DCC

DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2143 - INGENIERÍA DE SOFTWARE

2023 - 1º SEMESTRE

CLASE PRÁCTICA 1:

- Aspectos básicos de Ruby
- Métodos y bloques
- Arrays and hashes
- Estructuras de control
- Programación orientada a objetos

Alison Fernandez Blanco

CLASE PRÁCTICA 1:

- **Aspectos básicos de Ruby**
- Métodos y bloques
- Arrays and hashes
- Estructuras de control
- Programación orientada a objetos

3 ¿Qué es Ruby?

- Ruby es un lenguaje orientado a objetos.
  - Todo en Ruby es un objeto.
- Ruby fue creado por Yukihiro Matsumoto y presentado públicamente en 1995.
  - Alternativa a Perl y Python.

4 Variables

En Ruby, usamos variables para asignar etiquetas a objetos en nuestro programa. Se puede asignar una etiqueta a un objeto con el operador =

Code:

```
amount = 5
puts amount
```

Output:

```
5
```

5 Variables

Tipo de variable	Sintaxis	Ejemplo
Global	Comienza con $	$debug, $plan1
Local	Comienza con minúscula o _	name, _fish
Instancia	Comienza con @	@name, @point1
Clase	Comienza con @@	@@total, @@SINGLE
Constante	Comienza con mayúscula	PI, Columns

6 Tipos de datos

En Ruby, los tipos de datos representan categorías diferentes de datos como ser números, texto, etc. Dado que Ruby está orientado a objetos, los tipos de datos admitidos se implementan como clases.

Code:

```
sentence = "Hello students"      # Una cadena
other_sentence = 'Hello world!'  # Otra cadena
our_integer = 24                     # Un entero
our_float = 24.2                     # Un real
our_bool = true                  # Un boolean
our_symbol = :puc                    # Un símbolo
```

CLASE PRÁCTICA 1:

- Aspectos básicos de Ruby
- **Métodos y bloques**
- Arrays and hashes
- Estructuras de control
- Programación orientada a objetos

8 Métodos

Los métodos hacen que el código sea reutilizable al “empaquetar” el código y ponerle un nombre. Dado que Ruby está orientado a objetos, los métodos definen comportamiento.

Code:

```
def sum (n1, n2)
    puts n1 + n2
end
sum(3,4)
sum("cat", "dog")
```

Output:

```
7
catdog
```

9 Bloques

Un bloque es una pieza de código que puede aceptar argumentos y devuelve un valor. Un bloque siempre se pasa a una llamada de método.

Code:

```
2.times { puts "hello!" }
[1,2,3].each { |number| puts "#{number} was passed to the block" }
```

Output:

```
hello!
hello!
1 was passed to the block
2 was passed to the block
3 was passed to the block
```

CLASE PRÁCTICA 1:

- Aspectos básicos de Ruby
- Métodos y bloques
- **Arrays and hashes**
- Estructuras de control
- Programación orientada a objetos

11 Arrays

Estructuras de datos que pueden almacenar varios elementos de diferentes tipos de datos. En Ruby, los arreglos empiezan con el index 0.

Code:

```ruby
words = ["one", "two", "three"] # Un arreglo de cadenas
words.append("four")                    # Agrega "four" al arreglo
puts words[0]                           # Imprime el primer elemento
puts words.length                       # Imprime el # de elementos
puts words.index("four")                # Imprime el index de "four"
```

Output:

```
one
4
3
```

12 Arrays

Estructuras de datos que pueden almacenar varios elementos de diferentes tipos de datos. En Ruby, los arreglos empiezan con el index 0.

Code:

```
matrix = Array.new(2){Array.new(4){0}}  # Una matriz 2 x 4 con 0's
matrix[0][2] = 1                        # Actualiza valor en m[0][2]
puts matrix[0][2]                       # Imprime el valor en m[0][2]
puts matrix.length                      # Imprime el # de arreglos
puts matrix[0].length                   # Imprime el # de items en m[0]
```

Output:

```
1
2
4
```

13 Hashes

Estructuras de datos en donde se asignan valores a las llaves. En Ruby, se puede acceder al valor mediante la llave y tanto el valor como la llave son de cualquier tipo.

Code:

```
dict = {"one" => "eins", "two" => "zwei"}     # Un diccionario
dict["one"] = "uno"                          # Actualizar valor
puts dict["one"]                                 # Imprime el valor
puts dict.keys                               # Imprime las llaves
puts dict.length                                 # Imprime # de pares
```

Output:

```
uno
one
two
2
```

14 Métodos

Code:

```
['cat', 'dog'].each {|name| print name, " " }
puts [1,2,3].find { |x| x > 1}
puts (1...10).find_all { |x| x < 3}
puts [1...10].min()
```

Output:

```
cat dog 2
[1, 2]
1
```

CLASE PRÁCTICA 1:

- Aspectos básicos de Ruby
- Métodos y bloques
- Arrays and hashes
- **Estructuras de control**
- Programación orientada a objetos

16 If Control

La estructura de control más básica es el If. A continuación veremos ejemplos de su uso:

Code:

```
if number.between?(1, 10)
  puts "The number is between 1 and 10"
elsif number.between?(11, 20)
  puts "The number is between 11 and 20"
else
  puts "The number is bigger than 20"
end
```

Output:

```
The number is between 1 and 10
```

17 While loop

Con while se ejecuta un conjunto de líneas de código siempre que la condición especificada sea verdadera.

Code:

```
while x < y do
    puts x + " is less than " + y
    x += 1
end
```

18 For loop

Con for se ejecuta un conjunto de líneas de código cierto número de veces o se itera sobre un conjunto de elementos en específico.

Code:

```
words = ["one", "two", "three"] # Un arreglo de cadenas
for word in words do
    puts word
end
```

Output:

```
one
two
three
```

CLASE PRÁCTICA 1:

- Aspectos básicos de Ruby
- Métodos y bloques
- Arrays and hashes
- Estructuras de control
- **Programación orientada a objetos**

Clases y objetos

Code:

```ruby
class Song                                          # Definición de clase
    def initialize(name, artist, duration)
      @name = name                          # Variables de instancia @
      @artist = artist
      @duration = duration
    end
end
my_song = Song.new("Bicylops", "Fleck", 260)     #Creando objeto
```

21 Sobreescritura

Code:

```
class Song                                          # Definición de clase
    …
    def to_s                                        # Override to_s
      "Song: #@name--#@artist (#@duration)"
    end
end

puts my_song.to_s                                     #Llamando a to_s
```

Output:

```
Song: Bicylops--Fleck (260)
```

Herencia

Code:

```
class KaraokeSong < Song                    # Usa < para herencia
    def initialize(name, artist, duration, lyrics)
        super(name, artist, duration)
        @lyrics = lyrics
    end
end
karaoke_song = KaraokeSong.new("My Way", "Sinatra", 225, "And
now, the...")
karaoke_song.to_s
```

Output:

```
Song: My Way--Sinatra (225)
```

Herencia

Code:

```
class KaraokeSong < Song
    …
    def to_s                          # Override to_s
        super + " [#@lyrics]"         # Call to to_s de superclass
    end
end
karaoke_song = KaraokeSong.new("My Way", "Sinatra", 225, "And
now, the...")
puts karaoke_song.to_s
```

Output:

```
Song: My Way--Sinatra (225) And now, the...
```

24 Leer atributos

Code:

```
class Song
    …
    def name
        @name          # Retorna @name
    end
end
puts my_song.name
```

Output:

```
My Way
```

Leer atributos - otra forma

Code:

```
class Song
    attr_reader :name, :artist, :duration        # Permite acceder
    …
end
puts my_song.name
```

Output:

```
My Way
```

Escribir atributos - forma simple

Code:

```
class Song
    attr_writer :duration                    # Permite escribir
    …
end
puts my_song.to_s
my_song.duration = 257
puts my_song.duration
```

Output:

```
Song: Bicylops--Fleck (260)
257
```

27 Variables de clase y métodos de clase

Code:

```
class Example
    …
    @@no_of_examples = 0              # Variable de clase
    def instance_method               # Método de instancia
    end
    def Example.class_method          # Método de clase
    end
end
```

Control de acceso

- **public:** Accesible para todos. Es el acceso dado por defecto a los métodos (menos initialize).
- **protected**: Accesible desde la clase y la subclase.
- **private:** Accesible desde dentro del objeto (desde otros métodos que define la clase).

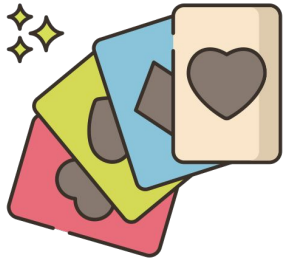
Control de acceso

Code:

```
class MyClass
protected                # subsequent methods will be 'protected'
    def method1          # will be 'protected'
    end
private                  # subsequent methods will be 'private'
    def method2          # will be 'private'
    end
public                   # subsequent methods will be 'public'
    def method3          # will be 'public'
    end
end
```

30 Ejercicios

- **Ejercicio 1 (Demostración):** Implementa el método **`max_number`**
  - Recibe un arreglo de números mayores o iguales a 0 y retorna el número mayor en el arreglo.
- **Ejercicio 2 (Por décimas extra):** Implementa el método **`completed_decks_in`**
  - Recibe un arreglo de cadenas (representan cartas) y retorna el número de mazos completos que se pueden crear con esas cartas.



31 Para la siguiente clase

- Instalar Ruby on rails (https://guides.rubyonrails.org/getting_started.html).
- Instalar Postman (https://www.postman.com/downloads/)
- ¡Traer sus computadoras para programar!

Material adicional

- Documentación de Ruby: https://www.ruby-lang.org/es/documentation/
- Prueba Ruby en tu navegador: https://try.ruby-lang.org/
- Aprende a programar: https://pine.fm/LearnToProgram/
- Otros libros: https://github.com/EbookFoundation/free-programming-books/blob/main/books/free-programming-books-langs.md#ruby

¿Consultas?