<!--Ingeniería de Software 2024-1-->

Introducción a Ruby {

<Por="Constanza Huerta y Cristóbal Moreno"/>

}

Introducción {

¿Qué es ruby?

- Simplicidad
- Sintaxis intuitiva
- Flexibilidad
- Orientado a objetos
- Uso de gemas
- Ruby on rails

}

Input {

```ruby
# Get user input
data = gets().chomp() # gets = input (leaves the "\n" character),
# chomp = strip

# Print user input previously recieved
puts(data)

# Print user input without the "\n" character
print(data)
print(", esto se imprime en la misma linea\n")

p(data)
```

}

Output{

```
1   # Get user input
2   data = gets().chomp() # gets = input (leaves the "\n" character),
3   # chomp = strip
4
5   # Print user input previously recieved
6   puts(data)
7
8   # Print user input without the "\n" character
9   print(data)
10  print(", esto se imprime en la misma linea\n")
11
12  p(data)
```

Resultado en consola:

```
ayudantia
ayudantia
ayudantia, esto se imprime en la misma linea
"ayudantia"
```

input

}

Python {

Codigo en python

```python
# Obtener la entrada del usuario
data = input()

# Imprimir la entrada del usuario
print(data)

```

}

Conversiones {

```ruby
# Define arbitrary strings that LOOK like integers
three = "3"
five = "5"

# Naive addition (returns the string concatenation)
naive = three + five

# Adds the integers after getting them from the strings
real = three.to_i() + five.to_i()  # Returns an integer

# Adds the floats after getting them from the strings
overkill = three.to_f() + five.to_f()  # Returns a float

# Log results
puts(naive) # 35
puts(real) # 8
puts(overkill) # 8.0
```

}

Output{

```ruby
# Define arbitrary strings that LOOK like integers
three = "3"
five = "5"

# Naive addition (returns the string concatenation)
naive = three + five

# Adds the integers after getting them from the strings
real = three.to_i() + five.to_i()  # Returns an integer

# Adds the floats after getting them from the strings
overkill = three.to_f() + five.to_f()  # Returns a float

# Log results
puts(naive) # 35
puts(real) # 8
puts(overkill) # 8.0
```

Resultado en consola:

```
35
8
8.0
```

}

Python {

Codigo en python:

```python
# Define cadenas arbitrarias que SE VEN como enteros
three = "3"
five = "5"

# Devuelve la concatenación de cadenas
naive = three + five

# Suma los enteros después de convertirlos desde las cadenas
real = int(three) + int(five)

# Suma los flotantes después de convertirlos desde las cadenas
overkill = float(three) + float(five)

# Imprime los resultados
print(naive)      # 35
print(real)       # 8
print(overkill)   # 8.0
```

}

Metodos de llamada{

```ruby

  puts("It's ruby time!")


  puts "It's ruby time!"
```

}

Output{

```ruby
# Calls method as expected
puts("It's ruby time!")

# Calls method WITHOUT parentheses... ¡¿WHAT?!
puts "It's ruby time!"
```

Resultado en consola:

```
It's ruby time!
It's ruby time!
```

}

Interpolación de strings {

```ruby
# Define arbitrary element years
time = 2

year = 2024
semester = 1


# Define element name
element = "Ingenieria de Software #{year}-#{semester}"


# Compose message
composition = "Llevo #{time} años esperando para hacerles esta ayudantia de #{element}"

puts composition

```

}

Output{

```ruby
# Define arbitrary element years
time = 2

year = 2024
semester = 1


# Define element name
element = "Ingenieria de Software #{year}-#{semester}"


# Compose message
composition = "Llevo #{time} años esperando para hacerles esta ayudantia de #{element}"

puts composition
# Llevo 2 años esperando para hacerles esta ayudantia de Ingeniería de Software 2024-1
```

Resultado en consola:

```
Llevo 2 años esperando para hacerles esta ayudantia de Ingenieria de Software 2024-1
```

}

Python {

Codigo en python:

```python
# Define arbitrary element years
time = 2

year = 2024
semester = 1


# Define element name
element = f"Ingenieria de Software {year}-{semester}"


# Compose message
composition = f"Llevo {time} años esperando para hacerles esta ayudantia de {element}"

print(composition)

```

}

Control de flujo {

```ruby
# Control flow in ruby works very similar to python,
# except we use "elsif" instead of "elif"

x = gets.chomp.to_i

if x < 0
    puts "The number #{x} is negative"
elsif x > 0
    puts "The number #{x} is positive"
else
    puts "The number #{x} is a portal"
end
```

}

Control de flujo {

```ruby
# When we want to use the logic of an "if not", we use "unless".
# The following two operations are equivalent

unless x.to_s.length > 2
    puts "The number #{x} is short"
else
    puts "The number #{x} is long"
end

# Boolean operators

puts true || false # or
puts true && false # and
puts !true         # not
```

}

Output{

```ruby
# Control flow in ruby works very similar to python,
# except we use "elsif" instead of "elif"

x = gets.chomp.to_i

if x < 0
    puts "The number #{x} is negative"
elsif x > 0
    puts "The number #{x} is positive"
else
    puts "The number #{x} is a portal"
end
```

```ruby
# When we want to use the logic of an "if not", we use "unless".
# The following two operations are equivalent

unless x.to_s.length > 2
    puts "The number #{x} is short"
else
    puts "The number #{x} is long"
end

# Boolean operators

puts true || false # or
puts true && false # and
puts !true         # not
```

Resultado en consola:

```
6
The number 6 is positive
The number 6 is short
true
false
false
```

input

}

Iteraciones {

```ruby
# Define a VERY realistic weekday list
week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Iter over the week and print each day
week.each do |day|      # for day in week -> python equivalent
  puts day              #     print(day)
end

puts "----------"  # Separator

for i in 0..6  # Both numbers (0 and 6) are included!
  puts week[i]     # Access the i element of week
end

puts "----------"  # Separator
```

}

Output{

```ruby
# Define a VERY realistic weekday list
week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Iter over the week and print each day
week.each do |day|      # for day in week -> python equivalent
  puts day              #     print(day)
end

puts "----------"  # Separator

for i in 0..6  # Both numbers (0 and 6) are included!
  puts week[i]     # Access the i element of week
end

puts "----------"  # Separator
```

Resultado en consola:

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
----------
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
----------
```

}

Iteraciones {

```ruby
week.each_with_index do |day, index|  # Remember Python's enumerate method?
  puts "#{index}: #{day}"
end

puts "----------"  # Separator

index = 0
while index < week.length
  puts week[index]
  index += 1
end
```

}

Output{

```ruby
week.each_with_index do |day, index|
  puts "#{index}: #{day}"
end

puts "----------"  # Separator

index = 0
while index < week.length
  puts week[index]
  index += 1
end
```

Resultado en consola:

```
0: Monday
1: Tuesday
2: Wednesday
3: Thursday
4: Friday
5: Saturday
6: Sunday
----------
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

}

Identación {

```ruby
x = gets.chomp.to_i
💡
if x < 0
puts "The number #{x} is negative"
elsif x > 0
puts "The number #{x} is positive"
else
puts "The number #{x} is a portal"
end

unless x.to_s.length > 2
puts "The number #{x} is short"
else
puts "The number #{x} is long"
end
```

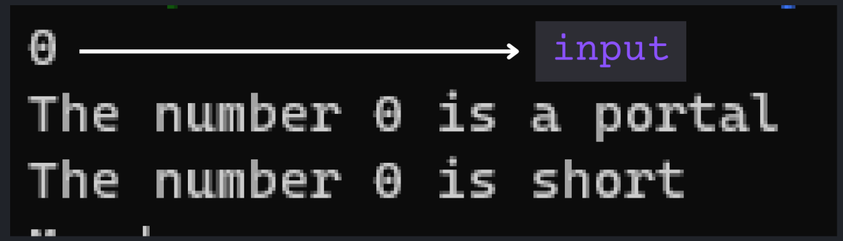
}

Output{

```ruby
x = gets.chomp.to_i
💡
if x < 0
puts "The number #{x} is negative"
elsif x > 0
puts "The number #{x} is positive"
else
puts "The number #{x} is a portal"
end

unless x.to_s.length > 2
puts "The number #{x} is short"
else
puts "The number #{x} is long"
end
```

Resultado en consola:



}

Identación {

```ruby
# Define a VERY realistic weekday list
week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Iter over the week and print each day
week.each do |day|      # for day in week -> python equivalent
puts day                #     print(day)
end

puts "----------"  # Separator

for i in 0..6  # Both numbers (0 and 6) are included!
puts week[i]     # Access the i element of week
end

puts "----------"  # Separator
```

}

Output{

```ruby
# Define a VERY realistic weekday list
week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Iter over the week and print each day
week.each do |day|      # for day in week -> python equivalent
puts day                #     print(day)
end

puts "----------"  # Separator

for i in 0..6  # Both numbers (0 and 6) are included!
puts week[i]     # Access the i element of week
end

puts "----------"  # Separator
```

Resultado en consola:

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
----------
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
----------
```

}

Identación {

```ruby
week.each_with_index do |day, index|  # Remember Python's enumerate method?
puts "#{index}: #{day}"
end

puts "----------"  # Separator

index = 0
while index < week.length
puts week[index]
index += 1
end

# In python, the indentation rules, in ruby, is the end keyword
```

}

Output{

```ruby
week.each_with_index do |day, index|  # Remember Python's enumerate method?
puts "#{index}: #{day}"
end

puts "----------"  # Separator

index = 0
while index < week.length
puts week[index]
index += 1
end

# In python, the indentation rules, in ruby, is the end keyword
```

Resultado en consola:

```
----------
0: Monday
1: Tuesday
2: Wednesday
3: Thursday
4: Friday
5: Saturday
6: Sunday
----------
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

}

Arrays {

```ruby
days =  ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]

# Get array length
puts days.length # prints 5

# Get an element"s index (nil if element is not in array)
puts days.index("Tuesday") # prints 1

# Access elements with arrays
puts days[0], days[3], days[-1] # prints Monday, Thursday, Friday

puts "----------"
# Access first and last elements without indexes
puts days.first, days.last  # prints Monday, Friday

puts "----------"
```

}

Output{

```ruby
days =  ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]

# Get array length
puts days.length # prints 5

# Get an element"s index (nil if element is not in array)
puts days.index("Tuesday") # prints 1

# Access elements with arrays
puts days[0], days[3], days[-1] # prints Monday, Thursday, Friday

puts "----------"
# Access first and last elements without indexes
puts days.first, days.last  # prints Monday, Friday

puts "----------"
```

Resultado en consola:

```
5
1
Monday
Thursday
Friday
----------
Monday
Friday
----------
```

}

Arrays {

```ruby
# Add elements to the end of arrays
days.push("Saturday")
days << "Sunday"

# Add elements to the start of arrays
days.unshift("Monday")

print days # prints Monday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
puts
# delete element using index
print days.delete_at(4) # prints Thursday
puts

# delete element
days.delete("Monday") # if element is repeated, it deletes all of them
print days # prints Tuesday, Wednesday, Friday, Saturday, Sunday
puts
# delete last element
puts days.pop # prints Sunday

# delete first element
puts days.shift # prints Tuesday
```

}

Output{

```ruby
# Add elements to the end of arrays
days.push("Saturday")
days << "Sunday"

# Add elements to the start of arrays
days.unshift("Monday")

print days # prints Monday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
puts
# delete element using index
print days.delete_at(4) # prints Thursday
puts

# delete element
days.delete("Monday") # if element is repeated, it deletes all of them
print days # prints Tuesday, Wednesday, Friday, Saturday, Sunday
puts
# delete last element
puts days.pop # prints Sunday

# delete first element
puts days.shift # prints Tuesday
```

Resultado en consola:

```
["Monday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
Thursday
["Tuesday", "Wednesday", "Friday", "Saturday", "Sunday"]
Sunday
Tuesday
```

}

Funciones {

```ruby
# Explicit return  definition
def addition(first, second)
    # Return the sum
    return first + second
end

# Implicit return definition
def implicit_addition first, second
  # Compute the sum as the last statement and DON'T return it
  first + second
end

# Log both results
puts addition 2, 3           # It obviously returns
puts implicit_addition 4, 5  # What happens with this call?
```

}

Output{

```ruby
# Explicit return  definition
def addition(first, second)
    # Return the sum
    return first + second
end

# Implicit return definition
def implicit_addition first, second
  # Compute the sum as the last statement and DON'T return it
  first + second
end

# Log both results
puts addition 2, 3           # It obviously returns
puts implicit_addition 4, 5  # What happens with this call?
```

Resultado en consola:

```
5
9
```

}

Python {

Codigo en python

```python

def addition(first, second):
    return first + second


def implicit_addition(first, second):
    return first + second


print(addition(2, 3))
print(implicit_addition(4, 5))

```

}

Hashes {

```ruby

# Define a hash like in Python (dictionaries are called hashes in ruby)
hash = {
    "one": 1,
    "two": 2,
    "three": 3
}

symbol_hash = { :one => 1, :two => 2, :three => 3 }

puts "HASHES"
# Log hashes generated... They are IDENTICAL!
puts hash
puts symbol_hash


# ¿¿WHAAAAAAT?? Strings get transformed to symbols when used inside
# the hash's definition...
puts symbol_hash[:one]
puts hash[:one]
```

}

Output{

```ruby

# Define a hash like in Python (dictionaries are called hashes in ruby)
hash = {
    "one": 1,
    "two": 2,
    "three": 3
}

symbol_hash = { :one => 1, :two => 2, :three => 3 }

puts "HASHES"
# Log hashes generated... They are IDENTICAL!
puts hash
puts symbol_hash


# ¿¿WHAAAAAAT?? Strings get transformed to symbols when used inside
# the hash's definition...
puts symbol_hash[:one]
puts hash[:one]
```

Resultado en consola:

```
HASHES
{:one=>1, :two=>2, :three=>3}
{:one=>1, :two=>2, :three=>3}
1
1
```

}

Hashes {

```
22  # Add some elements...
23  hash["four"] = 4
24  hash[:four] = 5
25
26  # Now, the string does not get converted to a symbol!
27  puts "HASH FOUR"
28  puts hash["four"]
29  puts hash[:four]
30  puts hash # WHAAAT?!?!?
31
32  # ProTip: when a hash is the last argument given to a method,
33  # {} brackets can be omitted inside the method call. For example:
34
35  puts "uno": 1, "dos": 2, "tres": 3
36
37  # Is the same as saying:
38
39  puts({"uno": 1, "dos": 2, "tres": 3})
40
```

}

Output{

```ruby
# Add some elements...
hash["four"] = 4
hash[:four] = 5

# Now, the string does not get converted to a symbol!
puts "HASH FOUR"
puts hash["four"]
puts hash[:four]
puts hash # WHAAAT?!?!?

# ProTip: when a hash is the last argument given to a method,
# {} brackets can be omitted inside the method call. For example:

puts "uno": 1, "dos": 2, "tres": 3

# Is the same as saying:

puts({"uno": 1, "dos": 2, "tres": 3})

```

Resultado en consola:

```
HASH FOUR
4
5
{:one=>1, :two=>2, :three=>3, "four"=>4, :four=>5}
{:uno=>1, :dos=>2, :tres=>3}
{:uno=>1, :dos=>2, :tres=>3}
```

}

Require {

```ruby
# Require a file from the directory
require_relative "env"

# Require a third party library (`gem install faker` first)
# https://github.com/faker-ruby/faker
require "faker"

# Log variable from the env.rb file

puts CONST_1
puts $var_3
# puts var_4 # What is gonna happen with this?
```

env.rb:

```ruby
CONST_1 = "I'm constant 1"
CONST_2 = "I'm constant 2"
$var_3 = "I'm global 3"
var_4 = "I'm local 4"

```

}

Output{

```ruby
# Require a file from the directory
require_relative "env"

# Require a third party library (`gem install faker` first)
# https://github.com/faker-ruby/faker
require "faker"

# Log variable from the env.rb file

puts CONST_1
puts $var_3
# puts var_4 # What is gonna happen with this?
```

```ruby
CONST_1 = "I'm constant 1"
CONST_2 = "I'm constant 2"
$var_3 = "I'm global 3"
var_4 = "I'm local 4"

```

Resultado en consola:

```
I'm constant 1
I'm global 3
```

}

Set {

```ruby
require 'set' # import set

newSet = Set.new # constructor of set class

newSet << 1 # add 1 to the set
newSet << 2 # add 2 to the set
newSet << 2 # add 2 to the set

puts newSet # log everything in the set

```

}

Output{

```ruby
require 'set' # import set

newSet = Set.new # constructor of set class

newSet << 1 # add 1 to the set
newSet << 2 # add 2 to the set
newSet << 2 # add 2 to the set

puts newSet # log everything in the set

```

Resultado en consola:

```
#<Set: {1, 2}>
```

}

Python {

Codigo en python

```python
new_set = set()

# Agregar elementos al conjunto
new_set.add(1)
new_set.add(2)
new_set.add(2)  # Los elementos duplicados se ignoran en un conjunto

# Imprimir el conjunto
print(new_set)
```

}

Clases{

```ruby
class Animal
    attr_accessor :name, :owner
    def initialize name, owner
        @name = name
        @owner = owner
        @steps = 0
    end

    def walk new_steps
        @steps += new_steps
    end

    def to_s
        "I am #{@name} and my owner is #{@owner}"
    end

end
```

}

Clases{

```ruby
class Cat < Animal
    attr_accessor :purrs
    def initialize(name, owner, purrs)
        super(name, owner)
        @purrs = purrs
    end

    def talk
        miau = "MI" + ("A" * rand(1..10)) + "U"
        puts miau
    end

    def pet
        @purrs += 10
        puts "Purr"
    end

end
```

}

Clases{

```ruby
random_animal = Animal.new("Steve", "Notch")
puts random_animal
puts random_animal.name


cat = Cat.new("Grogu", "Pedro", 0)
cat.talk

3.times do |_|
    cat.pet
end

puts cat.purrs
puts cat.name
```

}

Output{

```ruby
random_animal = Animal.new("Steve", "Notch")
puts random_animal
puts random_animal.name


cat = Cat.new("Grogu", "Pedro", 0)
cat.talk

3.times do |_|
    cat.pet
end

puts cat.purrs
puts cat.name
```

Resultado en consola:

```
I am Steve and my owner is Notch
Steve
MIAAU
Purr
Purr
Purr
30
Grogu
```

}

Clases Abstractas {

```ruby
module Animal
    attr_accessor :name, :owner # Try commentig this
    def initialize name, owner
        @name = name
        @owner = owner
        @steps = 0
    end
    # esto seria un metodo abstracto
    def talk
        raise NotImplementedError, "#{self.class} debe implementar el método 'speak'"
      end

      # Otro método que podría ser común a todas las clases que incluyan este módulo
    def move
        puts "#{self.class} se está moviendo"
    end

    def to_s
        puts "I am #{@name} and my owner is #{@owner}"
    end
```

}

Clases Abstractas {

```ruby
class Cat
include Animal
    attr_accessor :purrs
    def initialize(name, owner, purrs)
        super(name, owner)
        @purrs = purrs
    end

    def talk
        miau = "MI" + ("A" * rand(1..10)) + "U"
        puts miau
    end

    def pet
        @purrs += 10
        puts "Purr"
    end

end
```

}

Clases Abstractas {

```ruby
class Chicken
    include Animal
        def initialize(name, owner)
            super(name, owner)
        end

        def talk
            cocoroco = "CO" + ("CORO" * rand(1..10)) + "CO"
            puts cocoroco
        end



    end

# si intentas correr como antes solo animal este te enviara un error debido a que es abstracto
#random_animal = Animal.new("Steve", "Notch")
```

}

Clases Abstractas {

```ruby
cat = Cat.new("Grogu", "Pedro", 0)
chicken = Chicken.new("KFC", "Coronel Sanders")
chicken.talk
puts chicken.move
chicken.to_s

cat.talk

3.times do |_|
    cat.pet
end

puts cat.purrs
puts cat.move
cat.to_s
```

}

Output{

```ruby
cat = Cat.new("Grogu", "Pedro", 0)
chicken = Chicken.new("KFC", "Coronel Sanders")
chicken.talk
puts chicken.move
chicken.to_s

cat.talk

3.times do |_|
    cat.pet
end

puts cat.purrs
puts cat.move
cat.to_s
```

Resultado en consola:

```
COCOROCOROCOROCO
Chicken se está moviendo

I am KFC and my owner is Coronel Sanders
MIAAAAAU
Purr
Purr
Purr
30
Cat se está moviendo

I am Grogu and my owner is Pedro
```

}

Errores y excepciones {

```ruby
begin ## Equivalent to try on python
    puts "this will be shown"
    raise "an error"
    puts "this won't be shown"

rescue => error ## Except on python (you can also catch specific errors)
    puts error.message # "an error"
end

#Built-in exceptions
begin
    # Divide by zero to raise an exception
    result = 10 / 0
  rescue ZeroDivisionError => e
    puts "An error occurred: #{e.message}"
  end

# after running
# this will be shown
# an error
# An error occurred: divided by 0
```

}

Output{

```ruby
#Raise error
begin ## Equivalent to try on python
    puts "this will be shown"
    raise "an error"
    puts "this won't be shown"

rescue => error ## Except on python (you can also catch specific errors)
    puts error.message # "an error"
end

#Built-in exceptions
begin
    # Divide by zero to raise an exception
    result = 10 / 0
  rescue ZeroDivisionError => e
    puts "An error occurred: #{e.message}"
  end
```

Resultado en consola:

```
this will be shown
an error
An error occurred: divided by 0
```

}

Manejar archivos {

```ruby
require "csv"  # Import csv library

# Open file and print it line by line
File.open("files/read.txt", "r").each do |line|
    puts line
end

# Print to file!
File.open("files/write.txt", "w") do |f|
    # f.each do |file|
    #   puts file
    # end
    f.puts "Mi primera línea"
    f.puts "Esto se escribe en otra línea!"
end

# Just like in python, if the file doesn't exist, it creates it
```

}

Manejar archivos {

```ruby
File.open("files/write2.txt", "w") {|file| file.puts "Otra forma de escribir en archivos"}

# Append string to a file already written
File.open("files/write2.txt", "a") do |f|
    f.puts "Hola, soy yo de nuevo."
end

# Open a csv file with the module CSV
csv_file = CSV.read("files/read.csv")
puts csv_file
p csv_file
# Note that csv_file is a bidimensional array.

CSV.open 'files/write.csv', 'a' do |csv|
    csv << ["Hola","Mundo"]
    csv << ["Chao","mundo"]
end
```

}

Output{

```ruby
File.open("files/write2.txt", "w") {|file| file.puts "Otra forma de escribir en archivos"}

# Append string to a file already written
File.open("files/write2.txt", "a") do |f|
    f.puts "Hola, soy yo de nuevo."
end

# Open a csv file with the module CSV
csv_file = CSV.read("files/read.csv")
puts csv_file
p csv_file
# Note that csv_file is a bidimensional array.

CSV.open 'files/write.csv', 'a' do |csv|
    csv << ["Hola","Mundo"]
    csv << ["Chao","mundo"]
end
```

Resultado en consola:

```
Hola! soy un archivo txt.
Soy un ejemplo.
Tengo una serpiente en mi bota.
id
nombre
0
Nicole
1
Martin
[["id", "nombre"], ["0", "Nicole"], ["1", "Martin"]]
```

}

<!--Ingeniería de Software 2024-1-->

Gracias {

<Por="Constanza Huerta y Cristóbal Moreno"/>

}