Ruby Tutorial

...o como pasar un buen rato programando

Lección 1

Lección 2

- Los métodos
- Los métodos: argumentos
- Rangos
- Arrays

Lección 3

- Bloques
- Más malabares con strings
- Expresiones Regulares
- Condicionales
- Bucles
- Números Aleatorios

Lección 4

- Clases y Objetos
- Accesores
- Ficheros: lectura/escritura
- Cargando librerías
- Herencia de clases
- Modificando clases
- Congelando objetos
- Serializando objetos

Lección 5

- Control de acceso
- Excepciones
- Módulos
- Constantes
- Hashes y Símbolos
- La clase Time

Lección 6

- self
- Duck Typing
- Azúcar Sintáctico
- Test de unidades

contacto

e-mail

Los métodos: argumentos

Valores por defecto

Ruby deja especificar los valores por defecto de los argumentos, que son usados si no se especifica un valor explícitamente. Se hace esto mediante el operador de asignación (=):

```
#argumentos.rb

def mtd(arg1="Dibya", arg2="Shashank", arg3="Shashank")
   "#{arg1}, #{arg2}, #{arg3}."
end
puts mtd
puts mtd("ruby")
```

Hemos usado el operador interpolación #{ }: se calcula la expresión entre paréntesis, y el resultado se añade al string. Lo que obtenemos es:

```
>ruby argumentos.rb
Dibya, Shashank, Shashank.
ruby, Shashank, Shashank.
>Exit code: 0
```

Número de argumentos variable

Ruby permite escribir funciones que acepten un número variable de argumentos. Por ejemplo:

```
def foo(*mi_string)
  mi_string.each do |palabras|
    puts palabras
  end
end

foo('hola', 'mundo')
foo()
```

El **asterisco** indica que el número de argumentos puede ser el que se quiera. En este ejemplo, el asterisco toma los argumentos y los asigna a un array (o vector de elementos) llamado `mi_string`. Haciendo uso de ese asterisco, incluso se pueden pasar cero argumentos; que es lo que pasa con foo().

No hay máximo número de argumentos que podamos pasar a un método.

Argumentos opcionales

Si se quieren incluir argumentos opcionales, tienen que venir después de los argumentos no opcionales:

```
def arg_opc(a,b,*x) # bien
def arg_opc(a,*x,b) # mal
```

Los argumentos se interpretan de izquierda a derecha, por eso es importante que los argumentos no opcionales vayan en primer lugar. Si los pusiésemos en último lugar, no sabríamos decir donde acaban los argumentos opcionales y donde empiezan los no opcionales.

```
=begin
Ejemplo de como los argumentos s
```

```
e interpretan de izquierda a derecha
=end

def mtd(a=99, b=a+1)
  [a,b]
end
puts mtd
```


Other interesting sites

House Games

Marvel Untold MUX

Prämien teilen wiki