1. Ruby không có kiểu dữ liệu. Ta có thể gán cho biến bất kì giá trị nào.

VD: a = “Pie” , a = 4, a = true.

2. Chuỗi có thể dùng phép cộng để ghép chuỗi.

3. Mọi thứ trong Ruby đều là object.

VD: “Hello”.upcase => HELLO ; -32.abs => 32. (Lưu ý, ko có dấu () ).

4. Phân biệt puts and print.

The print method works just like puts, except that puts adds a newline character at the end of the string (if it doesn't already have one) to skip to the following line, whereas print doesn't

5. gets: nhập input từ console.

VD: input = gets

6. Dùng biến trong chuỗi => Dùng String interpolcation #{ variable }

VD: puts "The answer is #{6 \* 7}." => The answer is 42.

Lưu ý: Cách dùng này chỉ được áp dụng với dấu nháy kép. Với dấu nháy đơn, nó sẽ thành The answer is #{6 \* 7}.

7. Method chomp: Với mỗi #{ variable } thì phía cuối nó sẽ là \n. Ta sẽ xét VD sau:

input = Quyen

puts "Welcome, #{input}!!!"

=> Welcome, Quyen

!!!

Ta thấy !!! nhảy xuống dưới. Khi này, dùng input.chomp sẽ giải quyết bỏ đi \n.

8. Để xem 1 object có những method nào, ta dùng object.methods

VD: 42.methods => to\_s, abs, even,......

Cách xem object đang là loại class nào, ta dùng object.class

VD: 32.class => Fixnum

9. Điểu kiện IF

- if elsif thay vì if else

- unless = if not

10. Define function

**10.1 Syntax**

def [tên function] (param 1, param 2)

.....

end

VD:

Def greet (name)

puts “Welcome, #{ name }”

end

greet(Quyen)

=> Welcome, Quyen

**10.2 Gán dafault value cho param.**

VD:

Def greet (name = “Nobody”)

puts “Welcome, #{ name }”

end

greet

=> Welcome, Nobody

Lưu ý: Nếu có nhiều param, thì những param có default value nên để cuối cùng.

VD: def greet (age, location, name = “Nobody”)

**11. Class**

**11.1 Define class:**

class [tên class]

....

end

VD:

class Dog  
 def talk  
 puts "Bark!"  
 end  
 def move(destination)  
 puts "Running to the #{destination}."  
 end  
end

dog = Dog.new  
dog.talk

**11.2 Instance variables**

|  |  |
| --- | --- |
| attr\_writer :name | def name=(new\_value)  @name = new\_value end |
| attr\_reader :name | def name  @name end |
| attr\_accessor :name | def name=(new\_value)  @name = new\_value end def name  @name end |

An instance variable looks just like a regular variable, and follows all the same naming conventions. The only difference in syntax is that its name begins with an "at" symbol (@)

Tạo nhiều instance variable với method set và get: attr\_accessor :variable1, :variable2, :variable3,.....

VD:

class Dog  
 attr\_reader :name, :age

def name=(value)  
 if value == ""

raise "Name can't be blank!"

end

@name = value

end  
 def age=(value)

if value < 0

raise "An age of #{value} isn't valid!"

end

@age = value

end  
 def report\_age  
 puts "#{@name} is #{@age} years old."  
 end  
 def talk  
 puts "#{@name} says Bark!"  
 end  
 def move(destination)  
 puts "#{@name} runs to the #{destination}."  
 end  
end

Daisy = Dog.new

Daisy.name = “Daisy”

Daisy.age = 4

Daisy.talk

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vì ta cần validate cho instance variable nên ta chỉ nên dùng attr\_reader.

**11.3 Kế thừa**

**- Syntax: Subclass < Superclass**

**- Lưu ý khi dùng tên instance variable. Xét ví dụ sau:**

class Person  
 def name=(new\_value)  
 @storage = new\_value  
end  
def name  
 @storage  
 end  
end

class Employee < Person  
 def salary=(new\_value)  
 @storage = new\_value  
end  
def salary  
 @storage  
 end  
end

employee = Employee.new  
employee.name = "John Smith"  
employee.salary = 80000  
puts employee.name

=> 80000

Rõ ràng là biến name đã bị ghi đè, nguyên nhân là do lớp cha và lớp con cùng dùng @storage. Từ đó rút ra KN là nên đặt tên trùng với tên biến muốn gán. name sẽ là @name, salary là @salary.

**11.4 toString trong Ruby.**

**Override method to\_s.**

**VD:**

class Dog < Animal  
 def to\_s  
 "#{@name} the dog, age #{age}"  
 end  
end

puts lucy.to\_s, rex.to\_s

**12 Chia lấy kết quả là số thập phân => ít nhất 1 phần tử là số float.**

VD: 12/5 = 2 ; 12/5.0 = 2.4

Nếu muốn giới hạn độ rộng của kết quả, ta dùng format.

VD:

result = format("Rounded to two decimal places: %0.2f", 3.14159265)  
puts result

=> Rounded to two decimal places: 3.14

**13. Constructor**

Constructor trong Ruby chính là initialize. Cách dùng cũng giống như khai báo 1 method.

VD:

def initialize  
 @name = "Anonymous"  
 @salary = 0.0  
end

*hoặc*

def initialize(name, salary)  
 @name = name  
 @salary = salary  
end

employee = Employee.new("Amy Blake", 50000)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Nhưng để tránh trường hợp quên không truyền param vào, khi nãy sẽ có lỗi. Cách khắc phục là nên khai báo default value cho initialize. VD:

def initialize(name = “Anonymous”, salary = 0)  
 @name = name  
 @salary = salary  
end

employee = Employee.new

**14. Self**

Self ở đây tương đương với this, nó mang ý nghĩa là 1 đối tượng ta đang xét.

Giả sử ta muốn validate cho method initialize, ta làm như sau:

class Employee

attr\_reader: name

def name = (name = “Anonymous”) (\*1)

if name = “ ”

puts “Name can not be blank !”

else

@name = name

end

end

def initialize (name = “Anonymous”)

self.name = name (\*2)

end

John = Employee.new(“ ”)

=> Name can not be blank !

**Từ (\*2) ta thấy name được gán vào self.name, nó tương đương ta gọi hàm set name và truyền param name vào ở (\*1).**

**15. super**

class Employee  
 ...  
 def initialize(name = "Anonymous")  
 self.name = name  
 end  
 ...  
end

class SalariedEmployee < Employee  
 ...  
 def initialize(name = "Anonymous", salary = 0.0)  
 super  
 self.salary = salary  
 end  
 ...  
end

salaried\_employee = SalariedEmployee.new("Jane Doe", 50000)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Khi khởi tạo đối tương, ta sẽ nhận được lỗi:**

**in** initialize**: wrong number of arguments (2 for 0..1)**

**Nguyên nhân là do khi super không có params, nó tự động sẽ lấy hết params trong method initialize, mà với superclass thì super chỉ nhận 1 param.**

**Sửa lại đúng sẽ là super(name)**

**------**

**16. Class method**

**Khai báo trong class:**

self.[tên method](param1, param2,…)

…

end

**Cách dùng:**

[Tên class].[tên method](param1, param2,…)

*VD:*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class HourlyEmployee < Employee

def self.security\_guard(name)  
 HourlyEmployee.new(name, 19.25, 30)  
 end  
 def self.cashier(name)

HourlyEmployee.new(name, 12.75, 25)  
 end  
 def self.janitor(name)  
 HourlyEmployee.new(name, 10.50, 20)  
 end

attr\_reader :hourly\_wage, :hours\_per\_week

def hourly\_wage=(hourly\_wage)  
 if hourly\_wage < 0  
 raise "An hourly wage of #{hourly\_wage} isn't valid!"  
 end  
 @hourly\_wage = hourly\_wage  
 end  
 def hours\_per\_week=(hours\_per\_week)  
 if hours\_per\_week < 0  
 raise "#{hours\_per\_week} hours per week isn't valid!"  
 end  
 @hours\_per\_week = hours\_per\_week

end

def initialize(name = "Anonymous", hourly\_wage = 0.0, hours\_per\_week = 0.0)

super(name)

self.hourly\_wage = hourly\_wage  
 self.hours\_per\_week = hours\_per\_week

end

def print\_pay\_stub

print\_name

pay\_for\_period = hourly\_wage \* hours\_per\_week \* 2  
 formatted\_pay = format("$%.2f", pay\_for\_period)  
 puts "Pay This Period: #{formatted\_pay}"  
 end  
end

angela = HourlyEmployee.security\_guard("Angela Matthews")  
ivan = HourlyEmployee.cashier("Ivan Stokes")  
angela.print\_pay\_stub  
ivan.print\_pay\_stub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**------**

**17. Block**

**17.1**

def my\_method(&my\_block)  
 puts "We're in the method, about to invoke your block!"  
 my\_block.call  
 puts "We're back in the method!"

my\_block.call  
end  
my\_method do  
 puts "We're in the block!" (\*)  
end

=>***Kết quả:***

We're in the method, about to invoke your block!"

We're in the block!"

We're back in the method

We're in the block!"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Phần màu vàng (\*) được xem như là param đối với phương thức my\_method.

**17.2 Block có tham số:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def give(&my\_block)  
 my\_block.call("2 turtle doves", "1 partridge")  
end  
give do |present1, present2|  
 puts "My method gave to me..."  
 puts present1, present2  
end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

=> ***Kết quả:***

My method gave to me...  
2 turtle doves  
1 partridge

17.3 Yield

- Yield được dùng ở phần định nghĩa method.

- Khi dùng yield, ta sẽ bỏ param &block và .call

VD:

def give(&my\_block)  
 my\_block.call("2 turtle doves", "1 partridge")  
end

def give  
 yield "2 turtle doves", "1 partridge"  
end

**17.4 Sử dụng { } thay cho do…end**

- Chỉ sử dụng { } khi body của block chỉ có 1 dòng. Nếu nhiều hơn 1 dòng thì dùng do...end

**VD:**

def take\_this  
 yield "present"  
end

take\_this do |thing|  
 puts "do/end block got #{thing}" => 2 dòng  
end

take\_this { |thing| puts "brackets block got #{thing}" } => 1 dòng

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

VD sau là cách dùng mặc dù đúng syntax nhưng nhìn xấu:

take\_this do |thing| puts "do/end: got #{thing}" end

take\_this { |thing|  
 puts "brackets: got #{thing}"  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

17.5 Each method

**Cách dùng:**

[array].each |item| do

… *hoặc* [array].each { |item| …. }

end

**VD:**

alphabets = [a", "b", "c"]

alphabets.each { |param| puts param }

=> a

b

c

VD toàn diện về method each:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def total(prices)

amount = 0  
 prices.each do |price|

amount += price

end  
 amount

end

def refund(prices)

amount = 0  
 prices.each do |price|

amount -= price

end  
 amount

end

def show\_discounts(prices)  
 prices.each do |price|

amount\_off = price / 3.0

puts format("Your discount: $%.2f", amount\_off)

end

end

prices = [3.99, 25.00, 8.99]

puts format("%.2f", total(prices))  
puts format("%.2f", refund(prices))  
show\_discounts(prices)

=> ***Kết quả:***

37.98  
-37.98  
Your discount: $1.33  
Your discount: $8.33  
Your discount: $3.00

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**II. Lời khuyên**

**1. Không cần thiết dùng dấu ( ) ở function.**

**Leave parenthesis off of a method call if there are no arguments. You can leave them off for method calls where there *are* arguments as well, but this can make some code more difficult to read. When in doubt, use parenthesis!**

**2. Không cần dùng dấu chấm phẩy ở cuối mỗi câu lệnh.**

**3. Tránh việc tạo 1 function với nhiều câu if elsif, nếu được ta nên tạo class.**

**VD:**

def talk(animal\_type, name)  
 if animal\_type == "bird"  
 puts "#{name} says Chirp! Chirp!"  
 elsif animal\_type == "dog"  
 puts "#{name} says Bark!"  
 elsif animal\_type == "cat"  
 puts "#{name} says Meow!"  
 end  
end  
def move(animal\_type, name, destination)  
 if animal\_type == "bird"  
 puts "#{name} flies to the #{destination}."  
 elsif animal\_type == "dog"  
 puts "#{name} runs to the #{destination}."  
 elsif animal\_type == "cat"  
 puts "#{name} runs to the #{destination}."

end  
end

def report\_age(name, age)  
 puts "#{name} is #{age} years old."  
end

**4. DRY: Don’t repeat yourself**

**III Resource**

<https://daynhauhoc.com/t/ruby-on-rails-8-cach-hoc-tot-nhat/947>