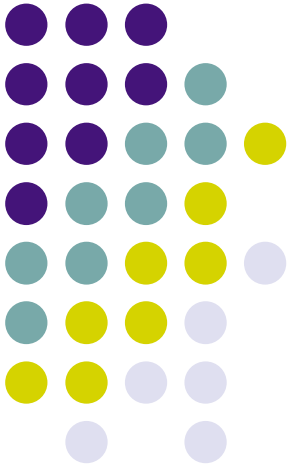
FSAB 1402: Informatique 2

Introduction et Concepts de Base

Peter Van Roy

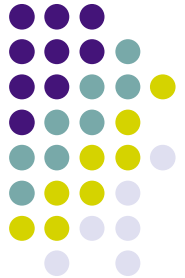
Département d'Ingénierie Informatique, UCL

pvr@info.ucl.ac.be



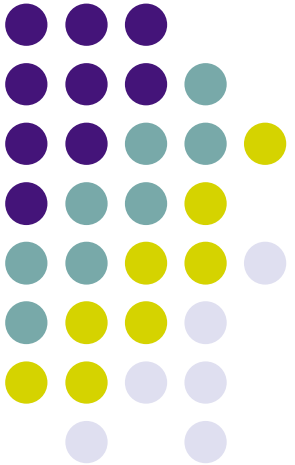
Transparents inspirés par Christian Schulte et Seif Haridi

Résumé du premier cours

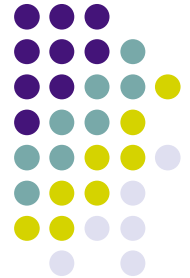


- Organisation du cours
- Résumé du cours
- Introduction aux concepts de base

Organisation du cours

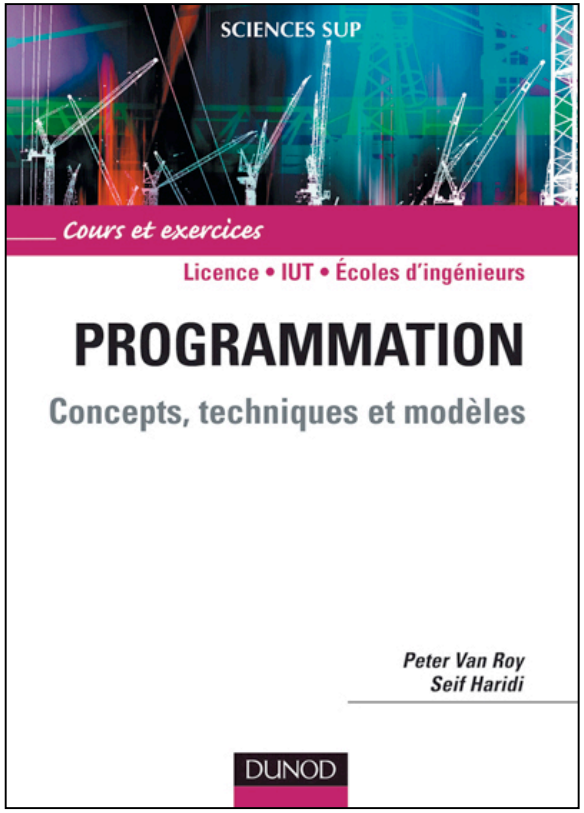


Cours magistraux et travaux pratiques



- Cours magistraux (semaines 1-13, sauf semaines 2 et 7)
  - Jeudi 8h30-10h30: BA91 (groupes 121X, 122X, 123X)
  - Jeudi 10h45-12h45: BA91 (groupes 124X, 125X)
  - Semaine 6 (24h vélo): le cours aura lieu dans un autre auditoire
  - Semaine 13: un deuxième cours le mercredi 10h45-12h45: SUD18
- Séances pratiques (semaines 2-13)
  - Les deux premières séances dans les salles CANDIX/IAO/DAO pour tous
    - Mardi 25 sept. 16h15-18h15 (groupes 124X, 125X et SINF12)
    - Mercredi 26 sept. 8h30-10h30 (groupes 121X, 122X, 123X)
    - Mardi 2 oct. 16h15-18h15 (groupes 124X, 125X et SINF12)
    - Mercredi 3 oct. 8h30-10h30 (groupes 121X, 122X, 123X)
  - Les séances suivantes
    - Les mêmes horaires qu'avant
    - Toutes en Pierre Curie: PCUR01, 02, 03, 04 ,05

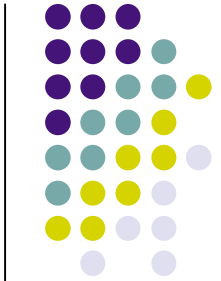
Livre du cours



- Le cours est basé sur le livre
  - Programmation: Concepts, techniques et modèles, de Peter Van Roy et Seif Haridi, édité par Dunod, sept. 2007, 354 pages
  - **La matière du cours = tout le contenu du livre**
- Attention! Chaque séance magistrale correspondra avec une partie du livre.
  - Les transparents du cours ne contiennent pas tout!
- Le livre est obligatoire pour le cours
  - Il est disponible au SICI pour environ 29€
  - Il y a aussi quelques exemplaires en bibliothèque en INGI et à la BSE
- Toutes les informations pratiques sur le cours sont disponibles sur le site Web:

  http://www.info.ucl.ac.be/Enseignement/Cours/FSAB1402/

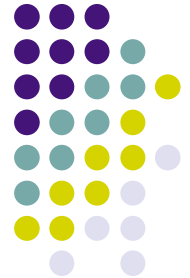
Logiciel du cours



- Labo interactif
  - Le Labo interactif a la même structure du livre et contient tous les exemples du livre et une centaine d'autres exemples
  - Il permet d'exécuter et de modifier tous les exemples
  - Il a été fait par nous (Yves Jaradin et Peter Van Roy) en collaboration avec l'éditeur ScienceActive
  - Pour les TPs nous ferons des exercices basés sur ces exemples
  - L'UCL a une licence de site pour ce logiciel; vous êtes encouragés de l'installer chez vous
- Le Labo interactif est basé sur le système Mozart

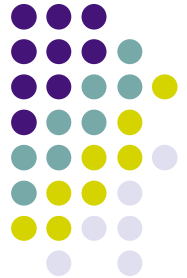
  - Mozart est la plate-forme sous-jacente qui implémente le langage Oz
  - Le Labo interactif permet une interaction qui est plus simple que l'interface de Mozart qui est basée sur l'éditeur emacs

Vous êtes des cobayes!



- Le livre en français vient de sortir en sept. 2007 et le Labo interactif est en version test
  - L'année dernière nous avons utilisé le livre en anglais "Concepts, Techniques, and Models of Computer Programming", MIT Press, et le système Mozart avec emacs directement
- Je compte sur vous pour donner du feedback
  - ScienceActive améliorera le logiciel selon nos commentaires
  - S'il y a des erreurs dans le livre, je ferai une page Web avec les corrections

Organisation des séances pratiques

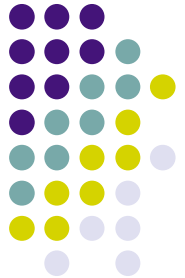


- Les deux premiers TPs seront des labos pratiques, pour maîtriser les concepts de base et l'utilisation du Labo interactif
  - Dans les salles CANDIX/IAO/DAO
- Les autres TPs seront faits *d'abord sur papier*
  - La réalisation sur l'ordinateur ne viendra qu'après que nous aurons vérifié vos solutions sur papier!
  - Nous voulons éviter le chipotage sur ordinateur et encourager la réflection
- Vous êtes encouragés d'installer le Labo interactif chez vous
  - Et d'explorer tous les exemples qu'il contient

Contenu des séances pratiques

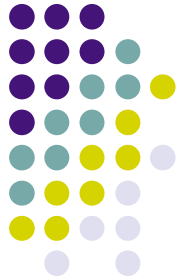
- Buts
  - Apprendre la **pratique** de la programmation
    - Écrire des programmes, écrire des programmes, écrire des programmes!
  - Répéter la matière des cours magistraux
  - Répondre aux questions sur la matière
  - Approfondir votre compréhension
- Exercices et problèmes
  - Comprendre les concepts et les mettre en pratique
  - Les encadrants vous aideront à trouver les réponses, mais ils ne donneront pas les réponses eux-mêmes!
- L'utilisation des ordinateurs
  - Il n'y en a pas beaucoup en Pierre Curie. Si vous avez un ordinateur portable, vous pouvez l'apporter aux séances pratiques.
  - Attention: le développement des programmes se fera d'abord sur papier – il ne faut pas tout de suite utiliser l'ordinateur!

Evaluations



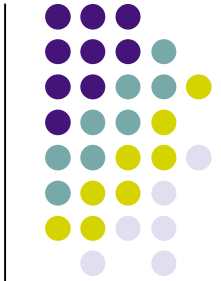
- Deux piliers du cours
  - La pratique: concevoir et écrire des programmes
  - La théorie: définir avec précision les concepts et la sémantique
- La syntaxe sera supposée connue
  - Ne pas faire d'erreurs!
- Interrogation
  - Vendredi 19 oct. dans l'après-midi (semaine 5)
  - Toute la matière des quatre premières séances magistrales
    - Y compris le jeudi de la veille!
- Petit projet
  - Après les interrogations, vers les semaines 10-12
- Participation dans les TPs
- Examen
  - Avec un formulaire de syntaxe que nous vous donnerons (éventuellement vous pouvez nous proposer un formulaire)

Equipe:
staff1402.fsa@student.uclouvain.be



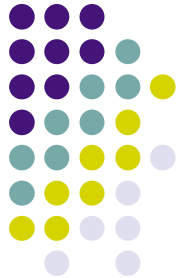
- Titulaire
  - Peter Van Roy (peter.vanroy@uclouvain.be)
- Assistants
  - Raphaël Collet (raphael.collet@uclouvain.be)
  - Yves Jaradin (yves.jaradin@uclouvain.be)
  - Boris Mejias (boris.mejias@uclouvain.be)
  - Stéphane Zampelli (stephane.zampelli@uclouvain.be)
- Etudiants/moniteurs
  - Pierre Bodson (pierre.bodson@student.uclouvain.be)
  - Laurent Lantsoght (laurent.lantsoght@student.uclouvain.be)
  - Adrien Dethioux (adrien.dethioux@student.uclouvain.be)
  - Vincent Lefevre (vincent.lefevre@student.uclouvain.be)
  - Julien Marlair (julien.marlair@student.uclouvain.be)
  - Stéphane Adline (stephane.adline@student.uclouvain.be)
  - Marie-Gabrielle Wybou (marie-gabrielle.wybou@student.uclouvain.be)
  - Adrien Bourgeois (adrien.bourgeois@student.uclouvain.be)
  - Simon van der Linden (simon.vanderlinden@student.uclouvain.be)

Encadrement



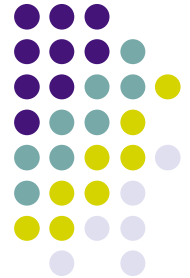
- Chaque encadrant aura trois groupes (ou tout les SINF12)
  - En principe, le même pendant tout le quadrimestre
- Si vous avez une question, adressez-vous d'abord à votre encadrant
  - Ne restez pas sur votre faim
    - Par exemple, dans les années précédentes il y a toujours beaucoup de personnes qui ne comprennent pas ce que c'est qu'un environnement contextuel. Pourtant, c'est très simple! Si vous ne comprenez pas
  - Vous pouvez aussi parler avec un assistant ou avec moi

Structure d’une séance



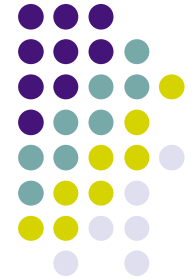
- Lecture pour le cours (sections du livre)
- Rappel de la dernière séance
- Résumé de ce qu’on va voir
- Le contenu
- Résumé de ce qu’on a vu

Lecture pour le premier cours



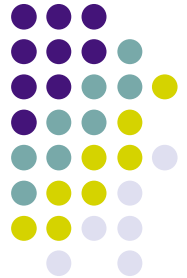
- Attention! Le livre va un peu plus loin que les transparents. Tout cela fait partie de la matière.
- Chapitre 1 (sections 1.1, 1.2, 1.3)
  - Introduction aux concepts de base
- Chapitre 2 (intro et section 2.1)
  - Définition des langages de programmation
  - La section 2.1.2 introduit la sémantique: on la verra en détail plus tard!
- Chapitre 2 (sections 2.2 et 2.3)
  - Le langage noyau et la mémoire à affectation unique
- Chapitre 2 (section 2.4.1)
  - Concepts de base (identificateurs, variables, environnement, portée, fonctions et procédures)

Des commentaires et suggestions sont bienvenus!



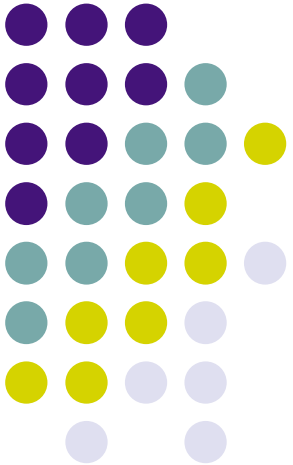
- Sur la structure ou le contenu du cours
- N'hésitez pas à me contacter ou à contacter un encadrant
  - Les encadrants doivent pouvoir répondre à toutes vos questions
- Si vous voulez me contacter, prenez un rendezvous s'il vous plaît!
  - Par email: peter.vanroy@uclouvain.be
  - Par téléphone: 010 47 83 74

Questions et freins!

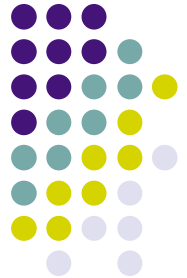


- Posez des questions quand ce n’est pas clair
  - Pour répéter une explication
  - Pour donner une meilleure explication
  - Pour donner un exemple
- Dites-moi quand je vais trop vite!
- Dites-moi quand je vais trop lentement!

Résumé du cours



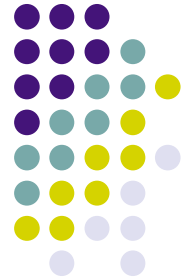
Objectif du cours



Donner une introduction à la discipline de la programmation, en cinq thèmes:

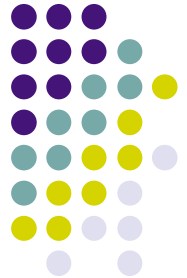
1. La programmation fonctionnelle (3 semaines)
   - Un programme est une fonction mathématique
2. Les techniques de programmation (3 semaines)
   - La complexité calculatoire
   - Les algorithmes sur les listes et les arbres
3. La sémantique formelle des langages (1 semaine)
   - On ne peut pas maîtriser ce qu'on ne comprend pas
4. L'abstraction de données (4 semaines)
   - Partitionner un problème pour maîtriser la complexité
   - Les deux approches: objets et types abstraits
   - Le langage Java
5. La concurrence et les systèmes multi-agents (2 semaines)

Il y a beaucoup de manières de programmer un ordinateur!



- Programmation déclarative
  - Programmation fonctionnelle ou programmation logique
- Programmation concurrente
  - Par envoi de messages ou par données partagées
- Programmation avec état
- Programmation orientée objet
- Programmation par composants logiciels

Modèles de programmation ("paradigmes")

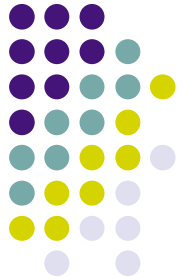


- Mettre ensemble
  - Des types de données et leurs opérations
  - Un langage pour écrire des programmes
- Chaque modèle/paradigme permet d'autres techniques de programmation
  - Les paradigmes sont complémentaires
- Le terme "paradigme de programmation"
  - Très utilisé dans le monde commercial; attention au sens (buzzword)!

Langages de programmation

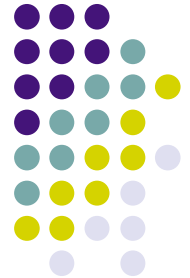
- Différents langages soutiennent différents modèles/paradigmes
  - Java: programmation orientée objet
  - Haskell: programmation fonctionnelle
  - Erlang: programmation concurrente pour systèmes fiables
  - Prolog: programmation logique
  - …
- Nous voudrions étudier plusieurs paradigmes!
- Est-ce qu'on doit étudier plusieurs langages?
  - Nouvelle syntaxe…
  - Nouvelle sémantique…
  - Nouveau logiciel…

La solution pragmatique…



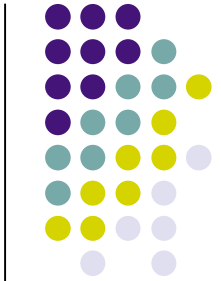
- Un seul langage de programmation
  - Qui soutient plusieurs modèles de programmation
  - Parfois appelé un langage “multi-paradigme”
- Notre choix est Oz
  - Soutient ce qu’on voit dans le cours, et plus encore
- L’accent sera mis sur
  - Les modèles de programmation!
  - Les techniques et les concepts!
  - **Pas** le langage en soi!

Comment présenter plusieurs modèles de programmation?



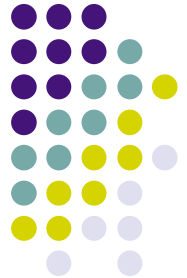
- Basé sur un langage noyau
  - Un langage simple
  - Un petit nombre de concepts **significatifs**
  - Buts: simple, minimaliste
- Langage plus riche au dessus du langage noyau
  - Exprimé en le traduisant vers le langage noyau
  - But: soutenir la programmation pratique

Approche incrémentale



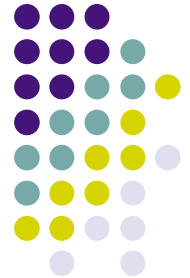
- Commencer par un langage noyau simple
  - Programmation fonctionnelle
- Ajouter des concepts
  - Pour obtenir les autres modèles de programmation
  - Très peu de concepts!
  - Très peu à comprendre!
- En FSAB1402 nous ne verrons que quelques modèles, à cause de la taille limitée du cours
  - Trois modèles principaux
  - Les autres modèles peuvent être vus dans d'autres cours (par exemple, en INGI1131 ou INGI2365) ou en lisant plus loin dans le livre

Les modèles que vous connaissez!



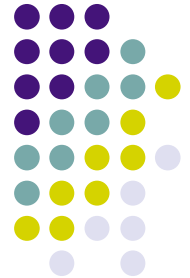
- Vous connaissez tous le langage Java, qui soutient
  - La programmation avec état
  - La programmation orientée objet
- C'est clair que ces deux modèles sont importants!

Pourquoi les autres modèles?



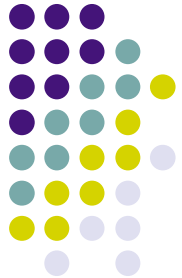
- Deux nouveaux modèles qu'on verra dans ce cours
  - Programmation déclarative (fonctionnelle)
  - Programmation concurrente (multi-agent) avec dataflow
- D'autres modèles pas vu dans ce cours
  - Programmation concurrente par envoi de messages ou par données partagées
  - Programmation logique (déterministe et nondéterministe)
  - Programmation par contraintes
  - Programmation par composants
  - …
- Est-ce que tous ces modèles sont importants?
  - Bien sûr!
  - Ils sont importants dans beaucoup de cas, par exemple pour les systèmes **complexes**, les **systèmes multi-agents**, les systèmes à **grande taille**, les systèmes à **haute disponibilité**, etc.
  - On reviendra sur ce point plusieurs fois dans le cours

Notre premier modèle



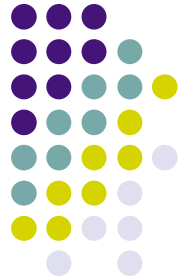
- La programmation **déclarative**
  - Deux formes: programmation fonctionnelle et programmation logique
  - On peut considérer la programmation fonctionnelle comme la base de la plupart des autres modèles
    - On regardera uniquement la programmation fonctionnelle
- Approche
  - Introduction informelle aux concepts et techniques importants, avec exemples interactifs
  - Introduction au langage noyau
  - Sémantique formelle basée sur le langage noyau
  - Étude des techniques de programmation, surtout la récursion (sur entiers et sur listes) et la complexité calculatoire

La programmation déclarative



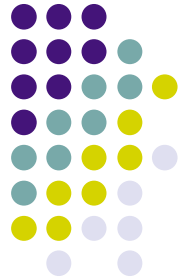
- L'idéal de la programmation déclarative
  - Dire uniquement **ce que** vous voulez calculez
  - Laissez l'ordinateur trouvez **comment** le calculer
- De façon plus pragmatique
  - Demandez plus de soutien de l'ordinateur
  - Libérez le programmeur d'une partie du travail

Propriétés du modèle déclaratif



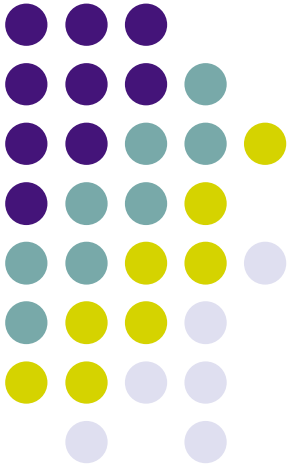
- Un programme est une fonction ou une relation au sens mathématique
  - Un calcul est l'évaluation d'une fonction ou une relation sur des arguments qui sont des structures de données
- Très utilisé
  - Langages fonctionnels: LISP, Scheme, ML, Haskell, …
  - Langages logiques (relationnels): Prolog, Mercury, …
  - Langages de représentation: XML, XSL, …
- Programmation "sans état"
  - Aucune mise à jour des structures de données!
  - Permet la simplicité

L’utilité du modèle déclaratif



- Propriété clé: “Un programme qui marche aujourd’hui, marchera demain”
  - Les fonctions ne changent pas de comportement, les variables ne changent pas d’affectation
- Un style de programmation qui est à encourager dans tous les langages
  - “Stateless server” dans un contexte client/server
  - “Stateless component”
- Pour comprendre ce style, nous utiliserons le modèle fonctionnel!
  - Tous les programmes écrits dans ce modèle sont ipso facto déclaratif: une excellent manière de l’apprendre

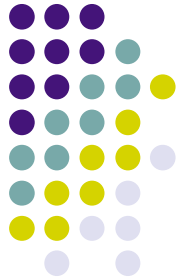
Introduction aux Concepts de Base



Un langage de programmation

- Réalise un modèle de programmation
- Peut décrire des programmes
  - Avec des **instructions**
  - Pour calculer avec des **valeurs**
- Regardons de plus près
  - instructions
  - valeurs

Système interactif



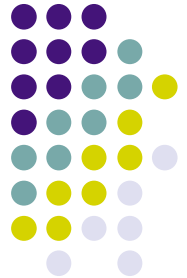
declare

X = 1234 * 5678

{Browse X}

- Sélectionner la région dans le buffer Emacs
- Donner la région sélectionnée au système
  - La région est compilée
  - La région compilée est exécutée
- Système interactif: à utiliser comme une calculatrice
- Essayez vous-même après ce cours!

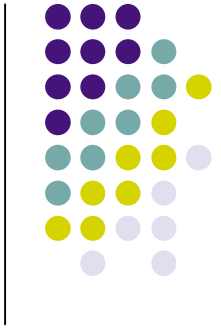
Système interactif: Qu'est-ce qui se passe?



```
declare
X = 1234 * 5678
{Browse X}
```

- **Déclarer** ("créer") une variable **désignée** par X
- **Affecter** à la variable la valeur 7006652
  - Obtenu en faisant le calcul 1234*5678
- **Appeler** la procédure Browse avec l'argument désignée par X
  - Fait apparaître une fenêtre qui montre 7006652

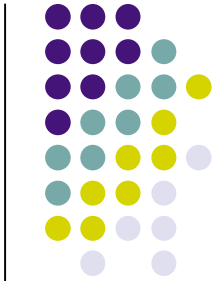
Variables



- Des raccourcis pour des valeurs
- Peuvent être affectées une fois au plus
  - Variables mathématiques
  - (Note: l'affectation multiple est un autre concept; on la verra plus loin dans le cours sous le nom de "cellule")
- Sont dynamiquement typées (type connu **pendant** l'exécution)
  - En Java elles sont statiquement typées: type connu **avant** l'exécution (à la compilation)
- Attention: il y a **deux** concepts cachés ici!
  - Premier concept: l'identificateur: le nom que vous tapez sur le clavier, c'est une chaine de caractères qui commence avec une majuscule

    Var, A, X123, OnlyIfFirstIsCapital
  - Deuxième concept: la variable en mémoire: une partie de la mémoire du système

Déclaration d'une variable

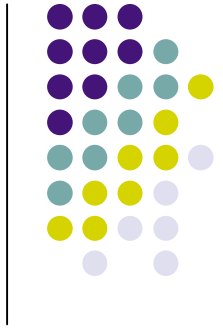


declare

X = …

- **declare** est une instruction ("statement")
  - Crée une nouvelle variable en mémoire
  - Fait le lien entre l'identificateur X et la variable en mémoire
- Troisième concept: l'environnement
  - Une fonction des identificateurs vers les variables
  - Fait la correspondance entre chaque identificateur et une variable (et donc sa valeur aussi)
  - Le même identificateur peut correspondre à différentes variables en différents endroits du programme!

Affectation

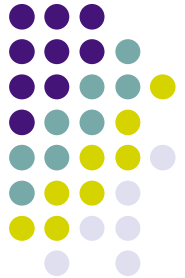


declare

X = 42

- L'affectation prend une variable en mémoire et la lie avec une valeur
- Dans le texte d'un programme, l'affectation est exprimée avec les identificateurs!
- Après l'affectation X=42, la variable qui correspond à l'identificateur X sera liée à 42

Affectation unique



- Une variable ne peut être affectée qu'à une seule valeur
  - On dit: **variable à affectation unique**
  - Pourquoi? Parce que nous sommes en modèle déclaratif!

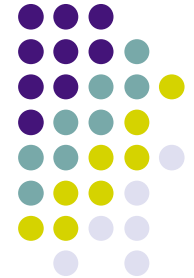
- Affectation incompatible: signale une erreur

  X = 43

- Affectation compatible: ne rien faire

  X = 42

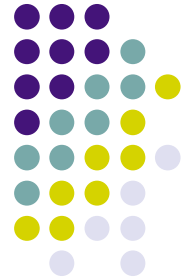
La redéclaration d'une variable (en fait: d'un identificateur!)



```
declare
X = 42
declare
X = 11
```

- Un identificateur peut être redéclaré
  - Ça marche parce qu'il s'agit des variables en mémoire différentes! Les deux occurrences de l'identificateur correspondent à des variables en mémoire différentes.
- L'environnement interactif ne gardera que la dernière variable
  - Ici, X correspondra à 11

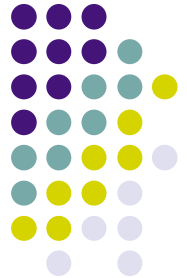
La portée d'une occurrence d'un identificateur



```
local
    X
in
    X = 42  {Browse X}
    local
        X
    in
        X = 11  {Browse X}
    end
    {Browse X}
end
```

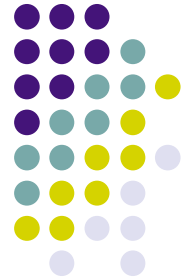
- L'instruction
  local X **in** <stmt> **end**
  déclare X qui existera entre **in** et **end**
- La portée d'une occurrence d'un identificateur est la partie d'un programme pour laquelle cet identificateur correspond à la même variable en mémoire.
- (Si la portée est déterminée par une inspection du code d'un programme, elle s'appelle portée lexicale ou portée statique.)
- Pourquoi n'y a-t-il pas de conflit entre X=42 et X=11?
- Le troisième Browse affichera quoi?

Structures de données (valeurs)



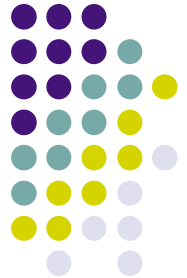
- Structures de données simples
  - Entiers 42, ~1, 0
    Notez: “~” pour entier négatif (!)
  - Virgule flottante 1.01, 3.14, 0.5, ~3.2
  - Atomes (constantes) foo, ‘Paul’, nil
- Structures de données composées
  - Listes
  - Tuples, enregistrements (“records”)
- Dans ce cours on utilisera principalement les entiers et les listes

Fonctions



- Définir une fonction
  - Donner une instruction qui définit ce que doit faire la fonction
- Appeler une fonction
  - Utiliser la fonction pour faire un calcul selon sa définition
  - On dit aussi: **appliquer** une fonction

Notre première fonction

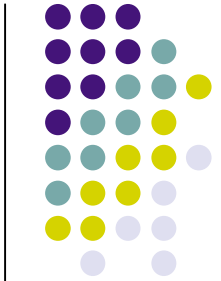


- Pour calculer la négation d'un entier
  - Prend un argument: l'entier
  - Rend une valeur: la négation de l'entier

- En notation mathématique:

$$\text{minus}: \begin{cases} \text{Integer} \rightarrow \text{Integer} \\ n \rightarrow \sim n \end{cases}$$

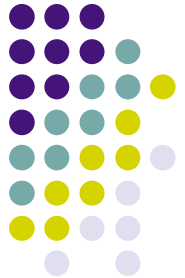
La définition de la fonction Minus



```
declare
fun {Minus X}
   ~X
end
```

- L'identificateur Minus sera lié à la fonction
  - Déclarer une variable qui est liée à l'identificateur Minus
  - Affecter cette variable à la fonction en mémoire
  - Attention! L'identificateur Minus commence avec une majuscule.
- La portée de l'argument X est le corps de la fonction

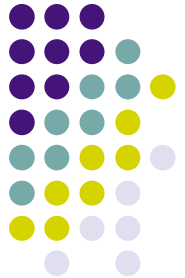
L’application de la fonction Minus



```
declare
Y = {Minus ~42}
{Browse Y}
```

- Y est affecté à la valeur calculée par l’application de Minus à l’argument ~42

Syntaxe



- Définition d'une fonction

  fun {*Identificateur Arguments*}

  Corps de la fonction

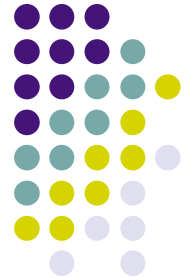
  end

  Le corps de la fonction est une expression qui calcule le résultat de la fonction

- Appel d'une fonction

  X = {*Identificateur Arguments*}

Fonction de maximum

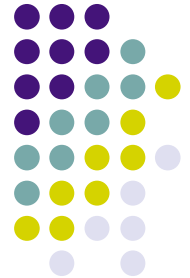


- Calculer le maximum de deux entiers
  - Prend deux arguments: entiers
  - Rend une valeur: l'entier le plus grand

- En notation mathématique:

$$\text{max}: \begin{cases} \text{Integer x Integer} \rightarrow \text{Integer} \\ \\ n, m \mapsto n, & n > m \\ \quad\;\; m, & \text{sinon} \end{cases}$$

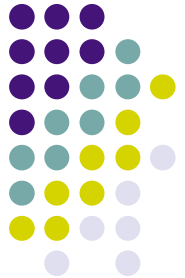
Définition de la fonction Max



```
declare
fun {Max X Y}
   if X>Y then X
   else Y
   end
end
```

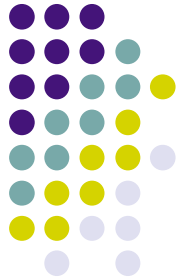
- Nouvelle instruction: conditionnel (if-then-else)
- Le conditionnel renvoie un résultat

Application de la fonction Max



```
declare
X = {Max 42 11}
Y = {Max X 102}
{Browse Y}
```

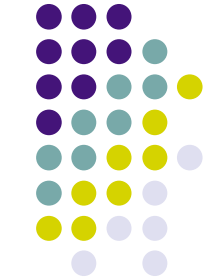
Maintenant le minimum



- Une possibilité: couper et coller
  - Répéter ce qu'on a fait pour Max
- Mieux: la **composition** de fonctions
  - Reutiliser ce qu'on a fait avant
  - C'est une bonne idée de reutiliser des fonctions compliquées
- Pour le minimum de deux entiers:

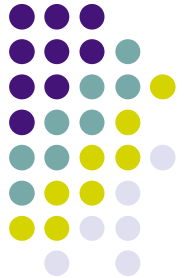
  min(*n*,*m*) = ~max(~*n*,~*m*)

Définition de la fonction Min



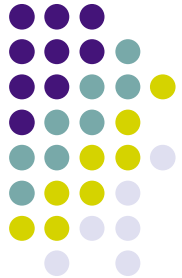
```
declare
fun {Min X Y}
   {Minus  {Max  {Minus X}
                 {Minus Y}}}
end
```

Définition de la fonction Min (avec ~)



```
declare
fun {Min X Y}
   ~{Max ~X ~Y}
end
```

Définition inductive d'une fonction



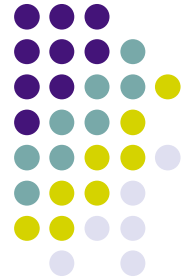
- Fonction de factorielle *n*!
  - La définition inductive est

    0! = 1

    n! = *n* * ((*n*-1)!)
  - Programmer ceci en tant que fonction Fact

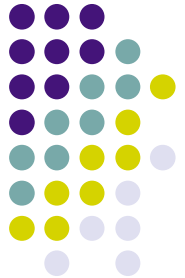
- Comment procéder?
  - Utiliser l'application **récursive** de Fact!

Définition de la fonction Fact



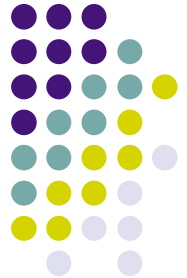
```
fun {Fact N}
   if N==0 then     % Test d'égalité
      1
   else
      N * {Fact N-1} % Appel récursif
   end
end
```

Récursion



- Structure générale
  - Cas de base
  - Cas récursif
- Pour un nombre naturel *n*, souvent:
  - Cas de base: *n* est zéro
  - Cas récursif: *n* est différent de zéro
    n est plus grand que zéro
- On verra d'autres programmes récursifs la semaine prochaine!
  - La récursion est la manière de faire une boucle en modèle déclaratif

Une fonction est un cas particulier d'une procédure

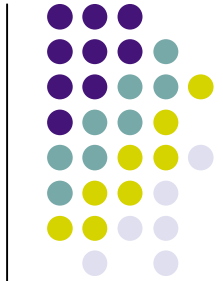


- Le vrai concept de base est la **procédure**
- Une fonction est une procédure avec un argument en plus, qui est utilisée pour renvoyer le résultat
- Z={Max X Y} *(fonction Max)*

  est équivalent à:

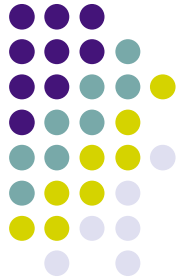
  {Max X Y Z} *(procédure Max, avec un argument de plus)*

Résumé



- Système interactif
- “Variables”
  - Déclaration
  - Affectation unique
  - Identificateur
  - Variable en mémoire
  - Environnement
  - Portée d’une occurrence d’un identificateur
- Structures de données
  - Entiers
- Fonctions
  - Définition
  - Appel (application)
- Récursion

A la prochaine fois!



- La séance pratique pour aujourd'hui:
  - FSA12: mardi prochain 16h15-18h15
  - FSA12: mercredi prochain 8h30-10h30
  - SINF12: venez me voir!
  - J'attends de vous que vous vous familiarisez avec le Labo interactif et les concepts qu'on a vus aujourd'hui
- L'énoncé du premier TP est sur le Web
- Je vous conseille d'installer le Labo interactif chez vous