IIC 2143 - Ingeniería de Software

Ruby: Conceptos Básicos

M. Trinidad Vargas
mtvargas1@uc.cl

Ruby

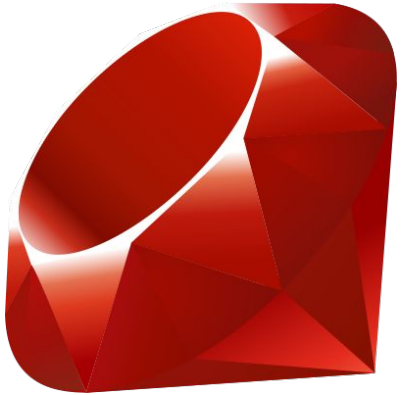
Un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad

Es un lenguaje **orientado a objetos**, todo es un objeto

Creado por el programador japonés Yukihiro Matsumoto y publicado en 1995

Inspirado en Python y Perl y se presenta como alternativa a estos lenguajes

Usaremos la **versión 3.3**



¿Cómo ejecutar Ruby?

Abrir la consola

```
irb

puts "Hello World!"
```

Ejecutar archivo

hello.rb

```
puts "Hello World!"
```

```
ruby hello.rb
```

Tipos de dato

```
# Un comentario
=begin
Un comentario de
múltiples líneas
=end
a_message = "Hello Ingeniería de Software"  # un string
another_message = 'Hello students :)'  # otro string
an_integer = 10  # un entero
a_float = 10.5  # un número de punto flotante
a_bool= true  # un booleano
```

Operadores básicos: números

```ruby
puts 1234.class  # => Integer
puts 0.3.class  # Float
puts 1 + 2
puts 0.1 + 4
puts 2 * 3
puts 5 / 2
puts 5.0 / 2
puts 5 / 2.to_f
```

Operadores básicos: strings

```ruby
puts "5" * 2
puts "4.2".to_i
puts "4.2".to_f
puts "qwer".to_f
puts "-" * 20
20.times {print "-"}
20.times {puts rand(10)}  # Entre 0 y 9
```

Operadores básicos: comparación

```ruby
10 == 10  # true
'a' == 'b'  # false
'10' == 10  # false
5 < 2  # false
2 <= 3  # true
2 < 3 && 10 > 5  # Y (and)
7 < 3 || 10 > 5  # O (or)
10 == 10  # true
10.eql?(10.0)  # false
```

Métodos

- definimos métodos entre def y end
- por defecto retorna la última línea
- para agregar un valor por defecto a un argumento se usa =valor

```
def greet(name='Pepito')
 "Welcome to Ingeniería de Software #{name} :)"
end
name = gets.chomp
puts greet(name)
> "Welcome to Ingeniería de Software Trinidad :)"
```

Rangos

Se usan para generar secuencias

- con dos puntos (..) incluye ambos límites
- con tres puntos (...) no incluye el límite superior

```
# El método to_a convierte el rango en un array
(1..9).to_a
("a"..."d").includes?("c")
# El operador === comprueba si un valor está entre los límites del rango
(1...4) === 3.141593  # true
```

Arrays

Un array es un conjunto ordenado: cada posición en la lista es una variable que podemos leer y/o escribir

```ruby
a = [1, "Ruby", 0.3, nil, 1..4]
# los índices parten desde 0
a[0]
# si el elemento no existe devuelve nil
a[5]
# podemos agregar más elementos
a[5] = 'Nuevo valor'
```

Arrays

```ruby
# %w nos permite crear un array de string
ciudades = %w{ Santiago Concepción Iquique Pucón }
# p, puts y print imprimen distintos formatos
p ciudades
puts ciudades
print ciudades
# el método each acepta un bloque de código y lo ejecuta una
# vez por cada elemento del array como parámetro
ciudades.each{|ciudad| puts '¡Me gustaba '+ ciudad +'!'}
```

Arrays

```
ciudades.sort
ciudades.length
ciudades.first
ciudades.last
ciudades.append("Valparaíso")
ciudades << "La Serena"
ciudades.index("Valparaíso")
ciudades.unshift("Santiago")
ciudades.uniq
ciudades.uniq!
```

Arrays

```ruby
ciudades.empty?
ciudades.include?("Concepción")
ciudades.push("")
ciudades.pop
ciudades.join
ciudades.join("-")
ciudades.shuffle
ciudades.select {|ciudad| ciudad.length > 8}
p ciudades.methods  # imprimimos todos los métodos
```

Todos los métodos: https://ruby-doc.org/core-3.0.1/Array.html

Hashes

Un hash es una colección de claves únicas y sus valores, similar a un diccionario

```
numeros = {"one" => "eins", "two" => "zwei"}
numeros["three"] = "drei"
numeros["one"] = "uno"  # actualizar el valor
numeros.keys
numeros.length
```

Bloques

Un bloque es una pieza de código que puede aceptar argumentos y devuelve un valor

Siempre es un parámetro en la llamada de un método

```
método {código}


método {|parámetro1, parámetro2| código}
```

Bloques

```
2.times {puts "Hello World!"}
> "Hello World!"
  "Hello World!"



["Sofía", "Tomás", "Juan"].each { |name| puts "Hello #{name}"}
> "Hello Sofía"
  "Hello Tomás"
  "Hello Juan"
```

Otros iteradores

```
1.upto(5) {|i| print i, " "}

99.downto(95) {|i| print i, " "}

50.step(80, 5) {|i| print i, " "}
```

Bloques

```ruby
def greet(name)
 puts "Hello #{name} :)"
 yield name
 puts "Goodbye #{name}!"
end

greet ("Jorge") { |name| puts "Nice to meet you #{name}"}
> "Hello Jorge :)"
  "Nice to meet you Jorge"
  "Goodbye Jorge!"
```

Bloques: alternativas

```ruby
# evita errores por llamar un bloque y que no exista
def example
 yield if block_given?
end

# puedes aceptar block como argumento explícitamente
def example(&block)
 block.call if block
end
example { puts "This is a block!" }
```

Control de flujo

En Ruby nil y false son falso, todo el resto es verdadero incluyendo 0, [], {} etc.

```ruby
condition_1 = true
condition_2 = false
if condition_1 && condition_2
 puts "Ambas condiciones verdaderas"
else
 puts "Al menos una condición es falsa"
end
```

Control de flujo

```ruby
condition_1 = false
condition_2 = false

if condition_1 || condition_2
 puts "Al menos una condición verdadera"
else
 puts "Ninguna condición verdadera"
end
```

Control de flujo

```ruby
condition_1 = false
condition_2 = false

if condition_1 && condition_2
 puts "Ambas condiciones verdaderas"
elsif condition_1 || condition_2
 puts "Al menos una condición verdadera"
else
 puts "Ninguna condición verdadera"
end
```

Control de flujo

```
número = 10
es_par = case
  when número % 2 == 0 then true
  when número % 2 != 0 then false
end
```

Bucles: For, While, Until

```ruby
for i in 1...5  # no incluye 5
  puts "Iteración #{i}"
end

i = 0
while i < 5
  puts "Iteración #{i}"
  i += 1
end
```

For and While loop

```ruby
i = 1
until i > 5
  puts "Iteration #{i}"
  i += 1
end
```

Clases y objetos

Un objeto es un contenedor de datos, que a su vez controla el acceso a dichos datos

Todo en ruby es un objeto, incluso nil.

Clases y objetos

```ruby
class Song  # definición de la clase
  def initialize(name, artist, duration)
    @name = name  # variables de instancia, empiezan con @
    @artist = artist
    @duration = duration
  end
end

my_song = Song.new("Love Story", "Taylor Swift", 235)
```

Clases y objetos

```ruby
class Song
  ...
  def to_s  # sobreescribimos to_s
    "Song: #{@name}--#{@artirst}--#{@duration}"
  end
end

puts my_song.to_s
```

Clases y objetos

```ruby
class Song
  attr_reader :name, :artist  # permite lectura
  attr_writer :name # permite escritura
  attr_accessor :duration  # permite ambos
  ...
end


puts my_song.name
my_song.name = "Love Story (Taylor's version)"
```

Clases y objetos: Herencia

```
class KaraokeSong < Song
 def initialize.new(name, artist, duration, lyrics)
   super(name, artist, duration)
   @lyrics = lyrics
 end
end

lyrics = "We were both young when I first saw you..."
karaoke_song = Song("Love Story", "Taylor Swift", 235, lyrics)
```

Clases y objetos: Herencia

```ruby
class KaraokeSong < Song        # usa < para herencia
  ...
  def to_s  # sobreescribimos to_s
    super + " [#{@lyrics}]"
  end
end

puts karaoke_song
```

Clases: Variables y métodos de clase

```ruby
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
```

Clases: Control de acceso

En ruby tenemos 3 niveles de acceso

- **Público (public):** Accesible para todos. Por defecto todos los métodos son públicos menos initialize
- **Protegido (protected):** Accesible desde la clase y subclase
- **Privado (private):** Accesible desde otros métodos dentro del objeto

Variables

```
var, _var        # variable local, empieza con [a-z] o
@var             # variable de instancia
@@var            # variable de clase
$var             # variable global
```

Clases: Control de acceso

```ruby
class Example
private  # métodos siguientes son privados
  def private_method
  end
protected  # métodos siguientes son protegidos
  def protected_method
  end
public  # métodos siguientes son públicos
  def public_method
  end
end
```

Documentación

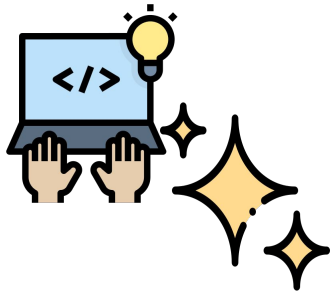
- Ruby https://www.ruby-lang.org/es/documentation/
- Aprende a programar https://try.ruby-lang.org/

Próxima clase

Clase práctica 2

No olvidar traer instalado ruby on rails y postman de escritorio en tu computador.

Traigan sus computadores para programar



Guía de instalación: https://github.com/IIC2143/Setup-Guides

Postman: https://www.postman.com/downloads/

Ejercicio

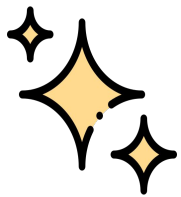
Demostración: Escriba una función llamada max_number que reciba un arreglo de enteros positivos y devuelva el mayor.

Ejemplo:

values = [8, 3, 2, 678, 1, 4, 4]

max_number(values)

> 678



Ejercicio

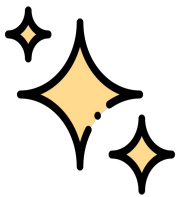
Ejercicio 1 (0.05): Escriba una función llamada count_cards_suit que recibe un arreglo de cadenas (que representan cartas) y retorna un diccionario con el número de cartas que hay en un mazo. Las llaves son 'corazon', 'diamante', 'trebol' y 'pica').

Ejemplo:

my_cards = ['1P', '2C', '13D', '12T', '3P']

count_cards_suit(my_cards)

> {'pica' => 2, 'corazon' => 1, 'diamante' => 1, 'trebol' => 1}



Ejercicio

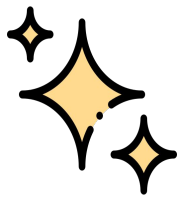
Ejercicio 2 (0.05): Escriba una función llamada is_deck_complete(cards) que recibe un arreglo de cadenas (que representan cartas) y retorna si es posible armar un mazo completo.

Ejemplo:

my_cards = ['1P', '2C', '13D', '12T', '3P']

is_deck_complete(my_cards)

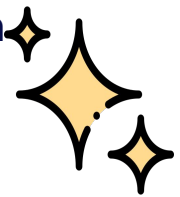
> false



Ejercicio

Ejercicio 1 (0.05): Escriba una función llamada count_cards_suit que recibe un arreglo de cadenas (que representan cartas) y retorna un diccionario con el número de cartas que hay en un mazo. Las llaves son 'corazon', 'diamante', 'trebol' y 'pica').

Ejercicio 2 (0.05): Escriba una función llamada is_deck_complete(cards) que recibe un arreglo de cadenas (que representan cartas) y retorna si es posible armar un mazo completo.



Entrega

- Un archivo .rb con los dos métodos
- Los métodos deben llamarse igual y aceptar el mismo formato que se muestra
- Se podrá entregar durante el día hasta las **22:00 hrs**