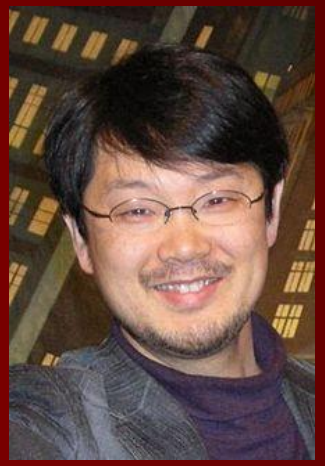
TUTORIAL DE RUBY

Lenguajes de programación
Universidad Nacional De Colombia
2017 - II

Laura P. Cerón M.
Camilo A. Dajer P.

INTRODUCCIÓN

- Ruby es un lenguaje interpretado
- Case sensitive
- Ruby es de código abierto
- Creado por Yukihiro Matsumoto.



“El lenguaje está enfocado en darle mayor importancia a las personas que programan las aplicaciones y a los usuarios que las manejan”.

Yukihiro Matsumoto

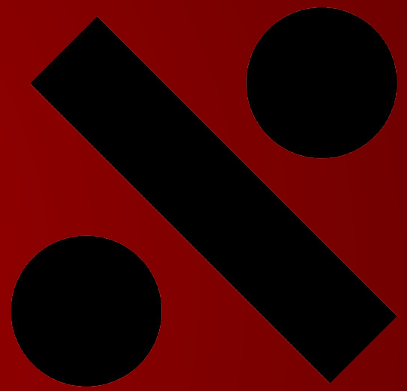
GRANDES EMPRESAS QUE IMPLEMENTAN RUBY

Y muchas más

ASPECTOS BÁSICOS DEL LENGUAJE

Operados básicos

- Aritméticos: + - / * ** %
- Relacionales: == != =
- Lógicos: and or !



Ruby no posee operadores pre/post incremento/decremento

IMPRESIÓN

- **puts**

Realiza un salto de línea

```
puts “a” , “b” #=>a
                 b
```

- **print**

```
print “a” , “b” #=> ab
```

OPERADOR !

- Ruby permite realizar funciones sobre objetos sin guardar cambios sobre el mismo.

```
array = ["Ruby", "Python", "Scala","Java"]

puts array.sort #=> [“Java”, “Python”, “Ruby”, ”Scala”]

puts array          #=> ¿Ordenados?
( array = array.sort  )  ==   array.sort!
```

CADENAS

- “Comilla doble”

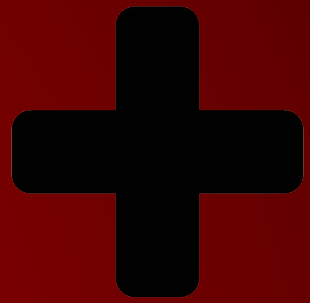
  Permiten de la presencia embebida de caracteres de escape precedidos por un backslash y la expresión de evaluación #{ }.

```
puts "a\nb\nc” #=> a
                   b
                   c
```

- ‘Comilla sencilla’

```
puts  ‘a\nb\nc’ #=> a\nb\nc
```

CONCATENACIÓN



- Se puede realizar con el carácter +:

```
puts “Ruby” + “ on Rails”  #=> Ruby on Rails
```

- ¿ Y qué pasa si multiplico una cadena por un número?

```
puts ‘Repeat me' * 2   #=> Repeat me Repeat me
```

LISTAS

- Es la estructura de datos más implementada
  - El acceso a elementos que no existen retorna nil
  - Concatenación: Usando el operador +
  - Agregar un elemento:
    - `.push`
    - `<<`

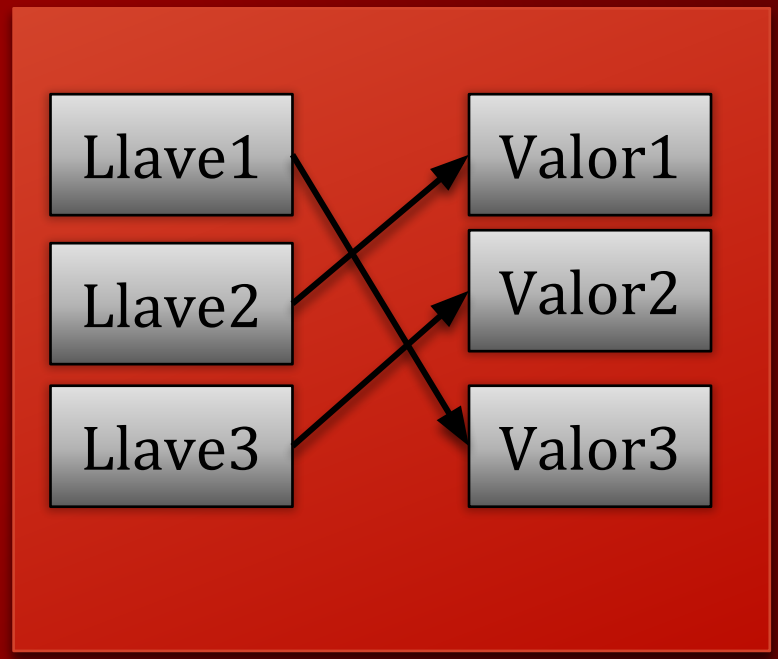
```
myList  = [12, 58, 93]
myList.push("Number")
myList << 481
#=> [12,58,93, "Number", 481]
```

COLECCIONES

Listas

Índice	Valor
0	“Palabra”
1	23
2	‘a’

Diccionarios



DICCIONARIOS

- En Ruby a los diccionarios se les denomina *hash*.

```
dict = {1 => 2, "2" => "4"}
puts dict   #=>{1=>2, "2"=>4}
```

- Agregar nueva llave y elemento
- Función *.invert*
- Función *.delete*

```
dict = {1 => 2, "2" => "4"}
dict[3] = "Nuevo"
puts dict
```

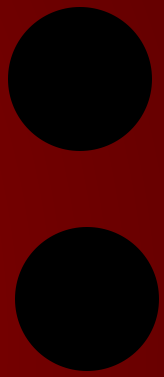
RANGOS

- Ruby permite implementar rangos de una manera muy sencilla:

```
a = (1..10).to_a  #=>  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

a = (1...10).to_a  #=>  [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

SÍMBOLOS

- Los símbolos son el objeto más básico de Ruby.



```
puts “3".object_id   #=> 39337164
puts “3".object_id   #=>39335436
puts :cadena.object_id    #=> 2136188
puts :cadena.object_id    #=> 2136188
```

Dado un símbolo, se refiere al mismo objeto en todo el programa. Por esta razón son más eficientes que las cadenas: dos strings con el mismo nombre son dos objetos distintos.

ESTRUCTURAS DE CONTROL

WHILE

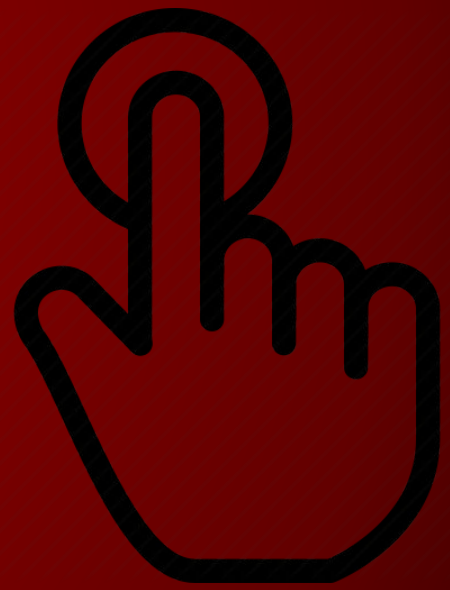
Se poseen 4 palabras reservadas para hacer operaciones especiales en el cada ciclo:

- **break:** Termina totalmente el ciclo
- **next:** Termina ejecución del bucle actual
- **redo:** Reinicia la iteración actual
- **return**

CASE

- La sentencia **case** es usada para comprobar un valor:

```
valor = 30
case valor
  when 30, (1..10)
    puts "1 - 10" + ", o puede ser 30"
  when 11..20
    puts "11 - 20"
end
```



FOR - IN

- En colecciones

```
numeros = [1,2,3,4,5]
for numero in numeros
  #Comandos
end
```

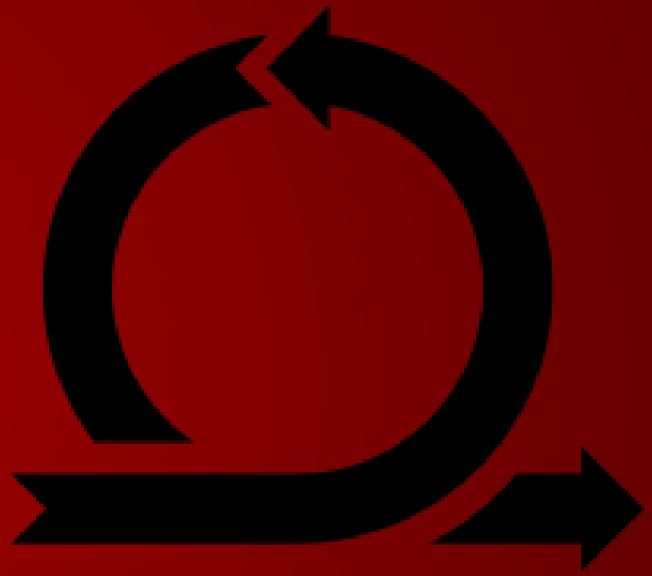
- En rangos

```
for i in (0..4)      #=> (0,1,2,3,4)
  #Comandos
end
```

ITERADORES

CADENAS

- `.each_byte`
- `.each_line`



COLECCIONES

- `.each`
- `.collect`

CLONE

- Ruby interpreta todo como un objeto. Cuando se desea tener dos objetos totalmente diferentes en cada variable se hace uso del comando **clone**.

```
a = [1,2]
b = a.clone
```

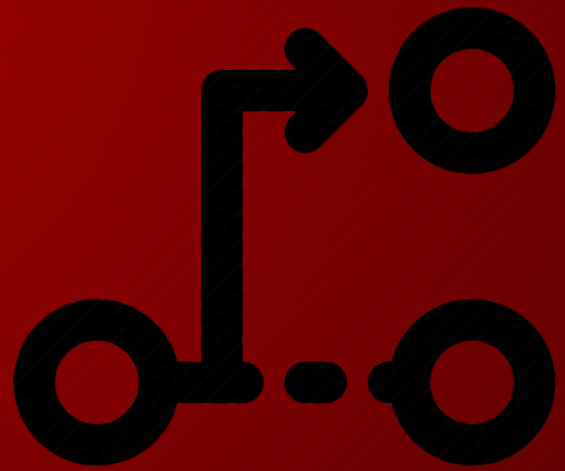
```
a = [1,2]
b = a
a[1] = 3
puts a    #=> [ 1, 3 ]
puts b    #=> [ 1, 3 ]
```

Cuando se iguala un arreglo a otro, estos quedan apuntando al mismo objeto, y los cambios realizados en uno también se reflejan en el otro.

PROGRAMACIÓN ORIENTADA A OBJETOS CON RUBY

ORIENTADO A OBJETOS

- Ruby es un lenguaje puramente orientado a objetos, como pudimos observar en la explicación anterior, todo en Ruby es interpretado como un objeto, a continuación veremos algunas características importantes de Ruby aplicando conceptos de la programación orientada a objetos.



MÉTODOS

- Métodos predefinidos:
  - *.length: Retorna el tamaño de la lista*
  - *.class: Retorna la clase del objeto*

- ¿Cómo se define un método?

```
def nombreDelMetodo
  #Comandos
end
```

```
def nombreDelMetodo(a,b)
  #Comandos
end
```

SOBRECARGA DE MÉTODOS

- Ruby no permite una manera convencional de realizar sobrecarga de métodos, para poder realizarlo se debe implementar de la siguiente manera:

```
def nombre(*args) # el símbolo * implica número variable de argumentos
 if args.size < 2 || args.size > 3
  puts 'ERROR: Este método recibe dos o tres argumentos'
 else
  if args.size == 2
    #Comandos
  else
    #Comandos
  end
 end
end
```

TIPOS DE VARIABLES

- Ruby permite declarar distintas clases de variables, las cuales se especifican de la siguiente manera:

    - $*[a-z]+[a-zA-Z0-9]**: Variable global
    - @*[a-z]+[a-zA-Z0-9]**: Variable de instancia
    - [a-z]+[a-zA-Z0-9]**:* Variable local
    - *[A-Z]+[a-zA-Z0-9]**: Constante

MÉTODO *.defined?*

- Este método nos permite saber si una variable a sido declarado y en dado nos devuelve una descripción de la variable.

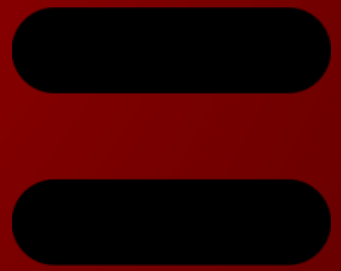
```
var1 = 1
Var2 = 1

puts defined? var1 #=> local-variable
puts defined? Var2 #=> constant
```

OPERADOR ||=

Este operador puede ser muy útil y ahorrar varias líneas de código

```
if not defined? var1
    var1 = 1
else
  puts “Ya existe la variable"
end
```

Var1 ||= 1



CLASES

- Las clases en Ruby son muy fáciles de declarar, su estructura es muy similar a la de Python por lo cual nos permite declarar atributos y métodos a una clase de manera intuitiva.

- Encapsulamiento
- Herencia
- Polimorfismo

```ruby
# Definimos la clase Persona
class Persona

  # Constructor de la clase
  def initialize(nombre,edad)
    # atributos
    @nombre = nombre
    @edad = edad
  end

  # método saludar
  def saludar
    puts "Hola! mi nombre es #{@nombre}"
  end

end
```

```ruby
andres = Persona.new("Andrés",20)
andres.saludar
```

HERENCIA

- Para poder implementar herencia en nuestro código solo debemos hacer uso del carácter **<** en la definición de la clase, de esta manera *extenderemos* de la clase declarada a la derecha del operador.

```
class Animal
  def respira
    puts "Inhalar y exhalar"
  end
end
```

```
class Perro<Animal
  def ladrar
    puts "Guau"
  end
end

Perro.new.respira
```

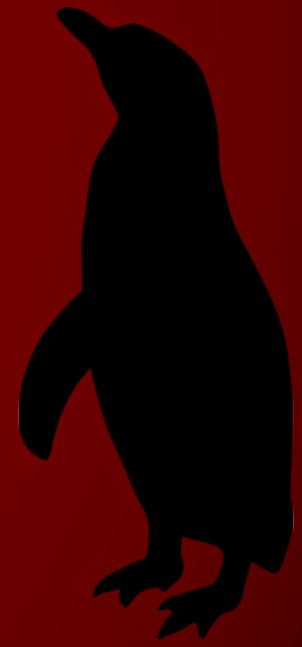
POLIMORFISMO

- Sin embargo hay casos en los que una subclase no debería heredar el comportamiento de la clase padre por lo que es posible sobreescribir el método.

La mayoría de aves vuelan, pero los pingüinos no.

```
class Ave
  def vuela
    puts "Volando"
  end
end
```

```
class Pinguino<Ave
  def vuela
    puts "Yo no vuelo"
  end
end
```



ENCAPSULAMIENTO

- Ruby nos permite aplicar encapsulamiento a métodos de nuestra clase declarando la palabra **private** y el nombre del método como símbolo.

```
class Persona
  def initialize(edad)
    @edad = edad
end
  def getEdad(clave)
    return getEdadReal - 5
  end
  def getEdadReal
    return @edad
  end
  private :getEdadReal
end
```

MÉTODOS SINGLETON

- Algunas veces es necesario modificar el comportamiento de un método de algun objeto, lo que nos implicaría tener que crear una nueva clase para ese objeto. Ruby nos permite modificar el comportamiento de algún objeto de manera individual.

```
class Persona
  def initialize(edad)
    @edad = edad
end
  def getEdad
    return getEdadReal
  end
  def getEdadReal
    return @edad
  end
  private :getEdadReal
end
```

```
persona_1 = Persona.new(20)

persona_2 = Persona.new(28)


def persona_2.getEdad
  return getEdadReal - 3
end
```

Ruby on Rails

¿Qué es Ruby on Rails?

Ruby on Rails (RoR) es un framework para el desarrollo de aplicaciones web basado en MVC el cual usa Ruby cómo lenguajes para el desarrollo.

Su filosofía: *Optimizing for programmer happiness with Convention over Configuration*

shopify

GRACIAS

BIBLIOGRAFÍA

- http://www.ubiqum.com/blog/las-mejores-aplicaciones-hechas-con-ruby-on-rails/
- https://skillcrush.com/2015/02/02/37-rails-sites/
- https://builtwith.com/github.com
- https://www.tutorialspoint.com/ruby/ruby_iterators.htm
- http://rubytutorial.wikidot.com/clases-modificar
- http://rubytutorial.wikidot.com/simbolos
- https://www.uco.es/aulasoftwarelibre/wiki/images/3/35/Curso_ruby_i.pdf
- https://www.ruby-lang.org/es/
- https://codesolt.com/rails/poo-ruby/
- http://ruby-doc.org/core-2.4.1/String.html
- http://www.rubyist.net/~slagell/ruby/