Clase 4: Clases y Objetos en Ruby

Clase 4: Contenido

- Recapitular clase anterior: else, elsif, for, while, until, array, each, hash
- Clases y objetos
- Tarea

Launchpad

Recapitulando la Clase 3

- Else
- Elsif
- Loops: for, while, until, each
- Array
- Hash
- Revisión de la tarea

Launchpad

Introducción a OOP

- OOP significa Object Oriented Programming
- En Ruby, todo es un objeto
- Abre la terminal, crea un string y llama a .class
- Llama a .class sobre el .class sobre el string
- Llama a .class sobre el .class sobre el .class sobre el string
- Como ves, Ruby organiza la información en lo que llamamos clases, y cada instancia de las diferentes clases se llama objeto

Introducción a OOP

- Los objetos en Ruby son como los objetos en la vida real, tienen atributos y tienen funciones

- Ej.: un lápiz:
  - Atributos: plástico, liviano, rojo, a tinta, etc.
  - Funciones: escribir, hacer un moño, marca la última página leída de un libro, etc.

- Ej.: un string:
  - Atributos: número de caracteres
  - Funciones: .reverse, .capitalize, etc.

Herencia

- Todas las instancias de una clase (objetos) tienen los atributos y funciones que esa clase les permite tener, además de los atributos y funciones que heredan de sus clases padre
- Un string común y corriente (ej.: “hola”) pertenece a la clase String, que a su vez hereda de la clase padre Class, que a su vez hereda de la clase padre Basic Object
- El concepto de herencia se va a tornar fundamental cuando trabajemos con Rails

Class method e instance method

- Para crear un nuevo string podemos usar un literal constructor o un name constructor (heredado de clase padre Class)

```
#literal constructor
s = "Hello"
#name constructor
s = String.new("Hello")
#literal constructor
my_array = [1,2,3]
#name constructor
```

- .new (name constructor) es un ejemplo de un class method, porque se llama sobre la clase String
- “Hello”.length (# de caracteres) sería un ejemplo de un instance method, porque se llama sobre un string en particular, no la clase String

Definiendo una clase

- Hasta ahora hemos usado clases prefabricadas de Ruby, como String, Array y Hash
- Probemos creando una clase propia, Word con una función very_long?

```ruby
class Word

    def very_long?(string)
        if string.length >= 10
           puts "true"
        end
    end

end


w = Word.new
puts w.very_long?("superduperlongword")


f=Word.new
puts w.very_long?("short")
```

Llamar .length sobre un w arroja un error porque Word es una Class y no tiene las mismas funciones que String a menos que lo explicitemos a través de una herencia

Launchpad

Heredando una clase

```ruby
class Word < String

   def very_long?(string)

       if string.length >= 10

           puts "true"

       end

   end

end


w = Word.new("superduperlongword")

puts w.very_long?("superduperlongword")

   #true

puts w.class

   #Word

puts w.class.superclass

   #String

puts w.length

   #10
```

Launchpad

Objeto: instancia de una clase (1/2)

- Definamos la clase Persona con los siguientes atributos y funciones

```
class Person

    attr_accessor :first_name, :last_name, :gender, :age

    def initialize(first_name, last_name, gender, age)

         @first_name = first_name

         @last_name = last_name

         @gender = gender

         @age = age

     end

    def introduction

         puts "#{@first_name} #{@last_name} is a #{@age} year old #{@gender}"

    end

end
```

Launchpad

Objeto: instancia de una clase (2/2)

- En el ejemplo anterior:
  - La primera línea, attr_accessor, indica qué información almacenaremos sobre las personas que creemos en el sistema. Esto crea métodos “getter” y “setter”, para modificar y leer la información sobre la Persona cuando queramos
  - El primer método, initialize, es llamado cuando el objeto es creado. Aquí se le asignan valores entregados a los atributos definidos en attr_accesor
  - Finalmente, la clase define un método introduction que retorna una impresión (puts) de una introducción bien formateada de la Persona en particular

```
p = Person.new(“John”, “Smith”, "male", 34)

p.introduction
```

Launchpad

Ejercicio #1

Instrucciones:

- Crea tu propia clase para un animal. Entrégale atributos y una función. Luego crea una instancia.

Launchpad

Más sobre herencia

- Recuerdas cuando heredamos las funcionalidades de String sobre Word?
- Ahora podemos crear una nueva clase Student que herede las funcionalidades de Person

```
class Student < Person

    def learning

            puts "#{@first_name} is learning!"

    end

end

s = Student.new("Jane", "Doe", "female", 33)

puts s.learning

s.introduction
```

- Ahora, además de usar learning sobre una instancia de Student, podemos además usar introduction porque lo hereda de Person

Launchpad

Ejercicio #2

Instrucciones:

- Crea una clase Teacher que herede de Person. Crea una función y ponla a prueba.

Launchpad

Ejercicio #3

Instrucciones:

- Crea un programa que:
  - Liste muchos items
  - Permita a un usuario seleccionar un item y ver el conteo del inventario, cambiar el número del inventario, borrar el item o incluso cambiar su nombre
  - Crea un nuevo item y entrégale un conteo de inventario, y permite que este item esté ahora en la lista de items que puedes mostrar
  - Hace un loop hasta que el usuario ya no quiera continuar editando la lista

Clase 4: Clases y Objetos en Ruby