AYUDANTÍA RUBY

¡Bienvenidos al curso Ingeniería de Software!

QUE VEREMOS

- I/O
- Type Conversions
- String Interpolation
- Control Flow
- Iterations
- Arrays and Hashes
- Methods and Method Calls
- Sets
- Require
- ARGV
- File handling
- Exceptions

INPUT/OUTPUT

```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)
```

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

CONVERSIONS

```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
```

METHOD CALLS

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

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

STRING INTERPOLATION

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

year = 2022
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 2022-1
```

CONTROL FLOW

```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 FLOW

```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
```

ITERATE

```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
```

ITERATE

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

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

ITERATE

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

INDENTATION

```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

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

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

# Access first and last elements without indexes
puts days.first, days.last # prints 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')

puts days # prints Monday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

# delete element using index
puts days.delete_at(4) # prints Thursday

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

# delete last element
puts days.pop # prints Sunday

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

METHODS

```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? (spoiler: it returns 9)
```

HASHES

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

# ¿¿WHAAAAAAT?? Strings get transformed to symbols when used inside
```

HASHES

```ruby
# the hash's definition...
puts symbol_hash[:one] # 1
puts hash[:one] # 1

# 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 # {:one=>1, :two=>2, :three=>3, "four"=>4, :four=>5}
```

VARIABLE SCOPE

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


# Local variables are local: they don't survive across a require.
# Global variables ($code_words), constants (CODE_WORDS)
# and instance variables (@code_words) do.

#https://stackoverflow.com/questions/37181901/require-relative-doesnt-pull-the-variables
```

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 # I'm constant 1
puts $var_3 # I'm local 3
# puts var_4 # What is gonna happen with this? (spoiler: it fails)

# Use a method from the third party library
(0..5).each do |_|
    puts " -> #{Faker::ChuckNorris.fact}"
end
```

SETS

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

ARGV

```ruby
# ARGV: array with console-given arguments

# Get first console-given argument
first_element = ARGV[0]

# Get second console-given argument
second_element = ARGV[1]

# ARGV[2] = "third element"

# Print both elements separated by a comma
puts "#{first_element}, #{second_element}"

print ARGV
```

```
ruby 13_argv.rb argumento1 argumento2 etc
```

CLASSES

```ruby
class Animal
    attr_accessor :name, :owner # Try commentig this
    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
```

```ruby
random_animal = Animal.new("Steve", "Notch")
puts random_animal # I am Steve and my owner is Notch
puts random_animal.name # Steve
```

CLASSES

```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
```

```ruby
cat = Cat.new("Michi", "Maggie", 0)
cat.talk # MIAAAAAAU

(0..3).each do |_|
    cat.pet # Purr
end

puts cat.purrs # 40
puts cat.name # Michi
puts cat # I am Michi and my owner is Maggie
```

FILE HANDLING

```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
```

FILE HANDLING

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

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
print csv_file
# Note that csv_file is a bidimensional array.

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

EXCEPTIONS

```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

# after running
# this will be shown
# an error
```