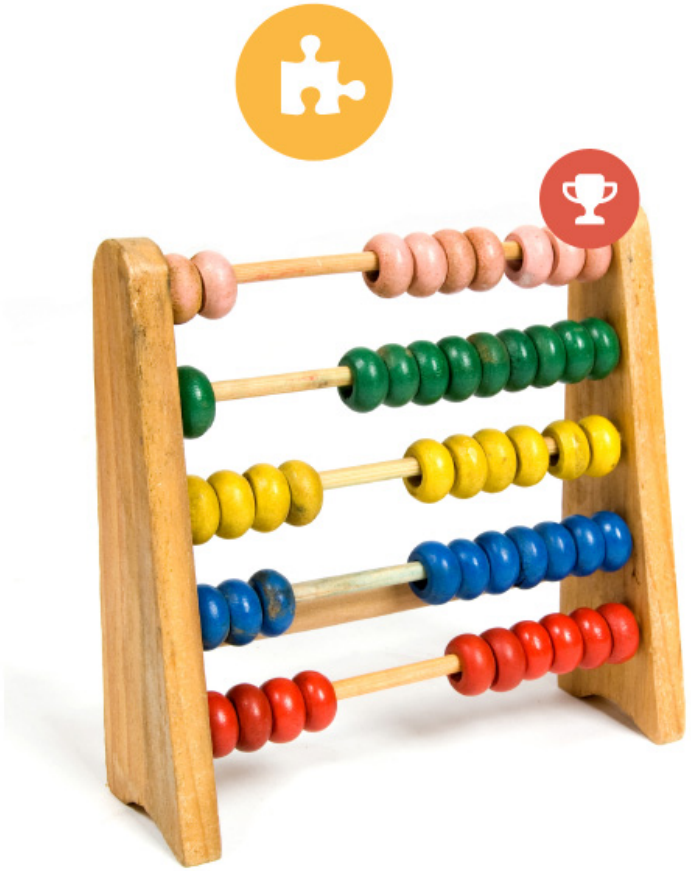
Programación en Lenguaje Ruby

Streams

Delio Tolivia

Gijón crece 2016 / 2019
AYTO. DE GIJÓN / CCOO / FADE / UGT
Ayuntamiento de Gijón
Gijón impulsa Empleo
dicampus TALENTO FORMACIÓN



Índice

I/O Streams

Clase IO

Salida, entrada y error estándar

Clase File

Lectura y Escritura

I/O Streams

- Un flujo de datos de entrada/salida (I/O Stream) es una secuencia de bytes a los que se accede de forma secuencial o aleatoria. Son utilizados para la comunicación hacia y desde el ordenador:
    - Sacar texto por pantalla
    - Recibir las teclas al escribir en el teclado
    - Emitir sonidos por los altavoces
    - Enviar y recibir datos en una red
    - Leer y escribir ficheros en el disco
    - …..

Clase IO

- La clase IO es la que permite inicializar los Streams

  # abrir el fichero "new-fd" y crear un descriptor de fichero
  fd = IO.sysopen("new-fd", "w")

  # Crear un nuevo flujo de entrada/salida usando el descriptor
  p IO.new(fd)

- Lo descriptores de fichero son un concepto heredado de UNIX. Son un número entero que hace referencia al objeto IO.

Salida, entrada y error estándar

- Ruby define las constantes ***STDOUT***, ***STDIN*** y ***STDERR*** que son objetos ***IO*** que apuntan a la salida (la consola), entrada (el teclado) y flujo de errores del programa (en nuestro caso la consola).

- Cuando se llama a ***puts*** su salida es redirigida al objeto que hace referencia ***STDOUT***. Lo mismo ocurre con el método ***gets*** que captura los datos de ***STDIN*** y el método ***warn*** que escribe hacia ***STDERR***.

- En Ruby hay un módulo accesible desde todos los sitios llamado ***Kernel***. Este modulo tiene las variables globales ***$stdout, $stdin*** y ***$stderr*** que apuntan a los mismos elementos que ***STDOUT***, ***STDIN*** y ***STDERR***.

- En realidad al llamar a ***puts*** se esta haciendo una llamada a ***Kernel.puts*** que llama a ***$stdout.puts***

- Podemos utilizar estas variables globales para asignarles otros objetos ***IO*** distintos de los que nos ofrece Ruby por defecto.

Clase File (I)

- En un ejemplo anterior hemos abierto un fichero con IO.sysopen e IO.new. Para hacerlo más facil Ruby nos ofrece la clase File.

```
mode = "r+" # modo en que accedemos al fichero
file = File.open("friend-list.txt", mode)

puts file.inspect #Mostrar la información del fichero

puts file.read #Leer el contenido del fichero

file.close #Cerrar el fichero
```

r -> solo lectura desde el principio del fichero
r+ -> lectura/escritura desde el principio
w -> escritura solo, vacía el fichero o crea uno
w+ -> escritura/lectura vacía el fichero o crea uno
a -> escritura solo desde el final del fichero o crea uno
a+ -> lectura/escritura desde el final o crea uno
b -> Modo byte combinado con los anteriores (salvo t)
t –> modo texto combinado con los anteriores (salvo b)

Clase File (II)

- El método ***File.open*** admite un bloque que nos permite ejecutar código y cierra automáticamente el fichero cuando termina.

```
what_am_i = File.open("clean-slate.txt", "w") do |file|
  file.puts "Call me Ishmael."
end
```

```
File.open("clean-slate.txt", "r") {|file| puts file.read }
```

Lectura y Escritura (I)

- El método ***File.read*** es el que se utilizará para leer de un fichero (o de otro I/O Stream). Recibe dos argumentos opcionales.
    - ***length:*** El número de bytes a leer
    - ***buffer:*** Se le pasa un ***String*** que rellenará con lo que lea y que puede reusarse al ir leyendo contenido del fichero.

```
file = File.open("master", "r+")

buffer = ""
p file.read(23, buffer)
p buffer

file.close
```

Lectura y Escritura (II)

- Para volver al inicio del fichero tenemos el método ***File.rewind***

```
file = File.open("master", "r+")

p file.read
file.rewind
```

- El método ***File.seek*** nos permite colocarnos en un byte en particular para seguir la lectura desde él.

```
p File.read("monk")

File.open("monk") do |f|
  f.seek(20, IO::SEEK_SET)
  p f.read(10)
end
```

IO:SEEK_SET Posición absoluta
IO:SEEK_CUR A partir de posición actual
IO:SEEK_END Desde el final (usar negativos)

Lectura y Escritura (III)

- El método ***readlines*** retorna un array con las líneas que ha leído del fichero. Se puede limitar el numero de líneas o el separador de las líneas.

```
lines = File.readlines(”master")
p lines
p lines[0] #Accederia a la primera línea que se ha leído
```

- Para escribir utilizaremos el método ***write***. Nos devuelve el número de bytes que ha escrito.

```
File.open("disguise", "w") do |f|
  f.write "Bar"
end
```