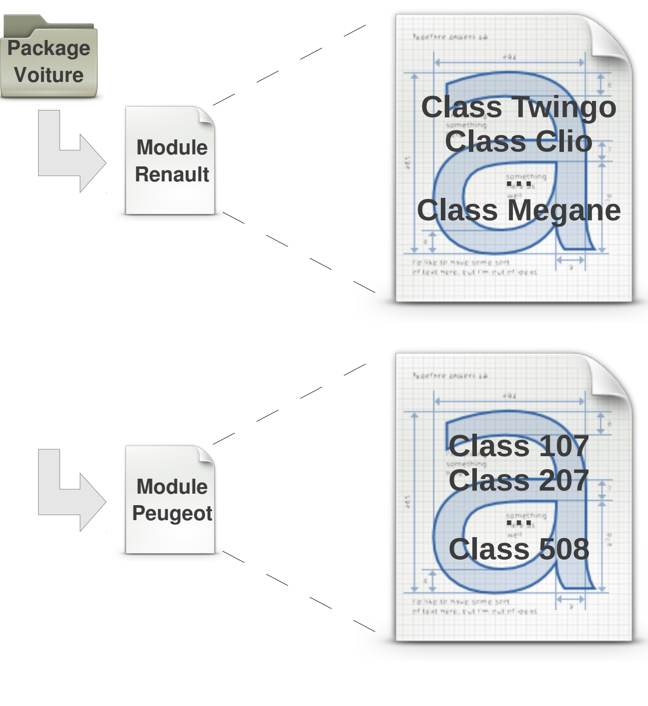
Si vous en avez l'utilité, vous pouvez par exemple écrire à l'intérieur de ce fichier **__init__.py**, du code qui sera exécuté à l'import du package.

Enfin, vous pouvez y placer une simple **docstring** afin de documenter le package (sous package, …). Cette documentation permettra une future maintenance que ce soit par vous même ou par un tiers (voir 3.15.3.6).

### 3.11.2.7 En résumé

Pour reprendre l'exemple des voitures, nous pourrions avoir un package voiture, puis un module par marque (Renaud, Jugeote, ...), puis une classe par modèle de voiture de la marque (Clio, 504, DS5, ...).

Chacune de ces classes, représentant une voiture, aurait alors des propriétés et des méthodes qui lui sont propres.




...

***Illustration 1: Principe de Package/Module/Classe***

### 3.11.3 Stockage d'objet dans des fichiers

Comme dans beaucoup de langage objet, en **PYTHON** nous pouvons enregistrer nos **objets** dans des fichiers pour nous en resservir plus tard.

Pour cela, nous utiliserons le module **pickle**. À l'intérieur de ce module, nous utiliserons deux classes: **Pickler** et **Unpickler**.

#### 3.11.3.1 Lecture

Après avoir ouvert un fichier en écriture, les commandes types à utiliser sont:

```
1 mon_pickler = pickle.Pickler(fichier) #Cree le pickler dans le fichier
2 mon_pickler.dump(objet) #stocke l'objet dans le pickler
```

#### 3.11.3.2 Enregistrement

Après avoir ouvert un fichier en lecture, nous utiliserons une commande de type:

```
1 mon_unpickler = pickle.Unpickler(fichier) #recupere le pickler dans fichier
2 mon_objet = mon_unpickler.load()
```

## 3.12 Les expressions régulières

Les **expressions régulières**, aussi appelées **REGEX** pour REGular EXpressions, sont très utilisées dans le monde de l'informatique. Elles permettent notamment de s'assurer qu'une saisie est conforme aux attentes et besoins du programme informatique.

Elles permettent de définir la structure d'une donnée attendue ou recherchée.

Le fonctionnement d'une **REGEX** est plus ou moins normalisé. Ci-dessous un résumé des éléments pouvant composer une **REGEX**.

| Paramètre | Description |
|---|---|
| **^** | Début de chaîne |
| **$** | Fin de chaîne |
| **.** | Tout caractère sauf retour à la ligne |
| ***** | Nombre d'occurrence indifférent (0 compris) |
| **+** | Nombre d'occurrence indifférent (0 exclus) |
| **?** | Nombre d'occurrence égal à 0 ou 1 |
| **\|** | OU logique |
| **( )** | Groupe avec fonction équivalente aux parenthèses en équation mathématique |
| **[ ]** | Intervalle |
| **{ }** | Répétition |

| Paramètre | Description |
|---|---|
| **A{n}** | A apparaît exactement n fois |
| **A{n,}** | A apparaît au moins n fois |
| **A{,n}** | A apparaît n fois maximum |

Côté fonctionnement, on commence par indiquer ce que l'on attend, puis on indique le nombre de fois désiré. Par exemple, si on attend 4 caractères entre 0 et F, alors on saisira **[0-9A-F]{4}**.

***Remarque: Nous avons indiqué ici deux intervalles de caractères potentiels. Le OU logique est ici implicite***

Ainsi, si l'on attend la saisie d'un numéro de téléphone tout attaché (soit 10 chiffres), la **REGEX** sera **^[0-9]{4}$**. Si l'on désire une forme avec des tirets, alors ce sera **^([ -][0-9]){4}$**.

De même, si on attend la saisie d'une adresse mail de notre société, comme identifiant par exemple, et que l'extension est en @societe.fr, alors nous pourrons saisir comme **REGEX** **^.{1,}(@societe\.fr){1}$**

***Remarque: On notera ici la présence de " \ ", caractère d'échappement pour les caractères spécifiques REGEX.***

## 3.12.1 Le module re

En **PYTHON**, nous avons la possibilité d'utiliser ces **REGEX**. Et pour nous faciliter le travail, il existe le **module re**.

Ce module met à notre disposition les méthodes **search** et **sub**.

### 3.12.1.1 Search

La première de ces méthodes, **search**, permet de rechercher une REGEX dans une chaîne de caractères.

```
1 >>> import re
2 >>> test = "ma_chaine"
3 >>> re.search("_cha", test)
4 <_sre.SRE_Match object at 0xa670870>
5 >>> re.search("toto", test)
6 >>>
```

Comme on peut le constater, rien de très compliqué. Il faut deux paramètres: le premier la **REGEX**, et le second la chaîne où l'on désire chercher. Si une occurrence est trouvée, re.search nous renvoie une

expression. Si aucune occurrence n'est trouvée, alors **re.search** nous renvoie None.

#### 3.12.1.2 Sub

La méthode **sub** permet, elle, d'effectuer un remplacement dans une chaîne de caractères.

Il s'agit ni plus ni moins d'une fonction rechercher/remplacer, mais en plus brut.

```
1 >>> mail = "toto@masociete.fr"
2 >>> re.search("^.{1,}(@masociete\.fr){1}$", mail)
3 <_sre.SRE_Match object at 0xa602860>
4 >>> re.search("^.{1}(@masociete\.fr){1}$", mail)
5 >>> re.sub("(@masociete\.fr){1}", "@societe.com", mail)
6 'toto@societe.com'
```

Le premier argument de **re.sub** est la **REGEX** recherchée, le second la chaîne de remplacement, et le troisième la chaîne où cherche la **REGEX** et effectue le remplacement.

Nous pouvons ici constater dans cet exemple que la méthode **re.sub** renvoie le résultat de la substitution.

## 3.13 Les exceptions

Les **exceptions** permettent d'intercepter une erreur dans un code (telle une division par zéro) et de réaliser une action donnée. Pour cela, nous utilisons **try:... except:... finally: ...**

```
1  try:
2      #code1 a tester
3  except:
4      #code2  execute en cas d'erreur
5  finally:
6      #code3 execute quoiqu'il arrive à la fin du code1
```

Dans cet exemple, le mot clé **finally** est utilisé. Dans la réalité, ce mot clé sert assez peu.

Après **except**, nous pouvons trouver un type d'erreur donné prédéfini tel **ZeroDivisionError**, ou encore un type d'erreur prédéfini personnalisé par nos soins et déclenché via l'instruction **raise:**

```
1  >>> def ma_div(num, denom):
2      try:
3          div = num / denom
4          if num % 2 is 0:
5              raise ValueError("numerateur impair obligatoire")
6      except ZeroDivisionError:
7          print "Division par zero"
8      except ValueError:
9          print "numerateur pair"
10     except:
11         print "une erreur a eu lieu"
12
13 >>> ma_div(2,0)
14 Division par zero
15 >>> ma_div(2,1)
16 numerateur pair
17 >>> ma_div("r",2)
18 une erreur a eu lieu
```

Regardons cet exemple de plus près. Si une division par 0 a lieu, alors elle est interceptée par le type prédéfini en ligne 6. Si le numérateur est pair, alors cela déclenche une erreur interceptée par un type que nous avons défini en ligne 8, sur le type d'erreur prédéfini. Enfin, si une erreur

a lieu et ne correspond à aucune de celle précisée, alors elle est capturée ligne 10.

Les exceptions peuvent être utilisées par exemple pour le debuggage ou pour des sections de code à risque. Tout code **PYTHON** doit comporter un minimum de ces exceptions afin de s'assurer que les erreurs ne passent pas inaperçues/silencieusement.

## 3.14 Les mots réservés

Il existe une série de mots réservés par **PYTHON** que nous ne pouvons jamais utiliser en tant que noms de variables, de classes ou autres.

| and | as | assert | break | class | continue | def | del | elif | else | except |
|---|---|---|---|---|---|---|---|---|---|---|
| False | finally | for | from | global | import | if | in | is | lambda | none |
| nonlocal | not | or | pass | raise | return | Tru<br>e | try | while | with | yield |

## 3.15 Convention de programmation

### 3.15.1 La PEP20

Le rôle de la **PEP 20** (PEP pour **PYTHON** Enhancement Proposal: proposition d'amélioration de **PYTHON**) est de donner des directives pour coder de la meilleure façon possible.

Énoncée sous forme d'aphorismes plus ou moins compréhensibles, cette PEP est une des briques de base pour tout programmeur **PYTHON**. Elle concerne cependant surtout l'aspect du code.

***>le beau est préférable au laid***
***>l'explicite est préférable à l'implicite***
***>le simple est préférable au complexe***
***>le complexe est préférable au compliqué***
***>du code trop imbriqué est plus difficile à lire***
***>l'aéré est préférable au compact***
***>la lisibilité compte***
***>les cas particuliers ne sont pas suffisamment particuliers pour casser la règle***
***>il est difficile de faire un code à la fois fonctionnel et « pur »***
***>les erreurs ne devraient jamais passer silencieusement***
***>à moins qu'elles n'aient été explicitement réduites au silence***
***>en cas d'ambiguïté, résistez à la tentation de deviner***
***>il devrait exister une et une seule manière évidente de procéder***
***>même si cette manière n'est pas forcément évidente au premier abord, à moins que vous ne soyez Néerlandais (humour: L'inventeur du PYTHON est néerlandais)***
***>maintenant est préférable à jamais***
***>mais jamais est parfois préférable à immédiatement***
***>si la mise en œuvre est difficile à expliquer, c'est une mauvaise idée***
***>si la mise en œuvre est facile à expliquer, ce peut être une bonne idée***
***>les espaces de noms sont une très bonne idée***

### 3.15.2 La PEP8

Le rôle de la **PEP 8** est de donner des directives claires quant à la manière même de rédiger le code. Une fois de plus, ce ne sont cependant que des conseils que vous êtes libre ou non de suivre.

Ci-dessous, une traduction française de ces conseils.

***>Une indentation doit équivaloir à 4 espaces (Configurez la touche tab)***
***>Il ne faut jamais mélanger espaces et indentations dans le même code***
***>Une ligne ne doit excéder 79 caractères***
***>La définition d'une fonction, classe ou autre doit être suivie de 2 sauts de lignes***
***>Il faut utiliser 1 import par package***
***>Ces imports doivent toujours être en début de code***
***>Ils doivent être répartis en 3 groupes dans l'ordre suivant, séparés par un saut de ligne***
***-Les bibliothèques standards***
***-Les bibliothèques tierces***
***-Les bibliothèques " maison "***
***>Toujours utiliser des chemins absolus pour l'import de modules***
***>Toujours utiliser un espace avant et après un opérateur***
***>Une seule instruction par ligne***

### 3.15.3 Règles de codage

L'ensemble des règles énoncées ci-après ne constitue en rien une obligation, mais uniquement de fortes recommandations, qui sont respectées par de nombreux programmeurs.

Le respect de ces **règles de codage** facilite autant le codage que la future maintenance potentielle par des tiers.

#### 3.15.3.1 Les variables

Le nom des variables ne peut pas commencer par un chiffre. Il doit n'être constitué que de lettres minuscules et les différents mots séparés par des underscores " _ ".

#### 3.15.3.2 Les fonctions/procédures

Les règles de nommage des fonctions et des procédures sont identiques à celles des variables.

#### 3.15.3.3 Les modules et packages

Les noms des modules et des packages doivent être courts et constitués uniquement de lettres minuscules.

De préférence, il faut éviter d'utiliser des underscores et n'avoir un nom ne tenant qu'en un mot, surtout pour les packages.

#### 3.15.3.4 Les classes

##### 3.15.3.4.1 Le nom des classes

Le nom d'une classe se compose d'un ensemble de mots, collés les uns aux autres, avec la première lettre de chaque mot en majuscule. Exemple: ClassName

##### 3.15.3.4.2 Les propriétés et les méthodes

On utilise les mêmes règles que pour les variables.

#### 3.15.3.5 Les exceptions

Les règles de nommage des exceptions suivent les mêmes règles de nommage que les classes.

### 3.15.3.6 Les DocStrings

En **PYTHON**, juste après la définition d'une fonction, procédure, classe, ou module, il faut insérer ce qu'on appelle une **docstring**.

Entourée d'un triple double quote, elle donne une description du code que nous écrivons, et peut s'écrire sur plusieurs lignes sans utiliser le caractère de césure. Si vous tapez **help**(fonction) c'est cette **docstring** que vous verrez apparaître.

```
1  >>> def ma_fonction(a):
2        """Multiplie a par 2"""
3        print a*a
4
5  >>> help(ma_fonction)
6  Help on function ma_fonction in module __main__:
7
8  >>> ma_fonction(a)
9  Multiplie a par 2
10
11 >>> ma_fonction(2)
12 4
```

***Remarque: Mettre une docstring bilingue est souvent apprécié dans les grandes entreprises.***

### 3.15.3.7 Début de code

Au début de chaque code, il est important de préciser certaines informations, surtout sur Linux.

Ces informations occupant systématiquement les deux premières sont le chemin de l’interpréteur, ainsi que l'encodage utilisé.

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
```

La première ligne indique à l'OS où trouver l'interpréteur **PYTHON** afin de traduire le code. Cela n'est à faire que sous LINUX.

La seconde ligne, non obligatoire, quoique vivement conseillée, permet de préciser à l'interpréteur le type d'**encodage** que nous utilisons pour le code. En effet, par défaut, les accents ne sont pas autorisés.

***Remarque: Ici nous avons précisé un encodage ISO. Ce n'est pas le seul puisqu'il existe également l'encodage LATIN1, UTF-8, ...***

### 3.15.3.8 Sortie de code

Vous l'avez peut-être parfois remarqué lors de l'exécution d'un code en ligne de commande, un petit " **exit code: 0** "

En fait, il s'agit pour l'OS de vous indiquer si votre traitement s'est bien déroulé (0) ou non (1).

En **PYTHON**, on utilise l'instruction **sys.exit(1)** pour quitter le programme si l'on sait qu'il va crasher. On s'assure ainsi de bien maîtriser le code, respectant ainsi le principe de la PEP indiquant qu'aucune erreur ne doit passer silencieusement.

De même, nous pouvons connecter le bouton de fermeture à un **sys.exit(0)** pour indiquer que tout s'est bien déroulé.

Il n'est cependant pas du tout obligatoire d'employer ce genre d'instructions, mais elles existent et il faut le savoir, car cela peut parfois s'avérer utile de les implémenter.

### 3.15.3.9 Autre

Certaines recommandations complémentaires existent. Par exemple, plutôt utiliser **is** et **is not** à la place de **==** et **<>**.

De même, il est déconseillé d'utiliser le L minuscule, le O majuscule ou le i majuscule. En effet, selon la police d'écriture ces caractères peuvent parfois mal s'interpréter.

Enfin, les constantes doivent systématiquement être écrites en majuscule: MA_CONSTANTE

### 3.15.3.10 En plus

En général, outre ces règles, on essaie de toujours faire commencer un nom par un préfixe (souvent une lettre) minuscule, suivie d'un underscore. Cette lettre permet d'identifier en un coup d'œil le type auquel nous avons à faire. En fond orange ceux qui s'avère réellement indispensables:

| Préfixe | Description |
|---|---|
| **vg_** | **Indique une variable globale** |
| **vl_** | **Indique une variable locale** |
| **f_** | **Indique une fonction** |
| **p_** | **Indique une procédure** |
| **pkg_** | Indique un package |
| **m_** | Indique un module |
| **c_** | Indique une classe |
| **c_p_** | Indique une propriété d'une classe (par exemple) |
| **c_m_** | Indique une méthode d'une classe (par exemple) |
| **e_** | Indique une exception |

### 3.15.4 Bonne structure type d'un programme

```
#!/usr/bin/PYTHON
# -*-coding:utf-8 -*

#===================================================================#
#-------------------------------------------------------------------#
#                                NOM                                 #
#-------------------------------------------------------------------#
#*******************************************************************#
#                  Société - Auteur - Date initiale                 #
#-------------------------------------------------------------------#
#                         Notes/Commentaires                        #
#                                                                   #
#-------------------------------------------------------------------#
#                             HISTORIQUE                            #
#   V0.1.0    Société - Auteur - Date                               #
#             Motif de la modification et/ou commentaire bref       #
#===================================================================#


#-------------------------------------------#
#          Importation des packages         #
#-------------------------------------------#
import time


#-------------------------------------------#
#          Declaration des variables        #
#-------------------------------------------#
a = "Hello World"


#-------------------------------------------#
#                    Code                   #
#-------------------------------------------#
def ma_fonction(a):
""" ma_fonction(a)
    permet d'afficher a à l'écran
"""
   print a

if __name__ == " __main__ ":
""" Main
    Permet d'afficher " hello world " toutes les 5 secondes
"""
   ma_fonction("hello world") #on peut aussi simplement passer 'a'
   time.sleep(5) #mise en veille pendant 5s
```

Comme on peut le voir sur cet exemple simple, une structure de code respecte une certaine mise en page et différentes règles, outre celles déjà énoncées précédemment.

#### 3.15.4.1 La mise en page

On veillera notamment à bien s'assurer de la présence d'un cartouche complet.

De plus, chaque section sera précédée d'un mini **cartouche** résumant l'utilité/l'action du code de cette section.

Les sections seront séparées par 4 sauts de lignes et les fonctions/procédures internes à ces sections par 2 sauts de lignes.

Les tabulations seront équivalentes à 4 espaces (les éditeurs sont paramétrables à ce niveau en général).

#### 3.15.4.2 Les règles

Sous Linux, UNIX, on n'oubliera pas les deux première lignes, qui doivent OBLIGATOIREMENT se trouver en ligne 1 et 2.

La première permet de stipuler où se trouve l'**interpréteur** **PYTHON**. La seconde, permet d'indiquer le type d'encodage à utiliser.

En plus de cela, on veillera à bien mettre en place des docstrings, les plus explicites possible.

Enfin, il est important de tenir à jour toutes les docstrings, cartouches et commentaires afin qu'ils correspondent au code en place.

Le module graphique **TKInter** est considérée comme le module graphique de base de **PYTHON**.

D'apparence un peu austère, elle ne possède que quelques widgets basiques, suffisants la plupart du temps. Certaines autres modules graphiques permettent d'améliorer l'aspect et/ou de rajouter des widgets complémentaires à TKInter, mais cela ne sera pas vu ici.

L'import se fait de la façon suivante:

```
1 >>>from tTkinter import *
```

# 4 Modules Complémentaires

Wikimedia Commons,
Gnome-mime-text-x-sh.png

## 4.1 Pypi

**Pypi**, pour " **PYTHON** Package Index " est un dépôt mettant à disposition des développeurs tout un ensemble de packages/modules.

Le but est triple: certifier de façon officielle un certain nombre de packages/modules, standardiser l'exécution de certaines actions, et doter notre langage préféré d'un minimum de fonctionnalités de base.

Le site, unique et en anglais http://pypi.python.org/pypi, met à disposition de tout programmeur, quel qu'il soit, des tutoriels, des exemples, des documentations, …

La communauté active permet de remonter l'ensemble des bugs identifiés et d'être sûr de toujours disposer d'une version la plus fonctionnelle possible d'un package/module.

De plus, le système **Pypi** simplifie l'installation/désinstallation, ainsi que la maintenance de l'ensemble des bibliothèques utilisées au sein d'un projet.

Au moment de la rédaction de ce livre, 26704 packages étaient disponibles via le dépôt **Pypi**.

Un flux RSS vous permettra également de vous tenir facilement au courant des évolutions de vos packages préférés et des nouveautés.

certain des modules présentés ici seront des modules dépendant de **Pypi**.

## 4.2 Le temps

On utilise ici le module **time**. Ce module permet de manipuler simplement les variables en rapport avec le temps.

### 4.2.1 Le timestamp

La variable de base pour la gestion du temps est ce qu'on appelle le **TIMESTAMP**. Ce **timestamp** correspond au nombre de secondes écoulées depuis la date de référence **UNIX**: le 1er janvier 1970 à 00h00m00s.

Pour obtenir ce timestamp rien de plus simple, il suffit d'utiliser la méthode **time**:

```
1  >>> import time
2  >>> time.time()
3  1341920301.923005
4  >>> type (time.time())
5  <type 'float'>
```

On peut remarquer que le **timestamp** est un float d'une grande précision. Cette précision peut éventuellement vous servir à calculer le temps d'exécution d'un code.

### 4.2.2 Date complète

Le module **time** possède une méthode **localtime** permettant de récupérer une date au grand complet avec les éléments suivants:

| Paramètre | Description |
|---|---|
| **tm_year** | L'année |
| **tm_mon** | Le numéro du mois |
| **tm_mday** | Le numéro du jour du mois |
| **tm_hour** | L'heure |

| | |
|---|---|
| **tm_min** | Les minutes |
| **tm_sec** | Les secondes |
| **tm_wday** | Le jour de la semaine (0 (lundi) à 6) |
| **tm_yday** | Le jour de l'année |
| **tm_isdst** | Indique un éventuel changement d'heure locale |

Il est vivement recommandé d'utiliser le **timestamp** pour tout ce qui est calcul et le **localtime** pour tout ce qui est affichage.

Vous pouvez utiliser le **timestamp** comme référence pour le **localtime**, et la méthode **mktime** pour récupérer un **timestamp** depuis un **localtime**.

```
1 >>> mon_timestamp = time.time()
2 >>> print mon_timestamp
3 1341921025.92
4 >>> mon_localtime = time.localtime(mon_timestamp)
5 >>> print mon_localtime
6 time.struct_time(tm_year=2012, tm_mon=7, tm_mday=10, tm_hour=13,
7 tm_min=50, tm_sec=25, tm_wday=1, tm_yday=192, tm_isdst=1)
8 >>> print time.mktime(mon_localtime)
9 1341921025.0
```

***Remarque: Comme on peut le voir en comparant la ligne 3 et la ligne 9, au cours de la conversion, nous perdons un peu de précision puisque il y a un arrondi à la seconde près.***

### 4.2.3 La mise en sommeil

Le module **time** permet également de mettre le code en pause pendant un temps déterminé par un float. Ce float représente un nombre de secondes

```
1 >>>time.sleep(5.5) #mise en pause pendant 5 secondes 1/2
```

## 4.3 Les mathématiques

### 4.3.1 Le module de base

Le module s'appelle **math**. Il possède différentes fonctions pouvant être utiles

```
>>>math.pow(4,2)  # 4 puissance 2
16
>>>math.sqrt(16) #racine carree de 16
4
>>> 2 * math.exp(5) #exponentiel
296.8263182051532
>>>math.fabs(-5) #valeur absolue
5
>>>ang_deg = 57.29
>>>math.radians(ang_deg) #conversion de degres en radians
0.9998991284675514
>>> math.degrees(0.9998991284675514) #conversion de radians en degres
57.29
>>>math.ceil(5.3) #arrondi par exces
6
>>>math.floor(2.4) #arrondi par defaut
2
>>>math.trunc (4.6) #ne renvoie que la parti entiere
4
>>>import random #import du module random complementaire a math
>>>random.random() #renvoie une valeur float aleatoire entre 0 et 1
0.535093545175585
```

### 4.3.2 Le module NumPY

**NumPy** est le module **PYTHON** spécialisée dans le calcul scientifique.

```
import numpy
```

Sa grande force tient dans le fait que l'on peut créer des tableaux ou des matrices **Numpy** et appliquer un calcul à cet ensemble, en une fois, permettant ainsi de réaliser un gain de temps important.

Bien que possédant de nombreuses spécificités, c'est surtout cette capacité à appliquer un calcul à l'ensemble des éléments d'un tableau que nous verrons ici.

#### 4.3.2.1 Tableaux multidimensionnels

Créer un **tableau multidimensionnel** NumPy est simple:

```
1 >>>import numpy
2 >>>tab1 = numpy.array([1,2,3]) #tableau a une dimension
3 >>>tab2 = numpy.array([[1,2,3],[4,5,6]]) #matrice 3*3
4 >>>tab2[0][0]
5 1
6 >>>tab2[1][2]
7 6
```

Il y a aussi possibilité de définir le type des éléments d'un tableau à sa création:

```
1 tab3 = numpy.array([1,2,3], dtype='i') #i pour integer
```

***Remarque: Un tableau de deux dimensions (matrice) est comparable à un tableau vertical contenant des tableaux horizontaux. Ainsi lors de l'appel d'un élément (ex: tab2[0][1]), le premier indice correspond à la ligne (emplacement dans le tableau vertical), et le second indice à la place de l'élément dans le tableau horizontal***

#### 4.3.2.2 Manipulation sur les tableaux

Un tableau **NumPy** se rapprochant des listes, on peut utiliser les mêmes méthodes de manipulation

##### 4.3.2.2.1 Ajout

Pour effectuer un ajout à un tableau, on utilise deux fonctions de **NumPy**: **hstack** et **vstack**. La première sert à ajouter des éléments en horizontal et la seconde en vertical du tableau

Elles prennent 2 paramètres sous forme d'un tuple: le tableau **NumPy** concerné, puis l'élément à rajouter

***Attention: Ces deux méthodes n'enregistrent pas les modifications effectuées. N'oubliez donc pas de récupérer le résultat***

```
>>>import numpy
>>>tab1=numpy.array([1,2,3])
>>>tab1
array([1,2,3])
>>>numpy.hstack((tab1,4))
array([1,2,3,4])
>>>tab1
array([1,2,3])
>>>tab1=numpy.hstack((tab1,4))
>>>tab1
array([1,2,3,4])
```

On peut également utiliser la méthode ***append***. Elle permet un ajout simplifié directement à la fin du tableau.

```
>>>tab2
array([[1,2,3],
       [4,5,6]])
>>>tab2 = numpy.append(tab2, [[7,8,9]], axis = 0)
>>>tab2
array([[1,2,3],
       [4,5,6],
       [7,8,9]])
```

***Remarque: Axis correspond au comportement désiré par NumPy. S'il vaut 0, alors le tableau garde des dimensions. S'il faut None, alors le tableau est mis à plat et ne possède plus qu'une dimension (tableau classique).***

##### 4.3.2.2.2 Modification

Pour modifier un élément rien de plus simple, il suffit de préciser l'élément à modifier puis de préciser sa nouvelle valeur

```
>>>tab2
array([[1,2,3],
       [4,5,6]])
```

```
4 >>>tab2[0][2] = 4
5 >>>tab2
6 array([[1,2,4],
7        [4,5,6]])
```

##### 4.3.2.2.3 SUPPRESSION

La suppression d'un élément d'un tableau **NumPy** passe par la méthode **NumPy.delete**. Cette méthode prend en paramètre le tableau ainsi que l'indice concerné. Dans un tableau à 1 dimension l'indice correspond à l'emplacement de l'élément à effacer. Dans un tableau à deux dimensions, on précisera uniquement le premier indice (suppression d'un tableau horizontal entier, cf remarque en 4.3.2.1.

```
1  >>>tab1
2  array([1,2,3,4])
3  >>>tab1 = numpy.delete(tab1,3)
4  >>>tab1
5  array([1,2,3])
6  >>>tab2
7  array([[1,2,4],
8         [4,5,6]])
9  >>>tab2=numpy.delete(tab2,1,axis = 0)
10 >>>tab2
11 array([[1,2,4]])
```

##### 4.3.2.2.4 COPIE

Une chose importante à savoir concernant **NumPy** est qu'il faut utiliser une fonction bien précise de **NumPy** pour réaliser une copie de tableau sous peine de ne posséder qu'une simple référence. On utilise la méthode **copy**.

```
1 >>>tab3 = tab2.copy()
```

#### 4.3.2.3 Transposition

**NumPy** offre la possibilité de réaliser simplement des transpositions. Pour rappel, il s'agit d'inverser lignes et colonnes d'une matrice ainsi que ses dimensions de fait

```
>>> tab2.transpose()
array([[1, 4],
       [2, 5],
       [3, 6]])
```

#### 4.3.2.4 Méthodes associées aux tableaux

Il existe également certaines méthodes associées aux tableaux.

| Paramètre | Description |
|---|---|
| **max()** | Récupère la ou les valeur(s) maximum(s) |
| **min()** | Récupère la ou les valeur(s) minimum(s) |
| **sum()** | Effectue la somme des éléments |
| **sort()** | Trie les éléments |
| **std()** | Calcul l'écart type |

Le paramètre axis=1 permet de travailler sur les lignes et axis=0 sur les colonnes et non pas l'ensemble du tableau.

```
>>> tab2.max()
6
>>> tab2.sum()
21
>>> tab2.sum(axis=1)
array([ 6, 15])
>>> tab2.max(axis=1)
array([3, 6])
>>> tab4=numpy.array([4,2,8,7])
>>> tab4.sort()
>>> tab4
array([2, 4, 7, 8])
```

#### 4.3.2.5 Calcul sur tableau

Comme indiqué au début de cette partie consacré à **NumPy**, ce qui va nous intéresser surtout est la capacité à réaliser des opérations de manière simplifiée:

```
1  >>> tab2
2  array([[1, 2, 3],
3         [4, 5, 6]])
4  >>> tab2 = 2 * tab2
5  >>> tab2
6  array([[ 2,  4,  6],
7         [ 8, 10, 12]])
```

Dans cet exemple, on peut voir que l'opération effectuée sur le tableau affecte la totalité du tableau. Nous économisons ainsi l'utilisation d'une boucle

#### 4.3.2.6 En plus...

Certaines fonctions fournies par **NumPy** permettent d'effectuer des opérations complexes sur des matrices

| Paramètre | Description |
| --- | --- |
| **dot(mat1, mat2)** | Permet d'effectuer un produit entre matrices |
| **inner(mat1, mat2)** | Permet de réaliser un produit scalaire |
| **solve(mat1, mat2)** | Permet d'obtenir la solution d'un système linéaire avec: mat1 $\begin{matrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{matrix}$ mat2 $\begin{matrix} reponse1 \\ reponse2 \\ reponse3 \end{matrix}$ |
| **numpy.sin(x)** | Permet d'obtenir le sinus de l'angle x (existe aussi numpy.arcsin) |
| **numpy.cos(x)** | Permet d'obtenir le cosinus de l'angle x |
| **numpyr.tan(x)** | Permet d'obtenir la tangente de l'angle x |
| **numpy.pi** | Renvoie la valeur de la variable pi |
| **numpy.abs(x)** | Renvoie la valeur absolue de x |
| **numpy.ceil(x)** | Arrondi à l'entier supérieur |
| **numpy.floor(x)** | Arrondi à l'entier inférieur |
| **numpy.round(x)** | Arrondi à l'entier le plus proche |

## 4.4 Imagerie

Pour travailler sur des images, nous utilisons le module **PIL** (**PYTHON** Imaging Library).

Ce module permet de manipuler les images comme par exemple pour effectuer des traitements sur ces dernières.

Nous verrons ici l'ouverture et la fermeture d'un fichier image, ainsi que leur création (pixel par pixel), leur enregistrement et enfin l'affichage de ces images. Il existe de nombreuses autres possibilités avec ce module. Pour les connaître toutes, rendez-vous sur http://effbot.org/imagingbook/image.htm

### 4.4.1 Ouverture d'une image

Pour ouvrir une image existante, il faut utiliser la méthode **open**

```
1 >>> from PIL import Image
2 >>> im = Image.open("/media/PERSO/test.png")
```

### 4.4.2 Création d'une image

Pour créer une image, c'est très simple. Il suffit de faire un appel au constructeur en précisant le mode (généralement " RGB ") et la taille désirée (un tuple(x,y)):

```
1 >>> from PIL import Image
2 >>> im=Image.new("RGB",(500,250))
```

***Remarque: Par défaut, l'image créée est entièrement noire***

Il ne reste ensuite plus qu'à renseigner chaque pixel de votre image.

```
1 >>> im.putpixel((0,0),(255,255,255))
```

Ici, le premier tuple correspond aux coordonnées (X,Y) du pixel dans l'image en partant du coin supérieur gauche.

Le second tuple, lui, correspond aux couleurs RGB, ou 0 représente le noir et 255 le blanc

***Remarque: Pour obtenir uniquement une image en dégradé de gris tel que du Depthmaps, il faut que R=G=B en toute occasion.***

### 4.4.3 Modification d'une image

Pour modifier une image c'est assez simple. Une fois l'image ouverte, il suffit d'utiliser la même méthode que pour créer une image. Le pixel sera alors remplacé.

Il est également possible de réaliser un traitement sur l'ensemble de l'image. Pour ce faire, il est alors conseillé de créer une matrice avec **NumPy** qui permettra alors de réaliser des traitements facilement (modification de teinte, détection de contours, ...)

### 4.4.4 Sauvegarde d'une image

Pour sauvegarder une image, il suffit d'utiliser la méthode **save**

```
1 >>> im.save("/media/PERSO/test", "PNG")
```

Comme on peut le constater, on passe ici deux paramètres. Le premier est le chemin avec le nom désiré pour le fichier mais sans extension. Le second est le type de fichier désiré, ce qui donnera l'extension du fichier.

### 4.4.5 Affichage d'une image

Pour afficher l'image via le visualiseur d'images par défaut du système, il suffit d'utiliser la méthode **show**.

```
>>> im.show()
```

### 4.4.6 Connaître les composantes d'un pixel

Pour des raisons diverses, il peut parfois être intéressant de connaître les composantes précises d'une image (RGB).

Pour cela, il faut utiliser la méthode **getpixel**. Elle prend en paramètre les coordonnées du point et renvoie un tuple contenant les composantes

```
r, g, b = im.getpixel(0,0)
```

### 4.4.7 Conversion en gris

Comme dit précédemment, une couleur de gris à une composante RGB (Red, Green, Blue) équilibrée. Entendez par là que R = G = B.

Cependant, il existe une formule donnée par la commission internationale de l'éclairage qui qualifie le gris dit luminance, avec la formule suivante:

$$gris = 0{,}2125 * R + 0{,}7154 * G + 0{,}0721 * B$$

### 4.4.8 Exemple

```
#!/usr/bin/PYTHON
# -*-coding:utf-8 -*

from PIL import Image



def Step1():
    """
    Cree une image noir de 64 * 64 pixels, puis l'enregistre.
    """
```

```
12      im = Image.new("RGB",(64,64))
13      im.save("/home/steph/demo.png")
14      del im #permet de supprimer l'objet image cree
15
16
17
18
19  def Step2():
20      """
21      Ouvre l'image, modifie 1 pixel sur 2, l'enregistre, puis l'affiche.
22      """
23      im = Image.open("/home/steph/demo.png")
24      i, j = 0, 0
25
26      while i < 64:
27          while j < 64:
28              im.putpixel((i,j),(255,255,255))
29              j = j + 2
30          i = i + 2
31          j = 0
32
33      im.save("/home/steph/demo.png")
34      im.show()
35      del im
36
37
38
39
40  if __name__ == '__main__':
41      Step1()
42      Step2()
```

***Illustration 2: Image générée avec le code exemple***

## 4.5 Les graphiques avec Matplotlib

Le module **Matplotlib** permet de créer des graphiques de type courbe.

Puissant, il n'est cependant pas forcément évident de prime abord. Capable de créer indifféremment des **courbes** 2D ou 3D, en couleur ou non, multiples ou simples, ce module est désormais la référence dans ce domaine.

Nous ne verrons ici que les courbes de base, en 2D, ainsi que la gestion des titres et des légendes associés. Pour des courbes plus complexes, je vous invite à visiter le site et à lire la documentation de Matplotlib.

### 4.5.1 Création d'une courbe

#### 4.5.1.1 Le conteneur

Pour créer une courbe, la première chose à faire est de créer un objet pour contenir cette courbe. Cet objet est ici appelé **Figure**.

Il peut prendre différents paramètres, mais nous les laisserons à leur valeur par défaut. Bien entendu, il faudra auparavant importer le module de **matplotlib**

```
1 from matplotlib.figure import Figure
2 ma_figure = Figure()
```

Une fois ce conteneur créé, il faut créer un support de courbe et l'ajouter au conteneur via la méthode **add_subplot**.

```
1 ma_courbe = ma_figure.add_subplot(111)
```

Comme on peut le voir, cette méthode prend un paramètre. Ce paramètre permet de stipuler l'emplacement de la courbe. En effet, un conteneur peut contenir 1,2,4, … courbes.

On définit donc leur emplacement à l'aide de ce paramètre. Ici, 111 signifie que le conteneur aura une définition de 1 courbe en Y, 1 courbe en X, et que ma_courbe occupera la position 1. Si on avait eu 212, on aurait eu 2 courbes en Y, 1 en X, et ma_courbe aurait été la seconde courbe. En résumé donc: YXPosition.

***Remarque: Si cela vous paraît encore un peu flou, je vous invite à faire quelques tests par vous même.***

#### 4.5.1.2 Ajout d'une courbe

Sur ma_courbe nous pouvons ensuite rajouter une ou plusieurs courbes. Nous utiliserons la méthode **plot(x, y, paramètres optionnels)**.

Cette méthode prend en paramètres optionnels, entre autres, un label, une couleur, un type de marqueur pour chaque point ou encore le type et la largeur de ligne. La couleur, le type de ligne et le type de marqueur peuvent être passés en un seul paramètre.

```
1  ma_courbe.plot([1,2,3],[1,2,3], 'go--', label='Test1', linewidth=2)
```

***Remarque: On peut voir que pour X et Y, on passe une liste de points. Une courbe se trace en une seule fois, c'est pourquoi on doit passer l'ensemble des points constituant la courbe en une fois. De plus, Il faut la même quantité de points entre x et y, sinon vous aurez un message d'erreur.***

Dans l'exemple ci-dessus, nous réalisons une ligne de 3 points. Le " go-- " signifie que nous traçons la ligne en vert, avec des marqueurs circulaires et que la ligne sera pointillée (cf tableaux ci-après). La ligne aura une largeur double à la normale. Notre ligne s'appellera 'Test1'.

| Paramètre | Description |
|---|---|
| **'-'** | Solide |
| **'--'** | Tiret |
| **'-.'** | Alternance tiret pointillé |
| **':'** | Pointillé |

| Paramètre | Description |
|---|---|
| **b** | Bleu |
| **g** | Vert |
| **r** | Rouge |
| **c** | Cyan |
| **m** | Magenta |
| **y** | Jaune |
| **k** | Noir |
| **w** | Blanc |

<table>
<tr><th>Paramètre</th><th>Description</th></tr>
<tr><td>'.'</td><td>Point</td></tr>
<tr><td>','</td><td>Pixel</td></tr>
<tr><td>'o'</td><td>Cercle</td></tr>
<tr><td>'v'</td><td>Triangle, pointe en bas</td></tr>
<tr><td>'^'</td><td>Triangle, pointe en haut</td></tr>
<tr><td>'<'</td><td>Triangle, pointe à gauche</td></tr>
<tr><td>'>'</td><td>Triangle, pointe à droite</td></tr>
<tr><td>'s'</td><td>Carré</td></tr>
<tr><td>'p'</td><td>Pentagone</td></tr>
<tr><td>'*'</td><td>Etoile</td></tr>
<tr><td>'x'</td><td>X</td></tr>
<tr><td>'|' (alt gr 6)</td><td>Barre verticale</td></tr>
<tr><td>'_' (touche 8)</td><td>Barre horizontale</td></tr>
</table>

Il est bien entendu possible de mixer ces symboles afin d'obtenir ce qu'on désire. Par exemple 'go-' pour une ligne verte avec des marqueurs circulaires et une ligne solide; ou encore 'r:x' pour une ligne rouge en pointillé, avec des marqueurs en x.

***Remarque: En cas d'absence d'information sur un type ou un style, le module prendra des valeurs par défaut.***

### 4.5.2 Paramétrage complémentaire

#### 4.5.2.1 Axes

Il est possible de paramétrer, dans une certaine mesure, les axes du graphique.

Par exemple, on peut définir le **Xmin**, le **Xmax**, le **Ymin** et le **Ymax**, via la commande **axis**:

```
1 ma_courbe.axis([Xmin,Xmax,Ymin,Ymax])
```

De même, nous pouvons désactiver l'affichage des axes:

```
1 ma_courbe.axis('off')
```

#### 4.5.2.2 Légende de la courbe

Les légendes permettent de facilement distinguer les différentes courbes présentes sur un graphique.

Pour cela il suffit d'utiliser la méthode **legend(loc='best')**. Cette dernière permet d'afficher la légende de la courbe au meilleur endroit possible.

Autre solution sinon, utiliser un coin du graphique.

```
1 ma_courbe.legend(loc='upper right')
```

#### 4.5.2.3 Labels

Chaque axe peut posséder son propre label. Pour donner un label à l'axe X ou Y, on utilise les méthodes **set_xlabel** et **set_ylabel**.

```
ma_courbe.set_xlabel('Axe X')
ma_courbe.set_ylabel('Axe Y')
```

#### 4.5.2.4 Grille

Il est possible de mettre une grille sur la courbe afin, par exemple, d'en faciliter la lecture. Pour cela, on utilisera la méthode **grid**. On lui passera comme paramètre:

```
ma_courbe.grid(color='r', linestyle='-', linewidth=2)
```

La couleur, le style de ligne sont sur le même principe que pour tracer une courbe.

Pour désactiver la grille, rien de plus simple, il suffit d'appeler **grid** sans paramètre.

```
ma_courbe.grid(False)
```

#### 4.5.2.5 Utilisation de date en X ou Y

Il peut être pratique pour tracer, par exemple, une courbe de données chronologiques de travailler directement avec des dates. Pour cela aussi **matplotlib** peut être utile.

En effet, le module **datetime** est spécialement là pour ça.

```
import datetime
import matplotlib
...
x= [datetime.datetime(2011,02,01), datetime.datetime(2011,03,01)]
y=[100,150]
ma_courbe.plot(X,Y,label = 'Ma courbe avec date en X')
ma_figure.autofmt_xdate(bottom=0.2, rotation=30, ha='right')
```

La dernière ligne mérite une petite explication complémentaire. Elle permet de modifier la position et l'inclinaison des annotations de l'axe X (dans notre cas, les dates).

Les valeurs par défaut placent le texte à 30°. Mais il est cependant possible de donner un autre angle de rotation.

#### 4.5.2.6 Image de fond

Dans certain cas, il peut être intéressant d'afficher une image en fond de notre courbe. Pour cela, il existe la méthode **imshow**. Cette méthode prend comme paramètre une image PIL (cf 3.14.3)

***Attention: La résolution peut poser problème. En effet, cette méthode va traduire l'image avec 1 pixel = 1 unité.***

```
1 imm = Image.open('/home/alex/test.bmp')
2 ma_courbe.imshow(imm)
```

#### 4.5.2.7 Effacement de la courbe

Effacer une courbe est très utile, lorsque l'on désire afficher une courbe de manière dynamique, ou simplement recharger une donnée sur un graphique.

Pour cela, la méthode **clf** est à utiliser.

```
1 ma_courbe.clf()
```

#### 4.5.2.8 Transformer un graphique en image

Pour transformer un graphique en image, c'est relativement simple: il suffit d'utiliser la méthode **savefig**.

Cette méthode peut prendre plusieurs paramètres. Nous ne verrons ici que la version simplifiée.

```
1 ma_figure.savefig(" /home/ag/mon_graphe.PNG ")
```

***Remarque: Les formats gérés pour l'export sont le png, le pdf, le ps, l'eps et le svg.***

***Attention: Lorsque vous voulez générer des graphiques en masse, n'oubliez jamais de supprimer chaque figure avant de passer à la suivante, car sinon votre mémoire se remplira rapidement. Utilisez pour cela pyplot.close(ma_figure)***

### 4.5.3 Exemple

```
1  #!/usr/bin/PYTHON
2  # -*-coding:utf-8 -*
3
4
5
6
7  from matplotlib.figure import Figure
8  import matplotlib.pyplot as plt #Remplace un conteneur GTK
9
10
11
12
13 def f_plot():
14     """
15     Cree et affiche une figure
16     """
17     ma_figure = plt.figure() #Implemente une interface basique autonome
18     ma_courbe = ma_figure.add_subplot(111)
19     ma_courbe.plot([1,2,3],[1,2,3], 'go--', label='Test1', linewidth=2)
20     plt.show()
21
22
23
24
25 if __name__ == '__main__':
26     f_plot()
```

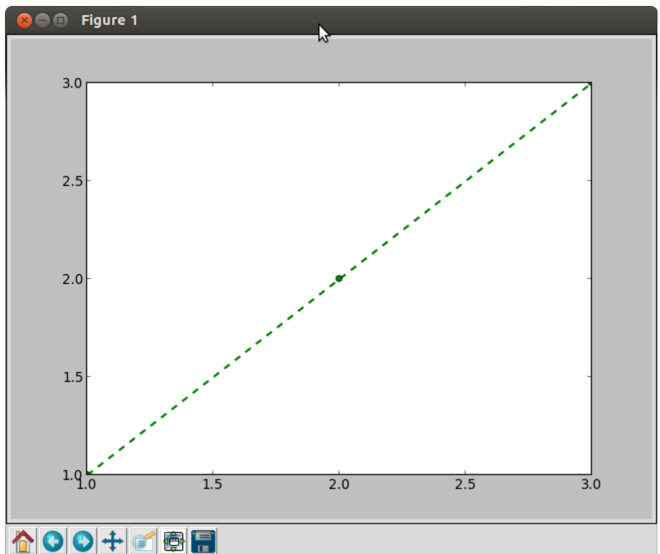



***Illustration 3: Resultat du code exemple de matplotlib***

## 4.6 Les bases de données

Lorsque l'on développe des applications, il est très pratique de pouvoir stocker simplement les paramètres de configuration.

Deux choix s'offrent alors à nous: utilisation d'un simple fichier texte ou l'utilisation d'une **base de données**.

C'est cette dernière solution que nous allons aborder ici.

### 4.6.1 Présentation rapide

Qu'est-ce qu'une base de données (BDD)? D'après wikipedia, c'est " un lot d'informations stockées dans un dispositif informatique ".

En informatique, la BDD est gérée par ce qu'on appelle un SGBD (Système de Gestion de Base de Données). Parmi les plus couramment utilisées de nos jours, nous trouvons **PostGreSQL**, **SQLite**, MySQL ou encore Oracle.

Les SGBD et les BDD sont un sujet très vaste et nous allons tâcher de rester assez simples, et ne verrons donc que les bases.

#### 4.6.1.1 Composants

Une BDD est constituée de différents éléments.

Tout d'abord la base. C'est elle qui accueille l'ensemble des données. Ces données sont réparties dans des tables. Chaque table est organisée en colonne.

Chacune de ces colonnes, tables, et bases portent un nom. Le nom des colonnes est propre à une table, et le nom des tables est propre à une base. Ainsi, par exemple, 2 tables de même nom ne peuvent coexister dans une même base, mais peuvent exister dans 2 bases distinctes.

#### 4.6.1.2 Fonctionnement générique

Une table peut être comparée à une feuille de Libre Office Calc, où chaque ligne possède un numéro unique, appelé ROW ID. Ce ROW ID est unique pour une ligne donnée, quelle que soit la table, à l'intérieur d'une base.

De plus, chaque table possède un index dit unique. Il peut être comparé à l'index d'un livre, à une exception prêt: l'index unique d'une table (un tuple sur 1 ou plusieurs éléments) ne peut contenir de doublons. Cela afin de garantir, ce qu'on appelle l'unicité de la table.

Quand nous cherchons une information, nous utilisons un langage en grande partie standardisé appelé SQL pour Structured Query Language.

***Remarque: Bien que grandement standardisé, chaque éditeur y va de ses touches personnelles, rendant malheureusement difficile l'inter opérabilité. C'est pourquoi, je vous invite fortement à utiliser des BDD Open Source telles PostGreSQL, ou encore SQLite pour les besoins plus faibles.***

Pour reprendre l'exemple d'un fichier classeur, imaginez que vous avez un fichier Calc nommé Carnet, contenant une feuille nommée Amis, contenant 50 lignes. Chacune de ces lignes correspond à l'un de vos amis et contient leur nom, leur prénom, leur adresse, leur mail, …

Ici, le fichier Calc correspond à la base, la feuille à la table, et le nom des colonnes ... au nom des colonnes. Chacun de vos amis y possède un numéro unique (N° de la ligne)

Maintenant imaginez que vous ayez besoin de l'adresse d'Albert Dupond. Vous vous dites: " je veux l'adresse d'Albert Dupond qui est stockée dans mon fichier carnet.ods, sur la feuille amis". Eh bien en SQL ce sera la même chose, sauf qu'on précise le nom des colonnes: " Je cherche l'adresse de la personne dont le nom est Dupond et le prénom Albert dans mon fichier carnet.ods, sur la feuille amis ".

Nous nous arrêterons là pour les explications sur les BDD. Il s'agit surtout d'une présentation succincte (pour ceux qui ne connaîtrait pas encore le sujet), mais néanmoins nécessaire et suffisante pour suivre le reste de ce point sur les BDD.

Pour apprendre le langage SQL, je vous renvoie vers Internet qui contient de nombreux cours fort bien conçus sur le sujet.

### 4.6.2 PYSQLITE

La base **SQLite** est une base de données extrêmement simple. Intégrée par défaut à **PYTHON**, elle présente cependant l'inconvénient de ne pas être multi user.

De fait, on la retrouvera plutôt au sein d'un programme autonome pour stocker les données. A titre d'information, c'est notamment souvent une base de données utilisée dans les appareils mobiles type smartphone.

Pour se connecter à une base **SQLite**, il faut importer le module sqlite3, qui s'installe en même temps que l'interpréteur **PYTHON**.

```
1 import sqlite3
```

Pour créer/manipuler des bases **SQLIte**, je conseille l'excellent **SQLite Manager**, PLUG IN de Firefox.

#### 4.6.2.1 Connexion

Pour se connecter à une base **SQLite**, on va définir le chemin du fichier **SQLite**.

```
1 ma_base = sqlite3.connect("./BASE.sqlite")
```

### 4.6.2.2 Exécuter une requête

#### 4.6.2.2.1 Curseur

La première chose à faire après s'être connecté à une base est de créer un curseur. Il s'agit d'une sorte de zone tampon entre notre programme et la base.

```
1 mon_curseur = ma_base.cursor()
```

Pour exécuter une requête SQL, rien de compliqué là non plus. On utilise la méthode **execute**, avec la possibilité de passer des paramètres dynamiques.

Imaginons que nos variables mon_nom et mon_prenom varient en fonction de la saisie utilisateur. Pour prendre en compte la saisie dans la requête on utilisera la syntaxe suivante

```
1 mon_curseur.execute("SELECT * \
2     FROM MA_TABLE \
3     WHERE NOM = ? \
4     AND PRENOM = ?",\
5     (mon_nom, mon_prenom))
```

Comme on peut le voir, nous avions ici deux paramètres dynamiques pour la requête. Nous les avons passés dans l'ordre, grâce au " ? ". Attention toutefois de ne pas oublier de les mettre entre parenthèses.

Le \ en fin de ligne, lui, permet d'indiquer que nous continuons l'instruction à la ligne.

#### 4.6.2.2.2 Récupération des résultats

Une fois l'interrogation effectuée, le résultat est disponible dans le curseur.

Pour l'exploiter pleinement, il est cependant préférable de transférer le résultat dans une liste temporaire, juste après la requête.

```
ma_liste = []
for r in mon_curseur:
    ma_liste.append(r[0])
```

Cette méthode fonctionne très bien si on n'attend qu'une seule colonne en retour. Mais dans le cas où on en attend plusieurs, on sera vite confronté au problème de savoir quoi correspond à quoi.

Pour y pallier, nous utiliserons donc le **row_factory**. En association avec sqlite3.Row, nous pouvons non seulement différencier les différentes colonnes, mais en plus connaître le nom des-dites colonnes, si ces dernières nous sont inconnues.

Pour connaître le nom des colonnes, nous procédons de la façon suivante:

```
ma_base = sqlite3.connect("./BASE.sqlite")
ma_base.row_factory = sqlite3.Row
mon_curseur = ma_base.cursor()
mon_curseur.execute("SELECT * FROM MA_TABLE)
r = mon_curseur.fetchone()
mon_listing = r.keys()
cur.close()
```

Ici, mon_listing est une pseudo liste contenant le nom des colonnes.

Pour récupérer les données d'une colonne dont nous connaissons le nom, tel que dans un logiciel à la recherche d'une donnée, un âge dans notre exemple:

```
for r in mon_curseur:
if r["Age"] == None:
    age = 0
else:
    age = int(r["Age"])
```

Dans la ligne 5, nous effectuons un transtypage afin de nous assurer que âge sera toujours un int.

#### 4.6.2.3 Sauvegarde

Comme indiqué plus haut, une mise à jour en base n'est effective que lorsque l'exécution **commit** a été exécutée.

```
ma_base.commit()
```

#### 4.6.2.4 Déconnexion

Enfin, une fois nos interrogations terminées, il ne reste qu'à se déconnecter. Pour cela, il suffit de fermer le curseur

```
mon_curseur.close
```

### 4.6.3 PSYCOPG

PSYCOPG est un module d'interfaçage entre **PYTHON** et **PostGreSQL**.

Ce SGBD, en plus d'être **Open Source**, est certainement l'un de ceux qui respecte le plus les normes SQL.

**PSYCOPG** nous permettra de dialoguer au mieux avec ce type de BDD, et le plus simplement possible.

```
Import psycopg2
```

#### 4.6.3.1 Connexion

La connexion à une base est extrêmement simple:

```
ma_connection=psycopg2.connect(database='test',user="name", \
                    password='mdp',host=127.0.0.1, \
                    port=5432) #par defaut
```

Vous voyez ici la forme complète. host et port peuvent être facultatifs selon votre utilisation. Le port 5432 est le port par défaut.

Afin de respecter le principe de **PYTHON** selon lequel " explicite est mieux qu'implicite ", je vous invite à toujours utiliser la forme complète.

### 4.6.3.2 Exécuter une requête

#### 4.6.3.2.1 Curseur

Une fois connecté à la base, l'exécution d'une requête est semblable à **SQLite**. Il faut d'abord passer par un curseur avant de pouvoir exécuter la requête.

```
1 mon_curseur = ma_connection.cursor()
2 mon_curseur.execute("SELECT %s FROM DUAL", ('NULL',))
```

Quelques explications concernant la ligne 2.

Nous voyons que nous avons utilisé un %s. Contrairement au langage C, il n'existe pas de %d, %f, … On n'utilisera donc uniquement que le %s, même si on doit passer la valeur 42 par exemple.

Les paramètres seront toujours sous la forme de tuples, ce qui explique que nous ayons "('NULL',)".

Nous avons également la possibilité de former la requête avant de l'exécuter grâce à la méthode mogrify:

```
1 requete = mon_curseur.mogrify(" SELECT %s FROM DUAL ", ('NULL',))
```

Ici, requête vaudra " SELECT NULL FROM DUAL "

#### 4.6.3.2.2 Récupération des résultats

```
1 a = mon_curseur.fetchone()
```

Nous récupérons ici le résultat de la requête.

Cela nous renvoie une seule ligne à la fois. Lorsque nous avons atteint la fin, nous récupérons la valeur None. Les résultats sont renvoyés sous forme de tuples.

Si l'on désire récupérer tous les résultats d'un coup, il faut utiliser la commande **fetchall()**.

#### 4.6.3.2.3 Lecture de la dernière requête exécutée

Pour connaître la dernière requête ayant été exécutée, il faut passer par la méthode **QUERY**:

```
1 last_requete = mon_curseur.query
```

#### 4.6.3.2.4 Procédures

Afin de faciliter un traitement, il est possible d'écrire des procédures **PostgreSQL** et de simplement les appeler au sein du script **PYTHON**.

Pour les appeler, il faut utiliser la méthode **callproc**:

```
1 ma_curseur.callproc(ma_procédure(mes_paramètres))
```

#### 4.6.3.2.5 Gestion d'erreurs

Il est possible de gérer simplement les erreurs sur une base **PostgreSQL**.

```
1 try:
2     mon_curseur.execute(" Select * from test ")
3 except Exception, e:
4     pass
5 print e.pgcode, ': ', e.pgerror
```

Ligne 5, nous afficherons le code erreur **PostGreSQL**, suivi de la description de l'erreur engendrée.

#### 4.6.3.3 Sauvegarde

Tout comme dans **SQLite**, la sauvegarde s'effectue via un commit:

```
1 ma_connection.commit()
```

#### 4.6.3.4 Déconnexion

Pour se déconnecter, la procédure est relativement simple. On commence par clore les curseurs ouverts, puis par clore la connexion à la base.

```
1 mon_curseur.close()
2 ma_connection.close()
```

***Remarque: Pour savoir si un curseur est clos ou non on peut utiliser mon_curseur.closed().***

## 4.7 Le module OS

Nous n'allons pas parler ici d'Open Source, ni de système d'exploitation, même si un lien existe.

En effet, de module sert à s'interfacer avec le **système d'exploitation** afin d'effectuer des opérations précises.

De nombreuses possibilités sont offertes par ce module, mais nous n'en verrons que quelques unes, celles qui nous servent le plus souvent.

```
1 import os
```

### 4.7.1 Taille d'un fichier

Pour connaître la taille d'un fichier nous utilisons **os.path.getsize()**.

```
1 taille_fichier = os.path.getsize(" ./mon_fichier.txt ")
```

### 4.7.2 Renommage d'un fichier

Pour renommer un fichier, nous utilisons **os.renames()**.

```
1 os.renames(" ./ancien_nom.txt ", " ./nouveau_nom.txt ")
```

***Remarque: Si le dossier stipulé dans le chemin de renommage n'existe pas, alors il sera créé.***

## 4.8 Scripting

Pour faire du scripting, nous allons utiliser les modules **sys** et **os**. Ces modules sont très utile en programmation **PYTHON** car ils permettent de réaliser nombre d'interactions avec la machine et l'OS.

Nous ne détaillerons pas dans ce livre les possibilités offertes par ces modules, car trop vaste, mais vous pourrez trouver beaucoup d'informations sur Internet.

```
1 import sys
2 import os
```

### 4.8.1 Les arguments

#### 4.8.1.1 Passage d'argument

Pour passer des **arguments**, nous utilisons l'attribut **sys.argv**. Cet attribut est une liste des arguments passé en paramètres.

Imaginons un script nommé script.py. Quand nous l'appelons en lui passant des arguments cela pourrait donné cela:

```
1 PYTHON script.py arg1 arg2 arg3 " arg4   arg5 "
```

Dans ce cas, sys.argv donnerait le retour suivant:

```
1 ['script.py', 'arg1', 'arg2', 'arg3', 'arg4   arg5']
```

Comme nous pouvons le voir la distinction entre les différents arguments est réalisée grâce aux espaces. Cependant, il est possible de passe un argument contenant des espaces, à condition de le mettre entre double quotes.

***Attention: Le premier argument (sys.argv[0]) est toujours le nom du programme.***

#### 4.8.1.2 Tester le nombre d'argument

Pour connaître le nombre d'arguments passés en paramètre, il suffit d'exécuter un len sur l'attribut.

```
1 >>>#sur notre exemple cela donne:
2 >>>len(sys.argv)
3 5
```

### 4.8.2 L'exécution de commande

Nous allons ici utilisé le module **os**, et plus précisément la méthode **popen()**. Cette méthode va prendre en paramètre la **commande** à exécuter (au format string), et vous renverra le résultat une fois la commande exécutée.

Le résultat, vous est renvoyé lui aussi sous forme d'une string. Il ne vous reste plus qu'a le lire grâce la méthode **read()**.

***Attention: La méthode read vous renvoie le contenu de la variable puis la réinitialise. Pensez donc à utiliser une variable de type string afin de récupérer le résultat de la commande.***

```
1 >>> ma_commande = os.popen("ls")
2 >>> ma_variable_string = ma_commande.read()
3 >>> ma_commande.read()
4 ''
5 >>> #la variable a été reinitialise
6 >>> ma_variable_string
7 'b\nBureau\nDocuments\nImages\nMod\xc3\xa8les\nMusique\nPublic\n'
```

### 4.8.3 Exemple

```
1 #!/usr/bin/PYTHON
2 # -*-coding:utf-8 -*
3
4
5
6
7 import sys
```

```
8  import os
9
10
11
12
13 def main():
14     """
15     Prend 2 arguments, les affichent à l'ecran, puis les enregistre.
16     """
17
18     nom = sys.argv[1]
19     prenom = sys.argv[2]
20
21     ma_commande = os.popen("ls -l /home/steph")
22     result = ma_commande.read()
23
24     print "programme ", sys.argv[0]
25     print "bonjour ", prenom, nom
26     print "contenu de /home/steph: "
27     print type(result)
28
29     file("/home/steph/log.txt","w")
30
31     with open("/home/steph/log.txt", "w") as mon_fichier:
32         mon_fichier.write(result)
33         mon_fichier.close()
34
35     print "Synthese sauvegardee dans log.txt"
36
37
38
39
40 if __name__ == '__main__':
41     main()
```

## 4.9 Les fichiers Zip

Nous allons ici parler du module **zipfile**. Ce module permet de gérer les archives ZIP mono fichier jusqu'à 4 GO.

Le module ici retenu, est loin d’être le seul et le plus performant existant. Cependant, il est très intéressant car le format ZIP est extrêmement répandu.

De plus, il possède une capacité de compression minimale non négligeable.

Enfin, comme nous le verrons dans la partie consacrée à l'**Open Document Format** (ODF), il est très utile pour s'interfacer avec des documents ODF.

```
1 import zipfile
```

### 4.9.1 Création/modification d'un ZIP

La création d'une archive Zip est extrêmement simple. La première étape est la création d'un objet zipfile. Ensuite, nous lui ajoutons les différents fichiers que nous souhaitons.

```
1 import zipfile
2
3 mon_zip = zipfile.ZipFile('mon_arch.zip','w',zipfile.ZIP_STORED)
4 mon_zip.write("fichier1.txt")
5 mon_zip.close()
```

Comme vous pouvez le constater ici, rien de compliqué. Ligne 3, nous passons en paramètre le nom de notre archive (et éventuellement le chemin), le type d'accès (r pour READ, w pour WRITE, a pour APPEND), et le type de compression. Ce dernier paramètre est optionnel et ne peut prendre que deux valeurs: **ZIP_STORED** (par défaut) ou **ZIP_DEFLATED**. Pour utiliser la compression le module ZLIB peut parfois être demandé.

A noter la possibilité d'une autre écriture

```
import zipfile

with zipfile.ZipFile('mon_arch.zip', 'w') as mon_zip:
	mon_zip.write('fichier1.txt')
mon_zip.close()
```

Bien que cette écriture soit plus propre, elle possède néanmoins l'inconvénient de devoir effectuer toutes les opérations d'écriture dans le **WITH**. A vous de voir la méthode la plus pratique dans vos projets.

***Attention: L'écriture dans le ZIP ne s'effectue qu'à l'appel du close()***

L'ajout de fichier à une archive s'effectue simplement en précisant le mode **append** lors de la création de l'objet **ZIPFILE**.

## 4.9.2 Ouverture d'un ZIP

L'ouverture d'un ZIP peut s'interpréter de deux façons différentes: ouverture du ZIP et accès aux données en lecture/écriture, ou bien une décompression complète.

#### 4.9.2.1.1 Ouverture et accès aux données

L'accès d'un fichier contenu dans un ZIP peut être utile, notamment quand on ne désire qu'ajouter un peu de texte à un fichier

Voici comment procéder: création d'un objet zipfile à partir de notre fichier ZIP (accès en lecture/écriture 'mode w)), puis accès en lecture au fichier depuis l'objet **ZIPFILE**, et enfin modification et fermeture du ZIP.

```
import zipfile

mon_zip = zipfile.ZipFile('mon_texte.zip', 'w')
fichier = mon_zip.read('./données.txt') #lecture du fichier
mon_zip.writestr('données.txt', b'ajout de données\n')	#on convertit en binaire
mon_zip.close()
```

***Remarque: Nous voyons ici que nous écrivons en mode binaire (wb). Il faut savoir en effet que les données dans un ZIP sont au format binaire.***

#### 4.9.2.1.2 Décompression complète

La décompression s'effectue de la manière suivante: nous créons un objet **ZIPFILE** avec un accès en lecture. Puis, pour chaque fichier/dossier trouvé, nous l'écrivons sur le disque au chemin désiré.

```
1  import zipfile
2
3  mon_zip = zipfile.ZipFile('mon_arch.zip','r')
4  for fichier in mon_zip.namelist():
5      contenu = mon_zip.read(fichier)
6      fichier_sortie = open(fichier, 'wb')
7      fichier_sortie.write(contenu)
8      fichier_sortie.close()
9  mon_zip.close()
```

Une autre écriture possible, plus directe, est la suivante:

```
1  import zipfile
2
3  mon_zip = zipfile.ZipFile('mon_arch.zip','r')
4  mon_zip.extractall('./') #on passe en parametre le chemin desire
5  mon_zip.close()
```

## 4.10 Le XML

**XML** n'est pas un langage de programmation mais un langage de description. Plus précisément, il s'agit d'un langage de balisage, dit générique.

De balisage, car il utilise des balises pour délimiter des zones. Générique car exceptées quelques règles, dont nous verrons les principales, chacun est libre de ses actions.

Ainsi, contrairement au **HTML** (autre langage de balisage), par exemple, nous donnons aux balises le nom que nous désirons.

De plus, de nombreux modules permettent de s'interfacer extrêmement facilement avec ce type de fichiers.

Ces éléments font que le **XML** est extrêmement apprécié dans le milieu informatique. En effet, c'est le langage par excellence qui permet à deux applications de s'interfacer sans difficultés.

La seule chose à connaître est la structure du **XML**. Cette structure est d'ailleurs ce qui permet de vérifier qu'un fichier **XML** est correct.

### 4.10.1 Codage

#### 4.10.1.1 L'entête

L'entête d'un **XML** est très simple et en voici un exemple que nous allons analyser. Cette entête s'appelle un **PROLOGUE**.

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

L'élément suivant est donc le langage utilisé (xml). Nous précisons ensuite la version. A l'heure actuelle seule la version 1 est vraiment utilisée, donc pas de questions à se poser. Enfin, le type d'encodage du fichier. Je vous conseille l'UTF-8, standard européen. Vous trouverez plus d'informations sur l'encodage de caractères sur Wikipedia.

Donc, comme vous pouvez le constater, rien de sorcier ici non plus.

#### 4.10.1.2 Le corps

Le corps du fichier même est organisé en arbre. Par arbre, j'entends ramification. Ainsi, chaque balise peut avoir un certain nombre de sous balises.

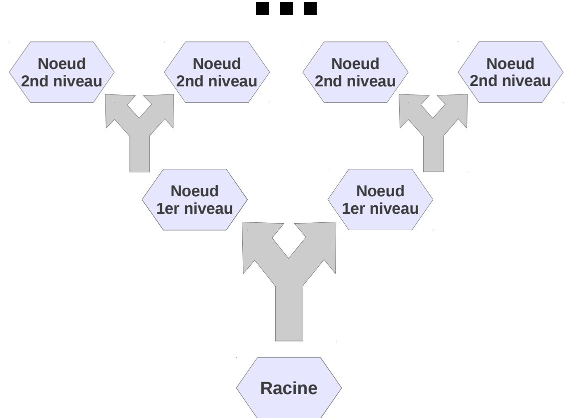



***Illustration 4: Structure en arbre d'un fichier XML***

#### 4.10.1.3 Exemple

Ci-dessous un exemple d'un annuaire codé en **XML**.

```
1  <?xml version= "1.0 " encoding= "utf-8 "?>
2  <annuaire>
3      <personne dpmt= "sciences ">
4          <nom> Dupond </nom>
5          <prenom> Jean-Baptiste </prenom>
6          <tel> 3637 </tel>
7      </personne>
8      ...
```

```
9       <personne dpmt= "langue ">
10          <nom> Martin </nom>
11          <prenom> Michel </prenom>
12          <tel> 5354 </tel>
13      </personne>
14  </annuaire>
```

## 4.10.2 Les principes de base

Nous allons voir ici les grands principes de base du **XML**. Certaines spécificités ne seront pas décrites ici. Ce n'est donc pas parce que vous ne trouverez pas d'informations sur ce que vous essayez de faire, que ce n'est pas possible. Internet est votre ami.

### 4.10.2.1 Les commentaires

En **XML**, les commentaires se font très simplement, avec des " <!-- " pour les commencer et un " --> " pour les clore. Cela permet d'avoir des commentaires sur plusieurs lignes.

```
1  <!--  Ceci est un commentaire  -->
```

### 4.10.2.2 La racine et les nœuds

La racine, aussi appelée nœud document, est la base du document **XML**.

Pour prendre une comparaison, un nœud peut être comparé à un dossier ou à un sous-dossier. Le nœud document, lui, est comparable à la racine système (" / " sous Linux)

### 4.10.2.3 Les balises

Les balises permettent de délimiter des zones au sein de l'arbre **XML**. La différence entre une balise ouvrante et une balise fermante tient à un caractère.

```
<ma_balise> <!-- balise ouvrante -->
    ...
</ma_balise> <!-- balise fermante -->
```

Dans le cas où il n'y aurait pas de données entre les balises ouvrantes et fermantes on peut utiliser un raccourci.

```
<ma_balise/>
```

***Remarque: Dans les faits, je vous recommande d'utiliser exclusivement la première syntaxe, qui sera dans tous les cas plus lisible, et conviendra à toutes les situations.***

#### 4.10.2.4 Chevauchement

Les balises n'ont pas le droit de se chevaucher.

```
<balise1><balise2> ... </balise1></balise2> <!-- Ceci est interdit -->
<balise1><balise2> ... </balise2></balise1> <!-- Ceci est la bonne syntaxe -->
```

#### 4.10.2.5 Les attributs

Les attributs, en **XML**, sont des informations complémentaires que l'on insère dans les balises.

Leur intérêt, par rapport à des sous-balises, est de potentiellement permettre une recherche plus rapide, d'un point de vue **XPATH**.

```
<Dupont region= "Bretagne " dpmt= "35 "> <!-- Présence ici de deux attributs -->
    ...
```

#### 4.10.2.6 Les instructions de traitement

Les instructions de traitement sont spécifiquement dédiées aux divers traitements de l'information contenue dans le fichier **XML**. Ces instructions commencent par un " <? " et finissent par un " ?> ".

L'instruction de traitement la plus courante en **XML** est l'entête.

#### 4.10.2.7 Le XPATH

Le **XPATH**, correspond à un chemin permettant d'accéder à une donnée bien précise (nœud). Il peut être assimilé à un chemin de fichier complet dans Linux.

Ainsi, si nous reprenons notre exemple de l'annuaire, le **XPATH** nous donnant le nom des personnes dans le fichier sera le suivant:

```
1 /annuaire/personne/nom
```

On dit qu'on accède aux nœuds " **nom** ". Cela nous ramène cependant l'ensemble des noms, et pas un en particulier. On aurait également pu utiliser " //**nom** ", mais ce type de notation est ambigu. En effet, il n'y a aucune description des balises mères, ce qui peut provoquer quelques soucis.

Voyons un peu plus en détail les possibilités offertes par le **XPATH**.

##### 4.10.2.7.1 Sélection d'un nœud précis

Il existe différentes façons de définir un nœud précis. Cependant, toutes sont liées à la position du nœud dans le fichier.

***Remarque: Ici la numérotation commence à 1 et non à 0.***

```
1 /annuaire/personne[2] <!-- Récupère la seconde personne du fichier -->
2 /annuaire.personne[position()<10] <!-- Récupère les 9 premières personnes -->
3 /annuaire/personne[nom=" Dupond "] <!-- Récupère les nœuds ou le nom vaut Dupond -->
```

Outre cela, nous pouvons également faire des recherches avec conditions, notamment au niveau d'un contenu partiel. Ainsi, si nous recherchons toutes les personnes dont le nom, ou le prénom, est ou contient " Jean ", nous aurons comme **XPATH**:

```
1 /annuaire/personne[contains(., " Jean ")]
```

#### 4.10.2.7.2 Sélection avec attributs

Si l'on désire chercher uniquement les personnes du dpmt science, le **XPATH** sera le suivant:

```
1 /annuaire/personne[@dpmt= "sciences "]
```

Si l'on désire récupérer les différentes valeurs des attributs dpmt, le **XPATH** sera le suivant:

```
1 /annuaire/personne/@dpmt
```

Nous pouvons effectuer également des recherches via les attributs. Par exemple, si nous voulons récupérer le nom de toutes les personnes du dpmt sciences:

```
1 /annuaire/personne[@dpmt=" sciences "]/nom
```

#### 4.10.2.7.3 Récupérer toutes les données d'un nœud

L'ensemble des données comprises dans un nœud peuvent être récupérées grâce au **XPATH** suivant, par exemple:

```
1 /annuaire/personne[1]/*
```

#### 4.10.2.7.4 Combinaison de XPATH

Il est possible de combiner des recherches afin de ne récupérer qu'une partie des informations d'un nœud. Il faut utiliser le caractère pipe, " | " (touches alt-gr + 6)

Ainsi, si nous ne désirons que le nom et le téléphone, nous aurons:

```
1 /annuaire/personne/nom | /annuaire/personne/tel
```

#### 4.10.2.7.5 En bref...

Comme nous venons de le voir rapidement ensemble, accéder à des informations d'un fichier **XML** est assez simple.

De plus il est possible de combiner les différentes méthodes de recherche afin d'affiner le nœud ciblé.

Cependant, précisons qu'un fichier **XML** n'est pas une BDD. Et cela, même si certaines BDD se basent sur des fichiers **XML**.

## 4.10.3 Le module LXML

Maintenant que nous avons vu les grands principes de base du **XML**, nous allons nous attaquer à l'interfaçage d'un fichier **XML** avec **PYTHON**. Comme vous le verrez, il n'y aura rien de bien difficile ici non plus.

Il existe en **PYTHON** deux façons d'aborder un fichier **XML**: de manière événementielle, ou par arborescence. Notre choix retenu ici est le type arborescence.

Notre choix est justifié par la volonté de coller le plus possible à la structure en arbre, définissant un fichier **XML**. Nous allons étudier le module **LXML**.

Ce module est un dérivé de librairie C. Il faut donc installer **lxml**, **libxml2**, et **libxslt**. Vous pouvez passer par Synaptic ou tout autre gestionnaire de paquets pour vous simplifier la vie sur Linux.

### 4.10.3.1 Utilisation de LXML

Nous n'utiliserons pas la totalité des possibilités offertes par le module. Nous allons ici nous concentrer sur **ETREE**.

**ETREE**, pour Element Tree, se rapporte comme son nom l'indique au principe d'arborescence en arbre. Grâce à cela, nous allons utiliser le fichier **XML** comme une source de données navigables.

***Remarque: Notez bien ici que j'ai utilisé l'expression " source de données " et non " base de données ". Un fichier XML, pour rappel, n'est nullement une BDD.***

```
1  from lxml import etree      #import du module
2
3  xml_file = etree.parse(" ./annuaire.xml ")       #connexion au fichier XML
```

#### 4.10.3.2 Création d'un fichier XML

La création d'un fichier **XML** diffère un peu de la création d'un fichier classique.

En effet, nous allons ici procéder en deux étapes. Tout d'abord nous allons créer la structure de notre fichier **XML**. Ensuite, nous écrirons cette structure au sein même d'un fichier.

Pour la création de la structure, il faut être particulièrement minutieux et respecter un ordre précis, telle l'arborescence d'un arbre.

Ainsi, la première chose à créer est la racine de l'arbre. Ensuite, nous créons les nœuds permettant de générer chaque branche de l'arbre. Enfin, pour finir, nous nous assurons de la bonne indentation du fichier.

```
1  from lxml import etree
2
3  racine = etree.Element(" annuaire ")             #Creation de la racine
4  personne1 = etree.SubElement(racine, " personne ")       #Creation d'un nœud
5  personne1.set(" dpmt ", " sciences ")            #Ajout d'un tag
6  nom1 = etree.SubElement(personne1, " nom ")
7  nom1.text = " Dupond "      #Initialisation de la valeur du noeud
8
9  file_str = etree.tostring(racine, pretty_print = True)       #Convertion en string
10 #ecriture dans un fichier texte
11 ...
```

Analysons un peu plus cet exemple.

Ligne 3, nous créons notre racine, en lui passant le nom de la balise en paramètre.

Ligne 4, nous créons une sous-balise, en passant en paramètre la balise mère, ou nœud père, ainsi que le nom de la balise.

Ligne 5, nous ajoutons un tag sur le même principe. Les paramètres sont le nom du tag et sa valeur.

Ligne 7, nous initialisons la valeur du nœud.

Ligne 9, nous convertissons notre structure, en chaîne de caractères, en précisant que nous voulons que la chaîne soit bien formatée, avec indentations. Pour cela, nous utilisons le paramètre **pretty_print**.

A partir de la ligne 10, il ne nous reste alors plus qu'à écrire la chaîne de caractères dans un fichier texte, à l'extension **.XML**.

#### 4.10.3.3 Lecture d'un fichier XML

Une fois connecté au fichier **XML**, tel que vu précédemment, nous pouvons naviguer au sein des données grâce au **XPATH**. Nous le ferons grâce à un **FOR**.

Un fois dans la boucle, nous utiliserons alors diverses méthodes afin de récupérer les informations qui nous intéressent.

```
1 for balise in xml_file.xpath(" /annuaire/personne "):
2     nom = nalise.xpath(" nom ")
3     print nom[0].text
```

Ci-dessous les principales diverses méthodes qui peuvent vous servir.

| Méthodes | Utilité |
|---|---|
| **.items()** | Récupère la liste des attributs |
| **.text** | Récupère le texte du nœud visé |
| **.tag** | Récupère le nom de la balise, du nœud |
| **.get(" attribut ")** | Récupère la valeur de l'attribut passé en paramètre |

#### 4.10.3.4 Modification d'un fichier XML

##### 4.10.3.4.1 Serialisation et deserialisation

Qui dit **XML**, dit **sérialisation** et **deserialisation**. Derrière ces deux termes se cache un principe fort simple: la conversion d'une chaîne de caractères vers un objet (désérialisation) ou depuis un objet (sérialisation)

Il existe en effet de nombreux modules capables d'effectuer des traitements sur les chaînes.

Convertir un objet en chaîne de caractères permet ainsi de simplifier les traitements.

***Remarque: Dans notre cas, pour sérialiser un objet de parse, il faut utiliser la méthode .write(<fichier>).***

##### 4.10.3.4.2 Méthode générale

Modifier un fichier **XML** n'est pas tellement plus compliqué qu'une création. Nous allons ici repartir sur notre annuaire **XML** d'exemple. Place au code que nous commenterons.

```
1  #!/usr/bin/env PYTHON
2  from lxml import etree
3  
4  xml_file=etree.parse("./annuaire.xml")
5  racine=etree.Element("annuaire")
6  personne=etree.SubElement(racine,"personne")
7  prenom=etree.SubElement(personne,"prenom")
8  for balise in xml_file.xpath("/annuaire/personne[nom=\"Dupond\"]"):
9        prenom = balise.xpath("prenom")
10       prenom[0].text="Charles"
11 
12 body = etree.tostring(xml_file, pretty_print = True)
13 entete = "<?xml version=\"1.0\" encoding=\"utf-8\"?>"
14 contenu = entete + "\n" + body
15 
16 mon_fichier = open("./annuaire.xml", "w")
17 mon_fichier.write(contenu)
```

Ligne 2, nous importons notre module.

Ligne 4, nous nous connectons à notre fichier **XML**, puis nous créons une liaison sur la racine, ligne 5.

Ligne 6 et 7, nous créons d'autres références aux sous balises personnes et prénom.

Ligne 8 à 10, nous recherchons dans les balises personnes, celles dont le nom est Dupond, afin de changer leur prénom en Charles.

Ligne 12, nous convertissons le fichier **XML** en chaîne de caractères.

Ligne 13, nous créons l'entête d'un fichier **XML**

Ligne 14, nous assemblons les deux parties précédentes afin de créer le corps de texte du fichier **XML**.

Ligne 16 et 17, enfin, nous générons/mettons à jour le fichier **XML**.

## 4.11 L'Open Document Format

Wikipedia dispose d'un très bon article sur l'**Open Document Format**, **ODF** en abrégé. Je vous invite à le lire en entier pour bien comprendre ce qu'il en retourne.

Pour ceux qui ne pourraient pas accéder à Wikipedia, nous allons résumer de manière succincte les grandes lignes de l'article Wikipedia.

### 4.11.1 Historique

**ODF** est un standard ouvert définissant des formats de bureautique. S'étant basé sur les formats d'Open Office, il en a repris les extensions en trois lettres (od pour les documents ou ot pour les templates, suivi d'une lettre indiquant le type de document).

Après de nombreuses années de mise au point, ce standard est certifié ISO en 2006 et de nombreuses sociétés ainsi que plusieurs pays ont déjà pris comme résolution de ne plus utiliser que ces formats bureautiques.

### 4.11.2 Les différents types de documents

Il existe plusieurs types possibles: texte (odt), classeur (ods), présentation (odp), graphique (odg). Il ne s'agit là que des principaux.

Ces formats sont des documents à part entière. Mais le standard **ODF** offre également la possibilité pour ces principaux types de réaliser des modèles, aussi appelés templates, aux extensions ott, ots, otp et otg.

### 4.11.3 Les templates

Ces **templates** permettent de réaliser une mise en page type, puis de venir remplacer différents champs du template par les valeurs désirées.

Ainsi, une sortie en fichier classeur, par exemple, se déroule en 2 étapes:

1-Création du fichier template avec des balises type

2-Copie du template et remplacement des balises par des données

### 4.11.4 Description du format ODF

Le format **ODF** n'est en soi qu'un simple fichier ZIP, contenant des fichiers **XML**, des fichiers de configuration et des dossiers.

En théorie, il suffit donc simplement de dézipper un fichier **ODF** afin d'accéder directement au contenu du fichier sans avoir à l'ouvrir.

### 4.11.5 Interaction avec PYTHON

A ce stade, il y a deux façons d'interagir avec des fichiers **ODF**: une simple modification de template ou une interaction totale.

### 4.11.6 Modification de template

#### 4.11.6.1 Théorie

Dans le cas d'une simple modification de template (par exemple, générer une fiche de données avec un format précis), nous allons utiliser la méthode directe:

1-Modification de l'extension en .ZIP
2-Dézippage
3-Modification des éléments désirés
4-Rezippage
5-Modification de l'extension au format **ODF**

Au niveau de la modification d'un template, ce qui va nous intéresser plus particulièrement ce sera **content.xml**, **meta.xml** et le dossier **Pictures**.

**meta.xml** contient toutes les informations relatives au document: application, titre, description, sujet, mots-clés, auteur, date de création, modèle utilisé, …

**content.xml** est le fichier qui contient le corps du document, autrement dit son contenu entier.

Le dossier **Pictures**, lui, contient les images du document, auxquelles fait référence **content.xml**.

La méthode directe consiste à fouiller dans les fichiers **XML** à la recherche de nos balises (par exemple MON_NOM, MON_PRENOM, …).

Ces balises doivent avoir un nom bien particulier et surtout unique au sein du fichier de manière a être sûr de les repérer simplement.

Une fois repérées, il suffit juste de les remplacer par les valeurs désirées.

Idem pour les images, qui sont prises entre des balises **XML** <Pictures>. Il suffit juste de remplacer la référence de l'image entre ces balises **XML**, puis de déposer l'image dans le dossier **Pictures**.

Ainsi, en partant d'un fichier odt ou ods, il est très facile de créer des templates de rapport très simplement en substituant simplement texte et image à notre convenance.

***Remarque: Il existe également des formats de fichiers TEMPLATE en ODF (ott, ots, …). Ces formats sont spécifiques à ODF, et visent plutôt à être modifiés via Libre Office (par exemple), ou via une méthode d'interaction complète.***

### 4.11.7 Interaction complète

L'interaction complète est très pratique lorsque vous souhaitez créer un document complet de zéro et de manière totalement dynamique (contrairement aux templates qui sont statiques).

Pour cela il faut utiliser des modules spécifiques. Nous verrons ici ensemble les bases de **EZODF**.

Comme d'habitude, nous ne prétendons nullement nous substituer à la documentation officielle. Les strictes bases pour l'odt et l'ods seront vues ici, rien de plus.

Il s'agit avant tout de vous faire une présentation succincte mais fonctionnelle de l'utilisation de ce package.

Vous serez la plupart du temps confronté à l'utilisation de template, plutôt qu'à la création de fichier **ODF** de toute pièce. Pour votre information personnelle, même si cela ne sera pas vu ici, sachez que **EZODF** sait très bien gérer les templates **ODF**.

L'utilisation de template fera l'objet de l'exemple général de cette partie sur **l'ODF**.

```
1 from ezodf import newdoc
```

#### 4.11.7.1 Création et ouverture d'un document ODF

Le constructeur est identique quel que soit le document:

```
1 mon_ods = newdoc(doctype='ods', filename='mon_ods.ods')
```

Cela se passe de tout commentaire, au vu de la simplicité. Les paramètres sont le type de document à créer, ainsi que le nom du fichier.

Si nous avons vu comment créer un document de toute pièce, il peut être utile aussi de savoir ouvrir un document existant pour le modifier.

```
mon_odt = ezodf.opendoc('mon_odt.odt')
```

Ici, seul le nom du document à ouvrir est à fournir.

#### 4.11.7.2 ODT

Voici l'exemple officiel pour créer un ODT basique:

```
from ezodf import newdoc, Paragraph, Heading

odt = newdoc(doctype='odt', filename='text.odt')
odt.body += Heading("Chapter 1")
odt.body += Paragraph("This is a paragraph.")
odt.save()
```

Avant de voir plus en détail les principaux éléments pour un ODT, nous allons commenter ce petit exemple.

Ligne 1, nous importons de quoi créer un nouveau document et de quoi créer un titre et un paragraphe. Il s'agit ici de styles prédéfinis.

Ligne 3, nous créons notre document ODT.

Ligne 4, nous ajoutons au corps de notre ODT, un titre en lui passant en paramètre le texte.

***Remarque: Ce package permet l'utilisation au choix de " .body += " ou de " .body.append(" . A vous d'utiliser celui que vous préférez.***

Ligne 5, nous ajoutons à notre corps de document un paragraphe en passant en paramètre le texte.

Ligne 6 nous sauvegardons notre document.

Pour effectuer un saut de page, il suffit d'utiliser:

```
odt.body += SoftPageBreak()
```

### 4.11.7.3 ODS

Voici l'exemple officiel pour la création d'un ODS:

```
1  from ezodf import newdoc, Sheet
2  
3  ods = newdoc(doctype='ods', filename='spreadsheet.ods')
4  sheet = Sheet('SHEET', size=(10, 10))
5  ods.sheets += sheet
6  sheet['A1'].set_value("cell with text")
7  sheet['B2'].set_value(3.141592)
8  sheet['C3'].set_value(100)
9  
10 pi = sheet[1, 1].value
11 ods.save()
```

Ligne 1, nous importons de quoi générer notre ODS.

Ligne 3, nous créons notre fichier CALC.

Ligne 4, nous créons une feuille CALC, en passant en paramètres le nom de la feuille et la taille désirée (lignes puis colonnes)

Ligne 5, nous ajoutons notre feuille à notre fichier CALC.

Ligne 6 à 8, nous remplissons plusieurs cellules de la feuille.

Ligne 10, nous récupérons une valeur dans une cellule. Otez que nous comptons à partir de 0. A0 est donc (0,0), B2 est (1,1), ...

Ligne 11, nous sauvegardons notre ODS.

Parmi les autres actions possibles au niveau d'une feuille ODS, nous retrouvons, entre autre les suivantes:

```
1  sheets = mon_ods.sheets  #recupere le nom de toutes les feuilles de l'ODS
2  nom = sheets[0].name              #recupere le nom de la 1ere feuille
3  count = len(sheets)       #recupere le nombre de feuilles dans l'ODS
4  rowcount = sheets[0].nrows()      #recupere le nombre de lignes de la feuille
5  colcount = sheets[0].ncols()      #recupere le nombre de colonnes de la feuille
6  valeur = sheets[0]['C1']    .value #recupere la valeur d'une cellule d'une feuille
7  del sheets[0]             #Supprime la 1ere feuille de l'ODS
```

### 4.11.8 Exemple

```python
#! /usr/bin/env PYTHON
# -*- coding: utf-8 -*-




import zipfile      #gere les fichiers zip
import shutil#permet de gerer des actions propre aux dossiers
import os     #interface avec l'os
import glob  #permet de gerer les chemins de maniere fine




#fonction d'archivage
def archive(filezip, chemin, leng):
    for i in glob.glob(chemin + '/*'):#os.listdir('./TEST/'):
        print i
        if os.path.isdir(i):
            archive(filezip, i, leng)
        else:
            filezip.write(i, i[leng:]) #Obligation de concevoir un chemin initial




#on dezip le fichier ods
os.rename("./TEST.ods", "TEST.zip")

mon_zip = zipfile.ZipFile('TEST.zip','r')
mon_zip.extractall('./TEST/')
mon_zip.close()


os.remove("./TEST.zip")


#on modifie notre balise $$NOM$$
with open("./TEST/content.xml", 'r') as mon_fichier:
    contenu = mon_fichier.read()
    mon_fichier.close()

contenu = contenu.replace("$$NOM$$", "DUPOND")

#Le fait d'ecrire sans append ecrase le contenu du fichier
with open("./TEST/content.xml", 'w') as fichier_out:
    fichier_out.write(contenu)
    mon_fichier.close()
```

```
49
50
51  #on rezip l'ods
52  with zipfile.ZipFile('TEST.zip', 'w',zipfile.ZIP_DEFLATED) as mon_zip:
53      path = './TEST'
54      pathsup = len(path)+1
55      archive(mon_zip, path, pathsup)
56      mon_zip.close()
57
58  os.rename("./TEST.zip", "./TEST.ods")
59
60
61  #on efface le dossier apres zippage
62  shutil.rmtree("./TEST")
```

## 4.12 Serveur FTP

Nous allons maintenant aborder la question des connexions FTP. En effet, il n'est pas rare qu'un traitement ait besoin d'aller chercher et/ou déposer de(s) fichier(s) sur un **serveur FTP**.

Nous ferons ici appel au module **ftplib**.

```
1 import ftplib
```

### 4.12.1 Connexion à un serveur FTP

Pour se connecter à un serveur **FTP**, il nous faut connaître au minimum:

=>l'adresse du serveur (et éventuellement le port)
=>vos identifiants d'accès

```
1 import ftplibimport ftplib
2
3 ftp_server = 'mon_serveur.com
4 login = 'invite'
5 passwd = 'invite'
6
7 ftpconnection = ftp.ftplib.FTP(ftp_server, login, passwd)
```

Vous voilà désormais connecté à votre serveur **FTP**. Voyons maintenant comment faire pour gérer les fichiers/dossiers à distance.

En général, pour chaque action réalisée, même une simple connexion, le serveur **FTP** vous renvoie des codes pour vous indiquer si l'opération s'est bien déroulée.

L'analyse de ces codes, que nous récupérerons ici dans la variable 'ret', vous permettra de détecter de potentielles erreurs.

### 4.12.2 Dépôt de fichiers/dossiers

Un dépôt de fichiers se passe en plusieurs étapes:
=>Ouverture du fichier en mode binaire
=>Envoi du fichier au serveur
=>Fermeture du fichier

Cette procédure est due au fait que l'envoi d'un fichier en mode binaire au serveur, impose que le fichier soit ouvert.

Le choix du transfert en mode binaire se justifie simplement par sa plus grande fiabilité.

```
1 file_name = './mon_fichier.txt'
2 fichier = open(file_name, 'rb')
3
4 ret = ftpconnection.storbinary('STOR ' + file_name, fichier)
5
6 fichier.close()
```

Une petite explication concernant la ligne 4. Le premier paramètre est une commande **FTP** classique. On demande au serveur **FTP** d'accepter le fichier dont on passe le nom en paramètre. Le second paramètre est le contenu du fichier que l'on désire transférer.

### 4.12.3 Récupération de fichiers/dossiers

En plus du dépôt, la récupération de fichiers/dossiers peut également être utile:

```
1 ret = ftpconnection.retrbinary('RETR '+ file_name, file_name.wite)
```

La commande **RETR** est également une commande de base **FTP**, qui demande à récupérer une copie du fichier dont le nom est passé en paramètre. Le second paramètre est ici le fichier ou l'on va écrire les données reçues.

### 4.12.4 Annulation d'un transfert en cours

Quel que soit le sens du transfert et le type du transfert (fichier ou dossier), la commande est unique:

```
1 ftpconnection.abort()
```

### 4.12.5 Liste des éléments du dossier

Afin de pouvoir se déplacer dans un serveur, il est important de savoir les éléments disponibles. Pour cela, il faut utiliser la commande dir():

```
1 listing = ftpconnection.dir()
```

### 4.12.6 Renommage d'un fichier/dossier

Pour renommer un fichier/dossier, il faut utiliser la ligne de code suivante:

```
1 ret = ftpconnection.rename(nom_actuel, nouveau_nom)
```

### 4.12.7 Création d'un dossier

Pour créer un dossier, il faudra utiliser la commande suivante:

```
1 ret = ftpconnection.mkd(repertory_name)
```

### 4.12.8 Effacement d'un fichier/dossier

Pour un fichier il suffit de réaliser l'action suivante:

```
1 ret = ftpconnection.delete(file_name)
```

Pour un dossier ce sera la commande suivante:

```
1 ret = ftpconnection.rmd(repertory_name)
```

### 4.12.9 Connaître la taille d'un fichier

Il peut parfois être pratique de connaître la taille d'un fichier afin de savoir si un transfert s'est bien effectué par exemple:

```
1 size = ftpconnection.size(file_name)
```

### 4.12.10 Savoir où l'on se trouve

Afin de savoir dans quel dossier/chemin on se trouve il suffit d'utiliser la commande **pwd**:

```
1 chemin = ftpconnection.pwd()
```

### 4.12.11 Utilisation de ligne de commandes

Il se peut que pour des raisons pratiques, ou pour utiliser une IHM simple, on désire simplement communiquer avec le serveur en ligne de commandes.

```
1 ret = ftpconnection.sendcmd(commande)
```

### 4.12.12 Déconnexion du serveur FTP

Le plus simple dans l'accès à un serveur **FTP**. Cela se passe de tout commentaire:

```
1 ftpconnection.quit()
2 ftpconnection.close()
```

Une petite explication ici: il n'y a pas d'erreur, il faut bien utiliser **quit()** puis **close()**.

La première commande indique au serveur que l'utilisateur se déconnecte, et la seconde commande que le programme se déconnecte du serveur même.

### 4.12.13 Les messages d'erreurs possibles

Comme évoqué plus haut, le serveur **FTP** renvoie des messages pour indiquer comment se déroulent les transferts.

Les principaux codes sont constitués de 3 chiffres. Le premier chiffre va de 1 à 5, le second de 0 à 5, et le dernier de 0 à 9.

Les codes vont ainsi de 100 à 559.

Il faut vous préoccuper principalement des codes 4xx et 5xx, indiquant des problèmes.

Voici, en résumé, la signification des principaux codes **FTP** (liste non exhaustive):

| Code FTP | Signification |
|---|---|
| **120** | Service prêt dans xxx minutes |
| **125** | Connexion de données déjà ouvertes |
| **150** | Fichier OK |
| **202** | Commande non prise en compte |
| **221** | Le service a fermé la connexion |
| **225** | Aucun transfert en cours |
| **226** | Action demandée effectuée avec succès |
| **227** | Mode passif entrant |
| **230** | User authentifié |
| **231** | User déconnecté avec succès |

| 232 | Demande de connexion prise en compte. |
|---|---|
| 331 | Mot de passe exigé |
| 332 | Autorisation demandée pour la connexion |
| 350 | Opération en suspens |
| 421 | Service non disponible / connexion fermée |
| 425 | Connexion impossible |
| 426 | Connexion fermée / transfert impossible |
| 430 | User ou mot de passe invalide |
| 450 | Fichier indisponible |
| 451 | Commande annulée |
| 452 | Anomalie espace mémoire |
| 501 | Erreur de syntaxe dans les paramètres |
| 502 | Commande non prise en compte |
| 503 | Mauvais ordre des commandes |
| 504 | Commande non prise en compte |
| 530 | User non reconnu |
| 532 | Autorisation exigée |
| 550 | Fichier indisponible |
| 551 | Commande interrompue |
| 552 | Commande de fichier interrompue |
| 553 | Action non prise en compte |

### 4.12.14 Exemple

L'exemple ici peut sembler superflu, mais néanmoins, il ne peut pas faire de mal.

```
1 #!/usr/bin/PYTHON
2 # -*- coding: utf-8 -*-
3
4
5
```

```
6
7  import ftplib
8
9
10
11
12 def connect_ftp():
13     host = 'mon_serveur.linux.net'
14     username = 'invite'
15     password = 'invite'
16
17     ftpconnection = ftplib.FTP(host, username, password)
18     ret = ftpconnection.getwelcome()
19     print ret
20     ftpconnection.quit()
21     ftpconnection.close()
22
23
24
25
26 connect_ftp()
```

Une petite précision concernant la ligne 18. La commande utilisée permet de récupérer le message de bienvenue envoyé par le serveur à la connexion.

## 4.13 Les mails avec SMTP

Nous allons rapidement voir ici comment envoyer des mails.

L'envoi de mails est couramment utilisé dans des scripts **PYTHON** d'administration afin de prévenir les personnes concernées de possibles anomalies dans les traitements, ou tout simplement leur adresser un compte rendu.

Pour la gestion de la réception, je vous renvoie à la documentation du module que nous allons utiliser ici : **smtplib**.

```
import smtplib
```

Le but ici est de vous donner les bases, non pas pour créer un client mail, mais pour savoir implémenter l'envoi de rapports par mail depuis vos scripts.

### 4.13.1 Données nécessaires

La première chose à faire est de préparer les données.

Pour envoyer un mail, vous aurez besoin des éléments suivants :
=>adresse d'un serveur **SMTP**
=>adresse d'expéditeur
=>adresse de destinataire
=>objet de mail
=>corps de mail
=>Nom et mot de passe d'utilisateur pour le serveur **SMTP**
=>Paramétrage de connexions au serveur **SMTP**

### 4.13.2 Exemple

Un exemple, valant mieux qu'un long discours, place au code :

```
#!/usr/bin/PYTHON
# -*- coding: utf-8 -*-
```

```
3
4
5
6
7  import smtplib
8  from email.MIMEText import MIMEText
9
10
11
12
13 def sendmail():
14     user = 'user'
15     passwd = 'pass'
16
17     msg = MIMEText('message du mail de test', _charset = 'utf8')
18     msg['From'] = 'testo@gmail.com'
19     msg['To'] = 'dest@yahoo.fr'
20     msg['Subject'] = 'Mail de test'
21
22     smtp = smtplib.SMTP('smtp.gmail.com:587')
23     smtp.starttls()
24     smtp.login(user,passwd)
25
26     smtp.sendmail(msg['From'], msg['To'], msg.as_string())
27
28     smtp.close()
29
30
31
32
33 sendmail()
```

Un peu d'explications maintenant.

Ligne 14 et 15, nous déclarons simplement notre nom d'utilisateur et notre mot de passe pour nous identifier sur le serveur désiré.

De la ligne 17 à la ligne 20, nous définissons les paramètres de notre message.

Tout d'abord nous créons un type **MIMEtext**. Il s'agit d'un format de texte adapté pour les envois mails avec **SMTP**. Il faut le voir ici comme un conteneur.

Ce **MIMEtext** contiendra le corps texte de notre mail. Nous précisons ici que nous utilisons le format d'encodage UTF-8.

Ensuite, nous précisons les divers paramètres : From, To et Subject.

Ligne 22 à 24, nous nous connectons avec noter user au serveur **SMTP** (ici, celui de gmail), sur le port 587, en TLS.

Ligne 26, nous envoyons notre mail proprement dit, et enfin ligne 28, nous nous déconnectons du serveur **SMTP**.

## 4.14 Le port série

Le module à utiliser est **pyserial**. L'ensemble de la documentation est disponible ici: http://pyserial.sourceforge.net/index.html

On peut se poser la question de l'utilité d'un tel module. Dans les faits, beaucoup de matériel USB ne sont en fait que du matériel série ou parallèle avec un pont usb. Savoir gérer correctement ces ports se révèle par conséquent utile.

### 4.14.1 Paramétrage

Voyons voir les éléments principaux de ce module. Voici un code présentant l'utilisation d'un **port série** sous Linux, de l'ouverture à la fermeture

```
1  import serial
2  ser = serial.Serial('/dev/ttyS0', 9600, timeout=0.008)
3  ser.write ("hello,world!!!")
4  x=ser.read()
5  ser.close()
```

Analysons cet exemple. Pour commencer nous importons le module **pyserial**, ligne 1. Ligne 2, nous déclarons notre port: le port utilisé, le débit souhaité, et le timeout (temps à attendre un octet avant de passer à la suite).

Concernant le port série, il s'agit ici de la notation Linux, sur une machine de dev DEBIAN.

Ligne 3, l'écriture sur le port, se passe de tout commentaire, tout comme la lecture ligne 4, ou encore la fermeture du port ligne 5.

Si vous n'avez besoin que d'émettre et recevoir sur le port série, cette configuration vous suffira amplement.

Pour ceux qui auraient besoin de quelque chose de plus complet, nous allons approfondir. Tout d'abord le constructeur tel que défini sur le site officiel:

***__init__(port=None, baudrate=9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False, rtscts=False, writeTimeout=None, dsrdtr=False, interCharTimeout=None)***

Nous pouvons d'ores et déjà constater que le port série est fortement configurable dans les moindres détails

Quelques précisions sur certain de ces paramètres: Bytesize peut prendre 4 valeurs différentes (FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS), Parity 5 valeurs (PARITY_NONE, PARITY_EVEN, PARITY_ODD PARITY_MARK, PARITY_SPACE) et stopbits 3 valeurs différentes (STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO ).

Côté débit, les valeurs standard sont 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

Enfin, vous trouverez ci-dessous l'essentiel de la documentation traduite en français

| Méthode | Utilité |
|---|---|
| **Open()** | Ouvrir le port |
| **Close()** | Fermer le port |
| **read(size=1)** | Lire un nombre d'octets égal à size (1 par défaut). Retourne le nombre d'octet lus |
| **write(data)** | Écrit des données sur le port. Retourne le nombre d'octets écrits |
| **inWaiting()** | Retourne le nombre d'octets en attente dans le buffer de réception |
| **setBreak(*level=True*)** | Sert à contrôler le TXD |
| **setRTS(*level=True*)** | Sert à contrôler le RTS |
| **setDTR(*level=True*)** | Sert à contrôler le DTR |
| **getCTS()** | Retourne un booléen indiquant l'état du cts |
| **getDSR()** | Retourne un booléen indiquant l'état du |

| | |
|---|---|
| | DSR |
| **getRI()** | Retourne un booléen indiquant l'état du RI |
| **getCD()** | Retourne un booléen indiquant l'état du CD |
| **name** | Retourne le nom du port ouvert |
| **baudrate** | Retourne les différentes valeurs de la configuration du port |
| **bytesize** | |
| **parity** | |
| **stopbits** | |
| **timeout** | |
| **writeTimeout** | |
| **xonxoff** | |
| **rtscts** | |
| **dsrdtr** | |
| **interCharTimeout** | |
| **BAUDRATES** | Retourne les différentes valeurs valides pour les paramètres. Idéal pour remplir une liste déroulante |
| **BYTESIZES** | |
| **PARITIES** | |
| **STOPBITS** | |
| **serial.tools.list_ports.comports()** | Renvoie une liste des ports disponibles (depuis la V2.6) |

### 4.14.2 Exemple

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4
5
6
7  import serial
8
9
10
```

```
11
12  def mon_port_serie():
13      """
14          Émet une chaine de test puis indique le paramétrage
15          A utiliser en mode root, pour les droits sur ttyS0
16      """
17
18      ser = serial.Serial('/dev/ttyS0', 9600, timeout=0.008)
19      ser.write ("Test emission")
20      print "Chaine emise sur port ", ser.name, " à ", ser.baudrate, "Bauds"
21      ser.close()
22
23
24
25
26  if __name__ == '__main__':
27      mon_port_serie()
```

## 4.15 Le port parallèle

Pour utiliser le **port parallèle** d'un PC nous utilisons le module **pyparallel**. Dans le code nous utiliserons un **import parallel**

Le port parallèle est ici utilisé dans sa version originelle, c'est-à-dire unidirectionnel (et non à la norme IEEE1284).

Il existe d'autre module pour piloter le port parallèle, mais celle-ci est la plus utilisé.

### 4.15.1 Paramétrage

Nous disposons donc d'un bus d'1 octet en sortie, plus trois sorties indépendantes (Data Strobe, Auto Feed, Initialize). Côté entrées nous disposons de Select, Paper Out, et Acknoledge.

Voici l'exemple fourni sur le site officiel:

```
1  import parallel
2  p = parallel.Parallel()    # open LPT1
3  p.setData(0x55)
```

Le code se passe de commentaire au vu de la simplicité. Outre le **setData()**, il existe d'autres méthodes:

| Methode | Utilité |
|---|---|
| **setDataStrobe(*level*)** | Configure la sortie Data Strobe |
| **setAutoFeed(*level*)** | Configure la sortie Auto Feed |
| **setInitOut(*level*)** | Configure la sortie Initialize |
| **getInSelected()** | Récupère l'état de l'entrée Select |
| **getInPaperOut()** | Récupère l'état de l'entrée Paper Out |
| **getInAcknowledge()** | Récupère l'état de l'entrée Acknoledge |

Ici, level vaut 1 ou 0.

### 4.15.2 Exemple

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4
5
6
7  import parallel
8
9
10
11
12 def mon_port_parallele():
13     """
14         Emet une chaine de test puis indique le parametrage
15         A utiliser en mode root, pour avoir les droits
16     """
17
18     p = parallel.Parallel()    # open LPT1
19     p.setData(0x55)
20     print "Donnée 0x55 emise. Paper out à ", p.getInPaperOut()
21
22
23
24
25 if __name__ == '__main__':
26     mon_port_parallele()
```

## 4.16 Le réseau: les bases

Le **réseau**, quel que soit le langage utilisé est un vaste sujet. Nous ne nous attarderons ici qu'aux bases. Cela vous permettra d’établir une connexion sur un réseau, d’émettre et de recevoir des données.

Pour des opérations plus complexes, je vous invite à rechercher sur le net, ou dans les sites répertoriés en fin de ce livre afin de trouver nombres d'exemples adaptés à vos besoins spécifiques.

### 4.16.1 Le module socket

Le module le plus usité pour le réseau est le module **socket**

```
1 import socket
```

Ce module est valable pour tous les OS modernes. Pour ceux qui ne seraient pas familiés avec ce concept, rappelons qu'un **socket** peut être résumé simplement en l'association d'une adresse et d'un port.

### 4.16.2 Ouverture d'un port

L'ouverture d'un **port** réseau se passe en réalité en trois étapes. Pour commencer, on définit le port que nous désirons utiliser. En général, on désire passer des adresses IP, et utiliser le port en type STREAM. La ligne suivante est donc généralement à copier coller directement.

```
1 sock= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Ensuite, nous définissons le timeout du port. Ce dernier est un float, que l'utilisateur définit comme bon lui semble.

```
1 sock.settimeout(3.0)
```

Enfin, nous ouvrons le port Ethernet en passant comme paramètres l'adresse réseau du destinataire, et le port sur lequel nous dialoguons.

```
1 sock.connect ((HOST,PORT))
```

Ici HOST vaudra par exemple 192.168.1.25 et PORT 9100, si nous dialoguons avec la machine dont l'adresse IP est 192.168.1.25, sur le port 9100. Notons que HOST pourrait tout aussi bien être une chaîne de caractères

### 4.16.3 Envoi de données

L'envoi de données ne représente rien de bien sorcier ici. Il suffit juste d'utiliser la méthode **send()**, avec comme paramètre la chaîne de caractères.

```
1 sock.send('beep\r')
```

### 4.16.4 Réception de données

Tout comme pour l'émission, la réception est très facile. La méthode à utiliser est **recv()**, qui prend comme paramètre le nombre maximum d'octets à lire en une fois

```
1 data=sock.recv(128)
```

Précisons qu'une fois la ligne précédente interprétée, si rien n'est reçu, passé le timeout, le programme poursuit sa route.

### 4.16.5 Fermeture d'un port

Tout comme pour le port série, il suffit d'appeler ici la méthode **close** pour fermer un port.

```
sock.close()
```

### 4.16.6 Exemple

```
#! /usr/bin/env PYTHON
# -*- coding: utf-8 -*-




import serial
import socket




HOST= '10.5.150.11'
PORT= 9100




def conversion():
    """
        Convertit une trame reseau en trame serie
    """

    ser = serial.Serial('/dev/ttyS0', 9600, timeout=0.008)

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3.0)
    sock.connect ((HOST,PORT))

    while True:
        data = sock.recv(10)
        ser.write (data)
        sock.send("ACK")

    ser.close()
    sock.close()




if __name__ == '__main__':
    conversion()
```

## 4.17 La webcam avec OpenCV

### 4.17.1 Utilisation basique

Pour utiliser la **webcam** avec **PYTHON**, via **OpenCV**, vous avez besoin de trois éléments: un objet de capture pour vous connecter au flux vidéo, un objet de frame pour réaliser un snapshot à un instant t du flux vidéo, et enfin un conteneur fenêtre (**GTK**, **KDE**, ou **OpenCV**, cas que nous verrons) afin d'afficher la vidéo.

Bien entendu, nous commencerons par l'import d'**OpenCV**:

```
1 import cv2.cv as cv
```

***Remarque: Selon les versions, l'import peut ne pas se faire de la même façon***

#### 4.17.1.1 Connexion au flux vidéo

Comme cité précédemment, la première chose à faire est de se connecter au flux vidéo. Pour cela nous allons créer un objet dédié à une caméra/webcam:

```
1 ma_caméra = cv.CaptureFromCAM(0)
```

le paramètre passé lors de la création de l'objet est un index permettant de définir quelle caméra utiliser. Lorsqu'il n'y a qu'une seule caméra, il faut passer 0 en paramètre.

***Remarque: Il existe la possibilité de remplacer le flux vidéo d'une webcam par le flux vidéo d'un fichier. Il ne faut alors pas utiliser cv.CaptureFromCAM(index) mais cv.CaptureFromFile(fichier).***

#### 4.17.1.2 Snapshot du flux

Bien que nous puissions afficher directement le flux vidéo à l'écran, cette solution est rarement retenue.

En effet, passer par une série de snapshot possède un certain nombre d'avantages tels l’application d'un traitement sur image, la possibilité de sauvegarder certain snapshots en direct, ...

Nous allons donc utiliser un objet nommé " **frame** ". Dans le monde vidéo, une frame est tout simplement une image. On ne parle donc pas de 25 images/seconde mais de 25 frames/seconde.

```
1 ma_frame = cv.QueryFrame(ma_caméra)
```

Lorsque nous créons cette **frame**, nous lui passons en paramètre le flux vidéo auquel elle est liée.

La **frame** est ainsi l'équivalent d'un snapshot à un instant t du flux vidéo.

#### 4.17.1.3 Affichage des snapshots

Enfin, pour afficher la vidéo, il faut une fenêtre conteneur. Dans vos futurs développements, vous incluerez cela sans nul doute dans une fenêtre **GTK** ou **KDE**.

Il existe aussi la possibilité toute simple de l'afficher à l'écran.

```
1 cv.NamedWindow("Test_Webcam")
```

Cela indique à **PYTHON** que nous utiliserons une fenêtre **OpenCV** pour l'affichage qui se nommera Test_Webcam.

Il ne reste ensuite plus qu'à afficher le conteneur en lui passant en paramètres la fenêtre à utiliser ainsi que la frame à afficher, puis tourner en boucle.

#### 4.17.1.4 Exemple d'affichage d'un flux webcam

```
#! /usr/bin/env PYTHON
# -*- coding: utf-8 -*-

import cv2.cv as cv

if __name__=='__main__':
    ma_caméra = cv.CaptureFromCAM(0)
    cv.NamedWindow("Test_Webcam")
    while True:
        ma_frame = cv.QueryFrame(ma_caméra)
        cv.ShowImage("Test_Webcam", ma_frame)

        if (cv.WaitKey(10) % 0x100) == 113:
            break
```

Lignes 13 et 14, il s'agit d'une méthode que nous verrons plus loin. Cela permet de lire le clavier et de sortir du programme lorsque la touche " q " est appuyée.

### 4.17.2 Utilisation avancée

Comme vous vous en doutez sûrement, **OpenCV** ne se contente pas de ces fonctionnalités basiques. Nous en verrons un peu plus plus loin, mais nous allons ici nous concentrer sur les trois objets de base que nous serons régulièrement amenés à utiliser.

#### 4.17.2.1 Les images

Le premier de ces objets est l'image. En effet, avec **OpenCV**, il faut différencier image et **frame**.

SI la **frame** est bien une image capturée à partir du flux vidéo, on ne peut réaliser aucun traitement dessus. Elle ne peut servir qu'a un affichage direct (comme vu précédemment), ou comme source pour un traitement d'imagerie.

Ces traitements sont nombreux, et nous en verrons quelques-uns, mais l'outil de base est un objet image.

```
1 mon_image = cv.CreateImage(cv.GetSize(ma_frame), cv.IPL_DEPTH_8U, 1)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SIZE | CVSIZE | - | Taille de l'image (largeur et hauteur) |
| 2 | DEPTH | INT | - | Type d'image |
| 3 | CHANNELS | INT | - | Nombre de canaux |

Le premier paramètre est un tuple indiquant la largeur et la hauteur de l'image. La plupart du temps pour éviter tous soucis, nous réalisons un **GetSize** depuis la source.

Le second argument est le type d'image: 8, 16, 32 ou 64 bits. Cela joue sur l'image finale. Par exemple, une image en nuance de gris sera en 8 bits. A noter qu'il s'agit ici de constante **OpenCV**.

| Paramètre | Description |
|---|---|
| **IPL_DEPTH_8U** | Image 8 bits non signée |
| **IPL_DEPTH_8S** | Image 8 bits signée |
| **IPL_DEPTH_16U** | Image 16 bits non signée |
| **IPL_DEPTH_16S** | Image 16 bits signée |
| **IPL_DEPTH_32S** | Image 32 bits signée |
| **IPL_DEPTH_32F** | Image 32 bits float simple précision |
| **IPL_DEPTH_64F** | Image 64 bits float double précision |

***Remarque: Beaucoup de traitements que nous verrons par la suite nécessitent des images de type cv.IPL_DEPTH_8U***

Enfin le dernier paramètre est le nombre de canaux. Entendez ici le nombre de couleurs: 1 pour une image en nuance de gris (grayscale), 3 pour une image classique RGB (couleur), 4 pour une image RGB avec canal alpha (transparence).

***Remarque: Si votre logiciel nécessitait de charger une photo, par exemple, afin d'y appliquer des traitements via OpenCV, vous pouvez utiliser cv.LoadImage(Fichier, -1).***

A noter que la valeur du pixel (x,y) d'une image est accessible via **img[x,y]**, que la largeur de l'image est **image.width** et sa hauteur **image.height**.

#### 4.17.2.2 Les enregistreurs vidéos

**OpenCV** offre la possibilité d'enregistrer les frames sous forme d'une vidéo de manière très simple.

Il faut tout d'abord créer l'objet dédié.

```
1  mon_rec = cv.CreateVideoWriter(" ma_video.avi ", mon_codec, 25,cv.GetSize(ma_frame), 0)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | FILENAME | CHAR | - | Nom du fichier de sortie |
| 2 | FOURCC | INT | - | TYPE de CODEC à utiliser |
| 3 | FPS | DOUBLE | - | Nombre de frames/seconde |
| 4 | FRAME_SIZE | CVSIZE | - | Taille des frames vidéo |
| 5 | IS_COLOR | INT | 1 | Si <> 0, mode couleur, grayscale sinon |

Le premier paramètre est le chemin et le nom de la vidéo. Ici la vidéo sera enregistrée dans le dossier courant.

Le second paramètre est un objet **CV_FOURCC**. Il s'agit d'un objet auquel on passe en paramètres les 4 lettres du codec désiré.

```
1  mon_codec = cv.CV_FOURCC('D','I','V','X')
```

Les **codecs** les plus courants sont les suivants:

| Paramètre | Description |
|---|---|
| **PIM1** | MPEG1 |
| **MJPG** | Motion JPEG |
| **DIVX** | MPEG4 |
| **H264** | H264 |
| **U263** | H263 |
| **I263** | H263 entrelacé |
| **FLV1** | FLV1 |

Le troisième paramètre est le nombre d'images désirées par seconde.

Le quatrième paramètre est la taille de la vidéo. Une fois de plus, on se calera sur la taille de la frame pour éviter tout problème.

Le cinquième et dernier paramètre permet d'indiquer si nous désirons un enregistrement en noir et blanc (0) ou en couleur (1).

Une fois l'objet enregistreur créé, il suffit d'y faire appel via une méthode **OpenCV**.

```
1 cv.WriterFrame(mon_rec, ma_frame)
```

Dans notre cas d'exemple, au bout de 25 appels à cette méthode, le film possédera donc 1 seconde de vidéo.

***Remarque: Dans le cas ou vous ne désireriez ne pas enregistrer une vidéo mais simplement une image, il faut utiliser la méthode cv.SaveImage(fichier, Image).***

#### 4.17.2.3 Les polices d'écriture

Le troisième et dernier objet complémentaire que nous verrons ici est le texte. Il peut être très utile de saisir du texte sur une image pour identifier la date et l'heure, voir le lieu.

Cependant, actuellement seule une police d'écriture est supportée.

Tout comme pour l'enregistrement vidéo, cela aura lieu en deux étapes: création de l'objet puis appel d'une méthode **OpenCV**.

```
1 ma_police = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX, 1,1,0,1,8)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | FONTFACE | INT | - | Nom de la police |
| 2 | HSCALE | DOUBLE | - | Échelle horizontale |
| 3 | VSCALE | DOUBLE | - | Échelle verticale |
| 4 | SHEAR | DOUBLE | 0 | Inclinaison |
| 5 | THICKNESS | INT | 1 | Épaisseur des traits |
| 6 | LINETYPE | INT | 8 | Type de traits |

Le premier paramètre est la taille et le type de la police.

| Paramètre | Description |
|---|---|
| **CV_FONT_HERSHEY_SIMPLEX** | Sans serif, taille normale |
| **CV_FONT_HERSHEY_PLAIN** | Sans serif, petite taille |
| **CV_FONT_HERSHEY_DUPLEX** | Sans serif, taille normale |
| **CV_FONT_HERSHEY_COMPLEX** | Taille normale avec serif |
| **CV_FONT_HERSHEY_TRIPLEX** | Taille normale avec serif |
| **CV_FONT_HERSHEY_COMPLEX_SMALL** | Petite taille avec serif |
| **CV_FONT_HERSHEY_SCRIPT_SIMPLEX** | Effet " écriture à la main " |
| **CV_FONT_HERSHEY_SCRIPT_COMPLEX** | Effet " écriture à la main " |

Les paramètres 2 et 3 sont les ratios en horizontal et vertical du texte par rapport à sa taille normale (100 % = 1.0).

Le quatrième paramètre est le degré d'inclinaison de l'italique. Sans italique, ce paramètre vaut 0. La valeur 1.0 indique une inclinaison d'environ 45°.

Le cinquième paramètre sert à gérer le paramètre " GRAS " du texte. Sa valeur par défaut est de 1. Il s'agit d'un entier.

Le dernier paramètre est également un entier, avec comme valeur par défaut 8. Cela permet d'indiquer si le texte doit être souligné ou non.

Une fois l'objet créé selon notre besoin, il ne reste plus qu'à incruster le texte dans l'image. Pour cela nous utiliserons la méthode **PutText()**.

```
1 cv.PutText(ma_frame, mon_texte, emplacement_texte, ma_police, 1)
```

Nous passons plusieurs paramètres à cette méthode.

Tout d'abord, la frame (ou l'image au choix) impactée. Vient ensuite le texte à afficher, puis l'emplacement du coin inférieur gauche du texte (en pixel).

Suit notre objet police,et un paramètre booléen. SI ce dernier vaut 1 (ou True), la référence (le pixel (0,0)) se situe au coin supérieur gauche, sinon au coin inférieur gauche.

#### 4.17.2.4 Utilisation du clavier

**OpenCV** vous laisse la liberté d'interagir avec le clavier, afin de déclencher des traitements, de sortie du programme, …

Pour cela, il faut utiliser la méthode **WaitKey(tps_en_ms)**. Pendant x ms, la méthode écoutera alors le clavier à la recherche d'une touche appuyée.

Le code renvoyé est un entier. Il faut effectuer un masque (% 0x100) avant de le comparer à une valeur décimale (ex: 113 pour " q ").

***Remarque: Le WaitKey peut ne pas fonctionner si vous n'avez pas créé de fenêtre conteneur OpenCV. Dans le cas d'une intégration dans une IHM GTK ou autre, pensez à utiliser votre IHM pour interagir avec l'utilisateur.***

### 4.17.3 Paramétrage de la webcam

Il est possible de paramétrer ou de lire le paramétrage de la webcam via les méthodes **SetCaptureProperty()** et **GetCaptureProperty()**.

Les paramètres utiles accessibles sont les suivants.

| Paramètre | Description |
|---|---|
| **CV_CAP_PROP_POS_MSEC** | Position dans le film en millisecondes |
| **CV_CAP_PROP_POS_FRAMES** | Position dans le film en nombre de frames |
| **CV_CAP_PROP_POS_AVI_RATIO** | Position dans le film en pourcentage |
| **CV_CAP_PROP_FRAME_WIDTH** | Largeur du flux vidéo |
| **CV_CAP_PROP_FRAME_HEIGHT** | Hauteur du flux vidéo |
| **CV_CAP_PROP_FPS** | Nombre d'images/seconde du flux vidéo |
| **CV_CAP_PROP_FOURCC** | Code du codec du flux vidéo |
| **CV_CAP_PROP_FRAME_COUNT** | Nombre total de frames (fichier vidéo uniquement) |
| **CV_CAP_PROP_BRIGHTNESS** | Luminosité de l'image |
| **CV_CAP_PROP_CONTRAST** | Contraste de l'image |
| **CV_CAP_PROP_SATURATION** | Saturation de l'image |
| **CV_CAP_PROP_HUE** | Teinte de l'image |
| **CV_CAP_PROP_GAIN** | Gain de l'image |
| **CV_CAP_PROP_EXPOSURE** | Exposition de l'image |
| **CV_CAP_PROP_CONVERT_RGB** | Image N&B ou RGB |

Tous ces paramètres ne sont pas forcément accessibles selon le périphérique utilisé.

#### 4.17.3.1 SetCaptureProperty

Il s'agit de la méthode de paramétrage. D'un emploi très simple, elle prend trois paramètres: l'objet de capture, la propriété à modifier, et la nouvelle valeur.

```
1  cv.PutText(ma_frame, mon_texte, emplacement_texte, ma_police, 1)
```

#### 4.17.3.2 GetCaptureProperty

La méthode pour lire le paramétrage, est également très simple d'emploi. Elle prend en paramètres l'objet de capture, ainsi que la propriété désirée.

```
1  nb_fps = cv.GetCaptureProperty(ma_caméra,  CV_CAP_PROP_FPS)
```

### 4.17.4 Traitements sur les frames

Comme explicité précédemment, le fait de travailler avec des frames plutôt qu'avec un flux vidéo brut va permettre d'appliquer des traitements aux images.

Vu le nombre important de possibilités avec **OpenCV**, nous nous limiterons aux principales méthodes.

#### 4.17.4.1 Les types de données OpenCV

Vous trouverez ci-après les principaux types de données que vous trouverez dans **OpenCV** et dans les pages qui suivent.

| Paramètre | Description |
|---|---|
| **CHAR** | Grand classique, caractères alphanumériques |
| **CVARR** | Type le plus utilisé. Toute image et frame en OpenCV est un CVARR, un simple tableau de données. |
| **CVMEMSTORAGE** | Espace de stockage servant aux calculs dynamiques |
| **CVPOINT** | Tuple contenant des coordonnées x & y |
| **CVSIZE** | Taille d'un rectangle ou d'une image |
| **CVSCALAR** | Tuple contenant de 1 à 4 doubles |
| **DOUBLE** | Grand classique également, nombre à virgule |
| **INT** | Autre grand classique, des entiers |

### 4.17.4.2 AbsDiff

Cette méthode permet d'effectuer une soustraction entre deux images puis de passer l'ensemble des pixels en valeur absolue.

Ainsi, les pixels n'ayant pas changés donneront en sortie un pixel blanc, et ceux ayant changés seront en grayscale.

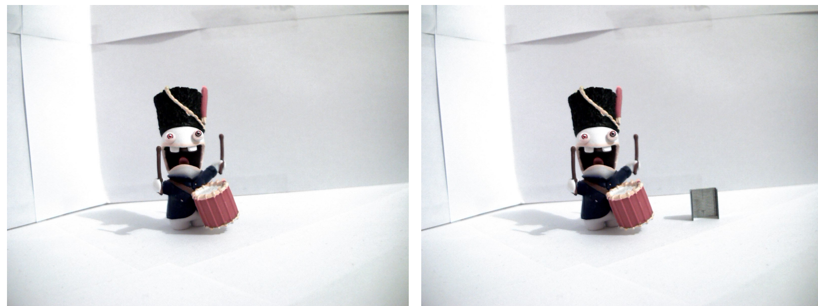
A l'issue de l'application de cette méthode, nous récupérons donc une image dont les pixels non blancs correspondent aux pixels différents entre les deux images.

```
1  cv.AbsDiff(img_1, img_2, img_cible)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC1 | CVARR | - | Image source 1 |
| 2 | SRC2 | CVARR | - | Image source 2 |
| 3 | DST | CVARR | - | Image de sortie |

Sans ambiguïté, nous voyons que les deux premiers paramètres sont les images/**frames** servant à la soustraction, et le dernier paramètre l'image où nous stockerons le résultat.

L'illustration ci après montre le résultat que l'on peut obtenir grâce à un **ABSDIFF**. Bien entendu, il faudra d'abord effectuer diverses opération dont des **CvtColor**, mais le résultat final est là.



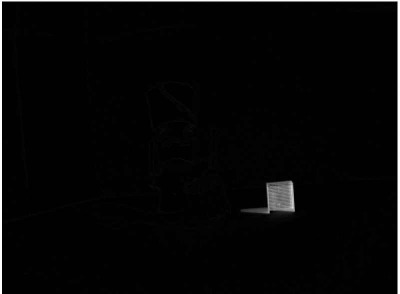


***Illustration 5: Application du ABSDIFF***

### 4.17.4.3 CvtColor

Cette méthode permet de convertir une image d'un espace de couleur à un autre, ou plus généralement, une image couleur en niveaux de gris ou grayscale.

```
1  cv.cvtColor(frame_source, frame_cible, cv.CV_RGB2GRAY)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image source |
| 2 | DST | CVARR | - | Image de sortie |
| 3 | CODE | INT | - | Type d'encodage |

Cette méthode est extrêmement simple, et convertit ici une image couleur en une image grayscale. Le dernier paramètre est une constante **OpenCV** stipulant le type de conversion désirée.

D'autres types de conversion que grayscale existent mais beaucoup moins usitées. Aussi nous ne verrons que la **HSV**.

La HSV pour Hue/Saturation/Value ou teinte saturation valeur est un type d'image qui nous servira plus loin pour la détection de couleur. Seule la constante est à remplacer. Nous utiliserons alors **CV_BGR2HSV** ou **CV_RGB2HSV**, la différence consistant en l'inversion des canaux bleu et rouge.

***Illustration 6: Avant/Après un CvtColor en grayscale***

#### 4.17.4.4 Dilate

L'opération de dilatation permet de diminuer la finesse d'une image.

***Attention: La référence ici n'est pas le pixel noir mais le pixel blanc. Aussi pourriez-vous avoir la sensation que la méthode de dilate érode et que la méthode d'érosion dilate.***

Prenez une image et un curseur carré. A l'intérieur du curseur la méthode va analyser les pixels et placer l'ensemble des pixels à la valeur la plus élevée disponible dans le curseur (Rappel, la référence est le pixel blanc, soit 0; donc plus un pixel sera proche de 0, plus il sera considéré comme élevé).

***Illustration 7: Avant/Après étape de dilatation***

L'intérêt de cette méthode est de supprimer l'ensemble des petits défauts parasites, et d'obtenir une image à la fois représentative, mais également plus brute, plus dense et donc plus simple à analyser.

```
1 cv.Dilate(img_source, img_cible, None, Nb_dilate)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image source |
| 2 | DST | CVARR | - | Image de sortie |
| 3 | ELEMENT | IplConv Kernel | NULL | Taille du curseur. |
| 4 | ITERATIONS | INT | 1 | Nombre d'itérations |

Les deux premiers paramètres sont classiques.

Le troisième sera la plupart du temps à None. Il s'agit de la dimension du curseur à utiliser. Par défaut, via la valeur None, il sera de 3*3 pixels.

Le dernier paramètre enfin indique le nombre de fois que l'on doit appliquer la méthode de dilatation.

#### 4.17.4.5 Erode

L'opération d'érosion, comme son nom l'indique, consiste à affiner une image.

Le concept est l'exact opposé de celui de la dilatation, à savoir qu'à l'intérieur du curseur carré, le pixel central prend la valeur du pixel le moins élevé du curseur.

Nous obtenons ainsi une image affinée et là aussi potentiellement débarrassée des pixels parasites isolés.

Appliqué après un **Dilate**, un **Erode** permet d'obtenir en sortie une image brute de décoffrage, simplifiée et plus rapide à analyser.

***Illustration 8: Avant/Après étape d'érosion***

```
1 cv.Erode(img_source, img_cible, None, nb_erode)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image source |
| 2 | DST | CVARR | - | Image de sortie |
| 3 | ELEMENT | IplConv Kernel | NULL | Taille du curseur. |
| 4 | ITERATIONS | INT | 1 | Nombre d'itérations |

Les paramètres ont la même fonction que pour la fonction **Dilate**.

#### 4.17.4.6 Combinaison: Ouverture

Le but d'une ouverture est de supprimer les petits objets isolés ou pixels.

Cette opération ne fonctionne que si l'objet est blanc sur fond noir, étant donné que la référence est le pixel blanc.

Le résultat est obtenu en effectuant une opération **Dilate**, suivie d'une opération **Erode**.

***Illustration 9: Avant/Après une opération d'ouverture***

#### 4.17.4.7 Combinaison: Fermeture

L'opération de fermeture vise à supprimer des tâches noires pouvant exister et parasiter notre corps blanc sur fond noir.

Elle est obtenue en effectuant d'abord une opération **Erode**, suivie d'une opération **Dilate**.

***Illustration 10: Avant/Après une opération de fermeture***

#### 4.17.4.8 Combinaison: Gradient

L'opération de gradient sert à déterminer les contours d'une forme.

Elle est obtenue en faisant l'opération suivante: **Dilate - Erode**

***Illustration 11: Avant/Après une opération de gradient***

### 4.17.4.9 Combinaison: Top Hat

L'opération Top Hat vise à isoler grossièrement notre forme.

Elle est obtenue en faisant: image - ouverture

***Illustration 12: Avant/Après un Top Hat***

### 4.17.4.10 Combinaison: Black Hat

L'opération de Black Hat permet, elle, d'inverser nos valeurs de pixels, en effectuant: fermeture - image

***Illustration 13: Avant/Après un Black Hat***

### 4.17.4.11 ContourArea

Il s'agit d'une des deux méthodes (avec **DrawContours**) que l'on peut exécuter directement après un **FindContours** (voir 4.17.4.14).

Le seul paramètre à passer est le retour du **FindContours**. La méthode nous renvoie alors la somme de toutes les aires détectées dans l'image.

***Remarque: La référence est le pixel blanc, et non le noir***

```
1 surface = cv.ContourArea(contours)
```

#### 4.17.4.12 DrawContours

La seconde méthode exécutable après un **FindContours**. Elle permet tout simplement de tracer les contours détectés par la méthode **Findcontours**.

```
1 cv.DrawContours(ma_frame, NewContours, (0,0,255),(0,255,0),1,2,8)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMG | CVARR | - | Image source (encodée en 8 bits) |
| 2 | CONTOURS | CVARR | - | Image de sortie |
| 3 | EXTERNAL_ COLOR | CVSCALAR | - | Couleur de ligne du contour externe |
| 4 | HOLE_COLOR | CVSCALAR | - | Couleur de ligne du contour interne |
| 5 | MAX_LEVEL | INT | - | Niveau de contour à dessiner. À 1, tous les contours sont dessinés |
| 6 | THICKNESS | INT | 1 | Épaisseur du trait. Si inférieur à 0 ou égal à cv.CV_FILLED, alors l'intérieur du contour est remplit |
| 7 | LINETYPE | INT | 8 | Type de trait |

#### 4.17.4.13 EqualizeHist

Cette opération vise à optimiser l'histogramme de l'image pour permettre le meilleur traitement possible de l'image.

Pour rappel, l'histogramme d'une image correspond à la répartition de la teinte, du contraste, de la luminosité, …

```
1 cv.EqualizeHist(img_source, img_cible)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image source (encodée en 8 bits) |
| 2 | DST | CVARR | - | Image de sortie |

Rien de bien compliqué ici comme vous pouvez le constater.

***Illustration 14: Avant/Après un EqualizeHist***

### 4.17.4.14 FindContours

Il s'agit d'une méthode permettant de trouver les contours d'une image, en lui fournissant simplement en entrée une image binaire (voir 4.17.4.23).

```
1 contours = cv.FindContours(grey,storage,cv.CV_RETR_EXTERNAL,\
2                            cv.CV_CHAIN_APPROX_SIMPLE)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMAGE | CVARR | - | Image source binaire (encodée en 8 bits) |
| 2 | STORAGE | CVMEM STORAGE | - | Espace de calcul dynamique (voir 4.17.4.19) |
| 3 | MODE | INT | CV_RETR _LIST | Mode de détection de contours |
| 4 | METHOD | INT | CV_CHAIN _APPROX _SIMPLE | Méthode de détection des contours |

**MODE** peut prendre différentes valeurs:

| Paramètre | Description |
|---|---|
| **CV_RETR_EXTERNAL** | Ne recherche que les contours externes |
| **CV_RETR_LIST** | Recherche tous les contours sans distinction |
| **CV_RETR_CCOMP** | Recherche tous les contours puis les organise par priorité |
| **CV_RETR_TREE** | Recherche tous les contours, les trie, puis créé un fichier descriptif |

**METHOD** peut également prendre différentes valeurs:

| Paramètre | Description |
|---|---|
| **CV_CHAIN_APPROX_NONE** | Détecte l'intégralité des pixels composants un contour |
| **CV_CHAIN_APPROX_SIMPLE** | Détecte les contours en simplifiant ces derniers à des vecteurs. Utilise alors les coordonnées des points des extrémités composants les vecteurs |
| **CV_CHAIN_APPROX_TC89_L1** | Utilisation d'un algorithme de Chin basé sur la détection des points dominants dans l'image |
| **CV_CHAIN_APPROX_TC89_KCOS** | Autre algorithme de Chin |

Vous verrez plus loin comment utiliser ces fonctions en 4.17.8.2, dans la partie sur les détections de contours avancés.

#### 4.17.4.15 HaarDetectObject

Cette méthode permet d'utiliser ce qu'on appelle des cascades Haar afin d'effectuer des détections diverses au sein d'une image.

C'est notamment via cette méthode que l'on peut réaliser de la détection de nez, des yeux, de visage, de main, …

```
1 face = cv.HaarDetectObjects(img_src, haarcasc, storage, 1.2, 2, 0, (20, 20))
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMAGE | CVARR | - | Image source |
| 2 | CASCADE | CVHAAR | - | Algorithme de détection à utiliser |
| 3 | STORAGE | CVMEM STORAGE | - | Espace de mémoire alloué pour le calcul |
| 4 | SCALE_ FACTOR | DOUBLE | 1.1 | Spécifie le diviseur appliqué à l'image à chaque recherche |
| 5 | MIN_ NEIGHBORS | INT | 3 | Indique la précision de recherche à utiliser. Plus le chiffre sera élevé, plus la recherche sera fine |
| 6 | FLAGS | INT | 0 | Non utilisé, laisser à 0 |
| 7 | MIN_SIZE | CVSIZE | (0,0) | Taille limite à laquelle la recherche doit s'arrêter |

Une petite précision ici. La recherche part avec la taille d'origine de l'image. Si rien n'est trouvé, alors le traitement applique une division de l'image par **SCALE_FACTOR**, et ce jusqu'à atteindre **MIN_SIZE**.

Aussi, en cas de problème de fluidité, vous pouvez jouer sur ces deux paramètres.

#### 4.17.4.16 InRangeS

Derrière ce nom de méthode énigmatique se cache une fonction permettant de traiter en masse une image.

Je m'explique: il suffit de passer une image en entrée puis d'indiquer une limite basse de pixel, et une limite hausse de pixel. L'ensemble des pixels dont la valeur se situe entre ces deux seuils sera blanc dans l'image de sortie. Les autres pixels seront noirs.

Cette fonction sert principalement pour isoler des couleurs, chose que nous verrons plus loin en détail.

```
1 cv.InRangeS(img_source, cv.Scalar(h,s,v), cv.Scalar(h,s,v), img_cible)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image source |
| 2 | LOWER | CVSCALAR | - | Seuil bas (en mode HSV) |
| 3 | UPPER | CVSCALAR | - | Seuil haut (en mode HSV) |
| 4 | DST | CVARR | - | Image de sortie |

*Illustration 15: Résultat d'un InRangeS*

### 4.17.4.17 Load

**Load** permet comme son nom l'indique de charger un fichier. Nous verrons plus loin comment l'utiliser pour la détection de visage.

```
1 cv.Load(filename)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | FILENAME | CHAR | - | Fichier source à charger |

### 4.17.4.18 MatchTemplate

**MatchTemplate** est une méthode qui vous permet de rechercher une image au sein d'une autre image.

```
1 resultat = cv.MatchTemplate(img, template, method)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMAGE | CVARR | - | Image d'entrée (8 ou 32 bits) |
| 2 | TEMPL | CVARR | - | Image à rechercher (même type que l'image d'entrée, et d'une taille inférieure ou égale) |
| 3 | METHOD | INT | - | Type de recherche |

Comme on peut le constater ici, nous récupérons une image en retour de l'appel dans résultat.

Résultat est en réalité une image un peu particulière de par sa construction. Pour commencer, elle doit obligatoirement être de type **IPL_DEPTH_32F**. Ensuite, en admettant que la résolution de img soit W*H et celle de template de w*h, alors sa taille doit être strictement de (W-w+1)*(H-h+1).

```
1 result = cv.CreateImage(((W-w+1),(H-h+1)),cv.IPL_DEPTH_32F, 1)
```

***Remarque: Pour rappel, la largeur d'une image est image.width et sa hauteur image.height***

Concernant les paramètres, précisons qu'il existe 5 types de méthodes différentes à essayer selon vos besoins.

| Paramètre |
|---|
| CV_TM_SQDIFF_NORMED |
| CV_TM_CCORR |
| CV_TM_CCORR_NORMED |
| CV_TM_CCOEFF |
| CV_TM_CCOEFF_NORMED |

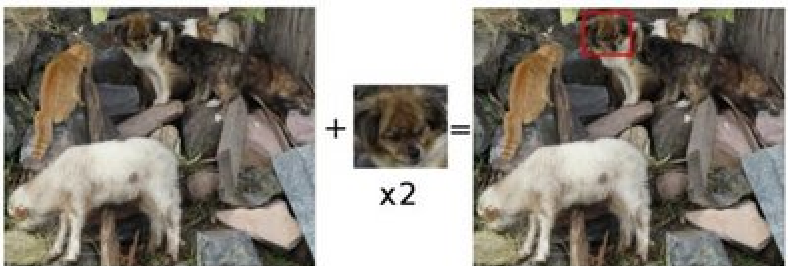



***Illustration 16: Résultat d'un MatchTemplate***

## 4.17.4.19 MemoryStorage

Cette méthode créé un espace de stockage dynamique utile pour les calculs effectués par certaines méthodes **OpenCV**.

L'unique paramètre correspond à la taille allouée. La valeur par défaut, 0, correspond à un espace de 64K.

```
1 storage = cv.CreateMemStorage(0)
```

### 4.17.4.20 MinMaxLoc

Cette méthode permet d'obtenir les coordonnées de points significatifs depuis une image binaire. Elle est très souvent utilisée après un **MatchTemplate**

***Remarque: Pour rappel, la référence est le pixel blanc.***

```
min_val, max_val, min_loc, max_loc = cv.MinMaxLoc(resultat)
(x,y) = min_loc
(x1,y1) = max_loc
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMAGE | CVARR | - | Image d'entrée (1 canal) |

En sortie, nous récupérons la valeur la plus faible puis la valeur la plus élevée. Ces deux retours sont rarement utiles dans les faits.

Suivent deux tuples: min_loc et max_loc.

Dans notre cas, x et y représenteront les coordonnées du coin supérieur gauche d'un rectangle et x1 & y1 les coordonnées du coin inférieur droit.

### 4.17.4.21 RunningAvg

Il s'agit ici de tenir compte de l’évolution d'une scène saisie au fil du temps.

Chaque nouvelle image capturée est additionnée à ce qu'on appelle un accumulateur, puis on fait la moyenne de cette somme pour chaque pixel. L'image résultante est alors injectée dans l'accumulateur.

Il en résulte une image qui garde pendant un certain temps des traces des variations de l'image dans le temps.

Pour vous donner une image plus parlante, comparer cela avec un avion en vol. Quand vous le regardé dans le ciel, ses moteurs laissent des traînées de nuages derrière eux, dessinant le chemin de l'avion. Au bout d'un certain temps ces nuages se dissipent à l'extrémité de la traînée.

Le concept est ici exactement le même.

S'il existe différentes méthodes concernant les accumulateurs, c'est celle du **RunningAvg** qu'il faut retenir.

```
1 cv.RunningAvg(img_source, accu, ratio)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMAGE | CVARR | - | Image d'entrée |
| 2 | ACC | CVARR | - | Image servant d'accumulateur |
| 3 | ALPHA | DOUBLE | - | Poids de l'image d'entrée |

Les paramètres passés sont l'image à ajouter à l'accumulateur, l'accumulateur lui-même et un ratio.

Pour information, l'accumulateur n'est rien d'autre qu'une image. Le ratio quant à lui est compris entre 0.0 et 1.0. Plus ce ratio sera élevé, et plus la nouvelle image aura d'importance dans le calcul.

**acc(x,y) = (1-ratio)*acc(x,y) + ratio * img(x,y)**

On voit de suite l'importance qu'a ce ratio dans le temps durant lequel les traces resteront en mémoire.

#### 4.17.4.22 Smooth

Effectuer un " **Smooth** ", c'est effectuer un lissage ou un moyennage.

Sur une image, cela revient, pour un pixel donné à prendre les x pixels autour, à en calculer la moyenne et à appliquer cette moyenne au pixel donné.

Quel intérêt cela peut-il avoir? Prenez une image N&B contenant des pixels parasites noir sur le fond blanc. Effectuer ce type de traitement va transformer les pixels parasites noirs en gris très clair.

En d'autres termes, lisser une image revient à atténuer les défauts. Nous pouvons considérer cela comme un nettoyage de l'image. Vous comprendrez tout l'intérêt de ce type de traitement à la lecture du **Threshold**.

```
1 cv.Smooth(img_source,img_cible,cv.CV_BLUR,p1,p2)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image d'entrée |
| 2 | DST | CVARR | - | Image de sortie |
| 3 | SMOOTHTYPE | INT | CV_GAUSSIAN | Type de lissage |
| 4 | PARAM1 | INT | 3 | Paramètre 1 de lissage |
| 5 | PARAM2 | INT | 0 | Paramètre 2 de lissage |

Les deux premiers paramètres ne posent, ici, pas de soucis.

Le troisième est le type de lissage désiré. Les principaux à retenir sont les suivants.

| Paramètre | Description |
|---|---|
| **CV_BLUR** | Moyennage linéaire sur un rectangle de p1 pixels par p2 pixels. |
| **CV_GAUSSIAN** | Moyennage gaussien sur un rectangle de p1 pixels par p2 pixels |
| **CV_MEDIAN** | Moyennage sur un carré de p1 pixels de côté |

P1 et p2 représentent la taille du curseur de traitement à utiliser sur l'image, en pixels.

***Illustration 17: Application d'un Smooth***

#### 4.17.4.23 Threshold

Appliqué sur une image grayscale, le **threshold** permet d'obtenir une image dite binaire.

Il s'agit ni plus ni moins que d'une image ne possédant que des pixels de deux couleurs: blanc ou non (noir en général).

En grayscale, chaque pixel possède une valeur comprise entre 0 & 255.

A partir d'un seuil passé en paramètre, et compris entre 0 & 255, le **threshold** générera une image binaire ou les pixels d'un côté du seuil seront blanc et ceux de l'autre seront d'une valeur définie par l'utilisateur.

```
1 cv.Threshold(img_source, img_cible, seuil, val, cv.CV_THRESH_BINARY_INV)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | SRC | CVARR | - | Image d'entrée |
| 2 | DST | CVARR | - | Image de sortie |
| 3 | TRESHOLD | DOUBLE | - | Valeur de seuil |
| 4 | MAXVALUE | DOUBLE | - | Valeur à utiliser, selon paramètre 5 |
| 5 | TRESHOLDTYPE | INT | - | Type de sortie |

Les trois premiers paramètres se passent de commentaires.

Le quatrième paramètre, val, est la valeur désirée pour les pixels non blancs. S'il est possible d'utiliser des nuances de gris, en général, nous passons toujours 255 comme valeur afin d'obtenir un pixel noir.

Le dernier paramètre définit le fonctionnement du **Threshold**.

| Paramètre | Description |
|---|---|
| **THRESH_BINARY** | Si pixel > seuil, alors pixel = val, 0 sinon |
| **THRESH_BINARY_INV** | Si pixel > seuil, alors pixel = 0, val sinon |
| **THRESH_TRUNC** | Si pixel > seuil, alors pixel = seuil |
| **THRESH_TOZERO** | Si pixel > seuil, alors pixel = pixel, 0 sinon |
| **THRESH_TOZERO_INV** | Si pixel > seuil, alors pixel = 0, pixel = pixel sinon |

### 4.17.5 Ajout d'élément sur une image/frame

#### 4.17.5.1 Rectangle

Pour dessiner un rectangle sur une image **OpenCV** c'est très simple via la méthode Rectangle.

```
1 cv.Rectangle(img, pt1, pt2, cv.CV_RGB(red, green, blue), epaisseur, type_ligne)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMG | CVARR | - | Image d'entrée |
| 2 | PT1 | CVPOINT | - | Coordonnées coin supérieur gauche |
| 3 | PT2 | CVPOINT | - | Coordonnées coin inférieur droit |
| 4 | COLOR | CVSCALAR | - | Couleur des traits |
| 5 | THICKNESS | INT | 1 | Épaisseur du trait |

| 6 | LINETYPE | INT | 8 | Type de trait |
|---|---|---|---|---|

Après avoir précisé à quelle image/**frame**, nous désirons appliquer le dessin, nous passons les coordonnées du coin supérieur gauche sous forme d'un tuple (x,y) (pt1) puis du coin inférieur droit (pt2) du rectangle.

Nous précisons ensuite la couleur via la méthode **cv.CV_RGB** qui nous permet de simplifier le choix de la couleur.

Épaisseur est de préférence à mettre à 1; -1 si vous souhaitez que le rectangle soit rempli.

type_ligne permet de préciser l'épaisseur du trait souhaité pour le traçage du rectangle.

Dans les faits, la nuance est subtile entre épaisseur et type_ligne et peu importante. Je vous conseille de gérer l'épaisseur du trait via type_ligne et de n'utiliser épaisseur que pour indiquer si le cercle doit être vide (valeur positive) ou non (valeur négative).

## 4.17.5.2 Cercle

Dessiner un cercle n'est pas plus difficile qu'un rectangle.

```
1  cv.Circle(img, centre, rayon, cv.CV_RGB(red, green, blue), epaisseur, type_ligne)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMG | CVARR | - | Image d'entrée |
| 2 | CENTER | CVPOINT | - | Coordonnées du centre |
| 3 | RADIUS | INT | - | Rayon |
| 4 | COLOR | CVSCALAR | - | Couleur des traits |
| 5 | THICKNESS | INT | 1 | Épaisseur du trait |
| 6 | LINETYPE | INT | 8 | Type de trait |

Comme nous pouvons le voir, ici rien de sorcier. Nous passons d'abord l'image/**frame** impactée, puis un tuple contenant les coordonnées du centre, le rayon du cercle, sa couleur puis ses épaisseurs.

Sur ce dernier point, quelques précisions. Le paramètre épaisseur est relatif au tracé lui-même alors que le type de ligne correspond à l'extérieur du tracé.

#### 4.17.5.3 Ligne

Pour finir, le traçage d'une ligne.

```
1 cv.Line(img, pt1, pt2, cv.CV_RGB(red, green, blue), epaisseur, type_ligne)
```

| N° param | Nom | Type | Valeur par défaut | Rôle / Description |
|---|---|---|---|---|
| 1 | IMG | CVARR | - | Image d'entrée |
| 2 | PT1 | CVPOINT | - | Coordonnées de départ |
| 3 | PT2 | CVPOINT | - | Coordonnées d'arrivée |
| 4 | COLOR | CVSCALAR | - | Couleur du trait |
| 5 | THICKNESS | INT | 1 | Épaisseur du trait |
| 6 | LINETYPE | INT | 8 | Type de trait |

En vous basant sur les rectangles et les cercles, je pense que le commentaire des paramètres est superflu.

### 4.17.6 Détection de mouvement

La détection de mouvement est un grand classique dans les exercices vidéos, au même titre que la détection de couleur.

Dans les fait comment cela se passe-t-il?

On considère qu'il y a mouvement à partir du moment ou nous constatons une différence entre deux scènes consécutive.

Au niveau d'une caméra/webcam, cela se traduit par un changement de pixels. Mais comment détecter cela, en informatique? Il suffit simplement de comparer les deux images.

Nous avons vu une fonction pour réaliser cela: ***AbsDiff***.

En réalisant la soustraction des deux images, nous récupérerons une image blanche contenant les pixels ayant changé.

Mais cela peut s'avérer alors encore difficile, surtout si vos images sont en couleur.

La solution dans ce cas là est de ne pas réaliser **l'AbsDiff** sur vos images couleur mais sur des images en grayscale.

L'image résultante de la comparaison peut alors être binariser.

Rappelez vous, cela passe par la méthode ***Threshold***, et consiste à comparer les niveau des pixels à un seuil puis selon leur position par rapport à ce dernier, à les passer en pixel blanc ou noir.

Il ne vous reste plus qu'a comparer la proportion de pixel noir rapport aux pixels blancs pour déterminer si l'image semble avoir détecter un mouvement digne d'attention.

Le problème que vous constaterez est que parfois des petits pixels parasites s'invitent et influent trop sur le résultat final. Il faut alors jouer avec des méthodes telle **Smooth**, **Erode** et **Dilate** pour optimiser l'image pour traitement.

Comme vous le constatez, détecter un mouvement est moins difficile qu'il n'y paraît.

Bien entendu, cela est ici simplifié mais le principe est viable.

Résumons tout cela par un schéma simple.

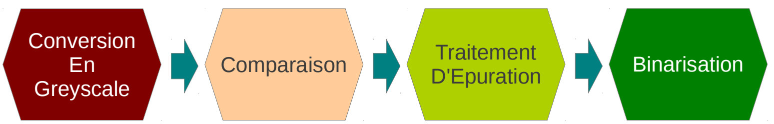



***Illustration 18: Chaîne de traitement pour la détection de mouvement***

## 4.17.6.1 Exemple

```
#! /usr/bin/env PYTHON
# -*- coding: utf-8 -*-




import cv2.cv as cv
import time




if __name__=='__main__':
    """
        Detection de mouvement, appui sur 'q' pour quitter
    """
    capture = cv.CaptureFromCAM(0)
    cv.NamedWindow("Test_Webcam")
    while True:
        frame = cv.QueryFrame(capture)
        grey1 = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
        grey2 = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
        res = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
        cv.CvtColor(frame, grey1, cv.CV_RGB2GRAY)
        time.sleep(0.001)
        frame = cv.QueryFrame(capture)
        cv.CvtColor(frame, grey2, cv.CV_RGB2GRAY)
        cv.AbsDiff(grey1, grey2, res)
        cv.Smooth(res, res, cv.CV_BLUR, 5,5)
        cv.Threshold(res,res,15,255,cv.CV_THRESH_BINARY_INV)
        cv.ShowImage("Test_Webcam", res)

        if (cv.WaitKey(10) % 0x100) == 113:
            break
```

### 4.17.7 Reconnaissance des couleurs

Après avoir vu la détection de mouvements nous allons monter un peu en difficulté: la détection de couleur.

Beaucoup d'entre vous se disent sans doute qu'il suffit de choisir sa couleur et de la traquer.

J'avoue que c'est également ce que je me disais avant d'attaquer la théorie et la pratique; mais malheureusement cela est un peu plus compliqué.

En effet, une image, telle que nous la connaissons est codée en RGB. Autrement dit, quelle que soit la couleur choisie, à l'exception du rouge/bleu/vert pur, il y aura toujours une composante d'une autre couleur venant parasiter le résultat.

Nous allons donc devoir changer de type d'encodage. Nous nous dirigerons vers du HSV (Hue/Saturation/Value, ou en français Teinte/Saturation/Luminosité).

Pourquoi ce format? A cause de sa simplicité dans ce type d'application.

En HSV, la teinte représente la couleur désirée, la saturation représente l'intensité de la couleur, et la luminosité la brillance de la couleur.

En d'autres termes, le seul paramètre dont nous avons à nous soucier dans ce cas est la teinte.

Comme le montre l'échelle de teintes ci après, il est simple de choisir sa couleur.

***Remarque: OpenCV utilise des demies valeurs sur le paramètre Teinte. Autrement dit, pensez à diviser les valeurs issues de l'échelle ci -après par 2.***

***Illustration 19: Échelle de teintes issues de wikipedia (Huescale.svg)***

Ou une fois converties en valeur **OpenCV**.

| **0 à 15** | Rouge |
|---|---|
| **15 à 45** | Jaune |
| **45 à 75** | Vert |
| **75 à 105** | Cyan |
| **105 à 135** | Bleu |
| **135 à 165** | Magenta |
| **165 à 180** | Rouge |

Bien entendu, il faut par la suite ajuster le tir par l'expérimentation afin de toujours diminuer les parasites, et/ou appliquer un traitement d'ouverture pour améliorer le résultat de sortie.

La saturation et la luminosité seront en général pris entre 0 et 255.

Nous utiliserons la méthode ***InRangeS*** afin d'isoler notre couleur recherchée et générer l'image finale.

***Remarque: Le seuil bas doit toujours contenir la valeur la plus petite et ne peut être inférieur à 0. Le seuil haut doit toujours contenir la valeur la plus haute et ne peut être supérieur à 180.***

***Illustration 20: Chaîne de traitement d'une détection de couleur***

#### 4.17.7.1 Exemple

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4  import cv2.cv as cv
5
6  if __name__=='__main__':
7      """
8          Permet de suivre la couleur bleu, appui sur 'q' pour quitter
9      """
10     ma_camera = cv.CaptureFromCAM(0)
11     cv.NamedWindow("Test_Webcam")
12     while True:
13         ma_frame = cv.QueryFrame(ma_camera)
14         ma_frame2 = cv.CreateImage(cv.GetSize(ma_frame), 8, 3)
15         ma_frame3 = cv.CreateImage(cv.GetSize(ma_frame2), 8, 1)
16         cv.CvtColor(ma_frame, ma_frame2, cv.CV_BGR2HSV)
17         #Recherche du bleu
18         cv.InRangeS(ma_frame2,cv.Scalar(90, 0, 0), \
19                 cv.Scalar(130, 255, 255), ma_frame3)
20         cv.ShowImage("Test_Webcam", ma_frame3)
21
22         if (cv.WaitKey(10) % 0x100) == 113:
23             break
```

### 4.17.8 Détection de contours

Montons encore d'un petit cran pour atteindre la détection de contours.

Si vous vous rappelez bien, nous avons vu cela un peu plus tôt. On appelle cela un **GRADIENT**. Il s'agit tout simplement d'une opération **d'Erode** soustraite à une opération de **Dilate**.

Pour effectuer la différence, il suffit d'utiliser un **AbsDiff**. Nous récupérons alors en sortie une image ne contenant que les contours des formes, en blanc sur fond noir.

#### 4.17.8.1 Exemple basique

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4
5
6
7  import cv2.cv as cv
8  import time
9
10
11
12
13 if __name__=='__main__':
14     capture = cv.CaptureFromCAM(1)
15     cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_WIDTH , 1200)
16     cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_HEIGHT, 1200)
17     cv.NamedWindow("Test_Webcam")
18     cv.ResizeWindow("Test_Webcam", 200,200)
19     while True:
20         frame = cv.QueryFrame(capture)
21         grey1 = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
22         grey2 = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
23         grey3 = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
24         res = cv.CreateImage(cv.GetSize(frame), cv.IPL_DEPTH_8U, 1)
25         cv.CvtColor(frame, grey1, cv.CV_RGB2GRAY)
26         cv.CvtColor(frame, grey2, cv.CV_RGB2GRAY)
27         cv.Erode(grey1, grey1, None, 3)
28         cv.Dilate(grey2, grey2, None, 3)
29         cv.AbsDiff(grey1, grey2, grey3)
30         cv.ShowImage("Test_Webcam", grey3)
31
32         if (cv.WaitKey(10) % 0x100) == 113:
33             break
```

### 4.17.8.2 Exemple avancé

Sur **OpenCV**, il existe une façon encore plus simple d'effectuer une détection de contours.

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4
5
6
7  import cv2.cv as cv
8
9
10
11
12 if __name__=='__main__':
13     ma_camera = cv.CaptureFromCAM(1)
14     cv.NamedWindow("Test_Webcam")
15     cv.SetCaptureProperty(ma_camera, cv.CV_CAP_PROP_FRAME_WIDTH , 1200)
16     cv.SetCaptureProperty(ma_camera, cv.CV_CAP_PROP_FRAME_HEIGHT, 1200)
17     while True:
18         ma_frame = cv.QueryFrame(ma_camera)
19         ma_frame2 = cv.CreateImage(cv.GetSize(ma_frame), 8, 3)
20         ma_frame3 = cv.CreateImage(cv.GetSize(ma_frame2), 8, 1)
21         cv.CvtColor(ma_frame, ma_frame2, cv.CV_BGR2HSV)
22         #Recherche du rouge
23         cv.InRangeS(ma_frame2,cv.Scalar(175, 0, 0),cv.Scalar(180, 255, 255), \
24                                         ma_frame3)
25         #Traitement d'image
26         cv.Erode(ma_frame3, ma_frame3, None, 3)
27         cv.Dilate(ma_frame3, ma_frame3, None, 5)
28         cv.Smooth(ma_frame3, ma_frame3, cv.CV_BLUR, 5,5)
29         #Detection de contours
30         storage = cv.CreateMemStorage(0)
31         contours = cv.FindContours(ma_frame3, storage, \
32                     cv.CV_RETR_EXTERNAL, cv.CV_CHAIN_APPROX_SIMPLE)
33         NewContours = contours
34         cv.DrawContours(ma_frame, NewContours, (0,0,255),(0,255,0),2,2,8)
35         #IHM
36         cv.ShowImage("Test_Webcam", ma_frame)
37
38         if (cv.WaitKey(10) % 0x100) == 113:
39             break
```

***Illustration 21: Résultat d'une détection de contours avec traçage***

## 4.17.9 Détection d'une forme

La détection d'une forme précise est quelque chose d'également assez simplifié dans **OpenCV**.

Il suffit juste d'avoir une image/frame à analyser, et la forme/image que l'on recherche dedans.

### 4.17.9.1 Recherche d'une image dans l'image

Voici un petit code d'exemple:

```
#! /usr/bin/env PYTHON
# -*- coding: utf-8 -*-




import cv2.cv as cv




```

```
12 image = cv.LoadImage('test0.jpg')
13 reference = cv.LoadImage('test1.jpg')
14
15 image_size = cv.GetSize(image)
16 reference_size = cv.GetSize(reference)
17 result_size = [ s[0] - s[1] + 1 for s in zip(image_size, reference_size) ]
18
19 result = cv.CreateImage(result_size, cv.IPL_DEPTH_32F, 1)
20
21 cv.MatchTemplate(image, reference, result, cv.CV_TM_CCORR_NORMED)
22
23 min_val, max_val, min_loc, max_loc = cv.MinMaxLoc(result)
24 cv.Rectangle(image,max_loc,min_loc,(0, 0, 255), 3,8,0)
25 cv.SaveImage("test2.png", image)
```

Voilà, 25 petites lignes pour tout faire. L'ensemble des éléments constituant ce code ont tous été vus en détail.

Lignes12 & 13 nous chargeons nos images.

Lignes 15 à 17, nous déterminons la taille précise que doit avoir l'image de sortie. Il s'agit d'une autre façon de coder le calcul à effectuer, tel que vu en 4.17.4.18.

Ligne 19, nous créons notre image de sortie, avec un seul canal, pour ne fonctionner qu'en greyscale.

Ligne 21, nous recherchons notre référence dans l'image à l'aide de la méthode ***CV_TM_CCORR_NORMED***.

Ligne 23, nous récupérons différentes valeurs dont les coordonnées précises de détection permettant de tracer le rectangle, ligne 24.

Enfin, ligne 25, nous sauvegardons le résultat dans une nouvelle image.

### 4.17.10 Détection de visage

La détection de visage et de certain de ses éléments (yeux, bouche, ...) fait partie de ces choses assez complexes.

Avec **OpenCV**, cela se simplifie grandement. Nous allons essayer de détailler cela au mieux, mais Internet saura combler les questions qui subsisteront.

Comme dans tout traitement, la première chose à faire est de passer notre image en gris sur 8 bits.

Il faut ensuite créer un espace de stockage qui permettra de stocker les calculs de détection. Pour cela, on utilise ***CreateMemStorage(0)***.

On va ensuite égaliser l'histogramme de l'image greyscale via un EqualizeHist.

Puis nous allons définir l'algorithme de détection que nous allons utiliser. Nous utiliserons par exemple haarcascade_frontalface_alt.xml pour le visage

Cet algorithme est chargé via la méthode ***Load***

Enfin, il reste à détecter ce que l'on recherche à l'aide de l'algorithme via la méthode ***cv.HaarDetectObjects***.

#### 4.17.10.1 Exemple

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4
5
6
7  import sys
8  import cv2.cv as cv
9
10
11
12
13 def detect(image):
14     image_size = cv.GetSize(image)
15     grayscale = cv.CreateImage(image_size, 8, 1)
16     cv.CvtColor(image, grayscale, cv.CV_RGB2GRAY)
```

```
17      storage = cv.CreateMemStorage(0)
18      cv.EqualizeHist(grayscale, grayscale)
19      cascade = cv.Load('./haarcascade_frontalface_alt.xml')
20      faces = cv.HaarDetectObjects(image, cascade, cv.CreateMemStorage(0), \
21                                                  1.2, 2, 0, (20, 20))
22      if faces:
23          print 'face detected!'
24          for i in faces:
25              cv.Rectangle(image,(i[0][0], i[0][1]),(i[0][0] + i[0][2], i[0][1] + i[0][3]),\
26                                  (0, 255, 0), 3,8,0)
27
28
29
30
31  if __name__ == '__main__':
32      cv.NamedWindow('caméra')
33      capture = cv.CreatecaméraCapture(1)
34      while 1:
35          frame = cv.QueryFrame(capture)
36          detect(frame)
37          cv.ShowImage('caméra', frame)
38          k = cv.WaitKey(10)
39          if (cv.WaitKey(10) % 0x100) == 113:
40              break
```

### 4.17.11 Création d'un algorithme de détection

Les **algorithmes Haar**, quoique complexes, peuvent être conçus maison.

Nous allons voir ici les grandes lignes des étapes permettant d'aboutir à la génération d'un fichier **XML** servant pour une détection **Haar**. Si certain points vous semblent flous, je vous invite alors à rechercher sur Internet le terme Haartraining.

Vous pourrez trouver notamment, plus d'informations sur la mesure de performance des algorithmes générés entre autres.

Cependant, avant de commencer, sachez que cette procédure est longue et fastidieuse. Ne partez pas sur l'idée de le concevoir en une nuit.

#### 4.17.11.1 Création des positifs

Il s'agit là de la première étape. Nous allons créer ce que l'on appelle des positifs. Il s'agit ni plus ni moins que d'images contenant l'objet que nous souhaitons savoir détecter, pris sous différents angles, et éventuellement également différents types d'éclairage. Plus vous en aurez, plus la détection sera efficaces.

***Remarque: A titre d'information, des algorithmes tel que celui permettant de détecter les visages ont été conçus avec plusieurs milliers de positifs et mis plusieurs semaines à générer le fichier XML de détection de visages.***

Ces images serviront plus loin à générer le fichier vecteur.

Concernant le besoin d'avoir un fond neutre (couleur unie) ou non (par exemple une pièce avec ses meubles), cela ne semble pas avoir d'importance particulière.

Cependant, si un fond neutre vous simplifiera la vie plus loin, un fond non neutre permettra également, dans certain cas, d'améliorer la détection de l'objet désiré.

Mon conseil est de privilégier un fond neutre, mais d'augmenter la quantité de positifs et de négatifs pour compenser.

#### 4.17.11.2 Création des négatifs

Complément des positifs, les négatifs sont des images ne contenant absolument pas l'objet que nous souhaitons détecter.

Idéalement, il faut la même quantité de négatifs que de positifs. Cela permettra d'apprendre à l'algorithme (si l'on peut utiliser cette expression) à ne pas détecter l'objet.

#### 4.17.11.3 Marquage des positifs

Étape suivante indispensable, le marquage consiste à préciser dans les positifs où se trouve l'objet que nous souhaitons détecter, afin que l'algorithme sache quoi chercher.

Cela se fait avec l'utilitaire **ObjectMarker** (à charger sur le net, puis à compiler: objectmarker.cpp; ou charger l'equivalent **PYTHON**).

Pour le lancer depuis un terminal, il suffit de taper

```
1 ObjectMarker fichier_sortie.txt <chemin ou se trouve les positifs>
```

Dans cet exemple, fichier_sortie.txt est généré par le traitement. Il contiendra pour chaque image, la référence de l'image, ainsi que les coordonnées de l'objet à détecter.

Il s'agit là d'une étape longue et fastidieuse mais nécessaire. N'hésitez pas à vous appliquer. Plus cette étape sera faite avec soin, plus fiable sera la détection finale.

Pour marquer l'objet, sur l'image qui apparaît, imaginer un rectangle encadrant l'objet à détecter, et cliquer à l'emplacement du coin supérieur gauche, puis à l'emplacement du coin inférieur droit.

***Attention: Seuls ces deux coins doivent être utilisés. En effet, le traitement ne sait pas gérer les autres coins pour générer le fichier de sortie.***

Si jamais le rectangle tracé vous semble inadéquat, il suffit de recommencer l'opération en cliquant sur le coin supérieur gauche puis le coin inférieur droit.

Si le rectangle vous convient, il faut alors taper sur espace pour le valider, puis de taper sur <ENTREE> pour sauvegarder et passer à l'image suivante.

Une fois la totalité des marquages terminés, il suffit de taper sur <ESCAPE> pour sauvegarder le fichier de sortie et fermer ObjectMarker.

Le fichier de sortie contiendra entre autres le nom de l'image, les coordonnées des coins du rectangle sélectionné, et la largeur et hauteur de ce dernier.

#### 4.17.11.4 Création du fichier vecteur

Une fois les positifs marqués, il faut empaqueter l'ensemble des images dans un fichier vectoriel, appelé fichier vecteur.

Pour ce faire, nous utilisons un utilitaire fourni par **OpenCV** qui s'appelle opencv-createsamples.

Nous lui fournirons comme paramètres le chemin du fichier généré à l'étape précédente (via l'option -info); le chemin de sortie, ainsi que le nom désiré pour le fichier vecteur (via l'option -vec); la hauteur désirée de l'échantillon, pour la future détection, ainsi que sa largeur (via les options respectives -h & -w).

```
opencv-createsamples -info ./output.txt -vec ./pos.vec -w 50 -h 50
```

#### 4.17.11.5 Création du classificateur

L'ultime étape. Je ne saurais dire si le terme de " classificateur " correspond à une traduction juste. Je pense que nous parlerons d'algorithme de **Haar**.

Car c'est cela que nous allons voir ici: sa création.

Nous allons pour cela utiliser un autre outil fourni par **OpenCV**: opencv-haartraining.

```
Opencv-haartraining -data ./haar_algo -vec ./pos.vec -bg ./neg.txt \
                              -npos 1000 -nneg 1000 -nstages 20
```

L'option -data permet de spécifier où générer et comment appeler notre algorithme de sortie (un dossier et un fichier XML seront générés).

L'option -vec permet d'indiquer ou se trouve le fichier vecteur.

L'option -bg est un simple fichier texte contenant un listing des négatifs. Le format sera le suivant:

```
1 /negative/im00.jpg
2 ./negative/im01.jpg
3 ./negative/im02.jpg
4 ...
```

Les options -npos et -nneg servent respectivement à indiquer le nombre de positifs et de négatifs.

L'option -nstage, enfin, permet de stipuler au générateur d'algorithmes le nombre d’entraînements qu'il doit exécuter. Plus ce nombre sera élevé, meilleure sera la détection, mais plus longue sera la génération de l'algorithme.

***Attention: La génération de l'algorithme peut prendre jusqu'à plusieurs semaines, en fonction du nombre d'images et de la complexité de l'algorithme final.***

## 4.18 Les Threads

Un **thread** est une tâche de fond dans laquelle nous pouvons implémenter certaines actions précises.

Prenons un exemple concret: les messageries instantanées. Si nous respectons une programmation séquentielle, la messagerie attendrait que vous saisissiez un message, puis l'enverrait, puis attendrait une réponse avant de vous autoriser à saisir un nouveau message.

Cela ne serait pas pratique. Pour y palier, nous pouvons donc utiliser plusieurs threads. Le premier se chargera uniquement de prendre en compte votre saisie et de l'envoyer tout en émettant un écho sur l'écran, et le second se contentera d'attendre un message avant de vous l'afficher.

L'avantage ici réside dans le fait que les **threads** ne s'exécutent pas de manière séquentielle mais simultanée, en parallèle. Cela peut permettre de gagner beaucoup de temps et de fluidité dans une application.

***Attention: Il existe deux types de threads. Les premiers gèrent des processus totalement indépendants mais nécessitant des actions parallèles. Les seconds gèrent des processus exécutés en parallèle mais accédant aux mêmes ressources (par exemple, un champ texte). Les threads présentés ci-après correspondent aux premier type.***

### 4.18.1 Création

Pour créer un **thread**, il faut commencer par importer le module dédié

```
1 import threading
```

Ensuite, il suffit d'appeler le constructeur **threading.Thread()**. Ce constructeur prend plusieurs paramètres, 5 pour être plus précis:

=>group
=>target
=>name
=>args
=>kwargs

Le premier, group, sert lorsque l'on désire créer des threads pour les regrouper par ensemble. S'agissant d'une fonctionnalité avancée, nous le positionnerons toujours à None.

Le seconde, target, est la fonction/procédure (juste le nom sans parenthèses) que doit gérer le **thread**.

name est tout simplement le nom que nous désirons donner au **thread**. Si vous ne désirez pas lui en donner, vous pouvez saisir none.

args est un tuple contenant l'ensemble des paramètres pour la fonction/procédure donnée dans target

kwargs est un dictionnaire pour passer de manière nommée les arguments pour la fonction/procédure donnée dans target

***Remarque: Il est possible dans certain cas de passer une partie des paramètres dans args et une autre dans kwargs***

```
mon_thread = threading.Thread(None, ma_fonction, nom_thread, (P1,P2),{})
```

### 4.18.2 Lancement

Pour lancer un **thread**, rien de très compliqué. Il suffit simplement d'utiliser la méthode **start()** après avoir créé le **thread**

```
mon_thread = threading.Thread(...)
mon_thread.start()
```

### 4.18.3 Arrêt

Pour arrêter un **thread**, la méthode la plus simple et la plus directe (même si pas forcément la meilleure) consiste à utiliser

```
mon_tread = threading.Thread(...)
mon_thread.start()
…
mon_thread._Thread__stop()
```

### 4.18.4 Pause

Un **thread** ne se met pas en pause. Il peut par contre être temporairement arrêté, puis relancé, tel que vu précédemment.

### 4.18.5 Appel toutes les x secondes

Il existe une possibilité alternative aux **threads**. En effet, s'il s'agit uniquement de lancer périodiquement une fonction (MAJ IHM par exemple), on peut utiliser le module **gobject**.

```
import gobject
```

On utilisera alors sa méthode timeout_add, laquelle prend comme premier paramètre le nombre de millisecondes au bout desquelles il faut lancer la fonction passée en second paramètre. On récupère alors un entier indiquant si l'appel et l'exécution se sont bien déroulés.

```
Gobject.timeout_add(1000,ma_fonction) #Lancement de ma_fonction toutes les s
```

### 4.18.6 Exemple

```
#! /usr/bin/env PYTHON
# -*- coding: utf-8 -*-


```

```
5
6
7  import threading
8  import time
9
10
11
12
13 def fonction_a():
14     while True:
15         print "fonction_a"
16         time.sleep(0.5)
17
18
19
20
21 def fonction_b():
22     while True:
23         print "fonction_b"
24         time.sleep(0.5)
25
26
27
28
29 def mon_thread():
30     """
31         Demarre 2 thread qui effectue des print
32     """
33     a = threading.Thread(target=fonction_a)#version simplifié pour
34     b = threading.Thread(target=fonction_b)# fonction sans paramètre
35
36     a.start()
37     time.sleep(0.5)
38     b.start()
39
40
41
42
43 if __name__ == '__main__':
44     mon_thread()
```

## 4.19 Les PDF: Reportlab

**Reportlab** est un module **PYTHON** puissant vous permettant de générer proprement des fichiers **PDF**.

Nul besoin de présenter le format **PDF**. C'est devenu un standard utilisé par bon nombre d'administrations et de sociétés à travers le monde.

Nous allons voir ici les bases de ce module. Bien entendu, comme d'habitude, si cela ne vous suffisait pas, je vous invite à aller consulter la documentation complète sur le site dédié, et à faire quelques recherches sur Internet.

### 4.19.1 Principes de fonctionnement

Afin de pouvoir utiliser correctement **Reportlab**, et avant même de voir les bases de ce module, il est indispensable de prendre connaissance de ses principes de fonctionnement.

Il ne faut pas imaginer en effet que le module va nous permettre d'écrire comme dans un traitement de texte type Writer.

**Reportlab** part d'une feuille totalement vierge, à créer, et s'appelant **canvas**, sur laquelle nous allons venir déposer différents types d'objets (texte, image, …) afin de créer un template, ou plus simplement un fichier tel qu'on le désire.

Le fonctionnement se rapprocherait donc plutôt ici de Draw que de Writer.

De plus, comme dans Draw, si l'on dispose un élément hors de la page, aucune alerte ne sera levée. De fait, il est primordial de savoir précisément la disposition désirée afin d'éviter toute mauvaise surprise.

A l'image de Draw, vous devrez gérer l'ordre d'insertion afin d'éviter tout conflit de disposition: le dernier élément inséré est placé au-dessus des autres.

Il faut également savoir que cela implique que le code soit exécuté de manière procédurale. En clair, si à un moment, vous demandez une rotation de 90°, cela veut dire que tant que vous ne changerez pas cette rotation, tout le code interprété sera tourné de 90°.

Une fois ces quelques principes assimilés, l'utilisation de Reportlab vous semblera plus naturelle. Pour ceux qui trouveraient ces principes encore un peu flou, je ne peu que les inviter à travailler un peu avec Libre Office Draw afin d'illustrer concrètement les principes explicités ici.

### 4.19.2 Les bases

#### 4.19.2.1 Canvas, format de page et unités

**Reportlab**, sait gérer les différents formats de page standard existant. Vous possédez également la possibilité de définir de toute pièce la dimension de la page voulue.

Pour des raisons de lisibilité de code et de simplicité de codage, je vous recommande la seconde option. En précisant en commentaire le format utilisé (A4, letter, ...) le code n'en sera que plus lisible.

```
1 from reportlab.pdfgen import canvas
2 from reportlab.lib.units import *
3
4 canvas = canvas.Canvas("hello.pdf", pagesize=(210.0 * mm,297.0 * mm))        #A4
```

Comme nous le voyons dans notre exemple, créer un **canvas** est très simple. Le principal est de connaître le nom du pdf à générer, et les dimensions à utiliser.

Les dimensions passées permettent d'ailleurs de définir si l'on fonctionne en portrait ou bien en paysage.

Enfin, le dernier point non négligeable ici concerne Les unités. Les principales disponibles sont le cm, le mm et le inch (pouce).

Dans notre cas, fonctionnant en système **ISO**, je vous recommande d'utiliser le mm.

***Attention: Pour Reportlab, le point d'origine est le coin inférieur gauche.***

#### 4.19.2.2 Texte

##### 4.19.2.2.1 Polices d'écriture

Par défaut, seules 14 polices d'écriture sont disponibles avec Reportlab.

| Police |
| --- |
| Courier |
| Courier-bold |
| Courier-oblique |
| Courier-BoldOblique |
| Helvetica |
| Helvetica-Bold |
| Helvetica-Oblique |
| Helvetica-BoldOblique |
| Times-Roman |
| Times-Bold |
| Times-Italic |
| Times-BoldItalic |
| Symbol |
| ZapfDingbats |

Il est cependant possible, je vous rassure, d'utiliser d'autres polices. Pour cela deux solutions: connaître le chemin d'accès sur le poste, ou la mettre à disposition avec votre programme.

Le premier choix est assez cavalier, car cela impose que vous soyez sur que l'utilisateur final disposera de la police sur son poste. Aussi, je vous recommande plutôt la solution N°2.

```
1 from reportlab.pdfbase import pdfmetrics
2 from reportlab.pdfbase.ttfonts import TTFont
3
4 pdfmetrics.registerFont(TTFont('FreeSans', './FreeSans.ttf'))
5 canvas.setFont("FreeSans", 36)
```

Dans cet exemple, nous décidons d'utiliser la police d'écriture libre FreeSans, équivalent d'Arial. Cette police, nous en mettons un exemplaire (fichier TTF) à disposition dans le même répertoire que notre script.

La déclaration auprès de **Reportlab** prend en premier paramètre le nom sous lequel on désire avoir accès à la police dans le code, et en second paramètre le chemin d'accès à la police désirée.

##### 4.19.2.2.2 Simple ligne texte

Pour dessiner une simple ligne texte, il faut passer par la série de méthodes **DrawString**.

| Fonction | Coordonnées fournies |
|---|---|
| **drawString** | Gauche |
| **drawCentreString** | Centre |
| **drawRightString** | Droite |

Lors de l'appel à une de ces méthodes, il faut passer en paramètres, les coordonnées de référence, correspondant à l'endroit où insérer le texte sur la ligne, puis le texte lui-même.

```
1 mon_canvas.drawString(10*mm, 10*mm, " Hello World ")
```

#### 4.19.2.3 Paragraphe

On peut se poser la question de l'utilité des paragraphes et/ou leurs différences avec de simples zones texte.

Eh bien, pour simplifier et résumer les choses, disons qu'une zone texte ne gère pas les retours à la ligne, les paragraphes, si.

De plus, les paragraphes peuvent être pré-paramétrés, via les styles, pour être de suite disponibles quand on en a besoin.

##### 4.19.2.3.1 Les styles

Comme leur nom l'indique, les styles permettent de définir le comportement d'un paragraphe complet.

Ainsi, quand nous créons un objet style, nous récupérons de base un certain nombre de styles prédéfinis auxquels nous pouvons rajouter des styles que nous créerons nous-même.

Les styles de bases sont: BodyText, Bullet, Code, Definition, Heading1, heading2, Heading3, Italic, Normal, Title.

Je vous invite à les essayer tous pour vous faire votre propre idée de chacun.

```
1 from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
2 from reportlab.lib.enums import *
3 …
4 types_style = getSampleStyleSheet()
5 mon_style = types_style['BodyText']
```

Voici comment avoir accès au style de base.

Concernant l'ajout d'un style, il faut avant tout savoir de quoi se compose un style de paragraphe. Nous allons voir ici les principaux, ceux qui vous seront vraiment utiles pour créer vos styles.

| Composante de style | Description |
|---|---|
| **name** | Nom de la police |
| **alignment** | Justification du texte. TA_LEFT, TA_RIGHT, TA_CENTER, TA_JUSTIFY |
| **backColor** | Couleur de fond: Color(R,G,B) |
| **firstLineIndent** | Indentation de la 1ère ligne du paragraphe. Précisez l'unité |
| **fontName** | Nom de la police d'écriture retenue |
| **fontSize** | Taille (en point) de la police. Ne pas préciser d'unité. |
| **spaceAfter** | Espace après le paragraphe. Précisez l'unité |
| **spaceBefore** | Espace avant le paragraphe. Précisez l'unité. |
| **textColor** | Couleur du texte: Color(R,G,B) |

```
1  types_style.add(ParagraphStyle(name='mon_style_perso''
2          alignment = TA_JUSTIFY,
3          backColor = None,
4          firstLineIndent = 10 * mm,
5          fontName = 'FreeSans',
6          fontSize = 14,
7          spaceAfter = 5 * mm,
8          spaceBefore = 5 * mm,
9          textColor = Color(0,0,0)
10     ))
```

### 4.19.2.3.2 Création d'un paragraphe

La création d'un paragraphe est très simple et se passe de commentaire.

```
1  from reportlab.platypus import Paragraph
2  …
3  mon_para =Paragraph('This is a very silly example', mon_style)
```

### 4.19.2.3.3 Ajout d'un paragraphe à un canvas

L'ajout d'un paragraphe à un **canvas** se déroule en deux étapes: on commence d'abord par définir la taille du paragraphe via une méthode **wrap**, puis on l'ajoute au canvas à la position désirée.

***Attention: Au niveau du wrap, c'est surtout la largeur qui compte, et prévaut sur la hauteur.***

```
1 mon_para.wrapOn(mon_canvas, 80*mm, 10*mm)
2 mon_para.drawOn(canvas, 10*mm, 150*mm)
```

#### 4.19.2.4 Couleur

##### 4.19.2.4.1 Principe

Pour **Reportlab**, la couleur se définit en système RGB. Cependant nous ne fonctionnons pas ici sur un système allant de 0 à 255 pour chaque composante, mais de 0.0 à 1.0.

Cela peut être interprété comme un pourcentage. Pour vous simplifier la vie, je vous invite à notifier la valeur de la composante sur 255, puis à effectuer une division par 255 dans l'appel à la couleur afin de retomber sur un pourcentage.

```
1 (240.0/255.0)
```

***Remarque: Pour rappel, pour avoir du gris, il faut R=G=B.***

##### 4.19.2.4.2 Couleur de ligne

Pour choisir la couleur des lignes, on utilise la méthode **setStrokeColorRGB()**.

```
1 mon_canvas.setStrokeColorRGB(255.0/255, 0.0, 0.0)
```

##### 4.19.2.4.3 Couleur de remplissage

Pour définir la couleur de remplissage d'un élément dessin, on utilise la méthode **setFillColorRGB(R,G,B)**, ou R, G, B sont des floats.

```
1 mon_canvas.setFillColorRGB(0,0,196.0/255.0)
```

#### 4.19.2.4.4 Transparence

Comme nous le verrons plus loin, **Reportlab** sait gérer la transparence.

Si l'on désire remplir un élément dessin avec une couleur gérant en partie le canal alpha (canal de la transparence), il ne faut pas utiliser la méthode précédente, mais la suivante:

```
1 from reportlab.lib.colors import Color
2 ...
3 mon_canvas.setFillColor(Color(255,0,0,alpha=0.5))
```

***Attention: Même si la couleur est indiquée en RGB, on voit qu'ici nous fonctionnons sur un système allant de 0 à 255.***

***Remarque: Le canal alpha est une implémentation récente. Si vous avez un message indiquant un nombre erroné d'arguments c'est que votre version de Reportlab est trop ancienne.***

#### 4.19.2.4.5 Couleur de texte

Changer la couleur de texte est similaire à changer la couleur de remplissage d'un élément dessin.

```
1 mon_canvas.setFillColorRGB(0,0,196.0/255.0)
```

#### 4.19.2.4.6 Définir une couleur

Vous pouvez définir vos propres couleurs aisément:

```
1 from reportlab.lib.colors import Color
2 ...
3 red = Color(R,G,B)      #R, G, B, compris entre 0 et 255
```

#### 4.19.2.5 Rotation

Une rotation s'effectue via la méthode **rotate()**, en passant en argument la valeur de la rotation en degré. La rotation s'effectue dans le sens trigonométrique.

```
1 mon_canvas.rotate(90)
```

#### 4.19.2.6 Tableau

Nous allons maintenant aborder le sujet des tableaux dans les fichiers pdf.

##### 4.19.2.6.1 Tableau de base

Pour créer un tableau basique il faut peu de choses: avoir importer le bon module, et avoir un tableau de données.

```
1 from reportlab.platypus import Table, TableStyle
2 …
3 data = [['00', '01', '02', '03', '04'],['10', '11', '12', '13', '14'],
4       ['20', '21', '22', '23', '24'],['20', '21', '22', '23', '24'],
5       ['30', '31', '32', '33', '34']]
6 mon_tab = Table(data)
```

##### 4.19.2.6.2 Les styles de tableaux

Les styles de tableaux vous permettent d'appliquer du paramétrage avec une finesse pouvant descendre à la cellule.

```
1 mon_tab.setStyle(TableStyle([('BACKGROUND',(1,1),(-2,-2),colors.green), \
2                             ('TEXTCOLOR',(0,0),(1,-1),colors.red)]))
```

Comme vous pouvez le constater, nous indiquons 4 éléments à la fois.

Le premier élément est le style sur lequel nous désirons intervenir.

Viennent ensuite les éléments 2 et 3. Ils correspondent aux cellules visées par la modification que nous souhaitons effectuer.

Le premier tuple (X, Y) est la cellule de départ (on compte à partir de 0 depuis le coin supérieur gauche).

Le seconde tuple correspond à la cellule de fin. Toutes les cases comprises dans ce rectangle ainsi défini seront affectées par la modification.

Ce second tuple est particulier car il y a deux façons de procéder: on peut donner les coordonnées précises (ex: (3,2)) ou des lignes/colonnes à exclure, comme dans notre exemple. Dans les faits, je vous recommande la première solution, bien plus lisible.

***Attention: Dans un tableau Reportlab nous tournons en rond. Comprenez que quand vous atteignez la dernière colonne, vous recommencez à la première. Ainsi lorsqu'on vous dit -2, alors que vous êtes à la colonne 1, cela vous positionne sur la dernière colonne.***

Enfin, le dernier élément est le paramètre à utiliser pour le style.

Voyons maintenant plus en détails les principaux différents styles sur lesquels nous pouvons intervenir:

| STYLE | Description |
|---|---|
| **FONTNAME** | Prend en paramètre le nom d'une police définie dans Reportlab (voir partie sur le texte) |
| **FONTSIZE** | Indique la taille (en point) de la police. Ne pas préciser d'unité. |
| **TEXTCOLOR** | Choisir la couleur de police. Définissez vos propres couleurs, tel que vu précédemment. |
| **ALIGNMENT** | Permet la justification du texte au sein d'une cellule: LEFT, RIGHT, CENTER |
| **BACKGROUND** | Permet de définir une couleur de fond. |
| **VALIGN** | Permet de choisir l'alignement vertical dans une cellule: TOP, MIDDLE, BOTTOM (par défaut) |

#### 4.19.2.6.3 Taille de cellules

La taille des cellules se paramètre à la création du tableau. La méthode que nous avons vu jusqu'à présent est la méthode automatique, et la plus simple.

Il existe cependant une façon un peu plus complexe, mais plus paramétrable:

```
1 from reportlab.platypus import Table, TableStyle
2 ...
3 data = [['00', '01', '02', '03', '04'],['10', '11', '12', '13', '14'],
4       ['20', '21', '22', '23', '24'],['20', '21', '22', '23', '24'],
5       ['30', '31', '32', '33', '34']]
6 mon_tab = Table(data, colWidths=[50*mm,40*mm,30*mm,20*mm,10*mm], \
7               rowHeights = 10*mm)
```

#### 4.19.2.6.4 Les bordures

Les bordures sont un sujet très intéressant et important. Voici les principaux types de bordures:

| Paramètre | Description |
|---|---|
| **GRID** | Contours externes de toutes les cellules |
| **BOX** | Contours externes de la sélection |
| **INNERGRID** | Lignes internes de la sélection |
| **LINEBELOW** | Bordure inférieur de la cellule |
| **LINEABOVE** | Bordure supérieur de la cellule |
| **LINEBEFORE** | Bordure gauche de la cellule |
| **LINEAFTER** | Bordure droite de la cellule |

Pour utiliser ces paramètres, il faut également procéder de la même façon que pour les styles.

```
1 mon_tab.setStyle(TableStyle([('BOX',(1,1),(-2,-2),1*mm,colors.green)]))
```

Le premier élément est ici le paramètre à configurer.

Les éléments 2 et 3 sont les coordonnées des cellules de début et de fin de sélection.

Le 4ème élément est l'épaisseur du trait.

Le dernier élément est la couleur du trait.

#### 4.19.2.6.5 Les images dans les tableaux

L'import d'images n'a rien de bien compliqué au sein d'un tableau. Cependant, de manière simple, vous devrez choisir, au niveau d'une cellule, entre image et texte.

Vous avez cependant la possibilité d'insérer non pas du texte, mais des paragraphes avec l'image ([mon_image, paragraphe] par exemple).

```
1 from reportlab.platypus import Image
2 …
3 mon_image = Image('./test.png')
4 mon_image.drawHeight = 15*mm
5 mon_image.drawWidth = 15*mm
6 data = [['1','2',[mon_image]]
```

Nous pouvons voir ici que nous définissons notre image, puis paramétrons sa hauteur et sa largeur.

Ensuite nous l'insérons tout simplement dans notre base de données, qui servira à la création du tableau.

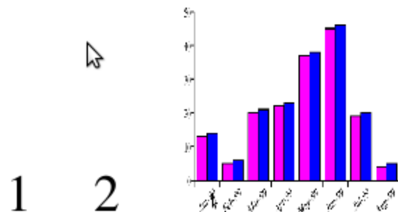


*Illustration 22: Le résultat du code*

##### 4.19.2.6.6 Insertion d'un tableau dans un canvas

Pour insérer notre tableau dans un canvas, nous allons utiliser 2 méthodes: **wrapOn** et **drawOn**.

La première méthode prend en paramètres le canvas cible et les dimensions maximums allouées au tableau, autrement dit, la taille maximum autorisée pour le tableau.

La seconde méthode, elle, prend en paramètres le canvas, et le positionnement, en X et Y, du tableau dans le canvas.

```
mon_tab.wrapOn(canvas, 210*mm, 29.7*mm)
mon_tab.drawOn(canvas,10*mm, 96*mm)
```

#### 4.19.2.7 Dessin

Exceptée la ligne, les différents éléments dessin présentés ici possèdent deux paramètres: stroke et fill.

Fill, a pour valeur par défaut 0. Il permet de préciser si l'élément dessin doit être rempli d'une couleur (1, couleur par défaut: noir) ou non (0)

Stroke a pour valeur par défaut 0. Il permet de préciser si l'élément dessin, même rempli d'un couleur doit être semi-transparent ou non.

#### 4.19.2.7.1 Ligne

Pour dessiner une ligne, il suffit d'appeler la méthode **line()**, en lui passant en paramètres les coordonnées de début,de fin, et l'épaisseur désirée, avec l'unité.

```
mon_canvas.line(10*mm, 10*mm, 100*mm, 100*mm)
```

#### 4.19.2.7.2 Cercle

Nous allons utiliser pour ce faire la méthode **circle()**.

```
mon_canvas.circle(x_centre, y_centre, rayon, stroke = 0, fill = 0)
```

Nous pouvons également tracer des ellipses:

```
mon_canvas.ellipse(x1_centre, y1_centre, x2_centre, y2_centre, fill = 0)
```

#### 4.19.2.7.3 Rectangle

Pour dessiner un rectangle, on utilise la méthode **rect()**, en passant comme arguments les coordonnées du coin inférieur gauche, sa largeur, sa hauteur, et en précisant s'il doit être rempli d'une couleur ou non.

```
mon_canvas.rect(10*mm, 10*mm, 100*mm, 100*mm, fill = 1)
```

Vous pouvez également dessiner un rectangle avec des bords arrondis, via la méthode **roundrect()**.

```
mon_canvas.roundRect(10*mm, 10*mm, 100*mm, 100*mm, 5*mm, fill = 0)
```

L'avant-dernier paramètre correspond ici au rayon des arrondis.

#### 4.19.2.7.4 Épaisseur de lignes

L'épaisseur des lignes des différents éléments dessin se paramètre via la méthode **setLineWidth()**. Cette méthode est a appeler avant de réaliser le tracé proprement dit.

```
1 mon_canvas.setLineWidth(2*mm)
2 mon_canvas.rect(1*mm,1*mm,10*mm,20*mm,fill = 0)
```

#### 4.19.2.7.5 Type de ligne

Vous pouvez choisir votre type de ligne très simplement. **Reportlab** fonctionne sur le principe de portion, concernant le type de ligne.

En clair, pour une portion donnée, vous indiquez comment le trait doit se comporter. On commence par ON, puis OFF et ainsi de suite.

Vous pouvez ainsi définir une simple alternance (ex: 50% ON, 50% OFF) ou quelque chose de plus complexe (ex: ON, OFF, ON, OFF, ON)

**Reportlab** prendra la totalité comme représentant 100% de la portion et recalculera les valeurs saisies en pourcentage.

```
1 mon_canvas.setDash(6,4) #60%ON, puis 40% OFF
2 mon_canvas.setDash(4,1,5)       #40% ON, 10% OFF, 50% ON
```

***Remarque: Pour que les exemples ci-dessus soient plus parlant, nous avons fait en sorte que la somme égale 10, mais ce n'est nullement une obligation. Vous pouvez très bien saisir setDash(5,7,9,3) par exemple.***

### 4.19.2.8 Image

L'import d'images est relativement simple également:

```
1 mon_canvas.drawImage(" mon_image.png ", x,y, width=100*mm,height=20*mm)
```

#### 4.19.2.9 graphique

***Attention: Chaque type de graphique possède son propre module, et nécessite donc son propre import.***

##### 4.19.2.9.1 Support de graphique

Les graphiques en **Reportlab** ont besoin de leur propre support, qui sera par la suite intégré au canvas.

```
1  from reportlab.graphics.shapes import Drawing
2  …
3  mon_draw = Drawing(400, 200) #support de 400 pixels de large et 200 de haut
```

***Remarque: Essayez de faire en sorte que votre support soit toujours plus grand que votre graphique, même si dans les faits, vous n'aurez pas toujours d'erreur.***

Une fois notre graphique dessiné, il ne nous restera plus qu'à l'ajouter au support via la méthode **add**:

```
1  mon_draw.add(mon_graph)
```

##### 4.19.2.9.2 Histogramme

Un histogramme se créé avec la méthode **VerticalBarChart()**. Il faut ensuite une liste contenant les données du graphique sous forme de tuple. Chaque tuple représente un jeu de données.

Une fois n'est pas coutume, place au code que nous allons commenter pour expliquer et clarifier les choses:

```
1  from reportlab.graphics.charts.barcharts import VerticalBarChart
2  from reportlab.lib.colors import Color
3
4  data = [ (13, 5, 20, 22, 37, 45, 19, 4), (14, 6, 21, 23, 38, 46, 20, 5) ]
5  bc = VerticalBarChart()
6  bc.x = 50 *mm
7  bc.y = 50*mm
8  bc.height = 125 *mm
```

```
9  bc.width = 300 *mm
10 bc.data = data
11 bc.strokeColor = colors.black
12 bc.valueAxis.valueMin = 0
13 bc.valueAxis.valueMax = 50
14 bc.valueAxis.valueStep = 10
15 bc.categoryAxis.labels.boxAnchor = 'ne'
16 bc.categoryAxis.labels.dx = 8
17 bc.categoryAxis.labels.dy = -2
18 bc.categoryAxis.labels.angle = 30
19 bc.categoryAxis.categoryNames = ['Jan-99','Feb-99','Mar-99', \
20                 'Apr-99','May-99','Jun-99','Jul-99','Aug-99']
21 violet = Color(255,0,255)
22 bleue = Color(0,0,255)
23 bc.bars[0].fillColor = violet
24 bc.bars[1].fillColor = bleue
```

Jusqu'à la ligne 5, rien de neuf. Nous notons que data possède deux tuples. Le diagramme représentera donc deux jeux de données.

Ligne 6 à 9, nous définissons l'emplacement de notre graphique sur le support, ainsi que ses dimensions.

Ligne 10, nous définissons la source de données.

Ligne 11, nous définissons la couleur du trait de l'histogramme.

Ligne 12 à 14, nous définissons notre axe y: son min, son max, et son pas.

Ligne 15, nous définissons le point d'accroche des labels. Les valeurs possibles correspondent aux points cardinaux et à leurs composantes: n, s, w, e, ne, nw, se, sw..

Ligne 16 et 17, nous paramétrons l'offset de positionnement du label sur le graphe.

Ligne 18, nous réglons l'inclinaison des labels de l'histogramme.

Les lignes 15 à 18 servent donc à bien paramétrer les labels de l'histogramme.

Ligne 19 et 20, nous " labellisons " l'axe X, en passant une liste contenant les labels. Il faut autant de labels en X qu'il y a de données dans la liste alimentant l'histogramme.

Ligne 21 et 22, nous définissons deux couleurs.

Ligne 23 et 24, nous colorons nos histogrammes aux couleurs choisies.

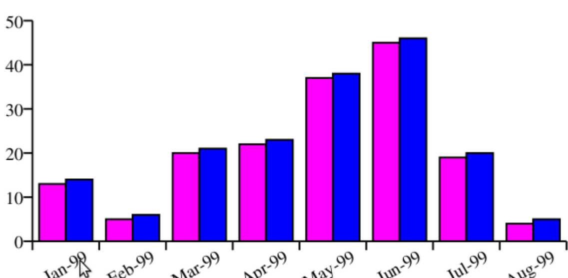



***Illustration 23: L'histogramme généré***

#### 4.19.2.9.3 Linéaire

Nous allons utiliser la méthode **HorizontalLineChart()**.

Comme précédemment, nous allons expliquer le fonctionnement au travers d'un code. Vous verrez un certain nombre de similitudes avec l'histogramme.

```
1  from reportlab.graphics.charts.linecharts import HorizontalLineChart
2  ...
3  data = [ (13, 5, 20, 22, 37, 45, 19, 4), (5, 20, 46, 38, 23, 21, 6, 14) ]
4  lc = HorizontalLineChart()
5  lc.x = 50 *mm
6  lc.y = 50 *mm
7  lc.height = 125 *mm
8  lc.width = 300 *mm
9  lc.data = data
10 lc.joinedLines = 1
11 
12 lc.categoryAxis.categoryNames = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug']
```

```
13 lc.categoryAxis.labels.boxAnchor = 'n'
14 lc.valueAxis.valueMin = 0
15 lc.valueAxis.valueMax = 60
16 lc.valueAxis.valueStep = 15
17 lc.lines[0].strokeWidth = 2 *mm
18 lc.lines[1].strokeWidth = 1.5 *mm
19 violet = Color(255,0,255)
20 bleue = Color(0,0,255)
21 lc.lines[0].strokeColor = violet
22 lc.lines[1].strokeColor = bleue
```

Ligne 3, nous retrouvons notre source de données.

Ligne 4, nous créons notre courbe.

Ligne 5 à 8, nous paramétrons sa position, ainsi que ses dimensions.

Ligne 9, nous définissons sa source de données.

Ligne 10, une nouveauté: nous précisons que nous désirons que les points de la source de données soient liés.

Ligne 12, nous définissons les labels de l'axe X.

Ligne 13, nous définissons l'ancrage des labels de l'axe X.

Ligne 14 à 16, nous paramétrons l'axe Y.

Ligne 17 et 18, nous paramétrons l'épaisseur des traits de chaque courbe.

Enfin, comme pour l'histogramme, ligne 19 à 22, nous colorons nos courbes.

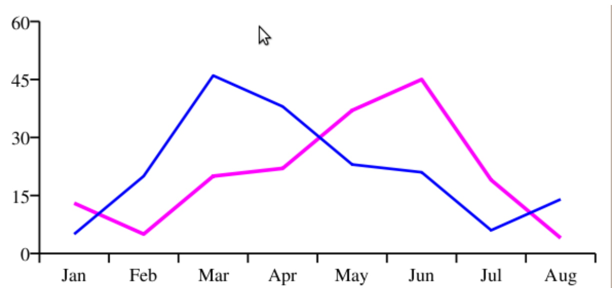



***Illustration 24: La courbe obtenue***

#### 4.19.2.9.4 Camembert

Le dernier classique en graphique est le célèbre graphique camembert. Nous le créons via la méthode **Pie()**.

Là aussi nous passerons par un code, et vous noterez des similitudes avec les graphiques précédemment passés en revue.

```
1  from reportlab.graphics.charts.piecharts import Pie
2  …
3  pc = Pie()
4  pc.x = 65
5  pc.y = 15
6  pc.width = 70 *mm
7  pc.height = 70 *mm
8  pc.data = [10,20,30,40,50,60]
9  pc.labels = ['a','b','c','d','e','f']
10 pc.slices.strokeWidth=0.5 *mm
11 pc.slices[3].popout = 10 *mm
12 pc.slices[3].strokeWidth = 2 *mm
13 pc.slices[3].strokeDashArray = [2,2]
14 pc.slices[3].labelRadius = 1.75
15 bleue = Color(0,0,255)
16 pc.slices[3].fontColor = bleue
17 pc.slices[3].fillColor = bleue
```

Ligne 3, nous créons notre graphique.

Ligne 4 à 7, nous paramétrons sa position dans le support, ainsi que ses dimensions.

Ligne 8, nous lui passons ses données. Vous aurez maintenant compris qu'il s'agit d'une simple liste qui peut être créée sur le moment ou plus en amont dans le code.

Reportlab considère l'ensemble des données communiquées comme représentant 100%, ou 360°. Il effectue ensuite un ratio sur les données passées.

Ligne 9, nous passons les labels respectifs.

Ligne 10, nous paramétrons l'épaisseur des traits du graphique.

Ligne 11 à 17, nous paramétrons plus en détails la donnée en indice 3, autrement dit en 4ème position: " d ".

Ligne 11, nous séparons la part de camembert " d " de 10 mm par rapport au centre. Cela est généralement utilisé afin d'effectuer une mise en valeur.

Ligne 12, nous modifions la largeur du trait de son contour.

Ligne 13, nous transformons le contour en trait discontinu.

Ligne 14, il s'agit d'un ratio par rapport au rayon du camembert indiquant où placer le label; dans notre cas " d ". Ici, il sera placé à une distance (du centre), de 1.5 fois le rayon du camembert. Il ne s'agit pas d'une dimension.

Ligne 15 à 17, nous colorons la part de camembert " d " de la couleur désirée.

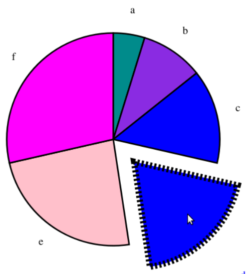



*Illustration 25: Le graphique final*

#### 4.19.2.9.5 Intégration du support dans un canvas

Pour intégrer un support, et donc le graphique qu'il abrite, dans un canvas, il faut utiliser une nouvelle méthode: le **renderPDF()**.

```
1 from reportlab.graphics import renderPDF
2 …
3 renderPDF.draw(mon_draw, mon_canvas, x*mm, y*mm)
```

Comme on peut le constater, le premier paramètre est le support de graphique, le second paramètre le **canvas** accueillant le graphique, puis la position du support graphique sur le **canvas**.

### 4.19.2.10 Changer de page

Pour fermer une page, on utilise la méthode **showPage()**.

```
1 mon_canvas.showPage()
```

***Attention: A chaque changement de page, tous les paramétrages sont remis à zéro (couleur, police, …).***

#### 4.19.2.11 Métadonnées PDF

Les métadonnées sont ces informations que l'on peut afficher dans tout lecteur PDF, et généralement nommées " propriétés du document ". Nous allons voir ici comment définir les principales.

##### 4.19.2.11.1 Auteur

Il faut utiliser la méthode **setAuthor()**:

```
1 mon_canvas.setAuthor(" toto ")
```

##### 4.19.2.11.2 Titre

Il faut utiliser la méthode **setTitle()**:

```
1 mon_canvas.setTitle(" titre ")
```

##### 4.19.2.11.3 Sujet

Il faut utiliser la méthode **setSubject()**:

```
1 mon_canvas.setSubject(" test pdf ")
```

#### 4.19.2.12 Sauvegarde

La sauvegarde d'un pdf, s'effectue à travers son canvas. Il est important de noter que tant que l'appel de la sauvegarde n'a pas été effectué, le pdf n'existe qu'en RAM. Toute coupure de courant supprimerait ainsi le contenu complet du PDF.

```
1 mon_canvas.save()
```

### 4.19.3 Exemple

```
2  #!/usr/bin/PYTHON
3  # -*-coding:utf-8 -*
4
5
6
7
8  from reportlab.pdfgen import canvas
9  from reportlab.lib.units import *
10 from reportlab.graphics.charts.barcharts import VerticalBarChart
11 from reportlab.lib.colors import Color
12 from reportlab.lib import colors
13 from reportlab.graphics.shapes import Drawing
14 from reportlab.graphics import renderPDF
15
16
17
18
19 canvas = canvas.Canvas("hello.pdf", pagesize=(210.0 * mm,297.0 * mm))
20 canvas.drawString(50*mm, 250*mm, " Test_pdf avec graphique ")
21 mon_draw = Drawing(50*mm, 200*mm)
22
23
24 data = [ (13, 5, 20, 22, 37, 45, 19, 4), (14, 6, 21, 23, 38, 46, 20, 5) ]
25
26
27 bc = VerticalBarChart()
28 bc.x = 50 *mm
29 bc.y = 50*mm
30 bc.height = 20 *mm
31 bc.width = 40 *mm
32 bc.data = data
33 bc.strokeColor = colors.black
34 bc.valueAxis.valueMin = 0
35 bc.valueAxis.valueMax = 50
36 bc.valueAxis.valueStep = 10
37 bc.categoryAxis.labels.boxAnchor = 'ne'
38 bc.categoryAxis.labels.dx = 8
39 bc.categoryAxis.labels.dy = -2
40 bc.categoryAxis.labels.angle = 30
41 bc.categoryAxis.categoryNames = ['Jan-99','Feb-99','Mar-99', \
42                                  'Apr-99','May-99','Jun-99','Jul-99','Aug-99']
43 violet = Color(255,0,255)
44 bleue = Color(0,0,255)
45 bc.bars[0].fillColor = violet
46 bc.bars[1].fillColor = bleue
47
48
49 mon_draw.add(bc)
```

```
50 renderPDF.draw(mon_draw, canvas, 50*mm, 150*mm)
51 canvas.save()
```

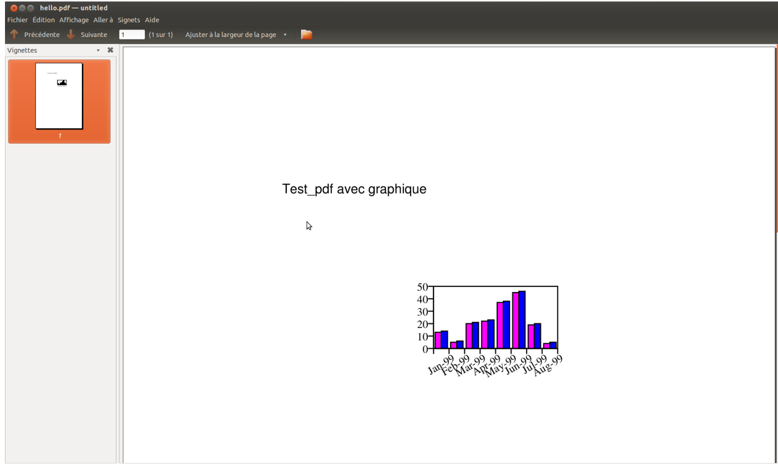



***Illustration 26: Rendu de l'exemple pdf***

# 5 Les interfaces graphiques

Wikimedia Commons,
GTK.png

A la recherche de modules dédiées **IHM**, nous trouvons de nombreux résultats.

Les deux plus connus sont **GTK** et **QT**. Nous nous concentrerons ici sur **GTK**.

Ces deux modules sont à la base des IHM Linux les plus connues, GNOME et KDE.

***Remarque: Si GTK est sous licence libre, QT dispose de diverses licences. Attention donc à vos DEV avec cette dernière à choisir la bonne version.***

## 5.1 PYGTK

**GTK**, pour Gimp ToolKit, est une bibliothèque graphique conçue à l'origine pour le célèbre logiciel GIMP. Par la suite adoptée par le projet GNOME, GTK lui est régulièrement (par erreur) associé.

D'un aspect un peu brut, diront certain, **GTK**, et **PYGTK** (son pendant **PYTHON**) n'en demeurent pas moins fonctionnels a 100% et suffisent à la plupart des besoins des développeurs.

Dans les faits, son interface simple et dénouée de toute fioriture inutile, telles que celles que l'on voit régulièrement dans les nouveaux OS (et qui consomme quantité de ressources), lui permet de remplir son office de façon optimale.

Nous verrons dans ce sous-chapitre l'essentiel de la bibliothèque. Pour de plus amples informations sur les widgets (y compris ceux qui ne serons pas vus ici) et leurs méthodes associées, je vous renvoie vers la page web dédiée: taper " **PYGTK** <NomDuWidget> " dans votre moteur de recherches préféré.

***Remarque: Il existe certain modules complémentaires permettant de rajouter des widgets à ceux de bases.***

```
1 import pygtk
```

```
2 import gtk
```

***Attention: Une fois un widget créé, il ne faudra pas oublier d'utiliser la méthode show() afin de l'afficher dans le conteneur. De même, pour le cacher, on utilisera la méthode hide(), et pour le détruire totalement, la méthode destroy().***

```
1 widget.show()
2 widget.hide()
3 widget.unparent()
4 widget.destroy()
```

***Concernant les fenêtres, on utilisera les méthodes run() et destroy(). Pour les fenêtres utilisant des boutons, on oubliera pas également de récupérer la valeur du bouton dans une variable (la fenêtre main est un cas à part et n'utilise ni run() ni destroy()).***

```
1 retour_bouton = fenetre.run()
2 fenetre.show()
3 fenetre.hide()
4 fenetre.destroy()
```

### 5.1.1 Les fenêtres

#### 5.1.1.1 Main

Il s'agit de la fenêtre principale de l'application, qu'on peut créer tout simplement

```
1 ma_fenetre = gtk.Window(type=gtk.WINDOW_TOPLEVEL)
```

Notez le paramètre type. Sa valeur par défaut est **gtk.WINDOW_TOPLEVEL**. De fait nous pouvons également créer la fenêtre principale sans préciser type.

```
1 ma_fenetre = gtk.Window()
```

***Remarque: Si type vaut gtk.WINDOW_POPUP alors cela générera une fenêtre particulière, généralement utilisée pour les***

***Splash Screen ou encore les fenêtres About, même si un constructeur spécifique existe pour ces dernières.***

On peut ensuite préciser certain paramètres tel que le titre

```
1  ma_fenetre.set_title("Ceci est mon titre")
```

On peut également préciser si la fenêtre est à maximiser ou non, ainsi que sa taille, sa position ou encore si elle peut être redimensionnable ou non.

```
1  #Taille de la fenêtre
2  ma_fenetre.set_default_size(L, H)
3  ma_fenetre.resize(L, H)
4
5  #Position
6  ma_fenetre.set_position(gtk.WIN_POS_CENTER) #au centre
7  ma_fenetre.move(X, Y)
8
9  #Maximiser une fenêtre
10 ma_fenetre.maximize()
11
12 #rendre une fenêtre redimensionnable ou non
13 ma_fenetre.set_resizable(True)
14 result = ma_fenetre.get_resizable()
15 ma_fenetre.set_size_request(650,450) # A utiliser avec un set_resizable(False)
```

Il reste ensuite les principales méthodes des fenêtres à connaître. L'ajout des widgets et/ou conteneurs se fait avec avec la méthode **add()**.

```
1  ma_fenetre.add(mon_conteneur)
```

Il ne reste ensuite qu'à préciser à l'interpréteur l'action à accomplir lorsque l'on clique sur la croix de fermeture de la fenêtre, puis afficher la fenêtre à l'utilisateur, et enfin lancer le programme complet

```
1  ma_fenetre.connect("destroy", gtk.main_quit)
2  ma_fenetre.show() #show_all() existe également
3  gtk.main() #lance le programme complet
```

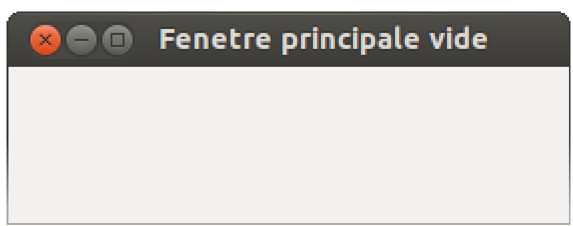



***Illustration 27: Une fenêtre principale vide***

#### 5.1.1.2 About

**PYGTK** vous propose comme n'importe quel autre interface graphique des fenêtres " A propos de ". Cependant, ici tout est pré-configuré.

De fait, vous avez juste à passer les paramètres pour obtenir une fenêtre " A propos de " de qualité professionnelle.

```
1  ma_fenetre = gtk.AboutDialog()
```

Il ne reste alors plus qu'à configurer chaque paramètre.

```
1  ma_fenetre.set_name(" Nom de mon programme ")
2  ma_fenetre.set_version(" 0.0.1 ")
3  ma_fenetre.set_copyright(" Copyleft ")
4  ma_fenetre.set_comments(" mon commentaire ")
5  ma_fenetre.set_license(" Licence GPL ")
6  ma_fenetre.set_website(" http://diablotronic.bzh.bz ")
7  ma_fenetre.set_website_label(" texte de mon lien ")
8  ma_fenetre.set_authors(liste_programmeurs)
9  ma_fenetre.set_documenters(Liste_auteurs_docs)
10 ma_fenetre.set_translator_credits(" listes des traucteurs ")
11 ma_fenetre.set_logo(logo) #logo est un pixbuf, voir ci-dessous
12 logo = PYGTK.gdk.pixbuf_new_from_file("mon_logo.png")
```

***Illustration 28: Une fenêtre A propos de (http://developer.gnome.org)***

#### 5.1.1.3 Info / erreur / question / Attention

Pour tous ces types de fenêtres, il n'y a qu'un seul constructeur, mais avec un paramétrage différent: **MessageDialog()**.

Ce constructeur prend 5 paramètres

```
1 ma_fenetre = gtk.MessageDialog(parent, flags, type, buttons, message_format)
```

Le premier permet de préciser si la fenêtre de dialogue dépend d'une autre fenêtre ou non. Sa valeur par défaut est None. En général, on précisera la fenêtre principale de notre programme.

Le second paramètre permet de configurer le comportement du parent vis-à-vis de la fenêtre de dialogue.

| Constante | Description |
| --- | --- |
| **gtk.DIALOG_MODAL** | Tant que la fenêtre de dialogue est ouverte, on n'a pas accès à la fenêtre parent |
| **gtk.DIALOG_DESTROY_WITH_PARENT** | La fenêtre de dialogue sera détruite en même temps que le parent |
| **gtk.DIALOG_MODAL\|<br>gtk.DIALOG_DESTROY_WITH_PARENT** | Combinaison des deux comportements précédemment décrits |
| **0** | Aucun (valeur par défaut) |

Le paramètre suivant, type, permet de paramétrer le type de la fenêtre de dialogue affichée à l'utilisateur.

| Constante |
| --- |
| **gtk.MESSAGE_INFO<br>(valeur par défaut)** |
| **gtk.MESSAGE_WARNING** |
| **gtk.MESSAGE_QUESTION** |
| **gtk.MESSAGE_ERROR** |

Viennent ensuite les boutons à afficher. Ici, vous ne créez pas vos boutons, ils sont prédéfinis et vous choisissez juste ceux que vous voulez parmi une liste de 6 configurations.

| Constante | Description |
| --- | --- |
| **gtk.BUTTONS_NONE** | Aucun bouton (peut alors servir de substitut à un splash screen) (valeur par défaut) |
| **gtk.BUTTONS_OK** | Affiche juste un bouton " Valider " |
| **gtk.BUTTONS_CLOSE** | Affiche juste un bouton " Fermer " |
| **gtk.BUTTONS_CANCEL** | Afficher un bouton " Annuler " |
| **gtk.BUTTONS_YES_NO** | Afficher un couple de bouton " Oui " et " Non " |
| **gtk.BUTTONS_OK_CANCEL** | Affiche un bouton " Valider " et un bouton " Annuler " |

Chacun de ces boutons est associé à des constantes **GTK** de type **RESPONSE**. Si cette constante vaut True, alors le bouton a été cliqué.

| Constante |
| --- |
| gtk.RESPONSE_YES |
| gtk.RESPONSE_NO |
| gtk.RESPONSE_CANCEL |
| gtk.RESPONSE_CLOSE |
| gtk.RESPONSE_OK |
| gtk.RESPONSE_ACCEPT |
| gtk.RESPONSE_APPLY |

***Remarque: Vous pouvez avoir accès aux constantes en recherchant sur Internet " PYGTK PYGTK-constants ".***

Le dernier paramètre, lui, est le texte à afficher dans la fenêtre de dialogue.

Il existe deux méthodes de gestion de la fenêtre: **run()** et **destroy()**, permettant respectivement de lancer et de détruire la fenêtre.

```
1 retour = ma_fenetre.run()
2 ma_fenetre.destroy()
```

Comme on le voit sur l'exemple, on récupère quelque chose lorsqu'on lance la fenêtre. Il s'agit du bouton cliqué par l'utilisateur dans la fenêtre de dialogue. Il suffit alors juste de comparer la variable retour avec une des constantes **GTK**.

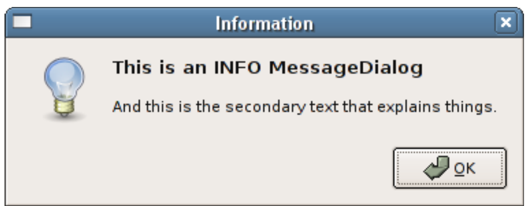



***Illustration 29: Un exemple de fenêtre de dialogue (http://developer.gnome.org)***

#### 5.1.1.4 Print

La fenêtre d'impression classique, telle que nous la connaissons, existe toute faite en **PYTHON**.

On peut de plus la configurer, comme bon nous semble, via une constante **GTK**.

| Constante | Description |
| --- | --- |
| **gtk.PRINT_OPERATION_ACTION_PRINT_DIALOG** | Fenêtre classique |
| **gtk.PRINT_OPERATION_ACTION_PRINT** | Impression directe |
| **gtk.PRINT_OPERATION_ACTION_PREVIEW** | Aperçu |
| **gtk.PRINT_OPERATION_ACTION_EXPORT** | Exporter vers un fichier (nécessite un export-filename) |

Il faut commencer par créer un conteneur d'impression, puis lancer l'affichage.

```
1 import gtk
2 fenetre = gtk.PrintOperation()
3 result = fenetre.run(gtk.PRINT_OPERATION_ACTION_PRINT_DIALOG)
```

Ligne 3, on peut voir qu'on récupère le résultat de l’opération pour savoir quoi faire.

Ce résultat peut prendre plusieurs valeurs. Il suffit de comparer la valeur de result avec les constantes **GTK** suivantes:

| Constante | Description |
|---|---|
| **gtk.PRINT_OPERATION_RESULT_ERROR** | Erreur |
| **gtk.PRINT_OPERATION_RESULT_APPLY** | Bouton imprimer sélectionné |
| **gtk.PRINT_OPERATION_RESULT_CANCEL** | Bouton annuler sélectionné |
| **gtk.PRINT_OPERATION_RESULT_IN_PROGRESS** | Impression non terminée |

```
if result = gtk.PRINT_OPERATION_RESULT_APPLY
```

Le travail ne s'arrête cependant pas là. En effet, nous n'avons fait que la moitié des choses. Une fois que l'opérateur a paramétré son impression, il reste à imprimer. Cela se fait de la manière suivante:

```
import gtkunixprint
filename = 'mon_fichier.txt'
ma_print = fenetre.get_selected_printer()
params = fenetre.get_settings()
config = fenetre.get_page_setup()
printjob = gtkunixprint.PrintJob('Print %s' % filename, ma_print, params, config)
printjob.set_source_file(filename)
printjob.send(print_cb)
pud.destroy()
```

Ce code ne fera pas l'objet de commentaires. Précisons tout de même qu'il ne fonctionnera que sous Linux/UNIX. Sous windows, basez-vous sur:

```
import win32api
win32api.ShellExecute (0, "print", filename, None, ".", 0)
```

Afin de satisfaire votre soif de connaissances concernant ces quelques lignes de code, je vous renvoie vers Internet.

#### 5.1.1.5 File

Au niveau des fenêtres de gestion de fichiers, vous avez plusieurs choix: Sélection de fichier(s) à ouvrir, sélection de fichiers pour la sauvegarde, sélection de dossiers, création de dossiers.

Pour toutes ces fenêtres, un constructeur unique.

```
1  ma_fenetre = gtk.FileChooserDialog(title=None, parent=None, \
2                    action=gtk.FILE_CHOOSER_ACTION_OPEN,\
3                    buttons=None, backend=None)
```

Comme on peut le constater, il faut jusqu'à 5 paramètres.

Title sera le titre que portera la fenêtre lorsqu'elle s'affichera à l'utilisateur.

Parent, backend sont pour une utilisation sur mesure et seront généralement à None. Pour plus d'informations sur ce paramètre, je vous renvoie vers la documentation officielle.

Action correspond au type de fenêtre souhaitée.

| Constante | Description |
|---|---|
| **gtk.FILE_CHOOSER_ACTION_OPEN** | Choisir un ou plusieurs fichier(s) à ouvrir |
| **gtk.FILE_CHOOSER_ACTION_SAVE** | Choisir un fichier pour l'enregistrement |
| **gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER** | Choisir un dossier ou effectuer une action |
| **gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER** | Permet de créer un dossier dans un dossier donné |

Enfin buttons est un tuple qui contient la liste des boutons désirés avec les signaux associés: (bouton1, signal1, bouton2, signal2, …). les boutons et les signaux sont identiques à ceux vus précédemment pour les fenêtres d'informations.

Du côté des méthodes, nous allons voir les principales.

```
ma_fenetre.set_select_multiple(True)
ma_fenetre.get_select_multiple()
```

Permet de déterminer si on peut sélectionner plusieurs fichiers (True) ou non.

```
ma_fenetre.get_filename()
```

Permet de récupérer le nom du fichier sélectionné (dans le cas d'un seul fichier possible). A None si aucune sélection.

```
ma_fenetre.get_filenames()
```

Permet de récupérer les noms des fichiers sélectionnés (dans le cas d'une sélection multiple). A None si aucune sélection.

```
ma_fenetre.set_filename(" monfichier.txt ")
```

Permet de définir le nom du fichier par défaut.

```
ma_fenetre.set_do_overwrite_confirmation(True)
```

Permet de demander une confirmation avant tout enregistrement.

```
ma_fenetre.get_current_folder()
```

Permet d'obtenir le chemin complet du dossier sélectionné, ou dans lequel se trouve le fichier sélectionné.

Nous avons également la possibilité de créer des filtres de fichiers pour la fenêtre de sélection de fichiers.

```
mon_filtre = gtk.FileFilter()
mon_filtre.set_name("fichiers texte")
mon_filtre.add_pattern("*.txt")
ma_fenetre.add_filter(filtre)
```

Enfin pour finir, on lance la fenêtre.

```
retour = ma_fenetre.run()
```

Ensuite, il faut analyser la variable retour et exécuter du code en fonction du choix de l'utilisateur.

```
1 if retour == gtk.RESPONSE_OK:
2     dossier = ma_fenetre.get_current_folde()
3     fichier = ma_fenetre.get_filename()
4     chemin = dossier + fichier
```

A la fin, on n'oublie pas de détruire la fenêtre.

```
1 ma_fenetre.destroy()
```

### 5.1.2 Les conteneurs

En **PYGTK**, il existe trois grands types de conteneurs. Les **Hbox**, divise l'espace de manière horizontale. Les **Vbox** divisent l'espace de manière verticale. Enfin les tableaux divisent l'espace en plusieurs zones, à la façon d'un damier, ou d'un tableau d'où leur nom.

Chacune des zones créées peut ainsi être utilisées de manière indépendante pour la répartition des différents widgets.

#### 5.1.2.1 Hbox / Vbox

La création d'une **Hbox** prend deux paramètres. Le premier, homogeneous, permet de faire en sorte que toutes les zones aient la même taille (True) ou non (False). Le second paramètre, spacing, permet de préciser le nombre de pixels que l'on désire avoir en séparation, entre chaque zone.

La création d'une **Vbox** prend les mêmes paramètres

```
1 ma_hbox = gtk.HBox(homogeneous=True, spacing=2)
2 ma_vbox = gtk.VBox(homogeneous=True, spacing=2)
```

Il ne reste ensuite plus qu'à charger nos widgets dans nos conteneurs. Pour cela nous utilisons la méthode **pack_start** (pour charger

le conteneur depuis la 1ère zone) ou **pack_end** (pour le charger depuis la dernière zone).

Ces méthodes prennent 4 arguments. Le premier est le nom du widget à placer dans la zone. Le second, expand, va dépendre de la valeur de homogeneous. Si ce dernier est à True, alors expand ne sera pas pris en compte. Par contre, dans le cas contraire, l'ensemble des widgets dont expand sera à True se partagera l'espace disponible de la fenêtre. Le troisième paramètre, fill, permet d'indiquer si le widget va occuper tout l'espace qui lui est alloué ou non. Le dernier, padding, permet de définir un espace (en pixel) entre le widget et la bordure de l'espace qui lui est alloué.

```
1 ma_hbox.pack_start(mon_widget, expand=True, fill=True, padding=2)
```

***Remarque: Si vous divisez un espace en 3 zones, le pack_end mettra le 1er widget dans la zone 1, le 2nd widget dans la zone 2, et le troisième dans la zone 3. Seul un widget peut exister par zone. Mais rien ne vous empêche de mettre un conteneur dans un autre afin de pouvoir mettre plusieurs widgets dans une zone initiale.***

***Attention: Les méthodes pack de ces conteneurs, ne permettent pas de commencer à les remplir par une case au choix.***

#### 5.1.2.2 Boîte à boutons

Les boîtes à boutons existent en mode vertical et horizontal. Elles ont le même rôle que les **Hbox** et **Vbox**, mais sont dédiées à la mise en place des boutons.

```
1 ma_hbuttonbox = gtk.HButtonBox()
2 ma_vbuttonbox = gtk.VButtonBox()
```

Une fois la boîte créée, il suffit juste de préciser le type de mise en page grâce à une des constantes prédéfinies.

| Constante | Description |
|---|---|
| **gtk.BUTTONBOX_SPREAD** | Permet de répartir les boutons de manière équitable dans l'espace dédié (la plus usité) |
| **gtk.BUTTONBOX_EDGE** | Permet de placer les boutons à l'extrême de l'espace |
| **gtk.BUTTONBOX_START** | Pour aligner les boutons sur la gauche ou le haut |
| **gtk.BUTTONBOX_END** | Pour aligner les boutons sur le bas ou la droite |

```
1 ma_hbuttonbox.set_layout(gtk.BUTTONBOX_SPREAD)
```

Pour ajouter des boutons, on utilisera là aussi les méthodes **pack_start()** et **pack_end()**, tel que vu pour les **Hbox** et **Vbox**.

#### 5.1.2.3 Tableau

Le constructeur des tableaux prend trois paramètres: le nombre de lignes (rows), le nombre de colonnes (columns), et homogeneous que nous venons de voir avec le **Hbox**.

```
1 mon_tableau = gtk.Table(rows=1, columns=1, homogeneous=False)
```

***Attention: La numérotation des colonnes et des lignes, pour la suite, commence à 0, en haut à gauche, tels des axes inversés. Le (0,0) correspond alors au coin supérieur gauche du tableau.***

L'avantage du conteneur tableau est que l'on peut attribuer un widget, non pas à un des espaces créés, comme c'est le cas avec **Hbox** ou **Vbox**, mais sur plusieurs espaces.

De plus, on peut placer chaque widget à n'importe quelle place, et dans n'importe quel ordre.

Pour ajouter un widget à notre tableau, on utilise la méthode **attach()**.

```
1 mon_tableau.attach(mon_widget, left_attach, right_attach, top_attach, \
2           bottom_attach, xoptions=gtk.EXPAND|gtk.FILL,\
```

```
3            yoptions=gtk.EXPAND|gtk.FILL, xpadding=0, ypadding=0)
```

Voyons voir maintenant les différents paramètres de cette méthode.

Avant tout, rappelons que nous commençons à compter à partir de 0, en partant du coin supérieur gauche. Il faut aussi savoir qu'on ne parle pas dans ce cas de ligne 0 ou de ligne 1, mais de ligne contenue entre la borne 0 et la borne 1 ou de ligne contenue entre la borne 1 et la borne 2.

•left_attach: N° de la borne à gauche du widget
•right_attach: N° de la borne à droite du widget
•top_attach: N° de la borne au dessus du widget
•bottom_attach: N° de la borne en dessous du widget
•xpadding, ypadding: espace en pixel entre chaque zone, en x et en y
•xoptions, yoptions: permet de définir le comportement du widget en x et en y. Pour cela, trois variables (tableau ci-après) sont disponibles et associables indifféremment, via un pipe " | ", selon le résultat souhaité

| Constante | Description |
|---|---|
| **gtk.EXPAND** | Le widget prendra le maximum de place disponible dans le tableau |
| **gtk.SHRINK** | Le widget prendra le minimum de place possible |
| **gtk.FILL** | Le widget prendra toute la place qui lui est allouée |

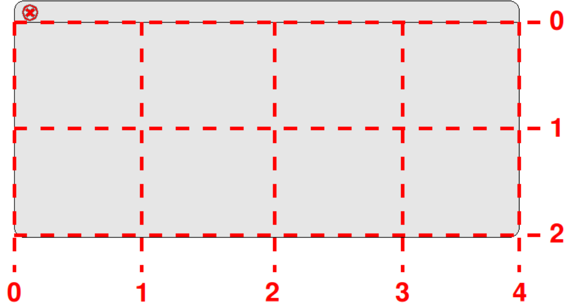



***Illustration 30: Le découpage d'un conteneur Tableau***

***Remarque: Il est possible de superposer plusieurs widgets sur la même zone, puis de choisir lequel on désire afficher.***

#### 5.1.2.4 Fixed

Le conteneur **fixed** permet de positionner les éléments avec des coordonnées. Très pratique, cela signifie néanmoins que votre application doit avoir une taille fixe pour tourner, si vous ne désirez pas avoir de mauvaises surprises en cas de redimensionnement.

```
mon_conteneur = gtk.Fixed()
```

Une fois créées, on retiendra 2 principales méthodes liées: l'insertion, et le déplacement des widgets.

```
mon_conteneur.put(widget, X, Y)
mon_conteneur.move(widget, X, Y)
```

#### 5.1.2.5 ScrolledWindow

Les **scrolledwindow** sont des fenêtres possédant des barres de défilement.

Ce type de fenêtre ne peut accepter qu'un seul enfant, qui sera la plupart du temps un conteneur (voir partie suivante)

##### 5.1.2.5.1 Création

La création d'une **scrolledwindow** est triviale:

```
ma_scroll = gtk.ScrolledWindow()
```

##### 5.1.2.5.2 Paramétrage

Parmi les paramétrages, nous retrouvons la possibilité d'afficher ou non les barres de défilement:

```
1 ma_scrol.set_policy(HorizontalScrol, VerticalScrol)
```

ici, HorizontalScrol et VerticalScrol, ne peuvent réellement prendre que deux valeurs: **gtk.POLICY_ALWAYS** (affiche toujours le scroll), ou **gtk.POLICY_AUTOMATIC** (affiche le scroll uniquement quand nécessaire).

##### 5.1.2.5.3 Ajout de l'enfant

Comme dit plus haut, on ne peut ajouter qu'un enfant. Cela s'effectue de la manière suivante:

```
1 ma_scrol.add_with_viewport(mon_conteneur)
```

L'ancrage de l'enfant a lieu sur le coin supérieur gauche de la **ScrolledWindow**.

### 5.1.3 Les widgets

#### 5.1.3.1 Boutons à cliquer

Les boutons permettent à l'utilisateur de déclencher des actions. A la création d'un bouton, on a la possibilité de lui passer 3 paramètres

| paramètre | Description |
|---|---|
| **label** | Texte affiché dans le bouton. Par défaut à None |
| **stock** | Icône à afficher en plus du texte, par défaut à None. Les icônes disponibles seront vues un peu plus loin en 5.1.3.20 |
| **use_underline** | Chaque lettre précédée d'un " _ " sera soulignée dans le texte du bouton. Par défaut à True |

La méthode pour créer un bouton est **Button()**

```
1 mon_bouton = gtk.Button('_Bouton', stock=None, use_underline=True)
```

Dans l'exemple ci-dessus, nous utilisons tous les paramètres, mais avec leur valeur par défaut. De fait, nous pouvons nous abstenir de les préciser pour arriver au même résultat.

```
1 mon_bouton = gtk.Button('_Bouton')
```

Une fois créé, il faut préciser l'action associée au bouton. Pour cela, on utilise la méthode **connect()** et l'un des trois signaux suivants

| Signal | Description |
|---|---|
| **pressed** | Quand le bouton est pressé |
| **released** | Quand le bouton est relâché |
| **clicked** | Quand le bouton est cliqué (pressé, puis relâché) |

Bien que pour certain, la subtilité ne soit pas forcément évidente, dans les faits, cela pourra parfois vous servir. Cependant, la plupart du temps on se contentera de **clicked**.

```
1 mon_bouton.connect(" clicked ", gtk.main_quit) #bouton pour sortir du software
2 mon_bouton.connect(" clicked ", ma_fonction)
```

***Remarque: Lors de l'utilisation de la méthode connect(), l'appel de la fonction est particulier. En effet, les arguments doivent être passés à la suite du nom de la fonction. De plus, le premier argument d'une fonction appelée par un connect() sera toujours widget (le widget appelant) (sauf si on est dans une classe, le 1<sup>er</sup> argument sera self, et le second widget). Résumons donc par l'exemple***

```
1 def ma_fonction(widget, arg1, arg2): #ou def ma_fonction(self,widget,arg1,arg2):
2       …
3 …
4 mon_bouton.connect(" clicked ", ma_fonction, arg1, arg2)
```

### 5.1.3.2 Boutons à commuter

Les boutons à commuter ne peuvent prendre que deux états (enfoncé ou non).

```
1 mon_bouton_commut = gtk.ToggleButton(label= "Bouton 2 ")
```

On peut définir l'état par défaut du bouton.

```
1 mon_bouton_commut.set_active(True)
```

On peut tester l'etat du bouton.

```
1 ...
2 if mon_bouton_commut.get_active():
3     ...
```

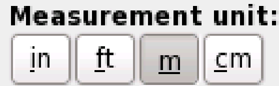



***Illustration 31: Les bouton a commutation (http://developer.gnome.org)***

### 5.1.3.3 Boutons à cocher

Les **checkbox**, ou cases à cocher permettent à l'utilisateur d'activer ou non certain labels. Il peut s'agir aussi bien d'un questionnaire, que d'une configuration.

Le constructeur est très simple. Le paramètre use_underline est à **True** par défaut, nous ne le préciserons donc pas.

```
1 ma_case = gtk.CheckButton(" Texte de la case')
```

Il existe derrière deux méthodes permettant de changer de manière logicielle l'état de la case, et également de le récupérer. On peut également déclencher certaines actions lorsqu'on active/désactive la case avec la méthode **connect**

```
ma_case.set_active(True)
ma_case.get_active() #vaut True si la case est cochee
ma_case.connect(" clicked ", ma_fonction)
```

***Illustration 32: Un bouton a cocher (http://developer.gnome.org)***

#### 5.1.3.4 Boutons radios

Les **cases radio** sont comparables aux **cases à cocher** à ceci près, qu'il est possible de créer des groupes et que chaque groupe ne peut posséder plus d'une case active.

Une **case radio** est construite sur le même principe qu'une case à cocher, mais prend un paramètre supplémentaire, passé en 1ère position: son groupe. En fait de groupe, nous passons le 1er élément du groupe aux éléments suivants

```
mon_radiob1 = gtk.RadioButton(None, "1er choix")
mon_radiob2 = gtk.RadioButton(mon_radiob1, "2nd choix")
mon_radiob3 = gtk.RadioButton(mon_radiob1, "3ème choix")
```

Tout comme les cases à cocher, on peut en cas de click, faire appel à une fonction

```
mon_radiob1.connect(" clicked ", ma_fonction )
```

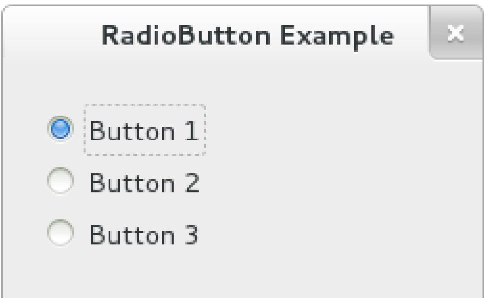



***Illustration 33: Quelques boutons radios (http://developer.gnome.org)***

#### 5.1.3.5 Combobox

Les **combobox**, ou listes déroulantes, permettent à l'utilisateur d'effectuer un choix parmi une liste prédéfinie.

```
1  mon_combo = gtk.combo_box_new_text()
```

Une fois la **combo** créée, on peut ajouter ou supprimer simplement des éléments.

```
1  mon_combo.append_text(" second élément ")
2  mon_combo.insert_text(0, " premier element ")
3  mon_combo.remove(1) # retire " second element "
```

Comme on peut le voir ligne 3, on utilise non pas le texte mais un index pour se repérer dans une **combobox**.

Ainsi pour savoir quel élément est actif, il est préférable d'avoir une liste ou un dictionnaire associé. Grâce aux méthodes associées on pourra alors facilement faire un lien.

```
1  mon_combo.get_active()
2  mon_combo.set_active(0) #prend l'index comme parametre
```

***Remarque: L'index commence à 0. De plus, l'index -1 signifie " affichage vierge "; en clair aucune sélection.***

Enfin, il est possible de savoir quand l'élément choisi change.

```
mon_combo.connect('changed', ma_fonction)
```

#### 5.1.3.6 Les onglets

Les onglets, ou **Notebook**, sont très pratiques en IHM.

Lorsqu'on a beaucoup d'informations à afficher à l'utilisateur, plutôt que de faire de multiples fenêtres, nous avons juste à créer un **Notebook**.

Ce dernier se comporte un peu comme un de ces annuaires, avec des signets qui dépassent en extrémité de page: vous sélectionnez votre signet, et vous arrivez sur une page dédiée.

Le principe utilisé ici sera le même.

```
mon_nb = gtk.Notebook()
```

Jusque là pas de soucis je pense. Alors passons aux choses sérieuses.

Il faut tout d'abord choisir où positionner nos onglets. 4 choix possibles:

```
mon_nb.set_tab_pos(POS_LEFT)          #a gauche
mon_nb.set_tab_pos(POS_RIGHT)         #a droite
mon_nb.set_tab_pos(POS_TOP) #en haut (par défaut)
mon_nb.set_tab_pos(POS_BOTTOM)        #en bas
```

Il faut ensuite ajouter des onglets à notre **notebook**. Par défaut, il est créé sans aucun signet.

```
mon_nb.append_page(widget, signet_texte)
mon_nb.prepend_page(widget, signet_texte)
mon_nb.insert_page(widget, signet_texte, pos)
mon_nb.remove_page(num_onglet)
```

La ligne 1 effectue un ajout d'onglet à la suite de ceux existant.

La ligne 2 effectue l'ajout d'onglet avant ceux déjà existant.

Dans les deux cas, le 1er paramètre correspond au conteneur ou widget inséré dans l'onglet, et signet_texte au label de l'onglet. Il est ainsi impossible de créer un signet totalement vide. Il faut au moins un conteneur dans l'onglet.

Ligne 3, nous créons également un onglet, mais en précisant cette fois sa position parmi ceux déjà existant. Le 1er onglet est le 0.

Ligne 4, nous enlevons un onglet grâce à son numéro.

Outre cela, nous pouvons également gérer l'onglet actif aisément:

```
1 onglet_actif = mon_nb.get_current_page()
2 mon_nb.set_current_page(num_onglet)
3 mon_nb.next_page()
4 mon_nb.prev_page()
5 mon_nb.set_scrollable(True)
```

Ligne 1, nous récupérons l'onglet actif (le 1er est 0 pour rappel).

Ligne 2, nous définissons quel onglet doit être affiché à l'utilisateur.

Ligne 3 et 4, nous nous déplaçons au sein des onglets du Notebook.

Ligne 5 enfin, nous précisons qu'en cas d'un nombre d'onglets trop important pour qu'ils soient tous affichés, le **Notebook** doit afficher une **scrollbar**.

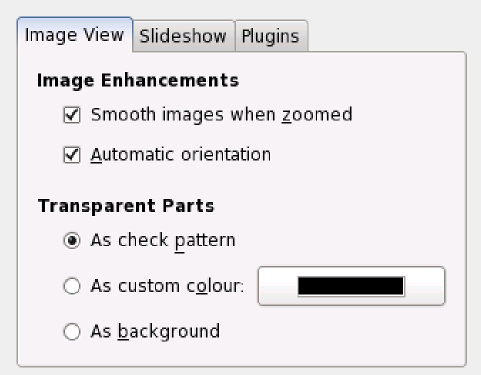



***Illustration 34: Exemple de notebook (http://developer.gnome.org)***

#### 5.1.3.7 Les tableaux de données

Les tableaux de données sont une composante souvent utile en IHM.

Quoiqu'en premier lieu un peu rebutant, vous verrez qu'en créer en **PYGTK** n'est pas si compliqué.

Un tableau de données en PYGTK repose sur 3 éléments: Le **TreeStore**, le **TreeView** et le **TreeViewColumn**.

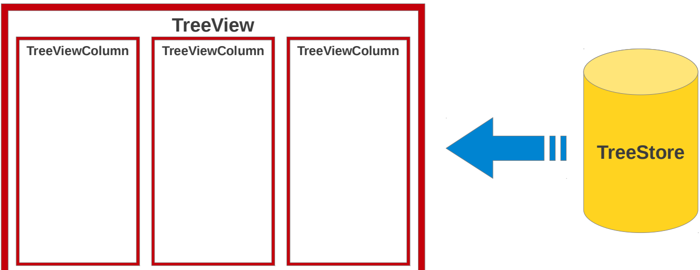



***Illustration 35: Principe d'un tableau de données GTK***

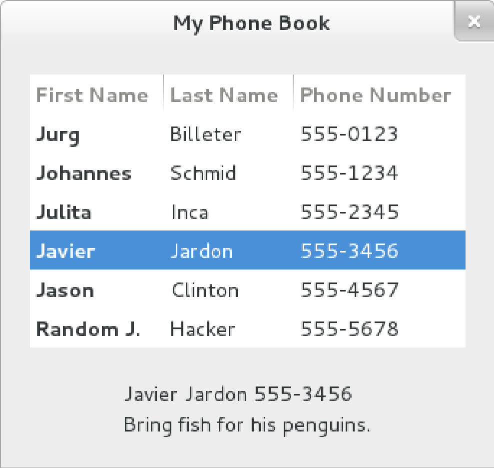



***Illustration 36: Exemple de rendu final (http://developer.gnome.org)***

#### 5.1.3.7.1 Les TreeStores

Il s'agit de l'objet contenant les données qui seront affichées dans le tableau. Il est comparable à une BDD spéciale pour le tableau. Il est comparable à une liste contenant elle-même d'autres listes, de façon à former une matrice.

***Remarque: Toutes les lignes d'un TreeStore doivent avoir le même nombre d'élément.***

On le créé en lui passant en paramètres les types des différents éléments qu'il contiendra.

```
1 mon_ts = gtk.TreeSore(int, float, str, str)
```

Ici nous venons de créer un **TreeStore** dont les lignes sont composées de 4 éléments: le premier élément est un entier, le second un float, et les deux derniers des chaînes de caractères.

***Remarque: Les autres possibilités à connaître sont: long, gboolean.***

Nous pouvons interagir avec les données d'un **TreeStore**:

```
1 mon_ts.append(None, [entier1, float1, string1, True])
```

Ici nous ajoutons des données au **TreeStore**. Le premier paramètre est le parent (en général None), et le second une liste contenant les éléments de la nouvelle ligne à ajouter.

Pour les actions suivantes, cela va se compliquer légèrement. En effet, il faudra passer par un **TreeIter**. Il s'agit simplement d'un tuple désignant l'accès à une donnée du **TreeStore**, ou à une de ses lignes.

Comparons, temporairement, nos **TreeStores** à des dossiers. Les dossiers peuvent à la fois contenir d'autres dossiers, et des données.

Sur le même principe, les **TreeStores** peuvent contenir d'autres **TreeStores**. Pour accéder à une ligne, nous utilisons ce qu'on appelle un **PATH** ou chemin.

La référence ici est le 0, une fois n'est pas coutume. Chaque entier correspond au nœud à rechercher dans le " dossier ". La chose à bien retenir est qu'on lit les éléments dans l'ordre où ils ont été insérés.

Le **PATH** pour accéder au fichier cible est alors (2,1). Si nous interprétons ce chemin, une fois dans notre **TreeStore** (dossier père dans l'exemple), nous devons rentrer dans le 3ème élément (le 2nd sous-dossier), afin d'atteindre notre cible qui est le 2nd élément du sous-dossier.

Dans un **TreeIter** basique, où nous n'avons pas de " sous-dossier ", le PATH est simplement le numéro de l'élément cible. Ainsi, le fichier placé directement dans le dossier maître de l'exemple est (1,).

Prenons un exemple visuel:

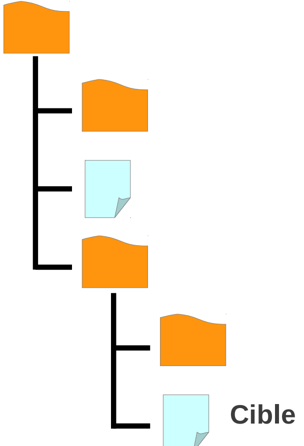



***Illustration 37: Principe du PATH d'un TreeStore***

Une fois notre **PATH** identifié, nous récupérons notre iter ainsi:

```
1 mon_treeiter = mon_ts.get_iter(PATH)
```

A partir de là, nous pouvons continuer sur nos interactions sur les **TreeStores**.

```
1 mon_ts.remove(mon_treeiter)
2 mon_ts.set_value(mon_treeiter, ma_colomne, valeur)
```

Ligne 1, nous enlevons une ligne de données.

Ligne 2, nous modifions la valeur d'un champ, en passant en paramètre le **TreeIter** permettant d'accéder à la bonne ligne, puis le numéro de la colonne impactée, et enfin la nouvelle valeur.

D'autre possibilités sont effectives sur les **TreeStores**. Si vous êtes amené à les considérer, je vous invite à lire plus amplement la documentation en ligne.

#### 5.1.3.7.2 Les TreeViews

Les **TreeView** servent d'interface entre le **TreeStore** (la base de données) et les **TreeViewColumns** (que nous verrons juste après), servant à afficher les dites données.

On créé un **TreeView** très simplement en l'associant à un **TreeStore**:

```
1 mon_tv = gtk.TreeView(mon_TreeStore)
```

Les **TreeView** possèdent beaucoup de méthodes et de propriétés. Voici les principales qui vous serviront:

```
1 mon_tv.set_rules_hint(True)
2 mon_tv.append_column(ma_colonne)
3 mon_tv.remove_column(ma_colonne)
4 mon_tv.insert_column(ma_colonne, position)
5 ma_colonne = mon_tv.get_column(n)
6 mon_treeiter = mon_tv.get_active_iter()
7 mon_tv.set_active_iter(mon_treeiter)
```

Ligne 1, nous indiquons que nous désirons un affichage au format tableau classique.

Ligne 2, nous ajoutons une colonne (point suivant) à notre **TreeView**.

Ligne 3, nous retirons notre colonne du **TreeView**.

Ligne 4, Nous insérons notre colonne à une certaine posisiton dans notre **TreeView** (on commence à 0).

Ligne 5, nous récupérons le **TreeViewColumn** placé à la position n dans le **TreeView**.

Ligne 6, nous récupérons le **TreeIter** correspondant à la ligne active dans le tableau.

Ligne 7, nous surlignons la ligne désignée par le **TreeIter**.

Voilà pour faire simple. Mais avant de passer aux **TreeViewColumns**, nous allons apporter quelques précisions sur la sélection dans un **TreeView**.

Cette dernière passe par un **TreeSelection**. Un **TreeSelection** est un objet créé en même temps que le **Treeview** et qui le renseigne sur les diverses actions liées à la sélection.

Cependant, en utilisant le **TreeView** pour nous interfacer avec, tel que vu lignes 6 et 7 précédemment, nous perdons nombre de possibilités. Nous allons combler ce manque maintenant.

Pour récupérer le **TreeSelection** associé au **TreeView**, on utilise:

```
1 mon_tselect = mon_tv.get_selection()
```

Le type de sélection se fait via **set_mode()**:

```
1 mon_tselect.set_mode(mode)
2 mode_actif = mon_tvselect.get_mode()        # Renvoi le mode actif
```

Les modes disponibles sont les suivants:

| Mode | Description |
|---|---|
| **gtk.SELECTION_NONE** | Aucune sélection autorisée |
| **gtk.SELECTION_SINGLE** | Sélection possible d'une seule ligne par clic |
| **gtl.SELECTION_BROWSE** | Sélection d'une seule ligne par survol du pointeur |
| **gtk.SELECTION_MULTIPLE** | Sélections multiples autorisées |

Si vous souhaitez cacher les entêtes du tableau, il faudra procéder ainsi:

```
1 mon_tv.set_headers_visible(False)
```

De manière logicielle il est possible de (dé)sélectionner tout ou partie du tableau.

```
1  mon_tselect.select_path(PATH)
2  mon_tselect.unselect_path(PATH)
3  mon_tselect.select_iter(mon_TreeIter)
4  mon_tselect.unselect_iter(mon_TreeIter)
5  mon_tselect.select_all()    #Necessite d'etre en mode selection multiple
6  mon_tselect.unselect_all() #Necessite d'etre en mode selection multiple
7  mon_tselect.select_range(START_PATH, END_PATH)
8  mon_tselect.unselect_range(START_PATH, END_PATH)
```

On peut aussi savoir si une ligne précise est sélectionnée ou non:

```
1  est_actif = mon_tselect.path_is_selected(PATH)
2  est_actif = mon_tselect.iter_is_selected(mon_TreeIter)
```

Pour connaître le nombre de lignes sélectionnées:

```
1  mon_tselect.count_selected_rows()
```

Enfin, le principal, pour récupérer les lignes sélectionnées:

```
2  mon_modele, mon_treeiter = mon_tselect.get_selected()
3  mon_modele, mon_treeiter = mon_tselect.get_selected_rows()
```

La ligne 1 est à utiliser dans les modes de sélection unique, la ligne 2 dans le mode de sélections multiples.

***Remarque: Les modèles sont une façades des TreeView que je n'ai volontairement pas évoqués, car cela complexifie considérablement l'approche, et n'est pas indispensable à l'usage des TreeView. Vous êtes obligé de le récupérer lors de l'appel de ces dernières méthodes, mais n'êtes nullement obligé de vous en servir par la suite. N'hésitez pas à consulter la documentation officielle PYGTK sur les TreeModel si votre curiosité vous le demande.***

##### 5.1.3.7.3 Les TreeViewColumn

Le dernier élément. Il permet d'afficher les données à l'utilisateur. Là aussi, la création est simple. Il ne prend qu'un seul paramètre qui s'avère être le titre de la colonne qu'il portera à l'écran.

```
1 ma_colonne = gtk.TreeViewColumn('Colonne 1')
```

parmi les méthodes les plus usuelles nous remarquerons les suivantes:

```
1 ma_colonne.set_sizing(gtk.TREE_VIEW_COLUMN_AUTOSIZE)
2 ma_colonne.set_fixed_width(60)
3 ma_colonne.set_resizable(True)
4 ma_colonne.set_expand(True)
5 mon_tv = ma_colonne.get_tree_view()
6 ma_colonne.set_title('Colonne 0')
7 ma_colonne.set_clickable(True)
8 ma_colonne.set_alignment(0.5)
```

Ligne 1, nous définissons le comportement que doit avoir la largeur de la cellule vis-à-vis de la largeur de son contenu. Les paramètres possibles sont les suivants:

| Paramètre | Description |
|---|---|
| **gtk.TREE_VIEW_COLUMN_AUTOSIZE** | La colonne prend automatiquement la bonne largeur |
| **gtk.TREE_VIEW_COLUMN_FIXED** | La largeur de la colonne est paramétrée en pixels par le programmeur (voir ligne2) |
| **gtk.TREE_VIEW_COLUMN_GROW_ONLY** | La taille de la colonne est définie par le modèle |

Il existe également la méthode **get_sizing()**.

Ligne 2, nous définissons la largeur de la colonne en pixels. Il existe aussi la méthode **get_fixed_width()**.

***Attention: Le paramètre de largeur doit être strictement supérieur à 0 afin de ne pas générer d'erreurs.***

Ligne 3, nous indiquons à **GTK** que nous autorisons l'utilisateur à modifier manuellement la largeur de la colonne. Utiliser la méthode **get_resizable()** pour connaître ce paramétrage.

Ligne 4, nous indiquons à **GTK** que la colonne doit prendre le maximum de place qu'elle peut prendre. La aussi, la méthode inverse existe: **get_expand()**.

Ligne 5, nous récupérons le nom du **TreeView** dont dépend notre **TreeViewColumn**

Ligne 6, nous modifions le titre de la colonne.

Ligne 7, nous transformons l'entête, contenant le titre, de la colonne en bouton. Le signal à utiliser alors est 'clicked'.

Ligne 8, nous centrons le titre de notre colonne.

***Remarque: L'ajout/insert/retrait de colonnes se gère au niveau du TreeView***

Pour la suite, nous allons compliquer légèrement, une fois de plus, les choses. En effet, même si cela n'est pas obligatoire, vous avez la possibilité de passer par ce qu'on appelle un **CellRendererText**.

Il s'agit d'un objet permettant des actions supplémentaires sur les colonnes. Par exemple, vous pouvez changer la couleur de fond, masquer les données, ...

```
1 mon_crt = gtk.CellRendererText()
2 mon_crt.set_property('cell-background', 'cyan')
3 mon_crt.set_property('''visible', False)
4 ma_colonne.pack_start(mon_crt, True)
5 ma_colonne.add_attribute(mon_crt, 'text', 0)
```

Ligne 1, nous créons simplement notre **CellRendererText**.

Ligne 2, nous paramétrons la couleur de fond. Les valeurs possibles sont entre autres: cyan, red, yellow, magenta, green, blue.

Pour définir votre propre couleur, utilisez: mon_crt.set_property('cell-background-gdk', gtk.gdk.Color(R,G,B,P)

Ici, RGB sont compris entre 0.0 et 1.0. P est simplement un index.

Ligne 3, Nous précisons que nous voulons cacher les données.

Ligne 4, nous positionnons le **CellRendererText** dans notre **TreeViewColumn**. Le 2nd paramètre à True est expand. Nous sommes ainsi assurés d'occuper tout l'espace disponible.

Ligne 5, nous configurons notre **CellRendererText** par rapport à notre **TreeViewColumn**. Le 1er argument est le **CellRendererText**, le 2nd le type de **CellRenderer** (toujours 'text'), et enfin le numéro de colonne auquel il doit être rattaché.

#### 5.1.3.8 Label

Les labels permettent d'afficher du texte non modifiable par l'utilisateur.

```
1 mon_label = gtk.Label("Hello_world")
```

Le texte, une fois établi est librement modifiable via une simple méthode

```
1 mon_lable.set_text(" nouveau texte ")
```

Vous avez également la possibilité de choisir l'emplacement du texte

```
1 mon_label.set_justify(gtk.JUSTIFY_CENTER)
```

On remarque ici l'utilisation d'une constante **PYGTK**. Les autres choix possibles sont **gtk.JUSTIFY_LEFT** (qui est la valeur par défaut), **gtk.JUSTIFY_RIGHT** et **gtk.JUSTIFY_FILL**.

***Remarque: Même si cela n'est pas vu ici, il existe la possibilité de choisir sa police, couleur, style de texte, ..., via du code***

***html dans la méthode set_text. Regardez sur Internet du coté de la méthode set_use_markup pour plus d'informations.***

#### 5.1.3.9 Textbox

##### 5.1.3.9.1 Monoligne

Les monolignes sont ce qu'on appelle des entrées. Le constructeur portera donc ce nom et prendra comme paramètre la longueur de texte qu'il doit supporter. Par défaut, paramétrer à 0 (infini).

```
1  mon_entree = gtk.Entry(Longeur)
```

Parmi ses méthodes, les plus usitées sont les suivantes

```
1  mon_entree.set_text(" Texte ") #renseigne le champ
2  mon_entree.get_text() #recupere le texte du champ
3  mon_entree.set_alignment(0,5) #Place le texte au centre (max=1 à droite,min=0)
4  mon_entree.set_visibility(True) #remplace l'écho des caractères par des points
5  mon_entree.set_max_length(10) #définie la taille max
6  mon_entree.get_max_length() #renvoie la taille max du champ
```

##### 5.1.3.9.2 Multilignes

Les textbox multilignes ne sont, à première vue pas si évidentes à maîtriser.

Si je dis cela, c'est qu'il ne suffit pas de créer un simple objet pour en disposer.

En effet, une textbox multilignes  est en réalité un assemblage de 3 objets: un **TextView**, un **TextBuffer**, un **TextTagTable**.

Le **TextTagTable** peut être vu comme un mini espace de stockage, abritant les données que l'utilisateur saisira et lira.

Le **TextBuffer** est l'objet **PYGTK** contenant le texte qui aura été saisi, et qui sera affiché à l'utilisateur. Le **TextBuffer** utilise comme aire

de stockage un **TextTagTable**. Il s'agit donc simplement d'un outil d'accès à des données.

Pour l'afficher à l'utilisateur, nous utilisons un **TextView**. Cet objet se contente de faire interface entre l'utilisateur et le **TextBuffer**.

Voilà pour la partie explication fonctionnelle. Le plus dur est fait. Pour la partie pratique, les choses se simplifient.

Il faut savoir que **PYGTK** nous permet une manipulation partiellement transparente de ces objets.

Ainsi, par exemple, la création d'un simple **Textview** entraîne la création automatique d'un **TextBuffer** et d'un **TextTagTable** associés. De plus, en général, nous n'avons nullement besoin d'intervenir sur le **TextTagTable**, simplifiant encore les choses.

Au final donc, nous interviendrons la plupart du temps au niveau du **TextView** pour paramétrer notre interaction avec l'utilisateur, et sur le **TextBuffer** pour gérer le texte.

```
1 mon_tv = gtk.TextView()
2 mon_buf = mon_tv.get_buffer()
```

Faisons simple pour commencer: la création des objets.

Ligne 1, nous créons notre **textview**, et ligne 2, nous récupérons la référence au buffer lié.

Commençons la revue en détail par le **TextView**.

Vous avez la possibilité d'autoriser ou non l'édition du texte afficher, et de connaître si cela est possible:

```
1 mon_tv.set_editable(True)#True ou False
2 edition = mon_tv.get_editable()
```

Vous avez également la possibilité de masquer le curseur:

```
1 mon_tv.set_cursor_visible(True) #True ou False
```

Vous pouvez aussi paramétrer le type de renvoi à la ligne automatique:

```
1 mon_tv.set_wrap_mode(gtk.WRAP_NONE) #Sans retour à la ligne
2 mon_tv.set_wrap_mode(gtk.WRAP_WORD) #Retour à la ligne au mot près
```

Enfin, reste le paramétrage de la justification:

```
1 mon_tv.set_justification(gtk.JUSTIFY_LEFT)
2 mon_tv.set_justification(gtk.JUSTIFY_RIGHT)
3 mon_tv.set_justification(gtk.JUSTIFY_CENTER)
4 justif = mon_tv.get_justification     #On recupere la justification parametree
```

***Remarque: Les TextView ne savent pas gérer les scrolls (barres de défilement). En cas de besoin, il vous faut donc passer par une fenêtre ScrolledWindow.***

Maintenant, place à la revue en détail du **TextBuffer**.

Comme dit précédemment, il s'agit de l'objet contenant le texte à afficher sur le **TextView**.

Vous pouvez grâce à lui connaître le nombre de lignes ou encore de caractères qu'il contient:

```
1 mon_buf.get_line_count()
2 mon_buf.get_char_count()
```

De manière tout aussi pratique, vous pouvez initialiser le texte:

```
1 mon_buf.set_text(mon_texte)
```

Le plus pratique maintenant: récupérer du texte. Cette partie se complique légèrement, mais rien d'inquiétant je vous rassure.

En effet; la méthode récupérant le texte demande de préciser le caractère de début et de fin, d'où l'utilité de connaître le nombre total de caractères dans le texte. Le dernier paramètre passé permet de préciser que l'on veut récupérer la totalité des caractères de la fourchette fournie.

```
1 mon_texte = mon_buf.get_text(start, end, True)
```

Finissons maintenant par l'effacement. Là aussi, il faut passer une fourchette de caractères:

```
1 mon_buf.delete(start, end)
```

Voici pour les explications d'une textbox multilignes. Le but ici n'est clairement pas de créer un éditeur de texte. D'autres paramétrages sont évidemment possibles, mais pour ces derniers plus complexes, et moins usuels, je vous renvoie vers la documentation officielle.

#### 5.1.3.10 Compteur

Les compteurs sont de petits widgets permettant de choisir une valeur numérique à l'aide de deux petits curseurs.

Pour créer un widget compteur, il faut avant créer un objet appelé **Adjustment**.

```
1 adjustment = gtk.Adjustment(value,lower,upper,step_incr,page_incr,page_size)
```

Comme on peut le voir, plusieurs paramètres sont ici passés. Il vont permettre de paramétrer précisément, par la suite, notre compteur. L'ensemble de ces paramètres sont par défaut à 0 (pouvant alors signifier infini pour certain paramètres)

| paramètre | Description |
|---|---|
| **value** | Valeur initiale |
| **lower** | Minimum possible |
| **upper** | Maximum possible |
| **step_incr** | Pas d'incrémentation (ou de décrémentation) lors de l'appui sur un bouton avec le click gauche |
| **page_incr** | Pas d'incrémentation (ou de décrémentation) lors de l'appui sur un bouton avec le click droit |
| **page_size** | Non utilisé |

Une fois ce paramétrage réalisé, il faut créer notre compteur

```
1 mon_compteur = gtk.SpinButton(adjustment, climb_rate, digits)
```

Ici, trois paramètres.

| paramètre | Description |
|---|---|
| **adjustment** | Notre objet créé précédemment |
| **climb_rate** | La vitesse d'avance quand on reste appuyé sur un des deux boutons (en %: de 0.0 à 1.0) |
| **digits** | Nombre de digits à utiliser pour l'affichage |

Vous pouvez reconfigurer ce compteur également à la volée.

```
1 mon_compteur.configure(adjustment, climb_rate, digits)
```

Vous pouvez aussi récupérer l'ajustement.

```
1 mon_compteur.get_adjustment()
```

Enfin, vous pouvez récupérer la valeur simplement avec un get.

```
1 mon_compteur.get_value()
```

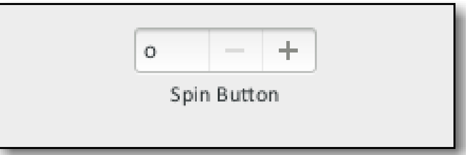



***Illustration 38: Un compteur (http://developer.gnome.org)***

### 5.1.3.11 Progressbar

Les **barres de progression** sont présentes un peu partout dans les logiciels afin de montrer le niveau d'avancement d'une tâche, ou encore pour représenter graphiquement certaines valeurs.

```
1 ma_pb = gtk.ProgressBar() #creation d'une pb à 0%
```

Ensuite, il est possible de changer l'orientation( et de fait le sens de la progression) grâce à la méthode set_orientation(). Les valeurs possibles sont les suivantes

| Constante | Description |
|---|---|
| **gtk.PROGRESS_LEFT_TO_RIGHT** | De gauche à droite (par défaut) |
| **gtk.PROGRESS_RIGHT_TO_LEFT** | De droite à gauche |
| **gtk.PROGRESS_BOTTOM_TO_TOP** | De bas en haut |
| **gtk.PROGRESS_TOP_TO_BOTTOM** | De haut en bas |

```
1 ma_pb.set_orientation( = gtk.ProgressBar(PYGTK.PROGRESS_LEFT_TO_RIGHT)
```

Il existe ensuite différentes façons de faire progresser une **ProgressBar**. La première méthode (la plus simple) consiste à lui indiquer quelle valeur elle doit représenter

```
1 ma_pb.set_fraction(0.5) #50%
2 ma_pb.set_fraction(1.0) #100%
```

La seconde méthode consiste à définir un pas d'avance, et à envoyer des impulsions

```
1 ma_pb.set_pulse_step(0.01) #pas de 1%
2 ma_pb.pulse() #chaque execution de cette ligne ajoute 1%
```

Par défaut, la **ProgressBar** n'affiche aucun texte. Il est cependant possible de le lire et de le définir.

```
1 ma_pb.set_text(" ma progressbar ")
2 retour = ma_pb.get_text()
```

#### 5.1.3.12 Calendrier

Il peut être utile de présenter à l'utilisateur un **calendrier**, ne serait-ce que pour s'assurer que la date qu'il va renseigner sera conforme à nos attentes. Pour cela, le widget calendrier est très utile.

```
1 mon_calendrier = gtk.Calendar()
```

Ici aussi, rien de très sorcier. Parmi les méthodes les plus utiles, nous retrouvons les suivantes

```
1 mon_calendrier.select_month(mois, annee) #affiche le calendrier
2 mon_calendrier.select_day(jour) #selectionne le jour dans le calendrier (de 1 à 31)
3 mon_calendrier.mark_day( jour) #met le jour en gras
4 mon_calendrier.get_date() #retourne la sélection user en tuple (an, mois, jour)
```

#### 5.1.3.13 Image

L'affichage d'images sous **PYGTK** est très simple. Il suffit de créer un widget Image, puis de le remplir avec une image. Il ne reste ensuite plus qu'à mettre le widget dans un conteneur.

```
1 mon_image = gtk.Image()
2 mon_image.set_from_file("./mon_image.png ")
3 mon_image.show()
```

Il faut savoir aussi que l'on peut insérer une image dans un bouton pour remplacer une icône par exemple (attention alors à la taille de l'image), en s'en servant comme d'un conteneur.

```
1 mon_bouton.add(mon_image)
```

***Remarque: il est également possible de mettre l'image avec un label dans une Hbox et de mettre la Hbox dans le bouton***

#### 5.1.3.13.1 Image en fond de fenêtre

Nous allons voir ici comment mettre une image en fond de fenêtre et faire en sorte que l'image s'adapte à la taille de la fenêtre, même si cette dernière est redimensionnée.

Nous allons commenter le code suivant, qui vous permettra de réaliser cette action. Nous verrons certaines méthodes dont nous ne détaillerons que le strict nécessaire. Cela dépasserait le cadre des simples éléments de bases, mais une petite recherche sur Internet vous aidera.

A partir de ce code, vous pourrez déduire par vous même comment faire dans le cas d'une fenêtre de taille fixe.

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3  import gtk
4
5  def draw_pixbuf(widget, event):
6      path = './logo.jpg'
7      allocation = widget.get_allocation()
8      pixbuf = gtk.gdk.pixbuf_new_from_file(path)
9      pixbuf = pixbuf.scale_simple(allocation.width, allocation.height,  \
10                 gtk.gdk.INTERP_TILES)
11
12     widget.window.draw_pixbuf(widget.style.bg_gc[gtk.STATE_NORMAL], \
13                          pixbuf, 0, 0, 0,0)
14
15     widget.window.invalidate_rect( allocation, False )
16
17     del pixbuf
18
19
20 window = gtk.Window()
21 window.set_title('Drawing Test')
22 window.set_size_request(64,48)
23 window.connect('destroy',gtk.main_quit)
24 hbbox = gtk.HBox()
25 window.add(hbbox)
26 hbbox.connect('expose-event', draw_pixbuf)
27 button = gtk.Button()
28 hbbox.pack_start(button, True, False, 50)
29 window.show_all()
30
31 gtk.main()
```

Ligne 1 et 3, vous reconnaîtrez des lignes classiques, dont nous passerons l'explication.

Ligne 20 à 23, nous créons et configurons notre fenêtre principale.

Ligne 24, nous créons un conteneur **Hbox**, et l'ajoutons à notre fenêtre principale ligne 25.

Ligne 27 et 28, nous créons et ajoutons un bouton à notre conteneur, puis ligne 29 et 31, nous affichons et démarrons notre logiciel.

Maintenant attaquons la partie intéressante de ce code. Tout d'abord la ligne 26. Nous connectons notre conteneur à la fonction draw_pixbuf lorsque l'événement " **expose-event** " est déclenché.

***Remarque: Il existe de nombreux événement possible (fenêtre détruite, pointeur souris qui bouge, bouton de souris pressé ou relâché, double ou triple click, touche de clavier pressé ou relachée, souris qui rentre ou sort d'une fenêtre, ...). Rendez vous sur cette page pour les retrouver: http://pygtk.org/docs/pygtk/class-gtkwidget.html***

Cet événement est généré lorsqu'un événement quelconque (déplacement, modification de taille, ...) a lieu sur le conteneur ou la fenêtre concernée.

Dans notre code, quand cela se produit nous appelons donc draw_pixbuf, fonction déclarée de la ligne 5 à 17.

Pour commencer, ligne 6, nous déclarons le chemin de l'image de fond.

Ligne 7, nous effectuons un get_allocation, de la classe widget. Cela permet de récupérer la largeur et la hauteur du widget (dans notre cas, allocation.height et allocation.width).

Ligne 8, nous créons un **pixbuf** à partir de notre image. Le pixbuf est un type d'image servant beaucoup en interne dans GTK.

Ligne 9 et 10, nous redimensionnons notre image à la taille du conteneur. Les paramètres sont la largeur, la hauteur, et le type de rendu. Ici, la constante **gtk.gdk.INTERP_TILES** est le meilleur compromis entre performance et rendu.

Ligne 12 et 13, nous dessinons notre pixbuf dans notre conteneur. Les 4 derniers paramètres sont des coordonnées, qui seront quasi exclusivement toujours à 0.

Ligne 15, nous effectuons une opération d'invalidation. Sans cette opération, votre dessin sera redimensionné mais également superposé avec tous les dessins intermédiaires du redimensionnement. Je vous invite à essayer pour en constater les effets concrets.

Ligne 17 enfin, nous effaçons notre objet pixbuf, qui une fois inséré en fond de conteneur, ne sert plus.

## 5.1.3.14 Cadre

Les **cadres**, comme leur nom l'indique, servent à encadrer un lot de widgets en leur donnant un nom précis.

```
1 mon_cadre = gtk.Frame(label= "Mon cadre ")
```

Peu de choses à connaître pour s'en servir. On peut changer le label en cours de code, jouer sur la position du texte par rapport au **cadre** (les paramètres X & Y varient alors entre 0.0 et 1.0, par défaut à 0.0), et enfin jouer sur le cadre lui-même, grâce aux constantes suivantes.

| Constante | Description |
|---|---|
| **SHADOW_ETCHED_IN** | Par défaut, trait du contour en creux |
| **SHADOW_ETCHED_OUT** | Trait du contour en relief |
| **SHADOW_IN** | Cadre de type creux |
| **SHADOW_OUT** | Cadre de type relief |
| **SHADOW_NONE** | Sans relief |

```
mon_cadre = gtk.Frame(label= "Mon cadre ")mon_cadre.set_label(" mon nouveau titre")
mon_cadre.set_label_align(X, Y)
mon_cadre.set_shadow_type(SHADOW_ETCHED_IN)
```

Comme d'habitude, on n'oubliera pas d’exécuter la méthode show sur notre widget afin de l'afficher.

***Remarque: S'il est un widget, le cadre joue également le rôle de conteneur.***

***Illustration 39: Un exemple de cadre (http://developer.gnome.org)***

## 5.1.3.15 Menu

Les **menus** permettent à l'utilisateur d'accéder à nombre de fonctions et de paramétrages au sein d'une application.

Selon l'application, on pourra se contenter d'une barre d'outils, ou l'on devra au contraire implémenter un menu complet

#### 5.1.3.15.1 Structure d'un menu

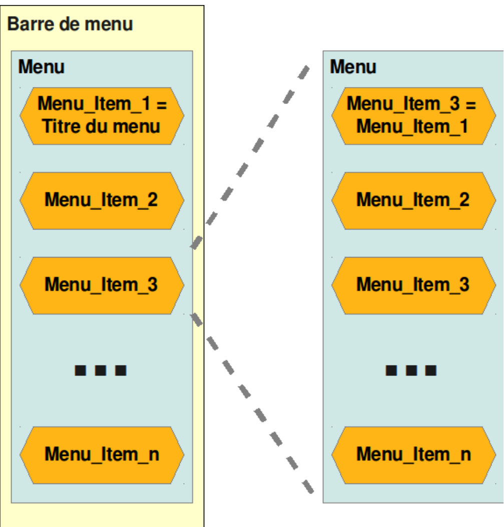



***Illustration 40: Fonctionnement des menus PYGTK***

L'illustration ci-dessus résume le fonctionnement des menus sous **PYGTK**.

Un menu **PYGTK** n'est qu'un assemblage de **menus** et de **menu_Items**, le tout rattaché à une barre de menu. Par menu_items nous entendons tout texte interprété du point de vue utilisateur comme une entrée menu.

Tout sous menu est un menu à part entière.

Le premier menu_items d'un menu est également le titre du menu

#### 5.1.3.15.2 Menu texte

```
1 mon_menu = gtk.MenuBar()
2 menu_fichier = gtk.Menu()
3 m_fichier = gtk.MenuItem("_Fichier")
4 m_fichier.set_submenu(menu_fichier)
5 exit = gtk.MenuItem("_Exit")
6 exit.connect("activate", gtk.main_quit)
7 menu_fichier.append(exit)
8 mb.append(m_fichier)
```

Ligne 1, nous créons notre barre de menus. C'est elle qui contiendra l'ensemble des menus.

Ligne 2, nous créons notre menu Fichier. Il est alors vide, sans aucune donnée.

Ligne 3, nous créons notre premier menu_items, Fichier. Le " _ " placé avant le F dans le label indique à **PYGTK** qu'il doit souligner le F. Cela peut servir à indiquer à l'utilisateur un raccourci clavier par exemple.

Ligne 4, nous connectons notre menu_items à notre menu afin d'en faire le titre du menu.

Ligne 5, nous créons un nouveau menu_items, et le configurons linge 6.

Ligne 7, nous l'ajoutons au menu Fichier.

Ligne 8, nous ajoutons notre menu à la barre de menu.

***Remarque: Une fois assimilé, le fonctionnement de création d'un menu n'est pas si complexe qu'en prime abord. Il est important de bien noter que le premier élément inséré dans le conteneur de menu sera l'élément que l'utilisateur considérera comme le nom du menu. L'ordre a donc son importance.***

#### 5.1.3.15.3 Sous menu

Pour éviter qu'un menu ne devienne trop brouillon, nous pouvons effectuer des regroupements par groupe. Cela donnera à l'utilisateur l'impression de disposer de menus et de sous-menus par thèmes.

Comme vu précédemment, un sous-menu n'est rien d'autre qu'un menu inséré dans un autre menu. Nous allons donc créer plusieurs menus, et les insérer les uns dans les autres afin de créer nos menus et sous-menus.

```
1  mon_menu = gtk.MenuBar()
2  menu_fichier = gtk.Menu()
3  m_fichier = gtk.MenuItem("_Fichier")
4  m_fichier.set_submenu(menu_fichier)
5  mb.append(m_fichier)
6
7  menu_choix = gtk.Menu()
8  m_choix = gtk.MenuItem("_Choix")
9
10 choix1 = gtk.MenuItem("Choix 1")
11 choix2 = gtk.MenuItem("Choix 2")
12 choix3 = gtk.MenuItem("Choix 3")
13
14 m_choix.set_submenu(menu_choix)
15 menu_choix.append(choix1)
16 menu_choix.append(choix2)
17 menu_choix.append(choix3)
18
19 menu_fichier.append(m_choix)
```

Nous pouvons voir, ligne 1 à 5, la création du menu principal.

Lignes 7 à 17, nous avons la création de notre menu2, qui va devenir notre sous-menu.

Enfin, ligne 19, nous intégrons notre menu 2 au menu 1 et en faisons donc le sous-menu.

### 5.1.3.16 Barre d'outils

Les **barres d'outils** sont les barres contenant généralement des icônes et vous permettant d'avoir un accès rapide à certaines fonctions.

```
1 ma_toolbar = gtk.Toolbar()
2 ma_toolbar.set_style(gtk.TOOLBAR_ICONS)
3 b1 = gtk.ToolButton(gtk.STOCK_NEW, 'Nouveau')
4 b2 = gtk.ToolButton(gtk.STOCK_QUIT, 'Quitter')
5 separator = gtk.SeparatorToolItem()
6 ma_toolbar.insert(b1, 0)
7 ma_toolbar.insert(separator, 1)
8 ma_toolbar.insert(b2, 3)
9 b2.connect("clicked", gtk.main_quit)
```

Dans cet exemple, ligne 1, nous créons notre barre d'outils.

Ligne 2, nous lui précisons que nous ne voulons que des icônes dans cette barre d'outil. Nous aurions pu également utiliser " **gtk.TOOLBAR_TEXT** " pour indiquer que nous ne voulions que du texte, ou encore " **gtk.TOOLBAR_BOTH** " pour indiquer que nous voulions du texte et des icônes.

Ligne 3 et 4, nous créons nos boutons dédiés à la toolbar, en précisant le type d'icônes utilisées. Dans notre exemple, il s'agit d'icône **PYGTK** prédéfinies. Nous précisons également le texte correspondant aux boutons.

Ligne 5, nous créons un séparateur de boutons pour la lisibilité de la toolbar.

Lignes 6 à 8, nous insérons nos objets dans la toolbar, en assignant à chacun une position. Le compteur commence à gauche, à partir de 0.

Ligne 9 nous connectons un de nos boutons à une fonction. Un **ToolButton** se configure comme un bouton classique.

### 5.1.3.17 Barre de statut

La barre de statut est une petite barre mince, située en bas de la fenêtre d'un programme et servant à donner des informations à l'utilisateur: aide à l'utilisation, avancée d'un traitement, espace mémoire, …

La création et la manipulation d'une barre de statut est très simple:

```
1 ma_statusbar = gtk.Statusbar()
2 ma_statusbar.push(ID_TEXT, mon_texte)
3 ma_statusbar.pop(ID_TEXT)
```

Ligne 1, nous créons notre barre de status.

Ligne 2, nous l'alimentons. Deux paramètres sont passés ici: un ID, et un texte. Nous pouvons passer un nombre important de textes à une barre de statut. L'ID permet de les différencier les uns des autres.

Ligne 3, nous supprimons un texte donné grâce à son ID.

### 5.1.3.18 Les curseurs

Les **curseurs**, qu'ils soient verticaux ou horizontaux, permettent à l'utilisateur de configurer facilement des paramètres numériques, tout comme les compteurs.

Leur création est extrêmement aisée:

```
1 cur_v = gtk.VScale()
2 cur_h = gtk.HScale()
```

Les paramétrages pour les deux sont identiques. Pour la suite nous utiliserons donc le curseur horizontal en exemple.

```
1 cur_h.set_range(0, 100)
2 cur_h.set_increments(1, 5)
3 cur_h.set_digits(0)
4 cur_h.set_size_request(150, 25)
```

```
5 cur_h.connect("value-changed", cur_h_changed)
```

Ligne 1, nous paramétrons le min et le max de notre curseur.

Ligne 2, nous paramétrons le pas par mouvement du curseur sur sa barre, via la souris (1er paramètre) et lorsque l'utilisateur utilise les flèches clavier ou clique sur la barre de défilement, sur le côté du curseur (2nd paramètre).

Ligne 3, indique le nombre de décimales désirées dans la valeur du curseur.

Ligne 4, nous paramétrons la taille du widget curseur en pixels (largeur puis hauteur).

Ligne 5, enfin, nous connectons le changement de la valeur du curseur à une fonction/procédure.

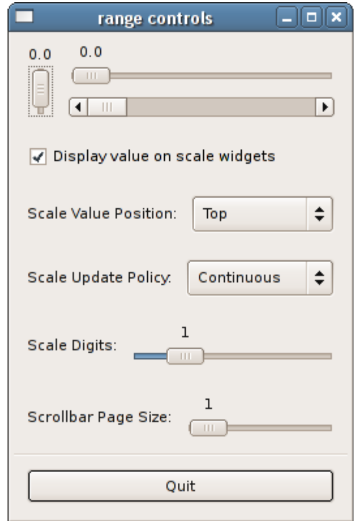



***Illustration 41: Assemblage de Combobox et de Scale (http://developer.gnome.org)***

### 5.1.3.19 Bulles d'aide

Pour chacun des widgets de **PYGTK**, on a la possibilité de créer des **infobulles**. Ces infobulles sont elles-même considérées comme des widgets.

```
1  mon_infob = gtk.Tooltips()
2  mon_infob.set_tip(mon_widget, " texte ", None)
```

### 5.1.3.20 Les stocks (icônes)

Les **stocks** sont des icônes prédéfinies dans **PYGTK**. Selon la version de **PYGTK** que vous utilisez, vous en disposerez de plus ou moins. Vous trouverez ici le nom et l'icône de ces stocks

| Constante | icône |
|---|---|
| gtk.STOCK_ABOUT | |
| gtk.STOCK_ADD | |
| gtk.STOCK_APPLY | |
| gtk.STOCK_BOLD | |
| gtk.STOCK_CANCEL | |
| gtk.STOCK_CDROM | |
| gtk.STOCK_CLEAR | |
| gtk.STOCK_CLOSE | |
| gtk.STOCK_COLOR_PICKER | |
| gtk.STOCK_CONVERT | |

| | |
|---|---|
| gtk.STOCK_CONNECT | |
| gtk.STOCK_COPY | |
| gtk.STOCK_CUT | |
| gtk.STOCK_DELETE | |
| gtk.STOCK_DIALOG_AUTHENTICATION | |
| gtk.STOCK_DIALOG_ERROR | |
| gtk.STOCK_DIALOG_INFO | |
| gtk.STOCK_DIALOG_QUESTION | |
| gtk.STOCK_DIALOG_WARNING | |
| gtk.STOCK_DIRECTORY | |
| gtk.STOCK_DISCONNECT | |
| gtk.STOCK_DND | |
| gtk.STOCK_DND_MULTIPLE | |
| gtk.STOCK_EDIT | |

| | |
|---|---|
| gtk.STOCK_EXECUTE | |
| gtk.STOCK_FILE | |
| gtk.STOCK_FIND | |
| gtk.STOCK_FIND_AND_REPLACE | |
| gtk.STOCK_FLOPPY | |
| gtk.STOCK_FULLSCREEN | |
| gtk.STOCK_GOTO_BOTTOM | |
| gtk.STOCK_GOTO_FIRST | |
| gtk.STOCK_GOTO_LAST | |
| gtk.STOCK_GOTO_TOP | |
| gtk.STOCK_GO_BACK | |
| gtk.STOCK_GO_DOWN | |
| gtk.STOCK_GO_FORWARD | |
| gtk.STOCK_GO_UP | |
| gtk.STOCK_HARDDISK | |

| gtk.STOCK_HELP | |
| --- | --- |
| gtk.STOCK_HOME | |
| gtk.STOCK_INDENT | |
| gtk.STOCK_INDEX | |
| gtk.STOCK_INFO | |
| gtk.STOCK_ITALIC | |
| gtk.STOCK_JUMP_TO | |
| gtk.STOCK_JUSTIFY_CENTER | |
| gtk.STOCK_JUSTIFY_FILL | |
| gtk.STOCK_JUSTIFY_LEFT | |
| gtk.STOCK_JUSTIFY_RIGHT | |
| gtk.STOCK_LEAVE_FULLSCREEN | |
| gtk.STOCK_MEDIA_FORWARD | |
| gtk.STOCK_MEDIA_NEXT | |
| gtk.STOCK_MEDIA_PAUSE | |
| gtk.STOCK_MEDIA_PLAY | |

| | |
|---|---|
| gtk.STOCK_MEDIA_PREVIOUS | |
| gtk.STOCK_MEDIA_RECORD | |
| gtk.STOCK_MEDIA_REWIND | |
| gtk.STOCK_MEDIA_STOP | |
| gtk.STOCK_MISSING_IMAGE | |
| gtk.STOCK_NETWORK | |
| gtk.STOCK_NEW | |
| gtk.STOCK_NO | |
| gtk.STOCK_OK | |
| gtk.STOCK_OPEN | |
| gtk.STOCK_PAGE_SETUP | |
| gtk.STOCK_PASTE | |
| gtk.STOCK_PREFERENCES | |
| gtk.STOCK_PRINT | |
| gtk.STOCK_PRINT_ERROR | |

| | |
|---|---|
| gtk.STOCK_PRINT_PAUSED | |
| gtk.STOCK_PRINT_PREVIEW | |
| gtk.STOCK_PRINT_REPORT | |
| gtk.STOCK_PRINT_WARNING | |
| gtk.STOCK_PROPERTIES | |
| gtk.STOCK_QUIT | |
| gtk.STOCK_REDO | |
| gtk.STOCK_REFRESH | |
| gtk.STOCK_REMOVE | |
| gtk.STOCK_REVERT_TO_SAVED | |
| gtk.STOCK_SAVE | |
| gtk.STOCK_SAVE_AS | |
| gtk.STOCK_SELECT_COLOR | |
| gtk.STOCK_SELECT_FONT | |
| gtk.STOCK_SORT_ASCENDING | |
| gtk.STOCK_SORT_DESCENDING | |

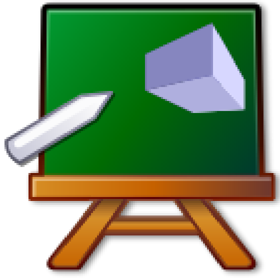
| | |
|---|---|
| gtk.STOCK_SPELL_CHECK | |
| gtk.STOCK_STOP | |
| gtk.STOCK_STRIKETHROUGH | |
| gtk.STOCK_UNDELETE | |
| gtk.STOCK_UNDERLINE | |
| gtk.STOCK_UNDO | |
| gtk.STOCK_UNINDENT | |
| gtk.STOCK_YES | |
| gtk.STOCK_ZOOM_100 | |
| gtk.STOCK_ZOOM_FIT | |
| gtk.STOCK_ZOOM_IN | |
| gtk.STOCK_ZOOM_OUT | |

## 5.1.3.21 Exemple

```
#!/usr/bin/PYTHON
# -*-coding:utf-8 -*




import pygtk
import gtk




def appui_but(widget):
	val = lab_nb.get_text()
	if val.isdigit():
		calcul = int(val) * int(val)
		lab_square.set_text(str(calcul))
	else:
		err_win = gtk.MessageDialog(None,gtk.DIALOG_MODAL,\
				gtk.MESSAGE_ERROR, gtk.BUTTONS_OK, \
				"Valeur saisie non numerique")
		err_win.run()
		err_win.destroy()



mw = gtk.Window()
mw.set_title("Calcul d'un carre (3 chiffres)")
mw.set_size_request(300,100)
mw.connect("destroy",gtk.main_quit)

nb = gtk.Frame("Nombre")

square = gtk.Frame("Carre")

but = gtk.Button("Calcul du carre")
but.connect("clicked", appui_but)

lab_nb = gtk.Entry(3)
lab_nb.set_alignment(0.5)
lab_square = gtk.Label("0")
nb.add(lab_nb)

square.add(lab_square)

table = gtk.Table(rows=2, columns=2, homogeneous=True)
table.attach(nb, 0,1,0,1)
table.attach(square,1,2,0,1)
```

```
49 table.attach(but,0,2,1,2, xoptions=gtk.SHRINK, yoptions=gtk.SHRINK)
50
51 mw.add(table)
52
53 table.show()
54 but.show()
55 square.show()
56 nb.show()
57 lab_nb.show()
58 lab_square.show()
59 mw.show()
60
61 gtk.main()
```

# 6 La 3D



Wikimedia Commons,
Nuvola_apps_edu_miscellaneous_3dCube.svg

## 6.1 Le format STL

Le format **STL** est un format très répandu dans l'univers du prototypage. Il s'agit d'un simple fichier ASCII ou binaire permettant de définir la forme d'un objet en 3D en le décrivant sous forme d'un assemblage de triangles.

Chacun de ces triangles est formé à partir de trois points 3D (appelé vertex), et d'une normale, généralement perpendiculaire à la surface du triangle, permettant d'indiquer au logiciel comment la lumière doit interagir avec cette surface.

### 6.1.1 STL ASCII

Un fichier STL commence avec le mot clé solid, suivi du nom de l'objet ; et il se finit avec le mot clé endsolid suivi du nom de l'objet.

Entre les deux, nous retrouvons la définition de chaque triangle à suivre, de ce type:

```
1 facet normal ni nj nk
2   outer loop
3     vertex v1x v1y v1z
4     vertex v2x v2y v2z
5     vertex v3x v3y v3z
6   endloop
7 endfacet
```

***Remarque: Les espaces sont importants dans la définition d'un fichier STL ASCII. Il faut respecter le décalage tel que dans l'exemple ci-dessous et ne surtout pas utiliser de tabulation pour remplacer les espaces.***

Imaginons maintenant un objet très simple type pyramide, donc quatre triangles uniquement. Voici à quoi pourrait ressembler son fichier ASCII STL:

```
1 solid Pyramid
2   facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
```

```
3       outer loop
4         vertex 0.000000e+00 0.000000e+00 0.000000e+00
5         vertex 1.000000e+00 0.000000e+00 0.000000e+00
6         vertex 0.000000e+00 1.000000e+00 0.000000e+00
7       endloop
8     endfacet
9     facet normal -1.000000e+00 0.000000e+00 0.000000e+00
10      outer loop
11        vertex 0.000000e+00 0.000000e+00 0.000000e+00
12        vertex 0.000000e+00 1.000000e+00 0.000000e+00
13        vertex 0.000000e+00 0.000000e+00 1.000000e+00
14      endloop
15    endfacet
16    facet normal 0.000000e+00 -1.000000e+00 0.000000e+00
17      outer loop
18        vertex 0.000000e+00 0.000000e+00 0.000000e+00
19        vertex 1.000000e+00 0.000000e+00 0.000000e+00
20        vertex 0.000000e+00 0.000000e+00 1.000000e+00
21      endloop
22    endfacet
23    facet normal 1.000000e+00 1.000000e+00 1.000000e+00
24      outer loop
25        vertex 1.000000e+00 0.000000e+00 0.000000e+00
26        vertex 0.000000e+00 1.000000e+00 0.000000e+00
27        vertex 0.000000e+00 0.000000e+00 1.000000e+00
28      endloop
29    endfacet
30  endsolid Pyramid
```

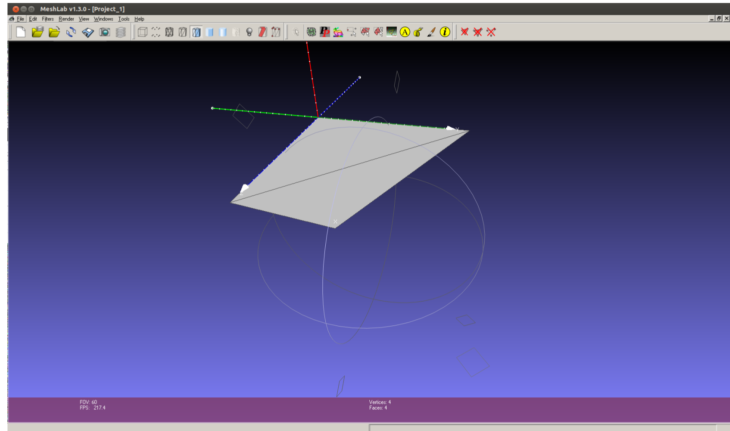



***Illustration 42: Une pyramide STL dans MESHLAB***

Outre sa simplicité, ce type de format est extrêmement pratique pour réaliser de la 3D avec Open Gl car ce dernier fonctionne lui aussi sur le principe de Vertex.

### 6.1.2 STL Binaire

Le format STL Binaire présente l'avantage de gérer en plus de la forme une couleur.

Ce format possède également une entête permettant de renseigner des informations sur le fichier.

Si dans les faits, le STL binaire est probablement le plus répandu, il n'est cependant pas le plus simple à ouvrir.

Voici comment est formé un fichier STL Binaire:

>80 octets d'entête
>1 mot de 4 octets pour indiquer le nombre de triangles
>puis pour chaque triangle
    >>3 mots de 4 octets pour la normale
    >>3 mots de 4 octets pour le vertex 1
    >>3 mots de 4 octets pour le vertex 2
    >>3 mots de 4 octets pour le vertex 3
    >>2 octets pour la couleur (si utilisé)

Il n'y a aucun retour à la ligne dans le fichier. Les octets se suivent les uns à la suite des autres.

## 6.2 Open GL et PYTHON: les bases

Nous allons voir ici l'ensemble des bases nécessaires pour pratiquer l'**OpenGL**. Nous détaillerons autant que possible les principaux éléments indispensables.

### 6.2.1 Le repère XYZ

En image de synthèse, le **repère** XYZ n'est pas celui que nous avons l'habitude d'utiliser en physique. En effet, si nous positionnons X à l'horizontal de gauche à droite, et Y en vertical de bas en haut, z devrait venir vers nous.

En imagerie, la convention veut qu'en réalité z s'éloigne de nous. Il faut donc bien en prendre pleinement conscience afin de ne limiter les potentielles erreurs.

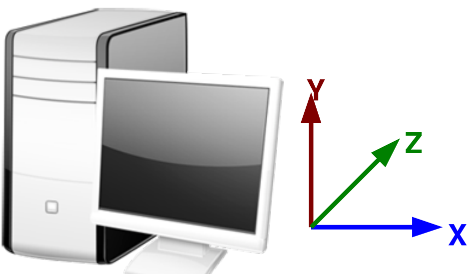



***Illustration 43: Le repère OpenGL***

En **OpenGL** (et en infographie en général), l'axe Z est également connecté à ce qu'on appelle communément le Z-Buffer.

Ce **Z-Buffer** permet au moteur 3D d'appréhender la notion de profondeur et évite donc des superposition hasardeuse.

L'exemple typique est un cube. Sans **Z-Buffer**, les faces sont affichées dans l'ordre ou vous les créer. Autrement dit, si vous

commencer par la face de devant et finissez par la face de derrière, la face de derrière apparaîtra en réalité devant la face avant.

Pour éviter ce quiproquo et ces interprétations hasardeuse, le **Z-Buffer** indique au moteur comment interpréter les instructions dans la globalité.

### 6.2.2 Notions de base

Dans cette partie, nous allons nous concentrer sur les éléments indispensables à l'utilisation courante **d'OpenGL**.

#### 6.2.2.1 Les vertex

Dans le monde de l'infographie 3D, un **vertex** (ou vertice), correspond simplement à un point dans l'espace.

Ce point possède des coordonnées qui lui sont propres et sert en général à créer un polygone 2D ou 3D dans l'espace.

**Vertex-Vertex Meshes (VV)**

Vertex List

| | | |
|---|---|---|
| v0 | 0,0,0 | v1 v5 v4 v3 v9 |
| v1 | 1,0,0 | v2 v6 v5 v0 v9 |
| v2 | 1,1,0 | v3 v7 v6 v1 v9 |
| v3 | 0,1,0 | v2 v6 v7 v4 v9 |
| v4 | 0,0,1 | v5 v0 v3 v7 v8 |
| v5 | 1,0,1 | v6 v1 v0 v4 v8 |
| v6 | 1,1,1 | v7 v2 v1 v5 v8 |
| v7 | 0,1,1 | v4 v3 v2 v6 v8 |
| v8 | .5,.5,1 | v4 v5 v6 v7 |
| v9 | .5,.5,0 | v0 v1 v2 v3 |

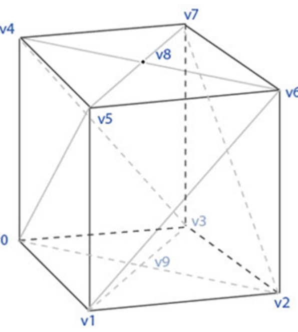



*Illustration 44: Exemple de vertex formant un cube (wiki. Commons, vertex-vertex meshes (VV).png)*

#### 6.2.2.2 Les polygones

Comme vu à l'instant, un **polygone** est définie par plusieurs vertex (un minimum de trois).

Cet assemblage de vertex permet de définir une forme dans l'espace. Chaque polygone peut lui-même être décomposé en une infinité de polygones plus petits. Cela peut parfois être utile, surtout lorsque l'on travaille sur des fichiers STL.

#### 6.2.2.3 Les formes de base

L'ajout d'une forme de base est relativement simple en **OpenGL**.

Une forme est décrite à l'intérieur d'un bloc **glBegin**()/**glEnd**(). Il faut un bloc par type de forme à afficher.

Au sein de chaque bloc, le type de forme est passé en paramètre au **glBegin()**. Les plus utiles sont les suivants:

| Constante | Description |
|---|---|
| **GL_POINTS** | Chaque vertex est un point unique |
| **GL_LINES** | Un ensemble de deux vertex forme une ligne. Fonctionne par groupe de 2 vertex |
| **GL_LINE_STRIP** | Une ligne est formée par deux vertex. Si trois vertex sont déclarés, cela forme deux lignes avec un point commun. Fonctionne par groupe de 2 vertex minimum. |
| **GL_TRIANGLES** | Trois vertex forment un triangle. Fonctionne par groupe de 3 vertex |
| **GL_QUADS** | Quatre vertex forment un carré. Fonctionne par groupe de 4 vertex |
| **GL_POLYGON** | Tous les vertex déclarés forment un polygone |

***Remarque: Pour le format STL, on fonctionnera avec le mode GL_TRIANGLES.***

Une fois le polygone déclaré, on le ferme avec la méthode **glEnd()**.

```
1 glBegin(GL_TRIANGLES)
2 …
3 glEnd()
```

***Attention: Un bloc glBegin/glEnd doit être déclaré pour chaque objet 3D que l'on désire afficher.***

#### 6.2.2.3.1 Les points

Élément de base, un point se déclare avec la fonction **glVertex2d**.

```
1 glBegin(GL_POINTS)
2 glVertex3d(x, y, z)
3 glEnd()
```

***Remarque: Les coordonnées d'un vertex sont des floats.***

Nous pouvons définir la taille d'un point de la manière qui suit:

```
1 glPointSize( 3.0 )       #Definition de la taille du point
```

#### 6.2.2.3.2 Les lignes

Si nous assemblons l'ensemble des éléments déjà vus, nous devinons aisément qu'une ligne se dessine grâce à un bloc du type:

```
1 glBegin(GL_LINES)
2 glVertex3d(x1, y1, z1)
3 glVertex3d(x2, y2, z2)
4 glEnd()
```

Nous pouvons définir la largeur d'une ligne de la manière qui suit:

```
1 glLineWidth( 3.0 )#Definition de la largeur du point
```

#### 6.2.2.3.3 Les triangles

Les triangles sont les éléments de bases pour un fichier STL. C'est pourquoi nous allons les voir ici.

Il n'y a rien de plus compliqué qu'avec les lignes. Un triangle se dessine avec le code suivant:

```
1 glBegin(GL_LINES)
2 glVertex3d(x1, y1, z1)
3 glVertex3d(x2, y2, z2)
4 glVertex3d(x3, y3, z3)
5 glEnd()
```

On peut jouer sur l'épaisseur des lignes constituant le périmètre du triangle, et sur la taille des vertex le définissant.

### 6.2.2.4 État de surface d'un polygone

#### 6.2.2.4.1 Les couleurs

Les couleurs, en **OpenGL**, fonctionnent en RGB. Une couleur peut être attribuée à chaque vertex, et **OpenGl** se charge alors de réaliser les dégradés de couleurs pour une surface formée par plusieurs vertex.

La couleur se configure avec **glColor3ub**

```
1 glColor3ub(R, G, B)
```

Comme on peut le constater dans notre exemple, on commence par définir la couleur, puis les vertex associés à cette couleur.

#### 6.2.2.4.2 Les textures

Il est pratique de pouvoir coloriser nos formes 3D, mais il peut être encore plus pratique de pouvoir utiliser des textures.

Nous allons voir ici comment utiliser des textures basiques. Entendez par là qu'il ne s'agit que d'une simple image en PNG.

Elles seront configurées pour s'adapter à la forme.

La première chose à savoir gérer est l'activation/désactivation de la texture. Cela s'effectue avec **glEnable**(**GL_TEXTURE_2D**) et **glDisable**(**GL_TEXTURE_2D**).

Il faut ensuite importer notre texture depuis notre image avec **gluint** ma_texture = loadTexture("ma_texture.png")

Enfin, il ne reste plus qu'à l'appliquer où on le désire.

***Attention: Une seule texture peut être appliquée par bloc. Il faut donc changer de bloc à chaque forme si la texture est différente.***

L'application de la texture sur notre objet est un peu plus complexe. Il faut tout d'abord préciser à **OpenGL** que nous désirons utiliser une texture sur notre bloc avec:

```
1 glBindTexture(GL_TEXTURE_2D, ma_texture)
```

Il faut ensuite simplement faire correspondre des coordonnées de la texture à des coordonnées du polygone grâce à **glTexCoord2d()**

Cela implique que vous connaissiez précisément les vertex de la texture et du polygone.

```
1 glBindTexture(GL_TEXTURE_2D, ma_texture)
2 glBegin(GL_TRIANGLES)
3 glTexCoord2d(texture_x1, texture_y1)
4 glVertex3d(x1, y1, z1)
5 glTexCoord2d(texture_x2, texture_y2)
6 glVertex3d(x2, y2, z2)
7 glTexCoord2d(texture_x3, texture_y3)
8 glVertex3d(x3, y3, z3)
9 glEnd()
```

***Remarque: Concernant la texture, vous êtes libre de prendre les coordonnées que vous désirez dans l'image. Quoiqu'il arrive, OpenGL déformera l'image pour s'adapter à votre demande.***

#### 6.2.2.5 La caméra

Au sein d'une application de visualisation 3D, la gestion de la caméra est un élément très important.

C'est par son intermédiaire que l'utilisateur visualise la scène 3D. Aussi de mauvais paramétrages, selon l'usage final de l'application, pourraient avoir de sérieuses répercutions.

##### 6.2.2.5.1 Les différents types de caméra

Il existe différents types de caméra pour visualiser une scène. Les principaux sont: **freefly** et **trackball**

La différence intervient lorsque l'on se déplace au sein d'une scène 3D.

L'exemple le plus typique est celui d'un simple cube. Lorsque nous nous déplaçons autour, celui-ci peut correspondre à deux mouvements différents:

=>L'objet tourne sur lui-même autour d'un axe et la caméra ne bouge pas (caméra trackball)

=>L'objet reste fixe et c'est la caméra qui tourne autour du cube (caméra freefly)

Selon le type de caméra utilisée, effectuer une rotation de l'objet selon un axe défini n'aura pas du tout le même rendu visuel à l'écran. Si on ne prend pas garde, non seulement l'objet en 3D va tourner, mais également les axes.

De fait, demander une rotation de l'objet selon un axe qui n'est plus là où on l'attend provoque un résultat surprenant.

Par défaut, c'est généralement le mode trackball qui est utilisé. Le Mode freefly se configure par l'intermédiaire du repositionnement de la caméra (voir point suivant).

L'utilisation du mode Freefly est recommandée pour avoir un mode de visualisation de type **ARCBALL**.

Il s'agit d'un type d'interaction programme/utilisateur, dans lequel, le programme fait généralement apparaître (mais pas nécessairement) trois cercles à l'utilisateur autour de l'objet. Le but est d'obtenir une utilisation intuitive.

Ainsi lorsque l'utilisateur déplacera son curseur vers la droite de l'écran, l'objet tourne vers la gauche (ou vers la droite selon la configuration souhaitée).

Dans les faits, réaliser une telle interface par des outils de rotation en OpenGL est complexe. C'est pourquoi il est plutôt recommandé de positionner la caméra autour de l'objet.

##### 6.2.2.5.2 Positionnement

Le **positionnement de la caméra** est important pour la visualisation d'une scène 3D. Cela n'est plus à démontrer.

```
gluLookAt(x1, y1, z1, x2, y2, z2, vx, vy, vz)
```

Le premier jeu de coordonnées correspond au point où sera fixée la caméra.

Le second jeu de coordonnées correspond au point visé par la caméra.

***Attention: N'oubliez pas que sur les axes des objets, l'axe Z est inversé. Z2 est donc en réalité -z2.***

Le vecteur fourni, enfin, indique l'inclinaison de la caméra. Par exemple, dans un plan XYZ, ou Z, positif, représente la verticale, on aura Vxyz(0,0,1). Si on veut regarder la scène à l'envers, on aura Vxyz(0,0,-1).

Ce **gluLookAt** s'appelle dans le DrawGLScene, juste après la réinitialisation de l'affichage, et juste avant de redessiner les objets dans la scène.

##### 6.2.2.5.3 Zoom, translation, et rotation

Le zoom peut s'effectuer de deux façons possibles: via le **gluPerpective** ou via le **glTranslatef**.

```
1 gluPerspective(fovy,ratio,near,far)
```

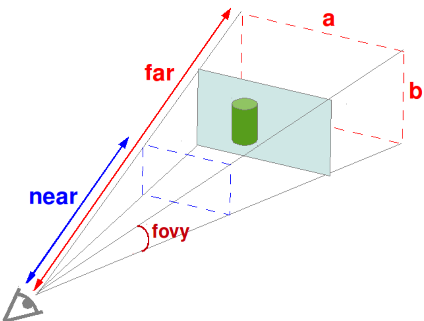



***Illustration 45: Fonctionnement du gluPerspective***

**fovy** est l'angle d'ouverture de la caméra, autrement dit l'angle de vision. En général, on saisit 45 (pour 45°). Plus cet angle sera petit, plus l'impression de zoom sera important.

**ratio** correspond au rapport a/b. En général ce ratio vaut 4/3 ou 16/9 selon les écrans.

**near** et far définissent les limites de vision. En deçà et au delà de ces distances, aucun objet n'apparaîtra à l'écran. Essayer les valeurs 0.1 et 100 pour commencer.

Le **glTranslatef** est à double usage. En effet, si on fixe la caméra dans l'alignement d'un axe, on peut s'en servir comme outil de zoom.

Cela n'est cependant pas son usage initial. Le **glTranslatef** permet d'effectuer des translations selon un vecteur passé en paramètre:

```
1 glTranslatef(X,Y,Z)
```

Concernant le zoom, si on est aligné sur un axe, une translation positive ou négative aura un effet de ZoomIn/ZoomOut.

***Remarque: Avec le LookAt, nous pouvons également effectuer un zoom. Il suffit simplement dans ce cas précis de déplacer la caméra plus ou moins loin de notre cible.***

Enfin, la rotation. On utilise **glRotated**. Cette méthode prend en paramètre un angle en degré (sens trigonométrique) et un vecteur définissant l'axe de rotation.

```
glRotated(alpha, X,Y,Z)
```

***Attention: Lorsque vous combinez des opérations de rotation/ translation le résultat peut changer selon l'ordre d'exécution.***

## 6.2.3 Fenêtre

### 6.2.3.1 Les indispensables

#### 6.2.3.1.1 Le constructeur de la classe

Il ne s'agit pas ici d'un indispensable réel mais plus d'une recommandation.

Pour nous, cela constituera un moyen simple de tester le code que nous réaliserons.

Il prendra la forme suivante:

```
if __name__ == '__main__':
    try:
        GLU_VERSION_1_2
    except:
        print "Need GLU 1.2 to run this demo"
        sys.exit(1)
    main()
    glutMainLoop()
```

Ce code ne doit normalement plus vous poser de questions importantes. Le constructeur est exécuté dès que nous lançons le module en autonome.

Un test permet de s'assurer de la bonne version minimale de **GLUT**. Cas échéant, un message d'erreur est affiché et nous sortons en erreur.

Sinon nous appelons notre main.

##### 6.2.3.1.2 Le main

Le main contient l'ensemble des éléments qui nous permettront de préciser à **GLUT** comment il doit démarrer et quelles sont les procédures/fonctions qui gèrent les différents évènements.

Un petit exemple:

```
1  def main():
2      global window
3      glutInit(sys.argv)
4      glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_ALPHA|GLUT_DEPTH)
5      glutInitWindowSize(640, 480)
6      glutInitWindowPosition(0, 0)
7      window = glutCreateWindow("Mon test a moi")
8      glutDisplayFunc(DrawGLScene)
9      glutReshapeFunc(ReSizeGLScene)
10     glutKeyboardFunc(keyPressed)
11     glutSpecialFunc(specialkey)
12     glutSpecialUpFunc(specialkey2)
13     glutMotionFunc(movemouse)
14     glutMouseFunc(buttmouse)
15     InitGL(640, 480)
```

Cet exemple ne sera pas commenté cette fois car nous allons voir chaque élément dans les points qui vont suivre.

#### 6.2.3.1.3 L'InitGL

L'**InitGL** est la fonction qui s'occupe, comme son nom l'indique, d'initialiser OpenGL, et plus précisément l'instance **d'OpenGL** à laquelle nous faisons appel.

Elle est appelée à la fin du main et prend en paramètres la largeur et la hauteur de la fenêtre souhaitée.

Elle va permettre de paramétrer un certain nombre d'éléments que nous allons voir ici. Ci-dessous une portion de code que nous allons détailler pour rendre plus concret cette fonction particulière:

```
1  def InitGL(Width, Height):
2      glClearColor(0.0, 0.0, 0.0, 0.0)
3      glClearDepth(1.0)
4      glDepthFunc(GL_LESS)
5
6      glEnable(GL_DEPTH_TEST)
7      glShadeModel(GL_SMOOTH)
8
9      glMatrixMode(GL_PROJECTION)
10     glLoadIdentity()
11     gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0)
12     glMatrixMode(GL_MODELVIEW)
```

Ligne 2, nous retrouvons une fonction qui nous permet de modifier la couleur du fond de la scène, ici à noir. Les paramètres sont respectivement R, G, B, alpha, compris entre 0 et 1

Ligne 3, nous réinitialisons le buffer de profondeur en lui donnant une valeur par défaut. Le buffer de profondeur est indispensable pour gérer les alignements de pièces dans l'espace.

Ligne 4, nous sommes toujours en train de configurer le buffer de profondeur. Nous lui indiquons que tout ce qui est inférieur à la valeur passée ligne 3, doit être pris en compte dans la gestion de la profondeur.

Ligne 6, cela concerne toujours le buffer de profondeur. Nous activons un lot de fonction de calcul GL.

Ligne 7, nous précisons à **OpenGL** que nous désirons que les faces d'un polygone doivent être pleines. Si on ne veut pas qu'elles soient remplies, il faut faire appel à **glShadeModel**(**GL_FLAT**).

Ligne 9, nous passons en mode projection. Cela correspond au **gluPerspective** vu précédemment. On va ainsi paramétrer la caméra. Nous réinitialisons le paramétrage ligne 10, et ligne 11 nous configurons la caméra.

La réinitialisation effectuée ligne 10 est extrêmement importante, car sans cela, les transformations et paramétrages s'accumulent et s'effectuent les uns après les autres. Il faut donc effectuer un **glLoadIdentity**() avant chaque nouveau jeu de transformation et/ou de paramétrage.

Ligne 12, enfin, nous repassons sur la gestion des objets mêmes.

##### 6.2.3.1.4 Le ReSizeGLScene

C'est la fonction qui est appelée lorsque l'on change la taille de la fenêtre afin de redessiner l'ensemble des éléments à la bonne échelle.

Dans l'exemple suivant nous détaillerons succinctement les différentes procédures qui seront examinées plus tard:

```
1  def ReSizeGLScene(Width, Height):
2      if Height == 0: #Eviter une division par 0.
3          Height = 1
4  
5      glViewport(0, 0, Width, Height)
6      glMatrixMode(GL_PROJECTION)
7      glLoadIdentity()
8      gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0)
9      glMatrixMode(GL_MODELVIEW)
```

Quelques explications. Tout d'abord on peut constater que les paramètres passés sont les nouvelles dimensions de la fenêtre.

En ligne 2 & 3, nous nous assurons d'avoir une hauteur minimale afin d'éviter toute erreur de type division par 0 ligne 8

En ligne 5, nous utilisons **glViewport()**. Cette procédure sert à définir la position de la fenêtre (coin supérieur gauche), ainsi que sa nouvelle taille.

En ligne 6, nous utilisons **glMatrixMode()**. Cette procédure permet de fixer la matrice devant accueillir la redéfinition du contenu graphique.

En ligne 7, **gLoadIdentitty()**. Cela permet de recharger la matrice dans la fenêtre pour redessiner les objets.

Ligne 8, **gluPerspective()**. Nous redéfinissons ici la position de la caméra. Un angle de vision de 45°, le ratio de la fenêtre, et pour finir les distances entre le devant et l'arrière de la scène.

Enfin, ligne 9, Nous recommutons le **glMatrixMode()**.

#### 6.2.3.1.5 Le DrawnGLScene

Le **DrawGLSCene** est la fonction qui s'occupe de dessiner l'ensemble de la partie 3D.

```
1  def DrawGLScene():
2      global rtri, rquad, yrtri, yrquad, zoom,dquad,dyquad, transf, \
3              astx, lasty, alpha, xc, yc, zc
4
5      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
6      glLoadIdentity()
7
8      gluLookAt(xc,yc,zc,xa,ya,za,vx,vy,vz)
9
10     glBegin(GL_QUADS)
11     glColor3f(0.0,1.0,0.0)
12     glVertex3f( 1.0, 1.0,-1.0)
13     glVertex3f(-1.0, 1.0,-1.0)
14     glVertex3f(-1.0, 1.0, 1.0)
15     glVertex3f( 1.0, 1.0, 1.0)
16
17     glColor3f(1.0,0.5,0.0)
```

```
18      glVertex3f( 1.0,-1.0, 1.0)
19      glVertex3f(-1.0,-1.0, 1.0)
20      glVertex3f(-1.0,-1.0,-1.0)
21      glVertex3f( 1.0,-1.0,-1.0)
22
23      glColor3f(1.0,0.0,0.0)
24      glVertex3f( 1.0, 1.0, 1.0)
25      glVertex3f(-1.0, 1.0, 1.0)
26      glVertex3f(-1.0,-1.0, 1.0)
27      glVertex3f( 1.0,-1.0, 1.0)
28
29      glColor3f(1.0,1.0,0.0)
30      glVertex3f( 1.0,-1.0,-1.0)
31      glVertex3f(-1.0,-1.0,-1.0)
32      glVertex3f(-1.0, 1.0,-1.0)
33      glVertex3f( 1.0, 1.0,-1.0)
34
35      glColor3f(0.0,0.0,1.0)
36      glVertex3f(-1.0, 1.0, 1.0)
37      glVertex3f(-1.0, 1.0,-1.0)
38      glVertex3f(-1.0,-1.0,-1.0)
39      glVertex3f(-1.0,-1.0, 1.0)
40
41      glColor3f(1.0,0.0,1.0)
42      glVertex3f( 1.0, 1.0,-1.0)
43      glVertex3f( 1.0, 1.0, 1.0)
44      glVertex3f( 1.0,-1.0, 1.0)
45      glVertex3f( 1.0,-1.0,-1.0)
46      glEnd()
47
48      time.sleep(0.002)
49      glutSwapBuffers()
```

Voici un exemple de DrawGLScene. Cette fonction est définie dans le main avec **glutDisplayFunc(DrawGLScene)**.

Ligne 5, nous appelons **glClear**. Cette fonction permet de réinitialiser les buffers passés en paramètres. Nous réinitialisons ici les buffers gérant la couleur et la profondeur.

Ligne 6, nous réinitialisons la scène.

Ligne 8, nous paramétrons la caméra. Nous fonctionnons ici en mode freefly. Aussi chaque paramètre du **glLookAt** est une variable.

Ligne 10 à 46, nous dessinons un cube.

La ligne 48 requiert une attention toute particulière. Cette fonction, déjà vue quelques chapitres plus tôt permet de forcer le programme à faire des pauses. Sans cela, si vous bougez votre cube, votre système tournera à 100% uniquement pour votre **OpenGL**.

En modulant cette valeur vous pouvez alléger la charge de votre processeur, dans une certaine limite. En effet, si vous montez trop en temps de pause, cela jouera sur la fluidité de votre programme.

Enfin, en ligne 49, nous mettons à jour l'affichage.

***Remarque: L'affichage en OpenGL est basé sur une série de buffers étant en réalité des matrices. Dans cet exemple, tant que nous n'avons pas appelé la ligne 49, l'affichage ne montre pas le cube que nous avons dessiné: nous n'avons rien à l'écran.***

### 6.2.3.2 Insertion dans une frame PYGTK

Il s'agit là d'un point très souvent difficile à appréhender, mais qui peut se révéler extrêmement pratique.

Sans insertion de la fenêtre **OpenGL** dans notre interface **GTK**, un programme peut vite se révéler pénible à utiliser.

Pour notre démonstration, nous utiliserons le module **pygtkglext**.

#### 6.2.3.2.1 Principe

Le principe ici est très simple avec le module sélectionné. Nous créons notre IHM **GTK** dans un coin, et notre code **OpenGL** dans un autre.

Pygtkglext va alors venir effectuer l'interface entre les deux en permettant à la fenêtre OpenGL d'être considérée comme un widget graphique à part entière par **PYGTK**

### 6.2.3.2.2 Mise en œuvre

Dans les faits, cela se complique légèrement par rapport à la théorie mais rien de sorcier non plus, je vous rassure.

Nous allons ensemble passer en revue le code ci-dessous:

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4  from OpenGL.GL import  *
5  from OpenGL.GLU import *
6  import gtk
7  import gtk.gdkgl as gdkgl
8  import gtk.gtkgl as gtkgl
9
10 def DrawGL (area, event):
11     zone = area.get_gl_drawable ()
12     gl_context = area.get_gl_context ()
13
14     if not zone.gl_begin (gl_context):
15         return False
16
17     allocation = area.get_allocation ()
18
19     glViewport (0, 0, allocation.width, allocation.height)
20     glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
21
22     glMatrixMode (GL_MODELVIEW)
23     glLoadIdentity ()
24
25     glMatrixMode (GL_PROJECTION)
26     glLoadIdentity ()
27     gluPerspective (45., float(allocation.width)/float(allocation.height), 2., 0.)
28     glTranslate (0, 0, -2)
29
30     glBegin (GL_TRIANGLES)
31     glColor3ub(0, 0, 255)
32     glVertex ( 0.5, -0.5, 0.0)
33     glColor3ub(0, 255, 0)
34     glVertex (0.0,  0.5, 0.0)
35     glColor3ub(255, 0, 0)
36     glVertex (-0.5, -0.5, 0.0)
37     glEnd ()
38
39     zone.swap_buffers ()
40
41     return True
42
```

```
43 if __name__ == '__main__':
44     window = gtk.Window()
45     window.set_title("Exemple: OpenGL dans PYGTK")
46     window.connect('destroy', gtk.main_quit)
47 
48     main_box = gtk.HBox(homogeneous=False, spacing=0)
49     window.add(main_box)
50 
51     config = gdkgl.Config(mode=(gdkgl.MODE_RGB|gdkgl.MODE_DOUBLE|\
52                                  gdkgl.MODE_DEPTH))
53     area = gtkgl.DrawingArea(config)
54     area.set_size_request(width=320,height=240)
55     area.connect('expose-event', DrawGL)
56     main_box.pack_start(area)
57 
58     area.show()
59     main_box.show()
60     window.show()
61     gtk.main()
```

Les 8 premières lignes ne devraient pas vous poser de problèmes. On est sous Linux, et on importe les modules dont nous avons besoin.

Ligne 10, nous avons notre fonction DrawGL qui va se charger de dessiner un triangle. Nous reviendrons dessus plus tard.

Ligne 43, nous avons notre main.

Nous commençons pas créer une fenêtre **GTK**, et un conteneur Hbox.

C'est lignes 51 à 56, que les choses vont commencer à devenir intéressantes.

Tout d'abord, ligne 51, nous paramétrons le futur affichage **d'OpenGL**: Couleur RGB, double buffers, et test de profondeur pour **OpenGL**.

Ligne 53, nous créons une zone dédiée **OpenGL**, en lui passant la configuration souhaitée par paramètre.

Ligne 54, nous paramétrons la taille de base par défaut de notre zone spéciale **OpenGL**.

Ligne 55, nous appellerons la fonction DrawGL, à chaque événement (ouverture de l'application, click de souris, frappe au clavier, ...)

Ligne 56, nous ajoutons notre zone spéciale à notre conteneur.

Lignes 58 à 61 enfin, nous retrouvons le fonctionnement d'une fenêtre **GTK** normale.

Nous allons maintenant revenir un peu à la fonction DrawGL. Elle récupère deux arguments: la zone de dessin, et l'événement qui l'a appelée.

Dans notre cas, nous ne tiendrons pas compte de l'événement appelant dans la suite du programme.

Ligne 11, nous commençons par nous approprier la zone de dessin.

Ligne 12, nous récupérons le contexte de la zone **OpenGL**. Le contexte correspond à un objet **GTK** contenant certaines informations liées, entre autres, au drag & drop.

Ligne 14, nous interrogeons **OpenGL** pour savoir s'il est prêt à recevoir nos instructions.

Ligne 17, nous récupérons la taille de notre objet, comme vu dans le 5.1.3.13.1.

Lignes 19 et 20, nous définissons la position et la taille de la fenêtre.

Lignes 22, 23, et 25 à 28, nous paramétrons la matrice d'affichage **OpenGL**.

***Remarque: Les lignes 19 à 28 correspondent au ReSizeGLScene.***

Lignes 30 à 37, vous reconnaîtrez sûrement maintenant le code permettant de dessiner un triangle multicolore.

Lignes 39 à 41 enfin, nous commutons les buffers et retournons True pour valider que tout s'est bien déroulé.

### 6.2.4 Gestion de la souris

La déclaration des fonctions de gestion de la souris s'effectue dans le main.

#### 6.2.4.1 Les boutons et la molette

Pour gérer les boutons et la molette de la souris, nous utilisons la fonction **glutMouseFunc(ma_procédure)**.

La procédure est du type: **def ma_procédure(button, state,mouse_x,mouse_y):**

Nous voyons ici 4 paramètres. Le premier est le type de bouton concerné, le second est l'état du bouton puis le troisième et le quatrième sont les coordonnées du curseur au moment de la détection.

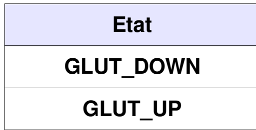
Les boutons sont prédéfinis et sont principalement:

| Constante |
|---|
| GLUT_LEFT_BUTTON |
| GLUT_RIGHT_BUTTON |
| GLUT_WHEEL_UP |
| GLUT_WHEEL_DOWN |

Les deux derniers concernent la molette. Ils sont à définir au début du programme:

```
GLUT_WHEEL_UP = Constant('GLUT_WHEEL_UP', 3)
GLUT_WHEEL_DOWN = Constant('GLUT_WHEEL_DOWN', 4)
```

Le paramètre state peut prendre lui deux états:



| Etat |
| --- |
| GLUT_DOWN |
| GLUT_UP |

```
def buttmouse(butt, state,x,y):
    global flag_mouse_left, flag_mouse_right, zoom

    if (butt==GLUT_LEFT_BUTTON) and (state==GLUT_DOWN):
        flag_mouse_left=1
    elif (butt==GLUT_LEFT_BUTTON) and (state==GLUT_UP):
        flag_mouse_left=0

    if (butt==GLUT_RIGHT_BUTTON) and (state==GLUT_DOWN):
        flag_mouse_right=1
    elif (butt==GLUT_RIGHT_BUTTON) and (state==GLUT_UP):
        flag_mouse_right=0

    if (butt == GLUT_WHEEL_UP): #gestion de la molette de la souris
        zoom=zoom+1
        DrawGLScene()

    if (butt == GLUT_WHEEL_DOWN):
        zoom=zoom-1
        if zoom < 0:
            zoom = 0
        DrawGLScene()
```

L'exemple ci-dessus montre la gestion des boutons et de la molette pour le zoom.

#### 6.2.4.2 Le deplacement de la souris

Le déplacement de la souris se gère simplement. Dans le main il faut utiliser une méthode permettant de définir quelle est la fonction s'occupant du mouvement de la souris:

```
glutMotionFunc(movemouse)
```

Cette fonction récupérera d'office les coordonnées X et Y de la souris sur l'écran.

La déclaration de la fonction devra donc être du type:

```
def movemouse(x,y):
```

Pour la suite la meilleure méthode consiste à comparer la position active du curseur à la précédente. Si cela dépasse un certain seuil, alors il faut effectuer une action, sinon ne rien faire.

En effet, tout mouvement de la souris, même d'un pixel activera cette fonction.

En général, au sein de cette fonction, on effectue des calculs de rotation et/ou de déplacement avant de rappeler le DrawGLScene.

### 6.2.5 Gestion du clavier

La déclaration des fonctions de gestion du clavier s'effectue dans le main.

#### 6.2.5.1 Touches ALT, SHIFT, CTRL

Ces trois touches sont prédéfinies dans **GLUT**. Ces définitions sont vraies quand la touche est pressée et maintenue:

| Constante |
|---|
| GLUT_ACTIVE_SHIFT |
| GLUT_ACTIVE_CTRL |
| GLUT_ACTIVE_ALT |

***Remarque: GLUT_ACTIVE_SHIFT est vrai également quand le verrouillage majuscule est actif. De plus si on appuie sur la touche***

***SHIFT alors que le verrouillage majuscule est actif, alors GLUT_ACTIVE_SHIFT sera faux***

#### 6.2.5.2 Touches alphanumériques

Les touches classiques du clavier se gèrent avec la fonction **glutKeyboardFunc(ma_procédure)** où ma_procédure représente la procédure que nous définissons pour gérer les touches du clavier.

Cette procédure aura la forme suivante: **def ma_fonction(*args):**

Le paramètre de cette procédure permettra de déterminer la touche appuyée. Ainsi pour savoir si nous avons appuyé sur la touche " a ", il suffit de comparer arg[0] avec 'a'.

```
1  def keyPressed(*args):
2      global zoom
3      # Si on appuie sur ESCAPE, alors on sort de l'application
4      if args[0] == ESCAPE:
5          glutDestroyWindow(window)
6          sys.exit()
7      elif args[0] == 'z':
8          zoom=zoom+2
9          if zoom>10:
10             zoom=2 #
11         DrawGLScene()
```

L'exemple ci-dessus montre par exemple la gestion de la touche **ESCAPE** et de la touche z pour gérer le zoom.

#### 6.2.5.3 Touches spéciales

Les touches spéciales sont principalement des touches prédéfinies et concernent surtout les touches telles que les Fx et les fléches de direction.

Pour les gérer nous utilisons la fonction **glutSpecialFunc(ma_procédure)** où ma_procédure représente la procédure qui sera appelée.

Cette procédure sera de la forme: **def ma_procédure(key, mouse_x, mouse_y):**

Trois paramètres sont passés à cette procédure. Le premier est la touche pressée, le second et troisième paramètres sont les coordonnées 2D de l'endroit où se trouvait la souris dans la fenêtre au moment ou la touche a été pressée.

Cette fonction détecte la pression de la touche. Il existe autrement la fonction **glutSpecialUpFunc(ma_procédure)** qui détecte elle le relâchement des touches spéciales.

***Remarque: La pression et le relâchement ne se distinguent que si l'on presse normalement une touche. En cas d'appui prolongé d'une touche, aucune différence n'est visible.***

Les touches spéciales prédéfinies sont les suivantes:

| Constante |
| --- |
| GLUT_KEY_F1 |
| GLUT_KEY_F2 |
| GLUT_KEY_F3 |
| GLUT_KEY_F4 |
| GLUT_KEY_F5 |
| GLUT_KEY_F6 |
| GLUT_KEY_F7 |
| GLUT_KEY_F8 |
| GLUT_KEY_F9 |
| GLUT_KEY_F10 |
| GLUT_KEY_F11 |
| GLUT_KEY_F12 |
| GLUT_KEY_LEFT |
| GLUT_KEY_UP |
| GLUT_KEY_RIGHT |

| GLUT_KEY_DOWN |
|---|
| GLUT_KEY_PAGE_UP |
| GLUT_KEY_PAGE_DOWN |
| GLUT_KEY_HOME |
| GLUT_KEY_END |
| GLUT_KEY_INSERT |

De plus, les touches escape, backspace et delete sont aussi des touches spéciales mais sont à déclarer au début du programme (non prédéfinies):

**ESCAPE = chr(27)**
**BACKSPACE = chr(8)**
**DELETE = chr(127)**

### 6.2.6 Exemple

Dans cet exemple tout simple, nous allons afficher un cube et gérer les divers événements liés.

Nous resterons volontairement basique au niveau du code, et ne respecterons pas la totalité des règles classiques afin que le code, déjà un peu complexe, soit accessible à tous.

```
1  #! /usr/bin/env PYTHON
2  # -*- coding: utf-8 -*-
3
4
5
6
7  from OpenGL.GL import *
8  from OpenGL.GLUT import *
9  from OpenGL.GLU import *
10 import sys
11 import time
12
13
14
15
```

```
16  ESCAPE = chr(27)
17  TURN= '\060'
18  zz=chr(122) #z
19  GLUT_WHEEL_UP = Constant('GLUT_WHEEL_UP', 3)
20  GLUT_WHEEL_DOWN = Constant('GLUT_WHEEL_DOWN', 4)
21
22  window = 0
23  flag=0
24
25  zoom=10.0
26  xc=2.
27  yc=2.
28  zc=2.
29
30
31
32
33  def InitGL(Width, Height):
34      glClearColor(1.0, 1.0, 1.0, 0.0)
35      glClearDepth(1.0)
36      glDepthFunc(GL_LESS)
37      glEnable(GL_DEPTH_TEST)
38      glShadeModel(GL_SMOOTH)
39
40      glMatrixMode(GL_PROJECTION)
41      glLoadIdentity()
42
43      gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0)
44
45      glMatrixMode(GL_MODELVIEW)
46
47
48  def ReSizeGLScene(Width, Height):
49      if Height == 0:
50          Height = 1
51
52      glViewport(0, 0, Width, Height)
53      glMatrixMode(GL_PROJECTION)
54      glLoadIdentity()
55      gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0)
56      glMatrixMode(GL_MODELVIEW)
57
58
59
60
61  def DrawGLScene():
62      global zoom, xc, yc, zc
63
64      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
65      glLoadIdentity()
```

```
66
67       gluLookAt(zoom*xc,zoom*yc,zoom*zc,0,0,0,0,0,1)
68
69       glBegin(GL_QUADS)
70
71       glColor3f(0.0,1.0,0.0)
72       glVertex3f( 1.0, 1.0,-1.0)
73       glVertex3f(-1.0, 1.0,-1.0)
74       glVertex3f(-1.0, 1.0, 1.0)
75       glVertex3f( 1.0, 1.0, 1.0)
76
77       glColor3f(1.0,0.5,0.0)
78       glVertex3f( 1.0,-1.0, 1.0)
79       glVertex3f(-1.0,-1.0, 1.0)
80       glVertex3f(-1.0,-1.0,-1.0)
81       glVertex3f( 1.0,-1.0,-1.0)
82
83       glColor3f(1.0,0.0,0.0)
84       glVertex3f( 1.0, 1.0, 1.0)
85       glVertex3f(-1.0, 1.0, 1.0)
86       glVertex3f(-1.0,-1.0, 1.0)
87       glVertex3f( 1.0,-1.0, 1.0)
88
89       glColor3f(1.0,1.0,0.0)
90       glVertex3f( 1.0,-1.0,-1.0)
91       glVertex3f(-1.0,-1.0,-1.0)
92       glVertex3f(-1.0, 1.0,-1.0)
93       glVertex3f( 1.0, 1.0,-1.0)
94
95       glColor3f(0.0,0.0,1.0)
96       glVertex3f(-1.0, 1.0, 1.0)
97       glVertex3f(-1.0, 1.0,-1.0)
98       glVertex3f(-1.0,-1.0,-1.0)
99       glVertex3f(-1.0,-1.0, 1.0)
100
101      glColor3f(1.0,0.0,1.0)
102      glVertex3f( 1.0, 1.0,-1.0)
103      glVertex3f( 1.0, 1.0, 1.0)
104      glVertex3f( 1.0,-1.0, 1.0)
105      glVertex3f( 1.0,-1.0,-1.0)
106      glEnd()
107
108      time.sleep(0.002)
109
110      glutSwapBuffers()
111
112
113
114
115  def buttmouse(butt, state,x,y):
```

```
116     global zoom
117
118     if (butt == GLUT_WHEEL_UP):
119         zoom=zoom+1
120         DrawGLScene()
121
122     if (butt == GLUT_WHEEL_DOWN):
123         zoom=zoom-1
124         if zoom < 1:
125             zoom = 1
126         DrawGLScene()
127
128
129
130
131 def main():
132     global window
133
134     glutInit(sys.argv)
135
136     glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_ALPHA | GLUT_DEPTH)
137
138     glutInitWindowSize(640, 480)
139
140     glutInitWindowPosition(0, 0)
141
142     window = glutCreateWindow("Mon test a moi")
143
144     glutDisplayFunc(DrawGLScene)
145
146     glutReshapeFunc(ReSizeGLScene)
147
148     glutMouseFunc(buttmouse)
149
150     InitGL(640, 480)
151
152
153
154
155 if __name__ == '__main__':
156     try:
157         GLU_VERSION_1_2
158     except:
159         print "Need GLU 1.2 to run this demo"
160         sys.exit(1)
161     main()
162     glutMainLoop()
```

## 6.3 Concepts mathématiques 2D

Travailler dans un plan 3D peut aisément se rapporter à travailler sur 2 plans 2D. C'est pourquoi, avant de se lancer dans les plans 3D, il est impératif de maîtriser les bases des plans 2D.

Ainsi, par la suite, en cas de difficultés sur un plan 3D, il est aisé de repasser sur des plans 2D pour modifier la vision de notre problème.

### 6.3.1 Les plans 2D

Tout le monde connaît les **plans 2D**. Nombre de jeux vidéos et d'éléments dans la vie courante sont basés sur ce principe.

Dans les fait, un **plan 3D** (XYZ) n'est rien d'autre qu'un ensemble de 3 plans 2D (XY, XZ, YZ).

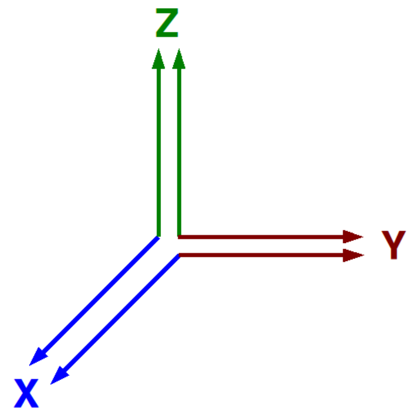



***Illustration 46: Assemblage des plans***

Il ne faut donc jamais perdre de vue cela, car un point en 3D, défini par ses coordonnées, peut de fait également être vu comme étant la fusion de 3 points de 3 plans 2D différents.

A(x,y,z) = fusion ( A(x,y), A(x,z), A(y,z) )

### 6.3.2 La trigonométrie

A partir du moment où l'on se met à manipuler des **objets 3D**, et dans la mesure ou un objet 3D est un assemblage d'**objets 2D**, la maîtrise de la **trigonométrie** de base est impérative.

Cela vous permettra de manipuler aisément (rotation par exemple) des objets en 3D.

#### 6.3.2.1 Cercle trigonométrique

Le cercle trigonométrique est la base de toute rotation dans l'espace.

La rotation en 3D fera l'objet d'un descriptif plus loin. Nous allons ici parler des rotations en 2D, servant de base pour la 3D.

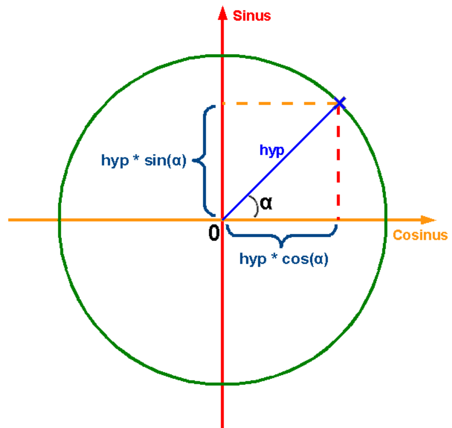



***Illustration 47: Le cercle trigonométrique***

Une rotation est simple à effectuer quand on maîtrise ce principe: les coordonnées de tout point tournant autour d'une origine définie sont inscrites sur un même cercle, le cercle trigonométrique.

Dans notre illustration, les axes cosinus et sinus pourraient être remplacés par les axes X et Y.

Ainsi, si l'on connaît les coordonnées d'un point, et que l'on sait que la pièce tourne autour de l'origine de x°, il faut suivre la procédure suivante:

1) A partir des coordonnées du point, détermination de l'angle initial
2) Addition de l'angle initial et de l'angle de rotation
3) Détermination des nouvelles coordonnées grâce aux cosinus et sinus (respectivement coordonnées en X et Y)

La seule chose à connaître est l'origine pour la rotation, qui est la plupart du temps simplement XY(0,0).

# 6.4 Concepts mathématiques 3D

## 6.4.1 Formules

### 6.4.1.1 Équation d'une droite en 3D

En 3D, une droite ne possède pas d'équation au sens propre de type " y = ax +b +c ". Elle est cependant définie par un ensemble de trois équations paramétriques:

$x = x_0 + \Phi tx$
$y = y_0 + \Phi ty$
$z = z_0 + \Phi tz$ ou $x_0,y_0,z_0$ sont les coordonnées d'un point de AB

Soit $A(x_1,y_1,z_1)$ et $B(x_2,y_2,z_2)$ les deux points définissant notre droite. Nous avons:

$tx = x_2 - x_1$
$ty = y_2 - y_1$
$tz = z_2 - z_1$

Par exemple: A(2,1,3) et B(4,-2,1)

$tx = 4 - 2 = 2$
$ty = -2 -1 = -3$
$tz = 1 - 3 = -2$

Les équations paramétriques de la droite AB sont donc:

$x = x_1 + 2\Phi$
$y = y_1 - 3\Phi$
$z = z_1 - 2\Phi$

### 6.4.1.2 Équation d'un plan en 3D

#### 6.4.1.2.1 Méthode 1

L'équation d'un plan 3D est de type ax + by + cz + d = 0

Un plan est défini par 3 vertex. On obtient donc:

L1 $ax_1 + by_1 + cz_1 + d = 0$ et $f_1 = \frac{x_2}{x_1}$

L2 $ax_2 + by_2 + cz_2 + d = 0$

L3 $ax_3 + by_3 + cz_3 + d = 0$ $f_2 = \frac{x_3}{x_1}$

Avec au final:

$$a = \frac{(-by_1 - cz_1 - d)}{x_1}$$

$$b = \frac{c*(z_2 - f_1*z_1) + d(1 - f_1)}{f_1*y_1 - y_2}$$

$$c = -d\left[\frac{(1-f_2)(f_1*y_1 - y_2) + (y_3 - f_2*y_1)(1-f_1)}{(z_3 - f_2*z_1)(f_1*y_1 - y_2) + (z_2 - f_1*z_1)(y_3 - f_2*y_1)}\right]$$

NB: prendre d = 1, puis calculer c, puis b, puis a

***Attention: Si $x_1 = 0$ cela pose problème pour $f_1$ et $f_2$. Échanger l'ordre des points pour avoir un $x_1 \neq 0$.***

Il faut également utiliser un algorithme similaire au suivant pour déterminer le signe de $f_1$ et $f_2$:

si $x_1 \neq 0$

  si $x_1 > 0$

    si $x_2 > 0$

      $f_1 = \frac{+x_2}{x_1}$

    sinon

      $f_1 = \frac{-x_2}{x_1}$

  sinon

    si $x_2 > 0$

      $f_1 = \frac{-x_2}{x_1}$

    sinon

      $f_1 = \frac{+x_2}{x_1}$

### 6.4.1.2.2 Méthode 2

La méthode 1 est à connaître, mais n'est cependant pas la plus simple. En effet, la plus simple est la méthode dite des produits mixtes qui nous fournit les équations suivantes:

a = $(Y_B - Y_A)(Z_C - Z_A) - (Z_B - Z_A)(Y_C - Y_A)$

b = $-((X_B - X_A)(Z_C - Z_A) - (Z_B - Z_A)(X_C - X_A))$

c = $(X_B - X_A)(Y_C - Y_A) - (Y_B - Y_A)(X_C - X_A)$

d = $-(a * X_A + b * Y_A + c * Z_A)$

z = $\dfrac{(-a * X - b * Y - d)}{c}$

### 6.4.1.2.3 Cas particuliers

Il existe quelques cas particuliers

>Plan parallèle à l'axe $\vec{Z}$ : ax + by + d = 0
>Plan parallèle à l'axe $\vec{Y}$ : ax + cz + d = 0
>Plan parallèle à l'axe $\vec{X}$ : by + cz + d = 0

De plus si l'origine est comprise dans le plan, alors cela simplifie encore les équations puisque d = 0.

Enfin, le calcul d'un plan 3D dont l'origine est dans le plan se fait avec:

a = $\dfrac{(y_2 * z_1 - y_1 * z_2)}{(x_2 * y_1 - y_2 * x_1)}$

b = $\dfrac{(-z_1 - a * x_1)}{y_1}$

c = 1

d = 0

### 6.4.1.3 Calcul de la normale au plan 3D

La **normale** au plan 3D est un vecteur perpendiculaire au plan, et servant dans les logiciels 3D principalement à indiquer comment réfléchir la lumière.

A partir d'un plan formé par les points $A(x_1,y_1,z_1)$, $B(x_2,y_2,z_2)$ et $C(x_3,y_3,z_3)$, le plan formé par ces trois points peut être défini selon les deux vecteurs $\vec{AB}$ $(x_2-x_1,y_2-y_1,z_2-z_1)$ et $\vec{AC}$ $(x_3-x_1,y_3-y_1,z_3-z_1)$.

La normale se calcule ainsi:
$Nx = (y_2 - y_1)*(z_3 - z_1) - (z_2 - z_1) * (y_3 - y_1)$
$Ny = (z_2 - z_1)*(x_3 - x_1) - (x_2 - x_1) * (x_3 - x_1)$
$Nz = (x_2 - x_1) * (y_3 - y_1) - (y_2 - y_1) * (x_3 - x_1)$

Avec la norme de la normale $\vec{N} = \sqrt{(x^2+y^2+z^2)}$

### 6.4.1.4 Calcul des coordonnées d'un point en 3D

Pour calculer les coordonnées d'un point en 3D, à partir d'une équation de plan auquel il appartient:

$$x = \frac{(-b*y-c*z-d)}{a}$$

$$y = \frac{(-a*x-c*z-d)}{b}$$

$$z = \frac{(-a*x-b*y-d)}{c}$$

A partir des équations paramétriques d'une droite:

$x = x0 + \Phi tx$
$y = y0 + \Phi ty$
$z = z0 + \Phi tz$

Il suffit juste de connaître x, y ou z afin de déterminer Φ. A partir de là, on peut trouver le reste des coordonnées du point.

#### 6.4.1.5 Rotation d'un point en 3D

Quand un point tourne en 3D, il peut tourner selon trois jeux d'axes: XY, XZ ou YZ.

Pour chacun de ces jeux, tout calcul n'est que trigonométrie de base constitué de sinus et de cosinus.

En résumé donc, une rotation 3D est simplement un assemblage de 3 rotations 2D.

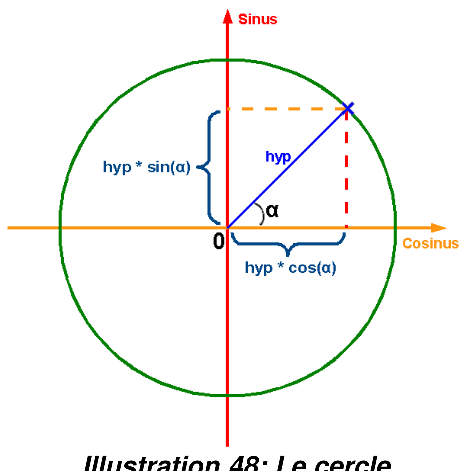



***Illustration 48: Le cercle trigonométrique***

Pour toute rotation en 3D, il faut donc pour chaque jeu d'axe, se rapporter à un plan 2D et simplement recalculer les coordonnées du point.

Vous connaissez systématiquement les coordonnées sur les deux axes. Vous pouvez donc déterminer la valeur de l'hypoténuse (h en pointillés sur l'illustration), puis par déduction la valeur de l'angle.

$$hyp^2 = \sqrt{a^2 + b^2} \quad \text{et} \quad \cos(\alpha) = \frac{b}{hyp}$$

Comme vous connaissez également la valeur angulaire de la rotation, vous pouvez donc en déduire les nouvelles coordonnées du point.

Pour une rotation de ϒ° :

$$\beta = \alpha + \Upsilon \quad \text{donc} \quad a = hyp * \sin(\beta) \quad \text{et} \quad b = hyp * \cos(\beta)$$

#### 6.4.1.6 Un point appartient-il à un triangle ?

Quand un point appartient à un triangle, quand nous parcourons les droites du triangle, ce point se trouve toujours du même côté des droites.

Si nous considérons notre triangle ABC, un point M se situe dans ce triangle si les $t_0$, $t_1$ et $t_2$ sont du même signe.

$t_0 = (x_2 - x_1) * (y_M - y_1) - (x_M - x_1) * (y_2 - y_1)$
$t_1 = (x_3 - x_2) * (y_M - y_2) - (x_M - x_2) * (y_3 - y_2)$
$t_2 = (x_1 - x_3) * (y_M - y_3) - (x_M - x_3) * (y_1 - y_3)$

## 6.5 Quelques techniques 3D

### 6.5.1 Depthmaps

Un **depthmaps** est une image totalement en nuances de gris ou chaque nuance de gris correspond à une profondeur différente. Le noir représente le 0 et le blanc le niveau le plus haut, ou l'inverse au choix.

Lorsque l'on désire traduire un depthmaps pour une machine, on précise alors la hauteur de l'objet final, ainsi que sa taille (hauteur et largeur).

Le logiciel se charge alors de calculer la hauteur correspondante à chaque nuance de gris, ainsi que la position de chaque pixel, puis effectue un lissage global pour obtenir un produit fini.

En absence de ce lissage, le produit final aurait un aspect relativement " pixellisé ".

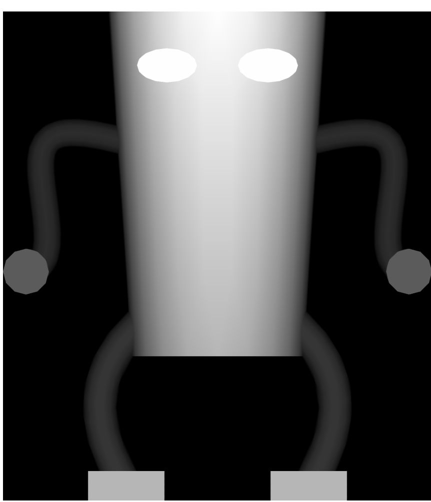


***Illustration 49: Un depthmaps issu d'un fichier STL***

### 6.5.2 Stéréogramme

Le terme de **stéréogramme** définit toute image qui permet à un être vivant, par n'importe quelle technique, de visualiser une image en 3D.

Les exemples les plus connus de stéréogrammes sont probablement les images en nuances de bleu et de rouge livrées avec une paire de lunettes adaptées.

Ces images sont conçues assez simplement sur le principe des yeux humains.

Prenez une feuille de papier et dessinez dessus un trait, puis sur ce trait placez deux points distancés de 10 cm.

Ensuite posez votre appareil photo sur le point de gauche et prenez une photo. Puis prenez une autre photo après avoir posé l'appareil sur le point de droite.

Vous obtenez deux photos. La première correspond à ce que voit normalement l’œil gauche humain et la seconde l’œil droit humain.

Via le traitement de ces deux images, vous êtes alors capable de concevoir un stéréogramme.

Avec un traitement informatique adapté vous pouvez même reconstituer un objet en 3D à partir de ces deux images.

### 6.5.3 Scanner 3D laser

Il est possible de scanner simplement un objet en 3D afin d'en déterminer son fichier STL équivalent par exemple.

Pour cela, il est nécessaire d'avoir un plateau tournant, une webcam, et une ligne laser.

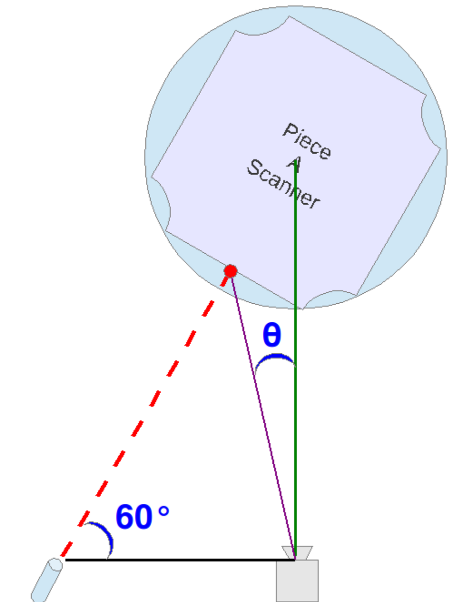



***Illustration 50: Principe du scanner laser 3D***

Le principe est le suivant: On connaît la distance entre la webcam et le centre du plateau tournant, de même que la distance entre la webcam et la ligne laser et enfin l'angle de la ligne laser, qui pointe vers le centre du plateau représentant l'origine.

Vu depuis la webcam, on constate que la ligne laser dessine le contour de l'objet. Si on fait tourner l'objet sur lui-même, on peut alors déterminer sa forme en 3D. Cela est encore plus flagrant dans le noir. En passant l'image en noir et blanc on obtient alors une ligne blanche sur fond noir.

Au niveau de l'analyse de la forme, cela est très simple. Notre webcam capture en réalité non pas un film mais une suite d'images. Ces images possèdent une certaine résolution. Il faut alors considérer cette résolution comme un quadrillage.

Si on aligne le centre de l'axe horizontal de la webcam avec le centre du plateau rotatif, alors chaque pixel, à droite ou à gauche, représente un certain angle. Il faut connaître ou mesurer l'angle de vision de la webcam pour déterminer la valeur angulaire d'un pixel.

Un pixel vaudra donc: $$pixel = \frac{angle\ vision\ webcam}{resolution\ native\ en\ X}$$

Le quadrillage vertical de pixels (en Y) permet donc de déterminer la forme de l'objet.

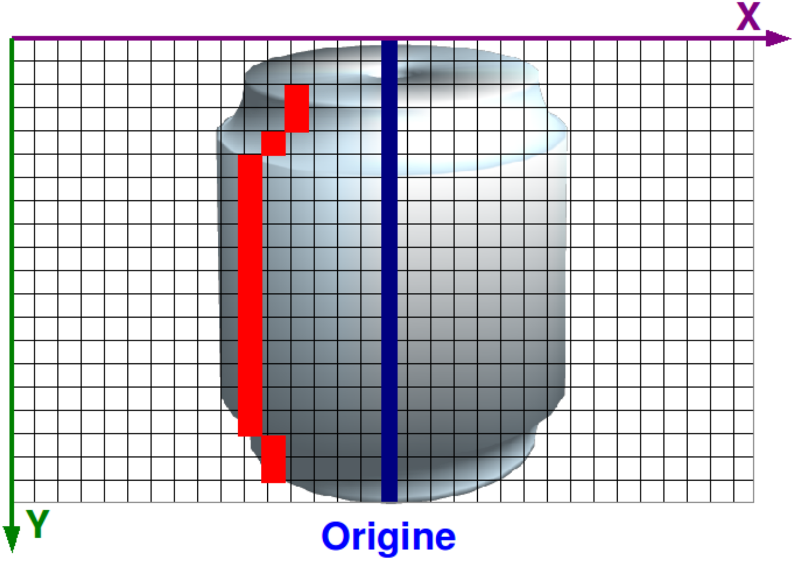
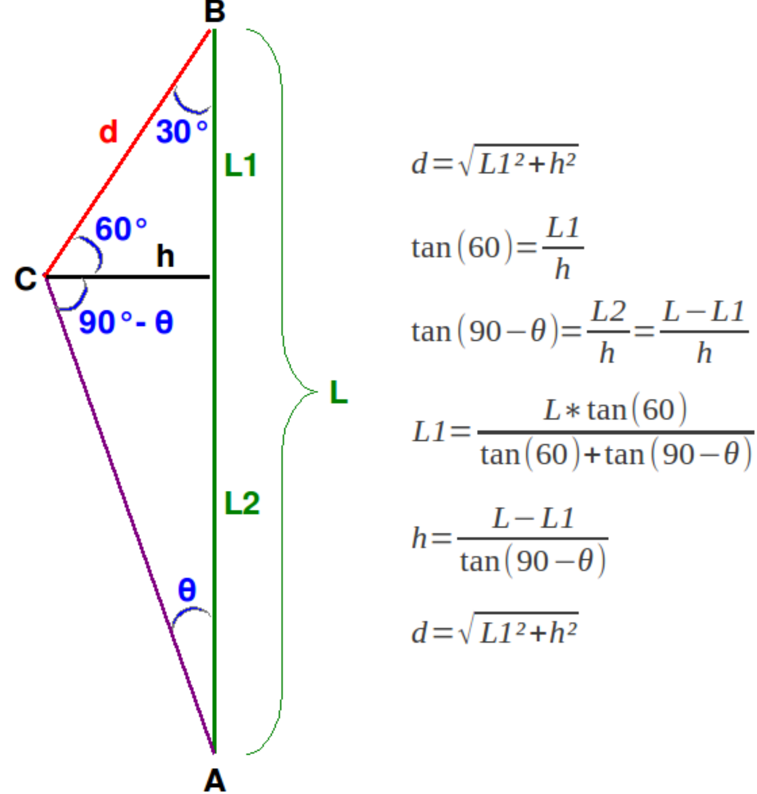



***Illustration 51: Matrice de points***

De fait, la précision d'un scanner d'un tel type dépend beaucoup de la qualité de la webcam. Plus la résolution sera haute, plus précis sera le scanner.

A partir de là, tout n'est alors que mathématique.




Le résultat final est un fichier de points 3D qui, via un logiciel type **MESHLAB**, ou un petit traitement maison, peut facilement devenir un fichier STL ou un ficher DEPTHMAPS. La référence de chaque valeur de d est le centre du plateau rotatif. Dans un plan 3D, cela correspondra à l'origine.

***Remarque: Pour avoir de bons résultats, il est préférable d'avoir une ligne laser de chaque côté de la webcam, de scanner l'objet dans un sens avec la première ligne laser, puis de le scanner dans l'autre sens avec la seconde ligne***

# 7 Gestion d'un projet

Wikimedia Commons,
Gnome-mime-application-x-gnumeric.png

Tout **projet** (**PJ**), quel qu'il soit, a besoin d'être bien géré pour aboutir.

Nous allons ici voir les principaux aspects de la gestion d'un projet afin de vous permettre, quel que soit votre objectif, de l'atteindre de manière la plus structurée possible.

## 7.1 Organisation

### 7.1.1 Description du PJ

La première chose à faire lorsque vous démarrez un projet est de réfléchir à ce que vous désirez vraiment. A ce stade il s'agit principalement d'une **description fonctionnelle**, mais néanmoins la plus détaillée possible.

Utilisez le document d'annexe pour vous aider:
" 01 - Specifications_Fonctionnelles_PYTHON "

Ce document est disponible également en téléchargement au format LibreOffice sur le site de l'auteur.

### 7.1.2 Structure du projet

Une fois l'étape de la description de votre projet effectuée, il faut réfléchir à la structure même de votre projet: regroupement de fonctions/procédures en packages et modules, répartition des différentes fonctions, les classes à créer, modules et/ou fonctions déjà existants, …

Pour résumer, il s'agit d'une **traduction fonctionnelle/technique** de l'étape précédente.

Utilisez le document d'annexe pour vous aider:
" 02 - Specifications_Techniques_PYTHON "

Ce document est disponible également en téléchargement au format LibreOffice sur le site de l'auteur.

L'expérience démontre qu'un projet mal décrit et/ou structuré peut aboutir, et ce même en fin de développement, à la réécriture partielle ou totale de code. De fait, il est important de passer du temps sur ces deux premières étapes afin de, par la suite, limiter les problèmes de développement.

### 7.1.3 Priorisation

Maintenant que vous avez terminé la description et la structure de votre projet, il va falloir prioriser les divers éléments.

En effet, tout comme dans le monde réel, il faut savoir dans quel ordre évoluer.

Pour prendre un exemple concret, imaginez un immeuble, où chaque étage représente un module différent. Il n'est pas vraiment recommandé de commencer par le module " Etage3 ". Il ne pourrait exister seul. Par contre, le module " Rez de chaussée ", lui, pourrait exister seul. Il s'agit de la brique de base.

Pour bien gérer un projet, il faut respecter la même logique et commencer par les briques de base avant d'évoluer progressivement vers les briques de sommet.

Bien entendu, il est possible pour certain cas particuliers de développer plusieurs modules en parallèle sans conflit ou problèmes particuliers.

Cet ordre que nous suivrons s'appelle la **priorisation**. Pour la symboliser, nous noterons, en commençant à 0, un indice de priorité dans le tableau précédent en face de chaque module, code, fonctions, …, que nous aurons à développer.

Utilisez le document d'annexe pour vous aider:
" 03 - Priorisation_PYTHON "

Ce document est disponible également en téléchargement au format LibreOffice sur le site de l'auteur.

### 7.1.4 Planification

Toutes les étapes précédentes passées, il ne vous reste plus qu'à **planifier votre développement** à l'aide de la documentation préalablement renseignée.

S'il est vrai que cela est indispensable dans le milieu professionnel, dans le cas d'un développement personnel, voir communautaire, cela peut parfois être moins significatif.

Quoiqu'il en soit, c'est une chose que je vous encourage à faire, même à titre privé, ne serait-ce que pour, à terme, avoir une vision générale du développement.

Pour planifier votre projet, le mieux est d'utiliser ce qu'on appelle un **diagramme de Gantt**. Il s'agit d'un graphique permettant visuellement d'identifier le temps alloué et passé à une tâche. Chaque tâche étant associée à une ressource, autrement dit un développeur, il est facile de suivre l'évolution d'un projet jour après jour.

De plus, chaque ressource pouvant être associée à un coût/heure ou coût/jour, un chef de projet, en entreprise par exemple, pourra rapidement avoir une vision globale du projet qu'il gère.

Dans notre cas, je vais vous présenter deux logiciels: **Planner**, qui est idéal pour les développements en solo ou en petite équipe, et **Calligraplan** qui s'adresse plutôt à des équipes de taille moyenne et importante.

#### 7.1.4.1 Planner

L'outil idéal pour un développeur solo, ou une petite équipe. Vous créez vos ressources, vos projets, vos tâches, et vous êtes prêt à planifier vos projets.

Cet outil se limite au strict minimum, mais le fait bien. Le strict nécessaire est suffisamment paramétrable pour un entrepreneur indépendant, et vous avez facilement l'ensemble des informations dont vous avez besoin, même si vous avez besoin de compiler l'ensemble des informations à la mano pour réaliser un rapport complet ou extrapoler les données brutes.

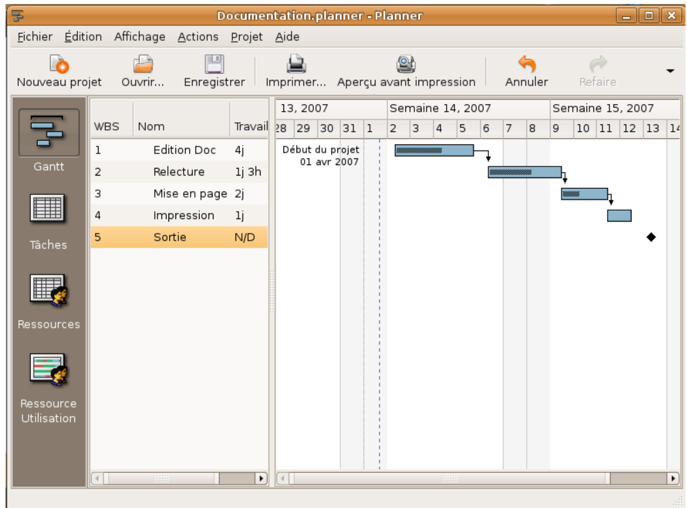



*Illustration 52: L'interface de Planner*

## 7.1.4.2 Calligraplan

Calligraplan est un logiciel similaire à Planner. Cependant, contrairement à ce dernier, il permettra de générer des synthèses, ce qui peut s'avérer très utile.

De plus, l'interface n'est pas la même. Si Planner est basé sur PYGTK, Calligraplan est lui basé sur KDE.

Enfin, comme précisé précédemment, le nombre de fonctionnalités avancées proposées par Calligraplan, rapport à celles de Planner, le prédestine plus à un usage de users avancés ou d'un groupe de développeurs un minimum élevé.

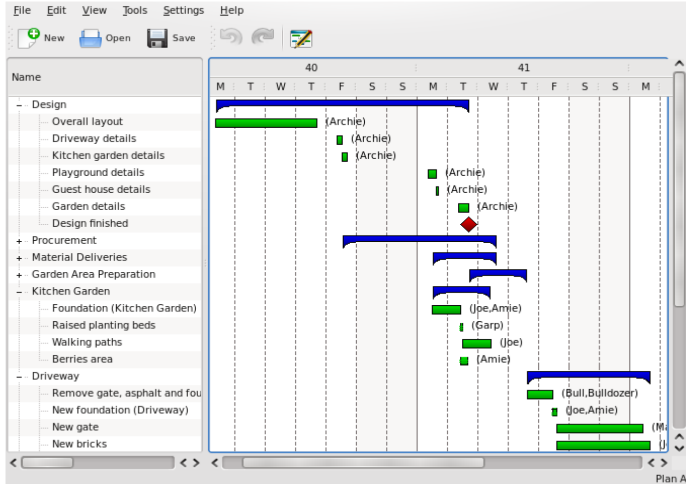



*Illustration 53: L'interface de Calligraplan*

## 7.2 Viabilité & pérennité

Lorsque vous développez, il faut penser en permanence à la **viabilité** et à la **pérennité de votre code**.

Par viabilité, il faut comprendre code simple, fonctionnel et aisément maintenable, et par pérennité, code aisément réutilisable.

Le manquement à ces deux principes peut compromettre un développement complet, une refonte, et/ou un colmatage de sécurité ou de fonctionnement.

### 7.2.1 Code

Un code doit en permanence être viable. Si vous codez n'importe comment, vous n'arriverez pas vous-même à vous replonger dans votre code quelques mois plus tard.

Pour cette raison, il est primordial de respecter les règles énoncées précédemment car la plupart des développeurs **PYTHON** les suivront.

C'est notamment ce respect du principe de code viable qui fait que l'on puisse autant échanger.

### 7.2.2 Tests

La partie des **tests** dans un projet est extrêmement importante. En effet, c'est cette dernière qui déterminera si votre code est conforme ou non à ce qui était attendu.

Dans les faits, on conseille souvent que les tests soient rédigés avant la réalisation, surtout si les testeurs et les développeurs sont les mêmes personnes.

Le but d'anticiper les tests est de ne pas être influencé par le code que l'on pourrait avoir développé ou participé à développer.

Les tests d'un projet peuvent se définir en trois phases: deux techniques visant à tester le code même, et une fonctionnelle.

#### 7.2.2.1 Test unitaire

Il s'agit du test de plus bas niveau, de la plus petite unité, autrement dit au plus proche du code.

Le test unitaire consiste à tester une fonction ou une procédure précise. Cela permet de s'assurer qu'une fois en cours d'exécution, chaque petite pièce de l'ensemble du code se comportera tel que désiré.

#### 7.2.2.2 Test d'intégration

Niveau supérieur au test unitaire, le test d'intégration vise à vérifier le bon fonctionnement d'un ensemble de fonctions/procédures/classes/..., et ce à différents niveaux.

Ainsi vous testerez d'abord un ensemble de fonctions/procédures d'une classe, puis un ensemble de classes, ..., jusqu'à votre logiciel complet.

Il ne s'agit pas de vérifier ici si en interne du code tout se déroule correctement, mais de s'assurer que la sortie est conforme à ce que l'on attend. Et cela même si en interne le fonctionnement n'est pas celui escompté.

En effet, cette dernière vérification est laissée au soin des tests unitaires.

#### 7.2.2.3 Test de validation ou fonctionnel

Le test de plus haut niveau. Ici, on se moque du code. Tout ce qui compte est le fonctionnement global de l'application. Il s'agit d'un test de fonctionnement d'un point de vue utilisateur.

Le logiciel réagit-il comme attendu? L'interface est-elle conforme? Les fichiers générés sont-ils au bon format?...

Toutes ces questions de " surface ", de fonctionnement, sont gérées par les tests fonctionnels.

#### 7.2.2.4 Construction d'un test

Chacun de ces tests se construit différemment. Le seul point commun est que, comme explicité au début, il faut écrire chaque test avant de concevoir. Cela vous aidera à coder au mieux et au plus proche du cahier des charges.

Ainsi, pour un test unitaire, vous devez décrire ce que vous attendez en entrée, ce que vous attendez en sortie, et le fonctionnement attendu en interne.

De même, pour un test d'intégration, vous devrez décrire les mêmes éléments: entrée, sortie, fonctionnement interne.

Enfin, pour le test fonctionnel, il faut purement vous appuyer sur le cahier des charges. Ce dernier est censé décrire de manière assez complète le côté fonctionnel attendu.

Une fois tout cela établi, le plus dur est fait. il ne vous reste plus éventuellement qu'à vous rédiger des petites fiches de tests et à passer ces derniers pour vérifier que tout est OK.

Vous trouverez en Annexe et sur le site de l'auteur un exemple de fiche pour chaque type de tests.

### 7.2.2.5 Logiciel dédié: Unittest

Côté **PYTHON**, vous avez la possibilité d'automatiser l'ensemble des tests techniques (unitaires et intégration).

Pour cela nous allons utiliser le module **UNITTEST**. Ce module vous permettra d'écrire un ou plusieurs modules dédiés.

Nous verrons ici comment utiliser les fonctions basiques de ce module.

#### 7.2.2.5.1 Fonctionnement de base

De quoi peut bien se composer un test? Et bien dans notre cas, ce dernier prendra l'apparence d'un module à part entière qui sera placé dans le même dossier (ou package) que notre code à tester.

Je vous conseille de nommer votre module de test d'un nom explicite; par exemple TestUnitaire_p_multiplication.py.

Le but est clairement que vous identifier d'un simple coup d'œil ce qui est testé dans ce module de test. N'oubliez pas qu'avec **PYTHON**, " Explicite est mieux qu'implicite ".

Une fois un nom explicite choisi, il ne reste plus qu'à remplir le module de test. Nous imaginerons ici que le module à tester s'appelle division, et que sa seule fonction interne s'appelle également division.

La première chose à faire est d'importer le module **Unittest**, puis de créer la boucle principale.

```
1  import unittest
2  …
3  if __name__ == "__main__":
4      unittest.main()
```

il faut ensuite importer le module que l'on désire tester.

```
1  from division import *
```

Enfin, il faut créer les tests mêmes. Cela passe par la création d'une classe, héritant du module Unittest.

```
1  class TestDivision(unittest.TestCase):
2      ...
```

Cette classe contiendra nos procédures de test.

```
1  def test_division(self):
2      retour = division(9,3)
3      self.assertEqual(retour, 3)
```

***Attention: La classe de test doit obligatoirement hériter de la classe unittest.TestCase. Par convention, son nom commencera toujours par Test. De même, toutes les procédures de tests devront commencer par test_ (attention à la casse). Cela permet à Unittest de savoir qu'il s'agit bien d'une procédure de test.***

#### 7.2.2.5.2 Possibilités offertes

Avant de voir plus en avant la réalisation d'un test, étudions ici les différentes possibilités.

A l'issue de l'exécution de votre portion de code, il vous faudra effectuer une opération de comparaison afin de déterminer si le fonctionnement est correct ou non.

Pour cela, il existe différentes opérations possibles.

La plus classique est le test d'égalité ou de presque égalité. Le mot clé diffère selon le type d'élément à comparer.

| Mot Clé | Type d'objet | Type d'égalité |
|---|---|---|
| **assertEqual** | Indifférent | = |
| **assertAlmostEqual** | indifférent | ≈ |
| **assertListEqual** | Liste | = |
| **assertTupleEqual** | Tuple | = |

| assertDictEqual | Dict | = |
|---|---|---|

La possibilité suivante est bien entendu, comme en mathématiques, le supérieur ou inférieur avec possibilité d'égalité.

| Mot Clé | opération |
|---|---|
| assertGreater | > |
| assertGreaterEqual | ≥ |
| assertLess | < |
| assertLessEqual | ≤ |

Voici pour les principaux types de comparaison. Bien entendu, il y en a d'autres. Je vous invite à étudier plus en détails la documentation officielle pour en apprendre plus.

#### 7.2.2.5.3 Test simple

Nous allons repartir sur l'exemple vu dans le fonctionnement de base. Imaginons que notre module " division " à tester soit le suivant.

```
4   def division(p1 = 0, p2 = 1):
5       if (p2 <> 0):
6           resultat = p1 / p2
7       else:
8           resultat = 0
9       return resultat
10
11  if __name__ = " __main__ ":
12      division()
```

Notre module de test ressemblerait alors, comme vu précédemment, à ceci.

```
1   import unittest
2   from division import *
3
4   class TestDivision(unittest.TestCase):
5       def test_division(self):
6           retour = division.division(9,3)
```

```
7         self.assertEqual(retour, 3)
8
9 if __name__ == "__main__":
10     unittest.main()
```

Comme nous pouvons le voir ici, ligne 7, nous testons le retour de la fonction division. Si le retour vaut bien 3, alors le test sera positif, négatif sinon.

Pour lancer le fichier de test, rien de plus simple. Dans un terminal, lancez-le comme n'importe quel fichier **PYTHON** (pour rappel: **PYTHON** <fichier.py>).

Vous pouvez également implémenter une redirection du résultat vers un fichier texte.

Pour votre information, vous verrez ci-dessous le résultat d'un test positif et d'un test négatif.

#### 7.2.2.5.4 Tests multiples

Maintenant que nous avons vu comment faire un test de base, nous allons voir comment les enchaîner.

En effet, il serait bien pénible de devoir les lancer un à un.

Eh bien, pour effectuer de multiples tests, il suffit simplement... de les coder les uns à la suite des autres. Cela n'est pas plus compliqué.

Chaque fonction de test définie dans la classe de tests sera exécutée, tout simplement.

### 7.2.3 Documentation

#### 7.2.3.1 Epydoc

Une bonne documentation est indispensable pour la pérennité de votre code. C'est elle qui vous permettra, plus tard, de réutiliser certain packages et/ou modules dans d'autres projets.

Il ne faut ainsi jamais se limiter à une vision à court terme, comme le font beaucoup de personnes et d'entreprises, mais au contraire, se projeter dans le temps et se demander comment rendre notre code le plus générique possible pour une future réutilisation.

Une fois cela fait, et vos docstrings réalisées clairement, vous avez fait 90% du travail. Les 10% restants seront assurés par un outil fort pratique nommé **EPYDOC**.

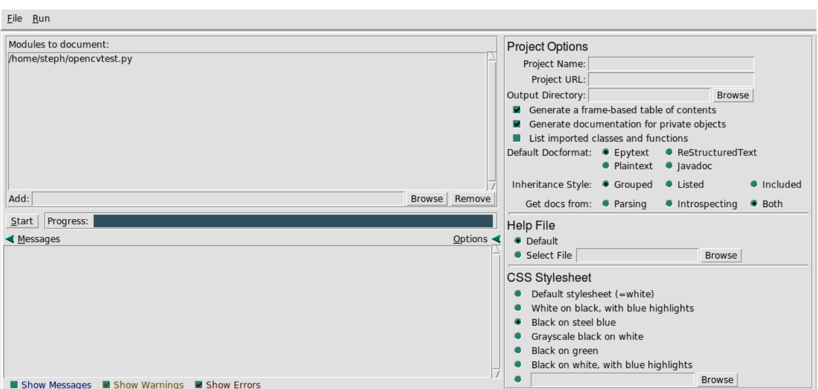



***Illustration 54: L'interface d'EPYDOCGUI***

**EPYDOC** est très simple à utiliser, même si son interface, simple mais un peu brute, peut ne pas être attirante de prime abord. Une fois installé, il suffit d'ouvrir un terminal et de saisir " **epydocgui** ". Vous avez aussi la possibilité de vous créer un raccourci.

Dans la fenêtre qui s'ouvre, via le bouton BROWSE, il faut sélectionner vos fichiers à documenter, ou vos dossiers packages, voir le

dossier de votre projet. Une fois votre sélection validée, il ne reste plus qu'à cliquer sur START, et attendre que la boîte à messages vous indique que le travail soit terminé.

Vous trouverez alors au même niveau que vos fichiers/dossiers un dossier nommé HTML. Ce dossier contient un fichier index qu'il vous suffit de lancer pour avoir accès à la documentation complète, générée par **EPYDOC**.

***Attention: EPYDOC ne générera de la documentation pour vos fichiers .py qu'à la stricte condition qu'il n'utilise que des tabulations pour l'indentation.***

***Remarque: En cliquant sur option sur la fenêtre principale de EPYDOCGUI, vous avez la possibilité de changer le thème de la documentation (en bas à droite).***

### 7.2.4 Fichier de log

Un **fichier de log** est un fichier qui garde en mémoire l'activité du logiciel, traçant les opérations, mais également les erreurs.

En cas d'anomalie provoquant le crash de votre application, vous disposez ainsi des informations nécessaires au debuggage.

Si votre fichier de log est bien construit, vous aurez alors la classe, fonction, données, … ayant provoqué le crash.

A l'aide de ces informations vous serez alors capable de reproduire le crash, d'en déterminer la cause, et d'y remédier.

Les fichiers de log, quoique parfois considérés comme superflus sont réellement utiles et permettent d'améliorer en permanence les logiciel.

Il ne faut donc pas que vous hésitiez à les implémenter au sein de vos applications.

Je vous conseille d'ailleurs de créer une classe dédiée dont les principales procédures seront l'ouverture/création du fichier de log, l'écriture dans ce dernier, et la fermeture propre du fichier de log.

## 7.3 Versionning

Le **versionning** est une façon simple de différencier les différentes versions d'un logiciel.

Comme l'explique l'article dédié au " version d'un logiciel " de wikipedia, il existe différentes façons de référencer les versions d'un logiciels. Je vous invite à lire cet article très bien documenté.

Nous allons ici détailler une seule façon de faire, mais très classique, souvent appelée système de version XYZ.

Ce système fonctionne sur trois chiffres, chacun ayant sa propre signification: logiciel Version X.Y.Z

Z représente une correction de bug(s)
Y représente un ajout/suppression de fonctionnalité(s)
X représente une évolution majeure ou bien une refonte partielle voir totale du logiciel

En règle générale, on dit que X est la version majeure, Y la version mineure et Z la version de correction.

La numérotation commencera toujours à 0 et chaque évolution d'une version entraînera la remise à zéro des niveaux inférieurs. Nous passerons ainsi par exemple, de la version 1.10.5 à la version 2.0.0

De plus, il faut savoir que X ne vaudra 0 que pour des versions dites alpha ou bêta, c'est-à-dire des versions de logiciels toujours en développement, encore non abouties et non recommandées pour utilisations courantes.

La première version stable sera toujours la 1.0.0

***Remarque: Si une version X peut ne pas être compatible avec une version X-1, une version X.Y doit absolument être compatible avec une version X.Y-1. Si ce n'est pas le cas, alors on changera de version majeure X***

## 7.4 Les licences libres

Qu'il s'agisse de tutoriels, de notices, de documentations, ou des logiciels, le choix judicieux d'une licence adaptée est primordial.

En effet, le choix d'une **licence** et son application sur vos œuvres, même sur un simple tutoriel peut vous éviter certain soucis juridiques et même vous protéger contre des abus ou de potentielles accusations.

Nous verrons ici, la licence Creative Commons, parfaitement adaptée aux écrits relatifs à l'informatique (livres, tutoriels, ou documentations) et la licence GPL pour les logiciels.

### 7.4.1 Licences Logicielles

Il y a deux questions importantes à prendre en compte pour les licences logicielles:
=>Permettent-elles de faire des logiciels libres?
=>Obligent-elles la redistribution en licence libre?

***Remarque: On parle souvent de copyleft dans le cadre de la redistribution. D'après wikipedia, " Le copyleft est l'autorisation donnée par l'auteur d'un travail soumis au droit d'auteur (œuvre d'art, texte, programme informatique ou autre) d'utiliser, d'étudier, de modifier et de copier son œuvre, dans la mesure où cette autorisation reste préservée. "***

Outre ces questions, on regarde aussi les différences par rapport à la licence libre de référence: la GNU GPL V3

***Remarque: il existe une différence entre libre et open source. Un logiciel libre sous licence GPL oblige les contributeurs à rendre accessible leurs modifications. Un logiciel open source n'obligera pas forcément cette divulgation, telle que la licence Apache. Sur le fond, il s'agit surtout, avec le terme " Open source " de faire moins peur aux investisseurs qu'avec le terme " libre ". En effet, ce terme peut avoir***

***mauvaise réputation dans le milieu professionnel. Dans les faits, même s'il existe une petite différence, " Libre " ou " Open Source " peuvent être employés indifféremment.***

#### 7.4.1.1 GPL

La licence **GPL**, est directement opposée au copyright. Quand ce dernier interdit toute copie, modification et redistribution d'une œuvre, la GPL l'autorise.

Parmi les principales obligations imposées par la GPL, on retrouvera le libre accès aux codes sources et la conservation de la licence quelle que soit l'action effectuée vis-à-vis du code source (modification, redistribution, ...).

La GPL garantit 4 libertés aux utilisateurs:
=>La liberté d'exécuter le programme, pour tous les usages
=>La liberté d'étudier le fonctionnement du programme et de l'adapter à ses besoins
=>La liberté de redistribuer des copies du programme (ce qui implique la possibilité aussi bien de donner que de vendre des copies)
=>La liberté d'améliorer le programme et de distribuer ces améliorations au public, pour en faire profiter toute la communauté

De plus, même si l'erreur est souvent commise, il y a une différence entre gratuit et libre. En effet, gratuit concerne le tarif, le prix du logiciel, et libre concerne son code source.

De fait, un logiciel peut être libre mais néanmoins payant. La licence GPL permet donc à l'utilisateur de facturer des copies du logiciel, à la condition qu'il livre le code source.

Dans les faits, les logiciels libres sont la plupart du temps gratuits et on trouvera plutôt la facturation sur des services que sur les logiciels.

Pour placer votre œuvre sous licence GPL, il faut mettre le bloc suivant au début de chaque fichier source.

```
1   Nom de votre programme + ce qu'il fait>
2  Copyright (C) <année sur 4 chiffres> <Nom des auteurs>
3
4     This file is part of <Nom de votre programme>.
5
6     <Nom de votre programme> is free software: you can redistribute it and/or modify
7     it under the terms of the GNU General Public License as published by
8     the Free Software Foundation, either version 3 of the License, or
9     (at your option) any later version.
10
11    <Nom de votre programme> is distributed in the hope that it will be useful,
12    but WITHOUT ANY WARRANTY; without even the implied warranty of
13    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14    GNU General Public License for more details.
15
16    You should have received a copy of the GNU General Public License
17    along with <Nom de votre programme>.  If not, see <http://www.gnu.org/licenses/>
```

Bien entendu, pour **PYTHON**, il faut laisser en ligne 1 & 2 le chemin d'accès à l'interpréteur et le type d'encodage. On placera donc les termes de la licence GPL à partir de la ligne 4 (lignes 1 & 2 pour l'interpréteur et l'encodage, ligne 3 en saut de ligne).

Il ne faut pas oublier également de placer dans le dossier de votre logiciel un dossier contenant au minimum la licence GPL complète au format texte.

## 7.4.2 Licences Documentaires

### 7.4.2.1 Creative Commons

La licence Creative Commons est un ensemble de six licences distinctives permettant à des auteurs de mettre leurs œuvres à disposition des personnes tierces de façon simplifiée.

Dans le cas de l'informatique, plutôt appliquée aux documentations ou livres dédiés au sujet, ces licences CC présentent beaucoup d'avantages.

Tout d'abord, ces licences copyleft ont une vrai valeur légale. Ensuite, elles vous garantissent la paternité de vos œuvres. Enfin, vous restez propriétaire de vos œuvres.

Ces licences sont basées sur 4 éléments, dont l'association donne naissance à 6 licences différentes.

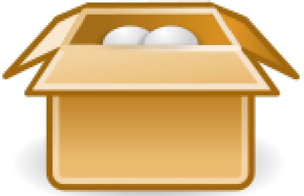
| Element | Nom | Description |
|---|---|---|
| **BY** | **Attribution** | Les tiers vous reconnaissent la paternité de l’œuvre. Lorsqu'ils rediffusent l’œuvre par leurs propres moyens, la paternité doit vous être attribuée. Vous êtes le seul propriétaire légitime de l’œuvre. |
| **NC** | **Pas d'utilisation commerciale** | Sauf à retenir l'option ND, vous autorisez la reproduction, la diffusion et la modification de vos œuvres à des fins exclusivement non commerciales. Vous êtes libre d'accorder une autorisation exceptionnelle sur demande. |
| **SA** | **Partage à l'identique** | Vous autorisez la reproduction, la diffusion et la modification de vos œuvres sous la condition que la nouvelle œuvre qui en découle utilise exactement la même licence Creative Commons. Vous êtes libre d’autoriser un changement de licence sur demande. |
| **ND** | **Pas de modification** | Vous autorisez la reproduction et la diffusion de votre œuvre stricto à l'identique. Vous êtes libre d'accorder une autorisation exceptionnelle sur demande |

Les licences disponibles sont les suivantes (définitions issues du site Creative Commons FR):

| Licence | Description |
|---|---|
| **BY** | Le titulaire des droits autorise toute exploitation de l’œuvre, y compris à des fins commerciales, ainsi que la création d’œuvres dérivées, dont la distribution est également autorisée sans restriction, à condition de l’attribuer à son auteur en citant son nom. Cette licence est recommandée pour la diffusion et l’utilisation maximale des œuvres. |
| **BY-ND** | Le titulaire des droits autorise toute utilisation de l’œuvre originale (y compris à des fins commerciales), mais n’autorise pas la création d’œuvres dérivées. |
| **BY-NC-ND** | Le titulaire des droits autorise l’utilisation de l’œuvre originale à des fins non commerciales, mais n’autorise pas la création d’œuvres dérivées. |
| **BY-NC** | Le titulaire des droits autorise l’exploitation de l’œuvre, ainsi que la création d’œuvres dérivées, à condition qu’il ne s’agisse pas d’une utilisation commerciale (les utilisations commerciales restant soumises |

| | |
|---|---|
| | à son autorisation). |
| **BY-NC-SA** | Le titulaire des droits autorise l'exploitation de l'œuvre originale à des fins non commerciales, ainsi que la création d'œuvres dérivées, à condition qu'elles soient distribuées sous une licence identique à celle qui régit l'œuvre originale. |
| **BY-SA** | Le titulaire des droits autorise toute utilisation de l'œuvre originale (y compris à des fins commerciales) ainsi que la création d'œuvres dérivées, à condition qu'elles soient distribuées sous une licence identique à celle qui régit l'œuvre originale. Cette licence est souvent comparée aux licences « copyleft » des logiciels libres. C'est la licence utilisée par Wikipedia. |

# 8 Déploiement



Wikimedia Commons,
Package-x-generic.svg

Le choix d'un bon **déploiement** est essentiel pour le succès d'une application.

En effet, les utilisateurs désirent plus que jamais de la simplicité, et on peut difficilement s'imaginer retourner en arrière au temps où, jadis, il fallait tout faire en ligne de commande.

Nous allons ici voir les paquets DEBIAN. Le choix de ce type de paquets plutôt qu'un autre tient en plusieurs réponses.

Tout d'abord, il est souvent considéré comme l'un des paquets les plus difficiles à générer, et comme dit l'adage, qui peut le plus peut le moins.

Ensuite, les désormais distributions connues Ubuntu, et Linux Mint sont basées sur la distribution DEBIAN.

Enfin, la distribution DEBIAN est très utilisée dans le monde industriel et professionnel.

## 8.1 Création d'un .deb Manuel

La création d'un package DEBIAN de base n'est pas aussi complexe qu'on peut le laisser entendre, spécialement, lorsque notre programme **PYTHON** consiste en un seul fichier.

Cette méthode est ici à titre informatif, car il existe désormais des méthodes plus simples pour créer nos paquets. Nous allons donc voir une création basique.

Si votre curiosité est plus forte, Internet vous fournira de plus amples informations.

Dans notre cas d'un seul et unique fichier, voici ce que l'on obtient comme structure pour un paquet DEBIAN.

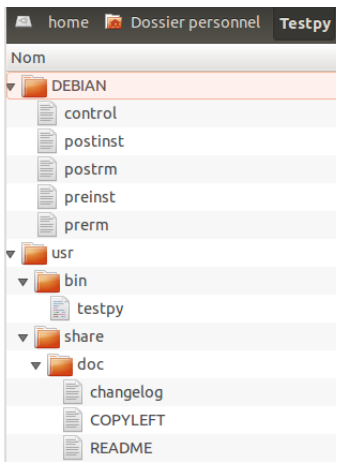



***Illustration 55: Composition d'un package DEBIAN***

***Remarque: Il faut scrupuleusement respecter la casse.***

Quelques explications seront, je le pense, les bienvenues.

Tout d'abord le dossier DEBIAN. Il contient les informations pour l'installation. Le fichier INDISPENSABLE: control.

Ce fichier possède la structure suivante:

```
control
1  Package: Nom_du_programme(ex:monSoftware)
2  Version: version_du_programme(ex:1.0.0)
3  Section: sections_ou_mettre_le_programme
4  Priority: optional
5  Architecture: all
6  Depends: les_dependances_de_votre_programme(ex:matplotlib (>= 1.2), opencv (>=2.0))
7  Maintainer: votre_nom <votre_mail>
8  Description: une_description_courte_de_votre_programme
9  Homepage: adresse_web_de_votre_site
```

***Illustration 56: Contenu du fichier CONTROL d'un paquet DEBIAN***

Vous pouvez recopier textuellement le contenu.

Vient ensuite les fichiers Preinst et Postinst, qui sont des shells permettant d'exécuter certaines actions avant et après l'installation. De même, Prerm et Postrm servent lors de la désinstallation.

Dans usr/bin, nous trouvons notre fichier **PYTHON**, sans extension. Cette absence d'extension permettra par la suite de n'avoir à saisir que le nom du programme pour qu'il se lance.

Dans usr/doc, nous trouvons quelques fichiers informatifs: README avec l'utilité qu'on lui connaît tous, changelog indiquant les évolutions des différentes versions, et COPYLEFT contenant la licence.

Une fois notre structure prête, il suffit de se placer là où se trouve le dossier (dans notre cas, /home/dossier personnel) puis d'exécuter la requête ***dpkg -b <nom_du_dossier>***

```
alex@alex-PC: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
alex@alex-PC:~$ pwd
/home/alex
alex@alex-PC:~$ dpkg -b Testpy
dpkg-deb : construction du paquet « testpy » dans « Testpy.deb ».
alex@alex-PC:~$
```

***Illustration 57: Génération d'un paquet DEBIAN via dpkg***

Le fichier .deb est alors créé dans votre dossier home. Vous pouvez l'installer comme bon vous semble. Ensuite un simple appel avec le nom du fichier contenu dans bin suffit à exécuter votre programme.

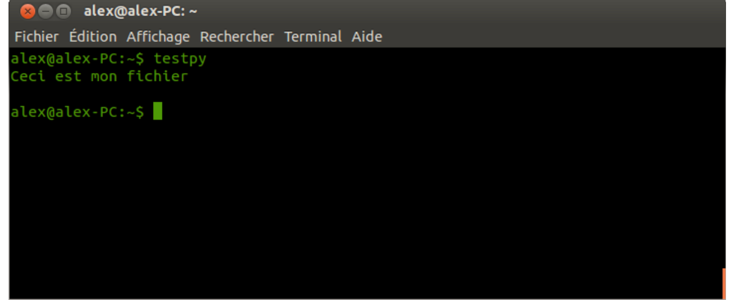



*Illustration 58: Test du soft une fois installé via le paquet DEBIAN*

## 8.2 Création d'un .DEB automatique

**DEBREATE** est un outil graphique permettant de créer simplement et facilement un paquet DEBIAN.

Divisé en 9 étapes, avec un passage simple de l'une à l'autre grâce aux flèches bleues en haut à droite, ce logiciel dispose d'une traduction disponible dans plusieurs langues, dont le français, le rendant ainsi particulièrement agréable d'utilisation, et rendant superflue l'utilisation des lignes de commande.

De plus, **DEBREATE** permet de renseigner la totalité des informations dans un paquet DEBIAN, ainsi que la structure finale. Bref, un outil plus qu'utile.

Voyons voir son fonctionnement plus en détail.

### 8.2.1.1 Accueil - Information

Sur cette première page, vous avez un petit descriptif du logiciel, ainsi qu'un lien vers une vidéo de démonstration (en anglais).

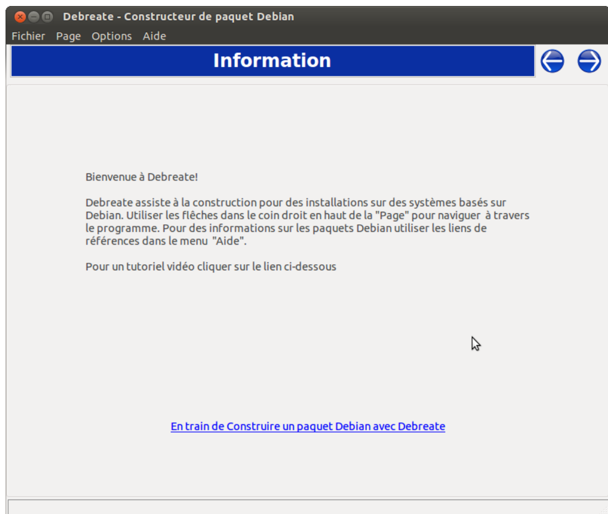



*Illustration 59: DEBREATE, Page d'acceuil*

### 8.2.1.2 Contrôle

Comme son nom l'indique, cette page va permettre de configurer notre fameux fichier *control* dans les moindres détails.

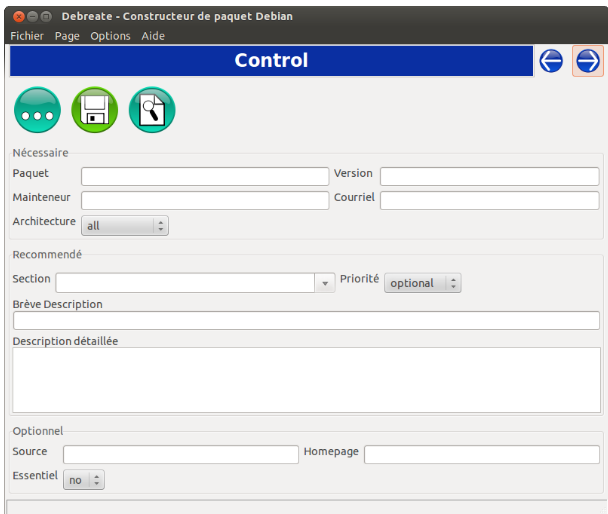



***Illustration 60: DEBREATE, Page de Control***

L'outil est relativement simple d'emploi et ne devrait pas vous poser de soucis particuliers.

Petite précision sur l'architecture: Cela est au cas ou vous concevriez une application spéciale pour système embarqué ARM par exemple. Dans la plupart des cas, on laissera à *all*.

De même, priorité sera en général laissée à *optionnel*. Cela sert à indiquer l'importance de notre logiciel par rapport à l'OS.

***Remarque: ESSENTIEL est à laisser à NO. En effet, s'il est déclarer en paquet essentiel vous rencontrerez des soucis de désinstallation.***

### 8.2.1.3 Dépendances et conflits

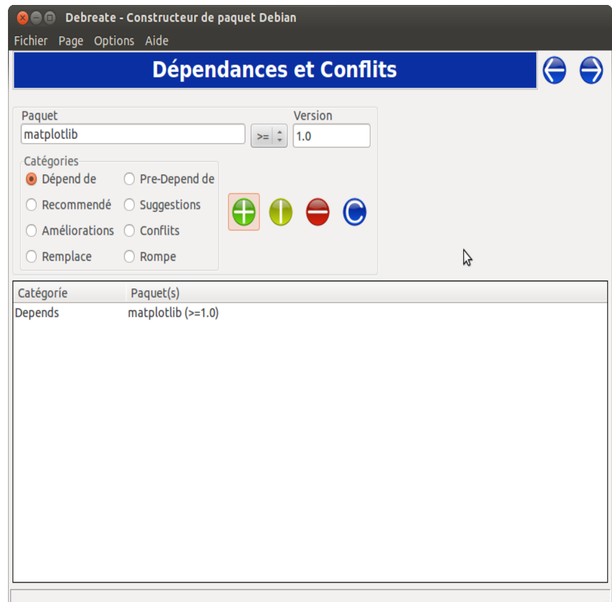



***Illustration 61: DEBREATE, Page des dépendances et conflits***

Ici non plus rien de sorcier. Vous indiquez le nom du paquet, puis sa version, sa catégorie rapport à votre logiciel, et enfin vous cliquez sur le bon bouton, et tout se fait tout seul.

***Remarque: Les bulles d'aides sont là pour vous aider, alors utilisez les.***

## 8.2.1.4 Création de la structure d'install

L'étape suivante est la création de la structure d'installation pour votre programme.

En effet, il se peut que vous ayez besoin d'avoir un sous-dossier images pour des boutons, dans le dossier de votre logiciel. C'est ici que tout va se passer.

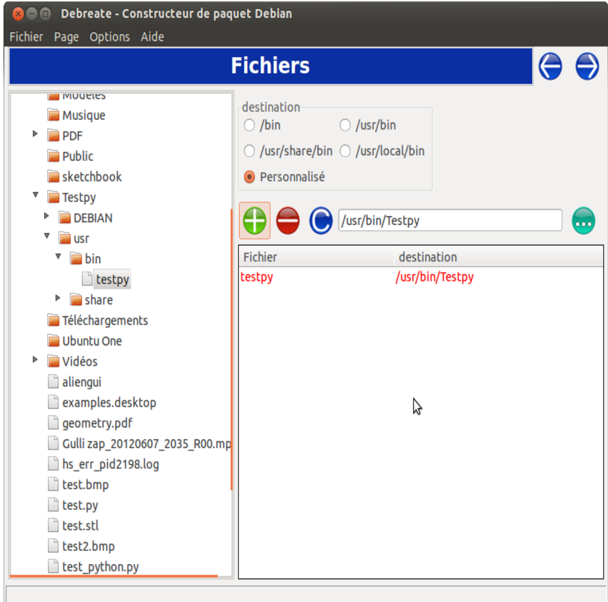



***Illustration 62: DEBREATE, Page de creation de structure***

On notera que le logiciel, pour simplifier la vie des cas les plus simples propose des emplacements pré-paramétrés.

#### 8.2.1.5 Scripts d'install et de désinstall

Nous arrivons maintenant aux scripts vus dans la création manuelle.

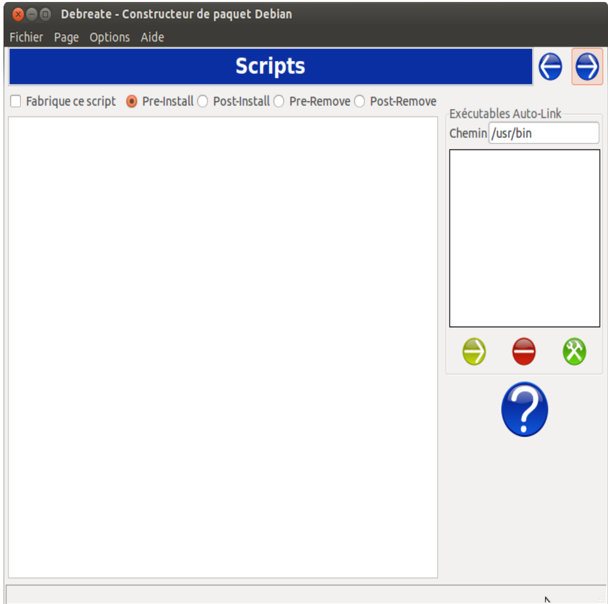



***Illustration 63: DEBREATE, Page de scripts***

Il faut ici utiliser la section à droite, et générer les scripts directement sur le moment.

Cliquez sur la flèche jaune, puis sélectionnez votre programme principal dans la liste et enfin cliquez sur le bouton vert. Vos scripts sont alors générés automatiquement

***Attention: Cette étape est importante pour avoir un lien dynamique. Sans ce lien, si vous saisissez le nom de votre programme dans un terminal, il ne sera pas lancé.***

### 8.2.1.6 Historique

La section historique, ou changelog. Il s'agit de tracer l'évolution du logiciel, afin de pouvoir suivre les différentes modifications, améliorations, ... du logiciel.

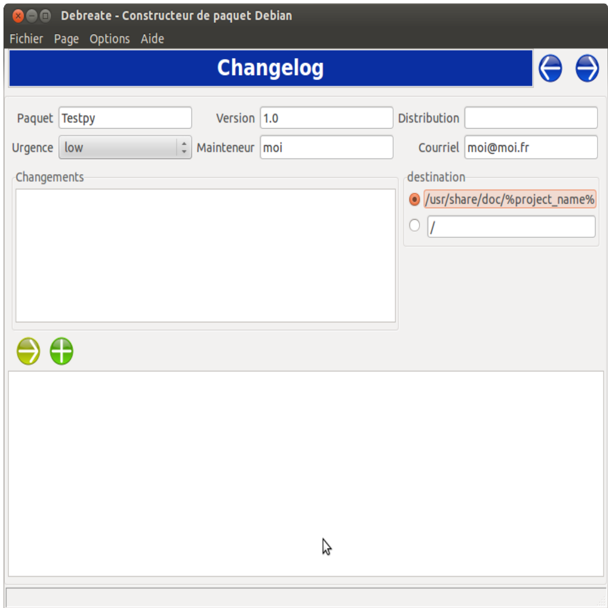



*Illustration 64: DEBEATE, Page d'historique*

A noter la petite flèche jaune ici qui permet d'importer directement les donner de la page de *Control*. Combiné avec la possibilité de sauvegarder le profil d'un logiciel à chaque fois, cela permet d’alimenter partiellement l'historique de manière systématique.

Votre tâche s'en trouve ainsi plus facile.

### 8.2.1.7 COPYRIGHT

Dans cette page, vous allez mettre le texte de la licence dont dépend votre logiciel. Je vous recommande l'excellente GPL.

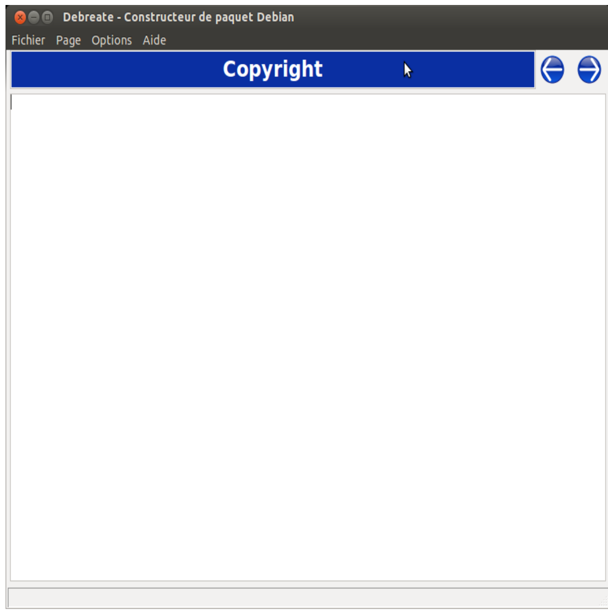



*Illustration 65: DEBREATE, Copyright*

### 8.2.1.8 Menu de lancement

Vient ensuite la création du menu

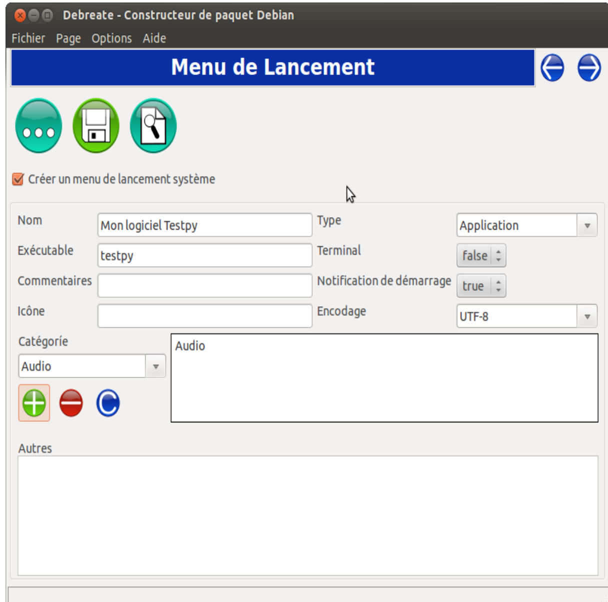



***Illustration 66: DEBREATE, Creation du menu***

C'est ici que l'on va paramétrer le lanceur de notre logiciel.

## 8.2.1.9 La création du .deb

La dernière étape. Il suffit juste de cliquer sur le bouton pour générer le .deb dans votre dossier personnel.

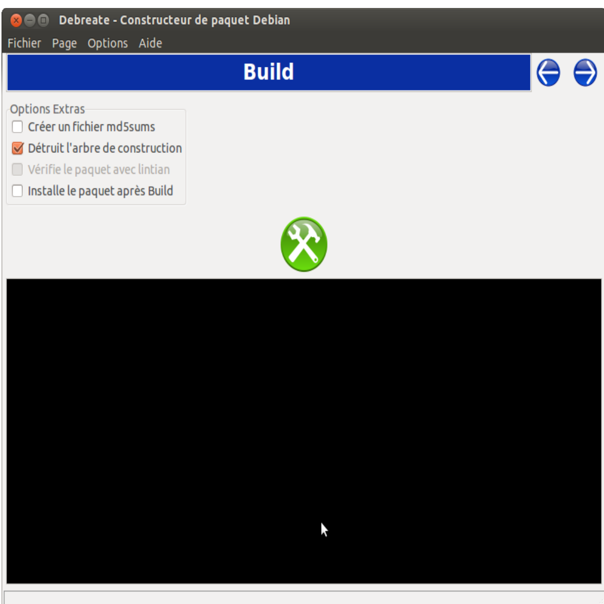



*Illustration 67: DEBREATE, La creation du .deb*

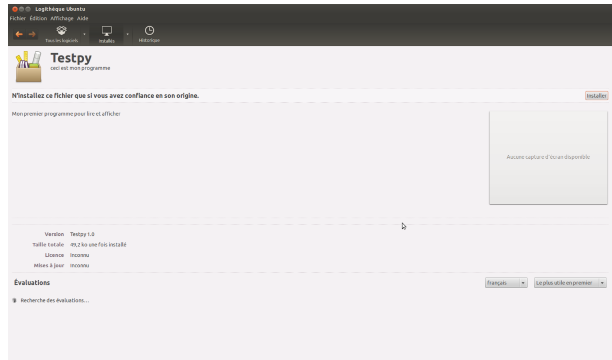



*Illustration 68: DEBREATE, Le .deb dans la logitheque*

## 8.3 Utilisation du logiciel Alien

Nous venons de voir comment créer un .deb permettant ainsi de diffuser simplement notre programme sur l'ensemble des distributions basées sur DEBIAN, tel Ubuntu ou encore LinuxMint, pour ne citer que les plus connues.

Cependant, le monde Linux est très vaste, et au delà de DEBIAN (.deb), il existe également des distributions basées sur Red Hat (.rpm), sur Slackware (.tgz), ou encore sur Solaris (.pkg) par exemple.

Bien que cette liste n'est pas exhaustive, elle couvre déjà un minimum de distribution. La plupart de ces distributions possèdent en effet un certain nombre de dérivées.

On pourrait ainsi se dire qu'utiliser un logiciel pour générer un paquet pour chacune de ces distributions résoudrait le problème.

Mais à chaque mise à jour, même mineure, vous passeriez un temps important pour régénérer les paquets et vous finiriez probablement par abandonner le support de certaine distribution.

Place donc à la solution alternative: **ALIEN**.

Il s'agit d'une commande permettant de réaliser des conversions de packages (et elle vous servira aussi parfois pour convertir un paquet n'étant pas destiné à votre distribution).

Ainsi, une fois notre .deb créé, il suffit d'y faire appel pour créer l'ensemble des paquets pour les autres distributions, permettant ainsi un déploiement simplifié sur les distributions Linux.

***Attention: Une conversion vers un autre système de paquets peut parfois être défectueuse. N'oubliez pas de tester votre paquet avant de le mettre à disposition.***

**Alien** étant un outil en ligne de commande, il est extrêmement simple d'emploi.

En effet, une fois installé sur le système, on se contentera des fonctionnalités de base.

Une fois **Alien** installé sur votre système donc, il suffit d'ouvrir un terminal et de vous placer dans le dossier ou se trouve le paquet à modifier.

Une fois cela fait, il ne reste plus qu'à appeler la commande alien, en précisant le type de conversion désirée et le nom du paquet à changer.

Les options sont les suivantes:

| Option | Description |
|---|---|
| **-d** | Transforme un paquet quelconque en paquet DEBIAN |
| **-r** | Transforme un paquet quelconque en paquet Red Hat |
| **-t** | Transforme un paquet quelconque en paquet Slackware |
| **-p** | Transforme un paquet quelconque en paquet Solaris |

Par exemple, pour convertir un paquet DEBIAN de votre création pour Red Hat, vous taperez dans votre terminal la commande suivante.

```
1 alien -r mon_paquet.deb
```

# 9 Ressources

Wikimedia Commons,
Book icon 1.png

| Langage | PYTHON |
|---|---|
| http://www.PYTHON.org/<br>http://docs.PYTHON.org/ | |

| Aide / Info | Developpez.com |
|---|---|
| http://PYTHON.developpez.com/ | |

| Aide / Info | Le site du zero |
|---|---|
| http://www.siteduzero.com/ | |

| Aide / Info | ODT |
|---|---|
| http://fr.wikipedia.org/wiki/OpenDocument | |

| Module | ezodf |
|---|---|
| http://pypi.PYTHON.org/pypi/ezodf/0.2.1 | |

| Module | FTPLib |
|---|---|
| http://docs.PYTHON.org/2/library/ftplib.html | |

| Module | GLUT |
|---|---|
| http://www.opengl.org/resources/libraries/glut/ | |

| Module | LXML |
|---|---|
| http://lxml.de/ | |

| Module | Math |
|---|---|
| http://docs.PYTHON.org/2/library/math.html | |

| Module | Matplotlib |
|---|---|
| http://matplotlib.org/ | |

| Module | Numpy |
|---|---|
| http://www.numpy.org/ | |

| Module | OpenCV |
|---|---|
| http://docs.opencv.org/ | |

| Module | OS |
|---|---|
| http://docs.PYTHON.org/2/library/os.html | |

| Module | PIL |
|---|---|
| http://www.PYTHONware.com/products/pil/ | |

| Module | PostGreSQL |
|---|---|
| http://www.postgresql.fr/ | |

| Module | PYGTK |
| --- | --- |
| http://www.PYGTK.org/docs/PYGTK/ | |

| Module | Pypi |
| --- | --- |
| http://pypi.PYTHON.org/pypi | |

| Module | Pyserial et Pyparallel |
| --- | --- |
| http://pyserial.sourceforge.net/<br>http://pyserial.sourceforge.net/pyparallel.html | |

| Module | ReportLab |
| --- | --- |
| http://www.reportlab.com/software/opensource/ | |

| Module | socket |
| --- | --- |
| http://docs.PYTHON.org/2/library/socket.html | |

| Module | SQLite |
| --- | --- |
| http://www.sqlite.org/ | |

| Module | SMTPLib |
| --- | --- |
| http://docs.PYTHON.org/2/library/smtplib.html | |

| Module | Sys |
|---|---|
| http://docs.PYTHON.org/2/library/sys.html | |

| Module | Thread |
|---|---|
| http://docs.PYTHON.org/2/library/threading.html | |

| Module | Time |
|---|---|
| http://docs.PYTHON.org/2/library/time.html | |

| Module | ZIPFile |
|---|---|
| http://docs.PYTHON.org/2/library/zipfile | |

| Livre | Apprendre à programmer avec PYTHON |
|---|---|
| De Gerard SWINNEN, ISBN-13: 978-2212134346 | |

| Livre | Apprenez à programmer en PYTHON |
|---|---|
| De Vincent Le Goff, ISBN-13: 979-1090085039 | |

| Logiciel | Debreate |
|---|---|
| http://debreate.sourceforge.net/?page=download | |

| Logiciel | Geany |
|---|---|
| http://www.geany.org/ | |

| Logiciel | Libre Office |
|---|---|
| http://fr.libreoffice.org/ | |

| Logiciel | Meshlab |
|---|---|
| http://meshlab.sourceforge.net/ | |

| Ressources | Lot d'icône Open Source |
|---|---|
| http://openiconlibrary.sourceforge.net/ | |

| Ressources | Wikimedia Commons |
|---|---|
| http://commons.wikimedia.org/wiki/Accueil | |

# 10 Annexes

Wikimedia Commons,
RedondoAdd.png

## 10.1 Spécifications Fonctionnelles

| SPECIFICATIONS FONCTIONNELLES | PJ PYTHON |
|---|---|
| PROJET: | |

| Société: | Date (début): | Date (fin): |
|---|---|---|
| AUTEUR(S): | | |

| Description du projet et du contexte |
|---|
| |

| <Société> | 1/2 |
|---|---|

| SPECIFICATIONS FONCTIONNELLES | PJ PYTHON |
|---|---|
| PROJET: | |

| Fonctionnalités souhaitée (Règles de Gestion) |
|---|
| F<N° sur 3 chiffres> - <Nom de la fonction><br><Description> |

## 10.2 Spécifications Techniques

| SPECIFICATIONS TECHNIQUES | PJ PYTHON |
|---|---|
| PROJET: | |

| Société: | Date (début): | Date (fin): |
|---|---|---|
| AUTEUR(S): | | |

| Ref. Fonctionnelle | Ref. Technique Associée | Existant? | Nom du composant si existant |
|---|---|---|---|
| Fxxx | Txxx | | |

<Société> 1/2

| SPECIFICATIONS TECHNIQUES | PJ PYTHON |
| --- | --- |
| PROJET: | |

| Description Technique |
| --- |
| T<N° sur 3 chiffres> - <Nom du composant><br><Description> |

# 10.3 Priorisation

| PRIORISATION | PJ PYTHON |
| --- | --- |
| PROJET: | |

| Société: | Date (début): | Date (fin): |
| --- | --- | --- |
| AUTEUR(S): | | |

| Priorisation des tâches | | | | |
| --- | --- | --- | --- | --- |
| Ref Technique | Charge Estimée | Jalon (Date limite) | Ressource | Indice de Priorité |
| Txxx | | | | |

<Société> 1/1

## 10.4 Tests Unitaires

| Tests Unitaires | PJ PYTHON |
|---|---|
| PROJET: | |

| Société: | Date: | Testeur: | Contrôleur: |
|---|---|---|---|
| | | | |

| Composant | Description du test | Résultat attendu | OK / KO | Capture Écran |
|---|---|---|---|---|
| - | - | 1) | 1) OKO | NO IMAGE AVAILABLE |
| - | - | 1) | 1) OKO | NO IMAGE AVAILABLE |
| - | - | 1) | 1) OKO | NO IMAGE AVAILABLE |

<Société> 1/*

## 10.5 Tests Intégrations

# Tests Intégration

PJ PYTHON

**PROJET:**

| Société: | Date: | Testeur: | Contrôleur: |
|---|---|---|---|
| | | | |

| Ref. Technique | Description du test | Résultat attendu | OK / KO | Capture Ecran |
|---|---|---|---|---|
| Txxx | - | 1: | 1) OKO | NO IMAGE AVAILABLE |
| Txxx | - | 1: | 1) OKO | NO IMAGE AVAILABLE |
| Txxx | - | 1: | 1) OKO | NO IMAGE AVAILABLE |

«Société» 1/*

## 10.6 Tests de Validation

| Tests Validation | PJ PYTHON |
|---|---|
| **PROJET:** | |

| Société: | Date: | Testeur: | Contrôleur: |
|---|---|---|---|
| | | | |

| Ref. Fonctionnelle | Description du test | Résultat attendu | OK / KO | Capture Ecran |
|---|---|---|---|---|
| Fxxx | - | 1) | 1) OKO | NO IMAGE AVAILABLE |
| Fxxx | - | 1) | 1) OKO | NO IMAGE AVAILABLE |
| Fxxx | - | 1) | 1) OKO | NO IMAGE AVAILABLE |

<Société> */*

## 10.7 Code ASCII

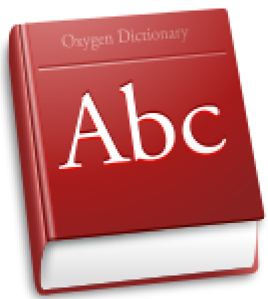
CODE ASCII

| Decimal | Hexa | ASCII | Decimal | Hexa | ASCII | Decimal | Hexa | ASCII | Decimal | Hexa | ASCII |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0x00 | ctrl+@ | 32 | 0x20 | Espace | 64 | 0x40 | @ | 96 | 0x60 | ` |
| 1 | 0x01 | ctrl+A | 33 | 0x21 | ! | 65 | 0x41 | A | 97 | 0x61 | a |
| 2 | 0x02 | ctrl+B | 34 | 0x22 | " | 66 | 0x42 | B | 98 | 0x62 | b |
| 3 | 0x03 | ctrl+C | 35 | 0x23 | # | 67 | 0x43 | C | 99 | 0x63 | c |
| 4 | 0x04 | ctrl+D | 36 | 0x24 | $ | 68 | 0x44 | D | 100 | 0x64 | d |
| 5 | 0x05 | ctrl+E | 37 | 0x25 | % | 69 | 0x45 | E | 101 | 0x65 | e |
| 6 | 0x06 | ctrl+F | 38 | 0x26 | & | 70 | 0x46 | F | 102 | 0x66 | f |
| 7 | 0x07 | ctrl+G | 39 | 0x27 | ' | 71 | 0x47 | G | 103 | 0x67 | g |
| 8 | 0x08 | ctrl+H | 40 | 0x28 | ( | 72 | 0x48 | H | 104 | 0x68 | h |
| 9 | 0x09 | ctrl+I | 41 | 0x29 | ) | 73 | 0x49 | I | 105 | 0x69 | i |
| 10 | 0x0A | ctrl+J | 42 | 0x2A | * | 74 | 0x4A | J | 106 | 0x6A | j |
| 11 | 0x0B | ctrl+K | 43 | 0x2B | + | 75 | 0x4B | K | 107 | 0x6B | k |
| 12 | 0x0C | ctrl+L | 44 | 0x2C | , | 76 | 0x4C | L | 108 | 0x6C | l |
| 13 | 0x0D | ctrl+M | 45 | 0x2D | - | 77 | 0x4D | M | 109 | 0x6D | m |
| 14 | 0x0E | ctrl+N | 46 | 0x2E | . | 78 | 0x4E | N | 110 | 0x6E | n |
| 15 | 0x0F | ctrl+O | 47 | 0x2F | / | 79 | 0x4F | O | 111 | 0x6F | o |
| 16 | 0x10 | ctrl+P | 48 | 0x30 | 0 | 80 | 0x50 | P | 112 | 0x70 | p |
| 17 | 0x11 | ctrl+Q | 49 | 0x31 | 1 | 81 | 0x51 | Q | 113 | 0x71 | q |
| 18 | 0x12 | ctrl+R | 50 | 0x32 | 2 | 82 | 0x52 | R | 114 | 0x72 | r |
| 19 | 0x13 | ctrl+S | 51 | 0x33 | 3 | 83 | 0x53 | S | 115 | 0x73 | s |
| 20 | 0x14 | ctrl+T | 52 | 0x34 | 4 | 84 | 0x54 | T | 116 | 0x74 | t |
| 21 | 0x15 | ctrl+U | 53 | 0x35 | 5 | 85 | 0x55 | U | 117 | 0x75 | u |
| 22 | 0x16 | ctrl+V | 54 | 0x36 | 6 | 86 | 0x56 | V | 118 | 0x76 | v |
| 23 | 0x17 | ctrl+W | 55 | 0x37 | 7 | 87 | 0x57 | W | 119 | 0x77 | w |
| 24 | 0x18 | ctrl+X | 56 | 0x38 | 8 | 88 | 0x58 | X | 120 | 0x78 | x |
| 25 | 0x19 | ctrl+Y | 57 | 0x39 | 9 | 89 | 0x59 | Y | 121 | 0x79 | y |
| 26 | 0x1A | ctrl+Z | 58 | 0x3A | : | 90 | 0x5A | Z | 122 | 0x7A | z |
| 27 | 0x1B | ctrl+[ | 59 | 0x3B | ; | 91 | 0x5B | [ | 123 | 0x7B | { |
| 28 | 0x1C | ctrl+\ | 60 | 0x3C | < | 92 | 0x5C | \ | 124 | 0x7C | \| |
| 29 | 0x1D | ctrl+] | 61 | 0x3D | = | 93 | 0x5D | ] | 125 | 0x7D | } |
| 30 | 0x1E | ctrl+^ | 62 | 0x3E | > | 94 | 0x5E | ^ | 126 | 0x7E | ~ |
| 31 | 0x1F | ctrl+_ | 63 | 0x3F | ? | 95 | 0x5F | _ | 127 | 0x7F | DELETE |

# 10.8 Nuancier RGB Hexa

| R/G/B | Teinte | R/G/B | Teinte | R/G/B | Teinte | R/G/B | Teinte | R/G/B | Teinte |
|---|---|---|---|---|---|---|---|---|---|
| 00 / 00 / 00 | | 40 / 00 / 00 | | 80 / 00 / 00 | | C0 / 00 / 00 | | FF / 00 / 00 | |
| 00 / 00 / 40 | | 40 / 00 / 40 | | 80 / 00 / 40 | | C0 / 00 / 40 | | FF / 00 / 40 | |
| 00 / 00 / 80 | | 40 / 00 / 80 | | 80 / 00 / 80 | | C0 / 00 / 80 | | FF / 00 / 80 | |
| 00 / 00 / C0 | | 40 / 00 / C0 | | 80 / 00 / C0 | | C0 / 00 / C0 | | FF / 00 / C0 | |
| 00 / 00 / FF | | 40 / 00 / FF | | 80 / 00 / FF | | C0 / 00 / FF | | FF / 00 / FF | |
| 00 / 40 / 00 | | 40 / 40 / 00 | | 80 / 40 / 00 | | C0 / 40 / 00 | | FF / 40 / 00 | |
| 00 / 40 / 40 | | 40 / 40 / 40 | | 80 / 40 / 40 | | C0 / 40 / 40 | | FF / 40 / 40 | |
| 00 / 40 / 80 | | 40 / 40 / 80 | | 80 / 40 / 80 | | C0 / 40 / 80 | | FF / 40 / 80 | |
| 00 / 40 / C0 | | 40 / 40 / C0 | | 80 / 40 / C0 | | C0 / 40 / C0 | | FF / 40 / C0 | |
| 00 / 40 / FF | | 40 / 40 / FF | | 80 / 40 / FF | | C0 / 40 / FF | | FF / 40 / FF | |
| 00 / 80 / 00 | | 40 / 80 / 00 | | 80 / 80 / 00 | | C0 / 80 / 00 | | FF / 80 / 00 | |
| 00 / 80 / 40 | | 40 / 80 / 40 | | 80 / 80 / 40 | | C0 / 80 / 40 | | FF / 80 / 40 | |
| 00 / 80 / 80 | | 40 / 80 / 80 | | 80 / 80 / 80 | | C0 / 80 / 80 | | FF / 80 / 80 | |
| 00 / 80 / C0 | | 40 / 80 / C0 | | 80 / 80 / C0 | | C0 / 80 / C0 | | FF / 80 / C0 | |
| 00 / 80 / FF | | 40 / 80 / FF | | 80 / 80 / FF | | C0 / 80 / FF | | FF / 80 / FF | |
| 00 / C0 / 00 | | 40 / C0 / 00 | | 80 / C0 / 00 | | C0 / C0 / 00 | | FF / C0 / 00 | |
| 00 / C0 / 40 | | 40 / C0 / 40 | | 80 / C0 / 40 | | C0 / C0 / 40 | | FF / C0 / 40 | |
| 00 / C0 / 80 | | 40 / C0 / 80 | | 80 / C0 / 80 | | C0 / C0 / 80 | | FF / C0 / 80 | |
| 00 / C0 / C0 | | 40 / C0 / C0 | | 80 / C0 / C0 | | C0 / C0 / C0 | | FF / C0 / C0 | |
| 00 / C0 / FF | | 40 / C0 / FF | | 80 / C0 / FF | | C0 / C0 / FF | | FF / C0 / FF | |
| 00 / FF / 00 | | 40 / FF / 00 | | 80 / FF / 00 | | C0 / FF / 00 | | FF / FF / 00 | |
| 00 / FF / 40 | | 40 / FF / 40 | | 80 / FF / 40 | | C0 / FF / 40 | | FF / FF / 40 | |
| 00 / FF / 80 | | 40 / FF / 80 | | 80 / FF / 80 | | C0 / FF / 80 | | FF / FF / 80 | |
| 00 / FF / C0 | | 40 / FF / C0 | | 80 / FF / C0 | | C0 / FF / C0 | | FF / FF / C0 | |
| 00 / FF / FF | | 40 / FF / FF | | 80 / FF / FF | | C0 / FF / FF | | FF / FF / FF | |

# 11 Index




Wikimedia Commons,
Oxygen480-apps-accessories-dictionary.svg

# A

Accesseurs....64, 68
Algorithmes Haar....207
ALIEN....392
And....56
ARCBALL....315
Arguments....121
Attributs....67

# B

Base de données....111
Booléens....42
Break....59
Byte Code....30

# C

Calligraplan....356
Cartouche....87
Chemin absolu....60
Chemin relatif....60
Classe....35, 66
Codecs....169
Commande....122
Continue....59
Courbes....103

# D

Date de référence UNIX....91
DEBREATE....382
Def....65
Déploiement....378
Depthmaps....346
Description fonctionnelle....354
Deserialisation....136
Diagramme de Gantt....356

Dictionnaire....47
Docstring....71, 83
DrawGLSCene....322

E

Encapsulation....37
Encodage....84
Enumerate....58
EPYDOC....366
ETREE....133
Exceptions....77
Expose-event....285
Expressions régulières....74
EZODF....141

F

Fichier de log....367
Float....43
Fonctions....65
For....58
Fovy....317
Freefly....315

G

GEANY....55
Getpass....53
GlBegin....311
GlColor3ub....313
GlDisable....314
GlEnable....314
GlEnd....311
GlLoadIdentity....321
Global....51
GlRotated....318
GlShadeModel....321
GlTranslatef....317

Gluint....................................................................................314
GluLookAt..............................................................................316
GluPerpective..........................................................................317
GluPerspective........................................................................321
GLUT.....................................................................................319
GlutMouseFunc.......................................................................328
GlVertex2d..............................................................................312
GPL.......................................................................................371
GTK.......................................................................................244
Guido VAN ROSSUM..................................................................31

# H

Help.........................................................................................83
Héritage.............................................................................38, 64

# I

If.............................................................................................57
IHM.......................................................................................244
Import......................................................................................64
InitGL....................................................................................320
Input.......................................................................................53
Integer.....................................................................................42
Interpréteur..............................................................................87
Is............................................................................................84
Is not.......................................................................................84

# L

Langages interprétés.................................................................30
Len..........................................................................................52
Licence..................................................................................370
Licences Documentaires..........................................................372
Licences Logicielles................................................................370
Liste........................................................................................46
Long........................................................................................42
LXML....................................................................................133

## M

Math....93
Matplotlib....103
METHODES....34
Méthodes....36
Module....70
Module re....75
Mutateurs....64, 68

## N

Near....317
Normale....343
Not....56
NumPy....93

## O

Objet....35, 73
Objets 2D....338
Objets 3D....338
ODF....138
Open Document Format....138
Open Source....116
OPEN SOURCE....27
OpenCV....165
OpenGL....309
Or....56
Os....121
OS....120

## P

Packages....71
PDF....216
PEP 20....80
PEP 8....81
Pérennité de votre code....359

PIL....................................................................................99
Pixbuf...............................................................................285
PJ....................................................................................354
Plan 3D.............................................................................337
Planifier votre développement.................................................356
Planner.............................................................................356
Plans 2D...........................................................................337
Polygone...........................................................................311
POO....................................................................................34
Port...................................................................................162
Port parallèle....................................................................160
Port série..........................................................................156
Portée des variables..............................................................51
Positionnement de la caméra..................................................316
PostGreSQL.......................................................................116
Print....................................................................................52
Priorisation.......................................................................355
Procédures..........................................................................65
Projet................................................................................354
Property...............................................................................70
PROPRIETES.......................................................................34
Propriétés............................................................................35
Pygtkglext..........................................................................324
Pyparallel...........................................................................160
Pypi......................................................................................90
Pyserial...............................................................................156

## Q

QT.....................................................................................244

## R

Raise....................................................................................77
Ratio..................................................................................317
Readline...............................................................................62
Readlines..............................................................................62
REGEX................................................................................74

Règles de codage....81
Repère....309
Reportlab....216
Réseau....162
ReSizeGLScene....321
Return....66

**S**

Self....65
Sérialisation....136
Serveur FTP....146
Smtplib....153
Socket....162
SQLite....113
Stéréogramme....347
STL....306
String....44
Sys....121
Système d'exploitation....120

**T**

Tableau multidimensionnel....94
Templates....139
Tests....359
Thread....212
Time....91
TIMESTAMP....91
TKInter....87
Trackball....315
Traduction fonctionnelle/technique....354
Transtypage....49
Trigonométrie....338
Try:... except:... finally: ....77
Tuple....48
Type....52

## U

UNITTEST....362

## V

Versionning....369
Vertex....310
Viabilité....359

## W

Webcam....165
While....59
With....61
Write....63

## X

XML....127
XPATH....131

## Z

Z-Buffer....309
Zipfile....124

## #

#....55