Scripting Languages

Intro to Ruby

Scripting Languages

Modern scripting languages have two principal sets of ancestors.

- Command interpreters or "shells" for traditional batch and "terminal" (command-line) processing
  - IBM's JCL, MS-DOS command interpreter, Unix sh and csh
- Tools for text processing and report generation
  - IBM's RPG, and Unix's sed and awk.

Scripting Languages

Glue languages – to "glue" existing programs together to build a larger system

- Combine a collection of existing components

General system administration tasks

Extension languages – like VB "macros" for MS Office

Web scripting

Common Characteristics

Batch and interactive – Perl does just-in-time compilation. Most others either compile or interpret input line-by-line

Minimum syntax overhead – Compare Ruby to Java, etc. for:

```
print "Hello world.\n"
```

Lack of declarations and dynamic typing

Easy access to other programs and the OS

Built in pattern matching and string manipulation

Example Scripting Languages

- Ruby
- Perl
- Tcl
- Python
- PHP
- Javascript
- Groovy
- Powershell
- Visual Basic
- Mathematica/ Maple/MathLab

Ruby

Designed in Japan by Yukihiro Matsumoto (a.k.a, "Matz")

Began as a replacement for Perl and Python

A pure object-oriented scripting language

- All data are objects

Most operators are implemented as methods, which can be redefined by user code

Purely interpreted

- Can be used interactively (irb)

Variables and Expressions

Nothing unusual, except variables are not declared

```
x = 3 + 2*(5 – 1)
x = 4**3
x = "x" + "y" + "z"
x = x < 'abc'
```

Constants and Variables

Names begin with a capital letter

```
Quarter = 0.25
```

They're not really constant, but changing them may give a warning

Variable names begin with a lower case letter

```
x = 10
```

Comments

```
# Extends to the end of the line

=begin
  This is a multi-line comment
  It has 2 lines
=end
```

Everything is an object

All arithmetic, relational, and assignment operators are implemented as methods

Numeric types are classes with methods

```
4.times { puts "Repeat that..."}
"Repeat that..." * 5
```

Conditionals

```
temp = 70
if temp >= 90
  puts "Wow, its HOT!"
elsif temp >= 70
  puts "This is nice"
elsif temp >= 50
  puts "A little cool"
else
   puts "Brrrrrrr!"
end
```

Conditionals

```
temp = 95
puts "Wow, its HOT!" if temp >= 90

temp -= 60
puts "Brrrrrrr" unless temp >= 50

temp += 40
puts "This is nice" if (70..90) === temp
```

Arrays

An array is a one-dimensional vector that holds zero or more values

```
fib = [1, 1, 2, 3, 5, 8]
mixed = ["cool", -4, 3.14, "hot"]
empty = []
five = fib[4]
pi = mixed[-2]
```

Two methods for the length of an array

```
len1 = fib.length
len2 = mixed.size
```

Array operations

Push and pop add/remove an element at the end of an array

```
list = ["first"]
list.push("second")
list << "third"
list.push "fourth"
list[list.length] = 'fifth'
list.pop
```

Now `list` contains:

```
["first", "second", "third", "fourth"]
```

Array operations

Shift and unshift to remove/add an element at the front of an array

Insert, delete, and replace anywhere in an array

```
x = list.shift
list.unshift 1234
list.insert(2, 'abc')
list.delete_at 3
list[2] = 'xyz'
```

Array operations

Building an array by splitting

```
b = "this is an array".split(" ")
```

Build a string by joining

```
s = b.join
```

Add an element at the beginning of the array

```
b.unshift s
```

Array operations

There are many other interesting array methods, including

```
clear, delete, empty?, first,
include?, index, last, reverse,
reverse!, sort, sort!, uniq,
values_at, +, -, &, |
```

Array iteration

Ways to iterate through an array

```
i = 0
while i != list.length
  puts list[i]
  i = i + 1
end
```

```
list.each do
   |item| puts item
end
```

```
for item in list
  puts item
end
```

Array Slicing

List of 26 characters in array b

```
b = ('A'..'Z').to_a
```

Slice an array

```
c = b[13..b.length]
c[2..10].each {|item| print item + " "}
d = b.values_at(1,2,5,8..15,20)
```

Hashes

A *hash* stores a mapping of (key, value) pairs

Ruby provides hashes to quickly look up the value given the key when there are many (key, value) pairs.

Similar to hash tables

Accessed similar to arrays, but uses keys as subscripts

Hashes

```
h = {}  # makes an empty hash
h['simpson'] = 'bart'
h["bunny"] = "bugs";
h[2.3] = 8.9;
h.keys.each {|key| puts key }
h[2.3] += 2
h.values.each do
  |val| puts val
end
h.each {|k, v| puts "#{k} : #{v}"}
```

Hashes

```
states = {"NJ" => "new jersey",
          "PA" =>  "pennsylvania"}
st = "NY"
if states[st]
   puts st + " is abbrev for " + states[st]
else
   puts "No known state for " + st
end
```

Methods

```
def weather(temp)
   if temp >= 90
     puts "Wow, its HOT!"
   elsif temp >= 70
     puts "This is nice"
   elsif temp >= 50
     puts "A little cool"
   else
      puts "Brrrrrrr"
   end
end

weather(75)
```