CoffeeScript

# About CoffeeScript

* A Programming language that compiles into JavaScript. <http://coffeescript.org/>
* Borrows some features from Ruby & Python.
* CoffeeScript is a transcompiler and results into formatted & readable JS
* Coffescript compiler is distributed as exe for node js & standalone js file.
* To start, you need Text editor + Coffescript compiler + JS environment (browser /node.js)
* Other languages that compile JS are: Traceur, ECMA Harmony / Narcisuss, Script#, GWT, Pyjamas.
* Features of Coffescript:
  + CoffeeScript Syntax (alternative syntax to JS):
    - Clean & minimal – reduced usage of ; , () {}
    - Significant Whitespace – uses indentation to mark blocks
    - Syntactic sugar & Expressive
  + Coffescript has class syntax, supports inheritance but no private members
  + Comprehensions are expressions to iterate over collections.
    - Return value is collected in result variable
    - With array, item & index can be iterated.
    - With Object, key & value can be iterated
  + Splats enable function arguments to be collapsed into an array.
    - Easy to deal with functions that accept any # of arguments
    - Allows arrays to be expanded into multiple arguments
  + Fixes JS Faults
    - == or != operator translates to === or !==. Removes data coercion mistakes
    - Provides ? existential operator to verify if variable is defined
    - Proper Lexical scoping as it ensures that variable will be declared with ‘var’

## Coffescript Installation

* It is optional.
* On Mac /linux , install using node package manager
  + **curl http://npmjs.org/install.sh | sh** //installs Node Package Manager
  + **npm install coffee-script** // install coffee-script Package
* On windows , install using coffeesharp. Run commend on prompt
  + **coffee** // starts the coffee environment
* **coffee <filename.coffee>** //executes a coffee script
* **coffee -c <filename.coffee>** //compiles the file & save it with extension js
* **coffee --help** // shows syntax of coffee command
* Avoid the compilation step by using file system watchers (Compiled only @ file change)
  + **coffee –o .\bin –c.\src -w** //-c what to compile(./ all files in current dir) –o where to save -w activates file watcher

# Basic

* **#** single line comment
* **###** multi line comment **###**
* **varname = value** //declares & assigns a variable if not already declared else assigns it
  + translates into local variable in JS i.e. lexical scoping
* [var1,var2] =[val1,val2] //declares & assigns multiple variables in single line
  + In JS it translates into 2 vars & an array
  + It works with arrays & objects
* **Blocks** // increased indentation marks new block whereas decreased indentation means end of block
* **if condition**

**action block**

**else**

**alternative block** // conditional statement

* + **action if condition** // if may come after the statement on same line
* **unless condition**

**action block** // avoid use of not operator. Conditional statement

* + **action unless condition** // unless may come after the statement on same line
* **variable = if** **condition then valT else valF** // translates into ternary operator
* **a<b<c** // chained comparisons. Translates into a<b && b<c
* **while condition**

**loop block** // Iterates while condition is true

* **until condition**

**loop block** // Iterates while condition is false

* **for item,index in array**

**iterate for each item** //array comprehension

* Logical operators: **and (&&), or (||), not (!), is (==), isn’t (!=)**

# Data types

Strings

* Single Quoted strings are literals ex **‘one plus one =#{1 +1}’** //prints one plus one =#(1 +1)
* Double quoted strings are interpolated ex **“one plus one =#{1 +1}”** //prints one plus one =2
  + **#{varname} or #{expr}** // gives value of var or expr. It can be part of string in “”
* Coffeescript supports multiline string.
  + **Enclose in “”** //concatenates the string in JS
  + **Enclose in ‘’** // keeps the newline character by treating as escape character
* Heredocs – String literal syntax **‘’’ multilines ‘’’**
  + Preserves line breaks & whitespace. Indentation is relative to start of heredoc.
  + Supports interpolation if string is double quotes **“”” multilines “””**

Numbers

* Same like JS Numbers. Floating point numbers. Not exact decimal fractions
* Imp functions : **parseFloat , ToFixed, Math.abs, Math.pow, Math.random, Math.round, Math.sqrt**
* Ranges – whole numbers ascending **[1..5]** descending **[5..1]** // translates into array in JS
  + **[1…5]** // exclude the second number . [1..3] = [1,2,3] whereas [1…3] = [1,2]

Arrays

* Similar to JS arrays with slicing & splicing capabilities
* Comma in the array can be omitted if indented on next line so **[1,2] is same as [1**

**2]**

* **arrObjSliced = arrObj [start..end]** // slices array from index=start to end. Index begin from 0
  + works on strings as it will return a new string.
* **arrObj [start..end] =[replace value]** //splicing the array removes the value from index=start to end and replaces it with replace value
  + it can change the length of the array.
  + It does not work on strings as strings are immutable

Regular Expressions

* Similar to JS with heregex (extended syntax). **/pattern/** is Regular expression literal
  + Heregex improves readability by ignoring whitespace & comments in literals **/// multiline pattern ///**

# Functions

* **funcName = (parameterList) ->**

**Code Block** // define a function

* **->** //declares a function & by itself is an empty function
* **()** required to invoke a function without any arguments. With arguments () is optional
* Last expression in function body is return value
* **funcName = (p1=v1,p2=v2…)** //Coffescript allows Default arguments unlike JS
* **this** // within function refers to object on which it is invoked
  + it’s value is lost if passed as callback or attached to new object. Use **=>** to bind the value of this when function was declared.

# Objects & Classes

* Object literal syntax is similar to JS except comma & {} not required if properties on new line
* Coffescript classes
  + objects used to instantiate other objects or special functions like constructors
  + Do not provide encapsulation. Convention is to start private variable with underscore
* **class ClassName** //class keyword declares a class. Its type is function

**constructor : (@P1,@P2) ->** //contructor keyword declares a constructor. Empty constructor

**methodName : -> console.log “#{@P1} , #{@P2}”** //@ is shorthand for this. Adds method to prototype of class

* **Obj = new ClassName ‘A1’,A2** //creates the object. its type is Object
* **Obj.methodname()** //calls the class method

Inheritance

* **extends** // implements inheritance
* **super** // to call parent’s member of same name
* Define static properties by adding the properties to class object