Ruby Review SL LIJIA XU

dynamic, everything is an object, Ruby on Rails ⇓ web framework

puts ⇒ add newline print ⇒ no new line

```
# inline comments or =begin
                      =end
x = 8 variable
X = 8 constants
```

All vars in Ruby can be of all types puts "is#{x}"

```
puts 2**3 # 8
parallel Assign ⇒ x, y, z = 1,2,3
                  a, b = b, a
puts "Ruby\'s syntax is \n fun" ⇒ Ruby's syntax is
                                   fun
puts '1'*2 = '11'
gets.chomp # does not include newline
gets.to_i # convert to int
```

only false and nill are falsey (even 0 is truthy)

* "false" is truthy

```
puts 3.eql?(3.0) #false
```

```
a=42
unless a<10
  puts "Yes"
else
  puts "No"
end #outputs "Yes"
```

```
a=2
case a
when 0,1
  puts "are"
when 2,3,4
  puts "two"
else
  puts "three"
end
```

```
x=0
while x<10
  puts x #0-9
  x+=1
end
```

```
a=0
until(a>10
  puts "a=#{a}"
  a+=2
end
```

```
(1...2) ⇒ 1
(1..2) ⇒ 1,2
to array
a=(1...3).to_a [1,2]
```

```
age=42
case age
when 0..14
  puts "~"
when 15..80
  puts "~"
else
  puts "..."
end
```

```
for i in (1..10)
  puts i
end # 1 to 10
```

break, next loop

```
x=0
loop do
  puts x
  x+=1
  break if x>10
end
```

Arrays ⇒ nums = ["1","2"]

Can contain different types

```
arr=[5, "Dave", 5.00, false]
Add new in arr ⇒ arr<<8
Insert with index 2
arr.insert(2, 8)
remove last
arr.pop
delete ⇒ arr.delete_at(2)
A range ⇒ print arr[1..2]
```

Hashes # associative arrays, maps, or dictionaries

```
ages = {"A" ⇒28, "B"⇒26, "C"⇒32}
puts ages["A"]
Symbols: a = :id # can not be changed
Use of symbols: puts h[:age]
h = {:name => "Dave", :age=28}
simplelify version:
h = [name: "Dave", age:28]
puts h[:age]
hash.delete(key)
hash.key(value) #hash.keys #hash.values
hash.invert # hash.values #hash.length
```

Iterators

```
arr =[2,4,6]
sum=0
arr.each do |x|
  sum+=x
end
puts sum #12
```

```
hash = {A:20, B:30, C:40}
hash.each do |key, value|
  puts "#{key} ⇒ #{value}"
end # simplify do ~ end 2
hash.each{ |key, value| puts
"#{key} ⇒ #{value}"}
```

```
10.times do
  puts "Hi"
end
```

Letter Frequency counter:

```
text = "a b cc dde"
text.downcase!
freqs = {}
freqs.defaults=0
text.each_char{ |char|
freqs[char] += 1}
("a".."z").each{|x|
puts "#{x}: #{freqs[x]}"}
```

```
def say
  puts "Hi"
end
```

Default Parameters:

```
def sum(a,b=8)
  puts a+b
end
x=5
sum(5) # 13
```

```
def sum(a=8,b)
  puts a+b
end
x=5
sum(x) #13
```

optional Parameters

```
def someMethod(*p)
  puts p
end # p is a Array
someMethod(25, "Hello", true)
```

```
def squares(a, b, c)
  return a*a, b*b, c*c
arr= squares(2,3,6)
#[4, 9, 36]
```

```
def demo(a, b)
  a = b-2
  b = a-3
end
#outputs
  ⇓
⇒ puts demo(5,6) ⇒ default return last line
```

Chaining Methods:

```
def add(a,b)
  a+b
end
def mult(a,b)
  a*b
end
def square(x)
  x**x
end
x = mult(add(2,3),add(4,7))
square(4).times{puts "Hi"}  # x ⇒ 55
```

scope: Local, Global, Instance, Class

```
Global Var ⇒ $x = 42
```

Classes Initialize method:

```
class Person
  def initialize
    puts "Hi there"
  end
end
```

Accessors attr_accessor

Instance Variables: preceded by the at sign @

```
class Animal
  @age = 0 # default value
  def initialize(name, age)
    @name = name
    @age = age
  end
end
```

```
class Person
  att_accessor :name, :age
  def initialize(name,age)
    @name = name
    @age = age
  end
end
p = Person.new("Bob")
P.name = "David"
Puts P.name
```

```
#also
attr_reader
attr_writer
```

```
# even no return key word, Ruby Auto return the last line of code
```

use self. to call methods (instance)

use @ to call instance vars

class method and class variable:

```
calass Person
  @@count = 0
  def initialize
    @@count += 1
  end
  def self.get_count
    @@count
  end
end
p1 = Person.new
p2 = person.new
puts Person.get_count
# outputs 2
```

Class Constants

```
class Calc  # Ruby Captalize for Constants
  PI = 3.14
end
puts Calc::PI # output 3.14
```

```
puts p is the same as puts p.to_s
# define own to_s
```

Inheritance:

```
class Dog < Animal
  # some code
end
```

super in a method of the subclass, method of the same name gets called from superclass.

*super is more commonly used in the initialize method.

```
class Animal
  def speak
    puts "Hi"
  end
end
class Cat < Animal
  def speak
    super
    puts "Meow"
  end
end
c = Cat.new
c.speak # Hi
        # Meow
```

```
# Instance vars always private
```

operator overloading c=a+b

```
a = Shape.new(2,3) b = Shape.new(5,6)
class Shape
  attr_accessor :h, :w
  def initialize(h, w)
    self.h = h
    self.w = w
  end
  def +(other)
    shape.new(self.h + other.h,
    self.w + other.w)
  end
end
```

public private protected: access subclasses

Modules: a collection of methods (mixins)

```
class cat
  attr_accessor :name, :age
  include Comparable
  def initialize(n, a)
    self.name = n
    self.age = a
  end
  def <=>(other)
    self.age <=> other.age
  end
end
c1 = Cat.new("A", 3)
c2 = Cat.new("B", 4)
puts c1<c2
# true
```

<=> Comparable (<, <=, ==, >=, >)

name spacing: organizing similar classes in a module

```
module Mammal
  class Dog  # can be many classes
    # codes
  end
end
a = Mammal::Dog.new
```

Mixins

```
module Flyable
  def fly
    puts "flying"
  end
end
```

Struct* is a built-in* Ruby class*

Name spacing methods #self

```
module MyMath
  PI = 3.14
  def self.square(x)
    x**x
  end
end
puts MyMath.square(15)
put MyMath::PI
```

```
Point = Struct.new(:x, :y)
point1 = Point.new(0,0)
point2 = Point.new(2,3)
puts point2.y
# outputs 3
```

OpenStruct (OStruct)

* must including required libraries

```
# Ostrust isn't as fast as Struct but mor flexible
```

```
require "ostruct"
person = OpenStruct.new
person.name = "John"
person.age = 42
puts person.age #42
```

initialize Ostruct with hash ⇒ require "ostruct"

```
person = OpenStruct.new(name: "John", age: 42)
puts person.age #42
```

```
t = Time.now
puts t.month
puts t.day
```

```
greet = Proc.new do |x|
  puts "Hi #{x}"
end
greet.call "Mary" # "Hi Mary"
```

Procs keyword

Procs can be passed into Methods

```
greet = Proc.new do |x|
  puts "welcome #{x}"
end
people = ["David", "Amy", "John"]
goodbye = Pro.new do |x|
  puts "Goodbye #{x}"
end
def say(arr. proc)
  arr.each{|x| proc.call x}
end
say(people, greet)
say(people, goodbye)
```

```
def calc(proc)
  start = Time.now
  proc.call
  dur = Time.now - start
end
```

```
someProc = Poc.new do
  num=0
  100000.times do
    num=num+1
  end
end
#puts calc(someProc)
```

lambda is an istance of the Proc class

```
talk = lambda{
  puts "Hi"}
equal
talk = ->(){puts "Hi"}
talk.call
```

```
#lambda check the number of arguments but procs do Not
```

proc, lambda different "return"

```
talk = lambda{|x| puts "Hello #{x}"}
talk_proc = Proc.new{|x| puts "Hello #{x}"}
talk.call = "David"
talk_proc.call = "Amy"
talk_proc.call
# Hello
talk.call #error
```

```
file = File.new("test.txt", "w+")
or
file = File.open("filename", "w+"
r r+
w w+
a a+
file.close # must close after modify
```

```
file.puts("some txt")
File.delete("test.txt")
File.open("file.txt", "w+"){
|file| file.puts("some txt")}
```

```
File.open("test.txt") if
File.file?("text.txt")
file.size
puts File.zero?("test.txt")
```