Ruby

Ventajas de `Ruby`

- Interpretado: No requiere compilación para interpretarse.
- Sintáxis sencilla: análogo al lenguaje en sintaxis.
- Meta Programming: Capacidad del código de transformar código en otro código.
- TryRuby: tryruby.org

Instalando `Ruby`

Ver qué versión de `ruby` estoy ocupando. En el terminal, se escribe

```
1 | which ruby
```

Y asegurarse que sea /.rvm/

Ver qué versión del `rails` estoy ocupando.

```
1 | which ruby
```

Interprete de `Ruby`

Interprete inline

```
1 | irb
```

```
1 | puts 'hola'
```

Dentro de una carpeta en terminal, escribimos el primer script de ruby, seguido de su extensión `.rb`

```
cd /
mkdir demoRuby
cd ./demoRuby
touch demoRuby.rb
```

En el archivo `demoRuby.rb`

```
puts 'hola'
```

Volvemos al terminal, y ejecutamos en la carpeta `./demoRuby`

```
ruby demoRuby.rb
```

Ejecutando dessde el editor de texto

En Sublime, `cmd + B`

Estructura léxica de `Ruby`

> Conjunto de palabras que definen un conjunto de reglas en el programa. el código se divide en tokens, que pueden ser comentarios, literales, etc...

- `ruby` es sensible a mayúsculas.
- Comentarios: texto omitido al leer el código.
- Literales: valores o textos.
- Identificadores: variables y nombres de funciones.
- Palabras reservadas: `start, stop, break, while, if`

Identificadores en `ruby`

> Identificadores: nombre de variables, métodos y clases. Son válidos en la medida que empiecen con una letra, o guión abajo. No puede empezar con números Sensible a las mayúscula.

```ruby
a = 5
puts a
a = 6
puts a
b1 = 'hola'
puts b1
```

También se le pueden asignar variables valores de otras variables

```ruby
a = 5
b = a
puts b

b = a + 1
puts a
puts b
```

Actualizar valores de una variable (forma concisa del snippet anterior)

```ruby
a += 1
puts a
```

Las constantes se escriben con mayúsculas. Advierte que el comportamiento no debiera modificarse:

```ruby
C = 'el diablo es magnífico'
```

Literales y captura de datos

> Valores que se asignan a las variables. Cuando hablamos de literales, hablamos de tipo de datos.

Descubir qué clase es un objeto

```ruby
a.class
```

Posibles clases

```ruby
'Hola'.class #string
10.class #Fixnum
(9.2).class #Float
```

Convertir formatos

```ruby
'100'.to_i #String to Fixnum
'9.4'.to_f #String to Float
(9.4).to_s #Float to string
```

Tipos de datos y métodos asociados

> Tipos de datos:
>
> - String:
> - Fixnum:
> - Float:
> - NilClass:
> - TrueClass, FalseClass:
> - Array:
> - Symbol
> - Hash:
> - Time:
>
> No existen datos nativos en ruby

If y unless

Bloques de control, en forma normal y express

```ruby

# Forma normal
if a == 2
    puts 'a equals 2'
end

# Forma express
puts 'a equals 2' if a == 2
```

```ruby

#forma normal
unless a == 2
    puts 'a not equals 2'
end

#forma express
puts 'a not equals 2' if a == 2
```

Elsif

```ruby
a = 4
b = 3

if a > b
    puts 'a greater than b'
elsif a == b
    puts 'both are equal?'
else
    puts 'b is greater than a'
end
```

While

```ruby
puts 'Add password'
pass = gets.chomp


while pass != 'secret'
    puts 'Add password'
    pass = gets.chomp
end

puts 'correct'
```

Validación

```ruby
numero = gets.chomp.to_i
numero = gets.chomp.to_i while numero < 0 || numero > 36
```

Times 1

```ruby
# forma normal
5.times do
    puts "repeaterbeater"
end

#forma express
5.times { puts "repeaterbeater"}
```

Times 2 (con indices)

```ruby
# forma normal

5.times do |i|
    puts "repeaterbeater: #{i}"
end

# forma compacta
5.times { |i| puts "repeaterbeater: #{i}"  }


100.times do |i|
    puts (i + 1).to_s + "lorem ipsum"
end


# Interpolar

100.times do |i|
    puts "#{i + 1} lorem ipsum"
end


100.times do |i|
    if i == 1
        puts "#{i + 1} lorem ipsum solito el unito"
    else
        puts "#{i + 1} lorem ipsum"
    end
end
```

For and each

> Preferir `each` sobre `for`.

```ruby
for i in 0..10
    puts i
end

(0..10).each do |i|
    puts i
end
```

Divisores

```ruby
(1..840). each do |i|
    puts i if (840 % i).zero?
end

```