Lua introduction for new programmers

Download slides at:
http://tstarling.com/presentations

Hello world

- In [[Module:Hello]] put:

```
local p = {}
function p.hello()
    return 'Hello, world!'
end
return p
```

- Then in a wiki page:

```
{{#invoke: Hello | hello }}
```

- Try it now at http://scribunto.wmflabs.org/

if

```
if colour == 'black' then
    cssColour = '#000'
end
```

if

```
if colour == 'black' then
    cssColour = '#000'
else
    cssColour = colour
end
```

if

```
if colour == 'black' then
    cssColour = '#000'
elseif colour == 'white' then
    cssColour = '#fff'
else
    cssColour = colour
end
```

Equality

- Single equals is only for assignment

```
if x = y then -- Error: 'then' expected near '='
    return 'something'
end
```

- Use double equals for equality

```
if x == y then
    return 'something'
end
```

for

```
f = 1
for i = 1, 5 do
    f = f * i
end
return 'The factorial of 5 is ' .. f
```

Types

- String
- Number
- Boolean (true/false)
- nil
- A few other things

Logic

- **and**: both are true

```
if beans and toast then
    return 'breakfast'
end
```

- **or**: one or the other or both

```
if chicken or beef then
    return 'dinner'
end
```

- **not**: the following thing is false

```
if not hungry then
    return 'nothing'
end
```

Functions

- Calling functions

```
colour = getDivColour()
```

- Defining functions

```
local function getDivColour()
    return 'blue'
end
```

Functions

- Functions let you avoid duplication
- Functions can have arguments:

```
local function plural(word)
    return word .. 's'
end
```

- German version left as an exercise to the reader

Functions

- Two types of functions
- Local functions for private use within the module

```
local function plural(word)
    return word .. 's'
end
```

- Exported functions

```
local p = {}
function p.hello()
    return 'hello'
end
return p
```

Local variables

```
function getDivStart()
    colour = getDivColour()
    return '<div style="background-color: ' ..
        colour .. '">'
end

colour = 'Fuschia'
return getDivStart() .. colour .. '</div>'
```

Local variables

```
function getDivStart()
    local colour = getDivColour()
    return '<div style="background-color: ' ..
        colour .. '">'
end

colour = 'Fuschia'
return getDivStart() .. colour .. '</div>'
```

Local variables

- If you don't set a variable to something, it will be nil by default

```
local x
if ready then
    x = 'GO!'
end
-- Now x has "GO!"
or nil
```

Tables

- Creating a table

```
numbers = {
    one = 1,
    two = 2,
    three = 3
}
```

- Accessing a table element

```
return numbers.one      -- returns 1
return numbers['one']   -- also returns 1
```

Numbered tables

```
africanFlatbreads = {
    'Aish Mehahra',
    'Injera',
    'Lahoh',
    'Ngome'
}

return africanFlatbreads[2] -- returns 'Injera'
```

Visiting each table element

- `pairs`: key/value pairs in random order

```
for name, number in pairs(numbers) do
    ...
end
```

- `ipairs`: Numeric keys in ascending order

```
for index, bread in ipairs(africanFlatbreads) do
    ...
end
```

Strings

- Length

```
s = 'hello'
return #s    -- returns 5
```

- sub

```
s = 'hello'
return s:sub(2, 3)     -- returns 'el'
return s:sub(2)        -- returns 'ello'
return s:sub(-2)       -- returns 'lo'
```

Further reading

- Programming in Lua: http://www.lua.org/pil/
- Reference manual: http://www.lua.org/manual/5.2/
- Scribunto: https://www.mediawiki.org/wiki/Extension:Scribunto
- lua-users.org

Lua introduction for programmers

Lexical

- Comments reminiscent of SQL

```
-- Single line comment

--[[
long comment
--]]
```

- Line breaks ignored
- Semicolons to terminate statements
  - optional, discouraged

Data types

- nil
- Numbers
  - Single type, floating point
- Strings
  - 8-bit clean
- boolean

Data types

- Functions
  - First class values
  - Return multiple values
  - Multiple return values are not bundled into a data type
  - Anonymous syntax:

```
x = function ()
    ...
end
```

Data types

- Tables
  - Implemented as a hashtable
  - Used for OOP, like JavaScript
  - Literal syntax: {name = value} or {a, b, c}
  - Access with foo.bar or foo['bar']

Operators

- Not equals: `~=` instead of `!=`
- Concatenation: `..`
- Length: `#`
- Logical: `and`, `or`, `not`
- Exponentiation: `^`
- Usual meanings: `<`, `>`, `<=`, `>=`, `==`, `+`, `-`, `*`, `/`, `%`

Operator omissions

- No assignment operators like +=
  - Even plain = is not really an operator
- No bitwise operators
- No ternary ? :

Assignment

- Like BASIC, assignment is a complete statement, not an expression
- Multiple assignment:

```
a, b = c, d
a, b = foo()
```

- Assignment with local variable declaration:

```
local a, b = c, d
```

- Not like this!

```
local a = b, c = d
```

Control structures

- Explicit block

```
do
    ...
end
```

- Precondition loop

```
while cond do
    ...
end
```

- Postcondition loop

```
repeat
    ...
until cond
```

Control structures

- If

```
if cond then
    ...
elseif cond then
    ...
else
    ...
end
```

Control structures

- Numeric for

```
for i = start, stop do
    ...
end
```

- Generic for

```
for i in iterator do
    ...
end
```

- Index variable is local to the loop

Variables

- Lexical scoping, almost identical to JavaScript
- An unset variable is identical to a nil variable
  - No special syntax for deletion, just x = nil
  - No error raised for access to undefined variables

Objects

- Made from tables using a variety of syntaxes, similar to JavaScript
- Private member variables implemented using lexical scoping, as in JavaScript
- Dot for static methods: `obj.func()`
- Colon for non-static methods: `obj:func()`

Objects

- Factory function style example

```
function newObj()
    local private = 1
    local obj = {}

    function obj:getPrivate()
        return private
    end

    return obj
end
```

Metatables

- Each table may have an attached metatable
- Provides operator overloading
- "index" metatable entry is used for object inheritance and prototype-based OOP

MediaWiki/Lua interface

Module namespace

- All Lua code will be inside the Module namespace
- Code editor provided
  - "ace" JavaScript code editor
  - Automatic indenting
  - Syntax highlighting

Invocation

- {{ #invoke: module_name | function_name | arg1 | arg2 | name1 = value1 }}
- #invoke instances are isolated, globals defined in one are not available in another
- Only caches are shared

Module structure

- A module is a Lua chunk that returns an export table
- `require`() provided
  - not isolated
- `package` library provided

Return value

- The exported function returns a wikitext string
- Multiple return values are concatenated
- Non-string return values are converted to string
  - Metatable entry "tostring" supported

Frame methods

- Argument access: args

```
local name1 = frame.args.name1
```

- argumentPairs()

```
local t = {}
for name, value in frame:argumentPairs() do
        t[name] = value
end
```

- getParent()
  - Provides access to the parent frame, i.e. the arguments to the template which called #invoke

Frame methods

- Wikitext preprocessing

```
frame:preprocess('{{template}}')
```

- Structured template invocation

```
frame:expandTemplate{
    title = 'template',
    args = {foo = foo}}
```

Avoiding double-expansion

- Arguments are already expanded
- Don't construct preprocessor input from arguments
- Use `frame:expandTemplate()`

Future directions

- Gabriel Wicke's interface:
  - frame:getArgument()
  - frame:newParserValue()
  - frame:newTemplateParserValue()
- All provided but only with stub functionality

Future directions

- Interwiki module invocation
- Languages other than Lua
- Date/time functions
- Direct access to other core parser functions and variables
  - {{PAGENAME}}
  - {{#ifexist:}}
  - etc.

Further reading

- Programming in Lua: http://www.lua.org/pil/
- Reference manual: http://www.lua.org/manual/5.2/
- Scribunto: https://www.mediawiki.org/wiki/Extension:Scribunto
- lua-users.org

Try it out

- Go to http://scribunto.wmflabs.org/
- Create a function that takes several arguments and does something to them

```
local p = {}
function p.hello(frame)
    return 'Hello ' .. frame.args[1]
end
return p
```