Introduction to Ruby

October 14,2014

Agenda

- About Ruby
- Comments
- Constants and variables
- Scopes
- Methods
- Ranges
- Arrays
- Operators
- Flow Control
- Iterators
- Math Library
- Date and Time
- Hashes
- Object and Classes
- Conclusion
- Hmw

About Ruby

- created by Yukihiro Matsumoto(Matz) in Japan in 1993
- Released 1995
- After 2000, it has grown in popularity, particularly because of the popularity of the Ruby on Rails.
- an **object-oriented** interpreted scripting language
- portable across multiple operating system platforms and hardware architectures

About Ruby

- a very intuitive and clean programming language
- A general-purpose language-> it can be used to create full scale, standalone GUI based applications
- also great for serving web pages, generating dynamic web page content and excels at database access tasks.

Commenting Ruby Code

Single line comments

#

Multi Line or Block Ruby Comments

=begin

=end

Define Constants, Variables

- declared by beginning the variable name with a capital letter (a common convention for declaring constants is to use uppercase letters for the entire name)

Ex : MYCONSTANT = "123456789 "

- Ruby is a dynamically typed language

Ex : a= 10
b=20
c = 30
d = 40

a, b, c, d = 10, 20, 30, 40
(parallel assignment)

Identifying a Ruby Variable Type

- *kind_of?* Method

Ex:

```
puts x.kind_of? Integer
puts x.kind_of? String
```

Converting Variable Values

Ex : x = 10

x.to_f → float

x.to_s → String

1234.to_s(2) →binary

1234.to_s(8)

1234.to_s(16)

Variable Scope

- find out the scope of a variable is to use the *defined?* method

Ex:

```
x = 25
puts defined? x

$x = 25
puts defined? $x
```

Name Begins With	Variable Scope
`$`	A global variable
`@`	An instance variable
`[a-z]` or `_`	A local variable
`[A-Z]`	A constant
`@@`	A class variable

Declaring and Calling a Ruby Method

```
def displayStrings( *args )
  args.each {|string| puts string}
end

displayStrings("one")
displayStrings("one", "two","three")
```

```
def name ( arg1, arg2, arg3, ... )
   .. ruby code ..
   return value
end
```

Ex:

```
def add(val1, val2 )
  result = val1 + val2
  puts result
end

puts add(10,20)
```

Ranges

- consisting of a start value, an end value and a range of values in between.

Ex: (1..10)

(1…10)

- **.to_a** method (converts to array)

Ex: ('a'..'l').to_a

Ex: print ('a'..'e') === 'c‘

Ex: Ranges in case statements

```
score = 90
result = case score
          when 0..40 then "Fail"
          when 41..60 then "Pass"
          when 61..70 then "Pass with Merit"
          when 71..100 then "Pass with Distinction"
          else "Invalid Score"
         end
puts result
```

Arrays

- A Ruby array is an object that contains a number of items. Those items can be variables (such as String, Integer, Fixnum Hash etc) or even other objects.

days =Array["Mon", "Tues", "Wed", "Thu", "Fri", "Sat", "Sun"]
days.size
days.**at**(2)
days[-1]
days. index("Wed")
days[1, 3] ←→ days.**slice**(1..3)
days1 = ["Mon", "Tue", "Wed"] → array combining
days2 = ["Thu", "Fri", "Sat", "Sun"]
days = days1 + days2 ←→days = days1.concat(days2)
days1 = ["Mon", "Tue", "Wed"]
days1 << "Thu" << "Fri" << "Sat" << "Sun " → array appending

Arrays(cont.)

```
flowers = ["Rose", "Daisy"]   → push,pop operations
flowers.push "Tulip"
flowers.push "Violet"
print flowers

flower = flowers.pop
puts  flowers
flowers.delete_at(1)
puts  flowers
flowers.delete("Tulip")
puts  flowers
```

Arrays(cont)

→ Sorting is very easy.

```
numbers = [12,1,-3,145,78,63,28]
puts numbers.sort
```

Some Operators

- Combined operators

```
x = 10
x += 5
y = 20
y -=10
x = 10;y = 5;
x /= y;

a, b, c = 10, 20, 30
```

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators

Comparison Operators

```
print 1.eql? 2
puts 50 <=> 50
puts 30 <=> 50
puts 70 <=> 50
```

Comparison Operator	Description
==	Tests for equality. Returns *true* or *false*
.eql?	Same as ==.
!=	Tests for inequality. Returns *true* for inequality or *false* for equality
<	Less than. Returns *true* if first operand is less than second operand. Otherwise returns *false*
>	Greater than. Returns *true* if first operand is greater than second operand. Otherwise returns *false*.
>=	Greater than or equal to. Returns *true* if first operand is greater than or equal to second operand. Otherwise returns *false*.
<=	Less than or equal to. Returns *true* if first operand is less than or equal to second operand. Otherwise returns *false*.
<=>	Combined comparison operator. Returns 0 if first operand equals second, 1 if first operand is greater than the second and -1 if first operand is less than the second.

Flow Control

Ex: if 10 < 20
  print "10 is less than 20 "
end
→print "10 is less than 20" if 10 < 20

if customername == "Ahmet"
    print "Hello Ahmet!"
else
    print "You're not Ahmet! Where's he?"
end

Flow Control(cont.)

```
customername = "Veli"
if customername == "Ahmet"
  print "Hello Ahmet!"
elsif customername ==" Fatma "
  print "Hello Fatma!"
elsif customername == "Marc"
  print "Hello Marc!"
else
  print "Hello guest"
end
```

Flow Control(cont.)

Conditional(Ternary) operator

```
customername = "Veli"
name = customername == "Fred" ?  "Hello Fred" :  "Who are you?"
puts name
```

Iterators

- for

Ex: for i in 1..4
  puts "i = #{i}"
end

Ex: limit = 10
for i in 1..limit
  puts "i = #{i}"
end

Iterators(cont.)

- loop

Ex: i=0

```
loop do
  break if i > 5
  puts i
  i += 1
end
```

Iterators(cont.)

- while

Ex : i = 0

```
while i < 5
i = i+1
puts i
end
```

Iterators(cont.)

- times

Ex: n = 10

n.times { |i| print i} ←→ 10.times { |i| print i}

Iterators(cont.)

- each

Ex: ('a'..'z').each {|ch| print ch}

Ex: colors = ["pink","purple","black","white"]
colors.each{|kind| puts kind}

Iterators(cont.)

- upto

```
n=0 ; max=10
n.upto(max) {|num| print num}
```

Some Math Functions and Methods

```
puts Math.constants
puts Math::PI , Math::E
puts Math.sqrt(144)
puts Math.exp(2)
```

Date and Time

```
require 'date'
date = Date.new(2008, 12, 22)
puts date.day , date.month, date.year
date = DateTime.now
puts date
```

Random numbers

```
r = Random.new
print  r.rand(10)
print r.rand(10...42)
```

Hashes

- called a dictionary or an associative array.

```
dict = {}  # create a new dictionary
dict['H'] = 'Hydrogen' #associate the key 'H' to the value 'Hydrogen'
dict['He'] = 'Helium'
dict['Li'] = 'Lithium'
p dict['H']       # prints "Hydrogen"
p dict.length
p dict.values
p dict.keys
p dict
```

Objects and Classes

```
class HelloWorld
  def hello()
    puts "hello"
  end
end

c = HelloWorld.new()
c.hello
```

Objects and Classes(cont.)

```
class BankAccount
      def initialize ()
      end

      def test_method
            puts " bank account test"
      end
end

account = BankAccount.new()
account.test_method
```

Objects and Classes(cont.)

```
class BankAccount

  def accountNumber
    @accountNumber = "1111111"
  end

  def accountName
    @accountName = "John Smith"
  end

  def initialize ()
  end

  def test_method
    puts  " bank account test"
    puts accountNumber
  end
end

account = BankAccount.new()
puts account.accountNumber
puts account.accountName
```

Objects and Classes(cont.)

```
class BankAccount

  def accountNumber
      @accountNumber
  end

  def accountNumber=( value )
      @accountNumber = value
  end

  def accountName
      @accountName
  end

  def accountName=( value )
      @accountName = value
  end
end
account = BankAccount.new()
account.accountNumber = " 456789"
account.accountName = " Mary Parker"
puts account.accountNumber
puts account.accountName
```

Objects and Classes(cont.)

```
class BankAccount

  def interest_rate
      @@interest_rate = 0.2
  end

  def accountNumber
      @accountNumber
  end

  def accountNumber=( value )
      @accountNumber = value
  end

  def accountName
      @accountName
  end

  def accountName=( value )
      @accountName = value
  end

 def calc_interest ( balance )
      puts balance * interest_rate
end
end
```

Objects and Classes(cont.)

```
account =BankAccount.new()
account.calc_interest( 1000 )
```

Conclusion

- Some core ruby components is sampled.

Hmw

- You will have an hmw !

Due Date : October 20, 2014 Monday at 8pm

Acknowledgements

This slides is collected from many sources.
Thanks to all of their authors.