MANUAL DE RUBY

(PARTE I)

Luis José Sánchez González

1. ¿QUÉ ES RUBY?

Ruby es un lenguaje de programación interpretado y orientado a objetos muy potente y al mismo tiempo muy sencillo.

Para apreciar la extrema sencillez de este lenguaje, veamos el código del típico programa que muestra por pantalla **hola mundo** en C, C++ y Ruby.

C	C++	Ruby

`#include <stdio.h>`
 	
`int main()`
`{`
`  printf("hola mundo");`
`  return (0);`
`}`
	`#include <iostream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  cout << "hola mundo";`
	`  return 0;`
	`}`
		`puts "hola mundo"`

2. ¿CÓMO PODEMOS EJECUTAR RUBY?

Hay muchas maneras de ejecutar código ruby, veamos algunas de ellas.

Podemos ejecutar comandos de ruby de forma interactiva con **irb** (interactive ruby). Simplemente tendremos que abrir una consola y teclear irb. Desde ese momento podemos introducir comandos sueltos que se irán ejecutando cuando le damos a INTRO.

Otra forma de ejecutar ruby (la que utilizaremos de aquí en adelante) será utilizar el comando **ruby** seguido del nombre del programa. Por ejemplo, si tenemos el fichero **saludo.rb** con el siguiente código

```
puts "hola mundo"
```

teclearemos

```
ruby saludo.rb
```

desde una consola para ejecutarlo.

Tenemos una tercera forma. Podemos ejecutar directamente el archivo con el código ruby, pero deberemos añadir en la primera línea la ruta hacia el intérprete de ruby de la siguiente forma:

```
#!/usr/bin/ruby

puts "hola mundo"
```

Ahora sólo tendremos que ejecutar el archivo como si de cualquier otro programa ejecutable se tratara:

```
./saludo.rb
```

Lógicamente, para ejecutar éste o cualquier otro programa hace falta que tenga permiso de

ejecución. Si no lo tuviera, tecleamos:

```
chmod +x saludo.rb
```

Otra forma más de ejecutar código ruby consiste en untilizar la opción **-e** con el comando **ruby**. Deberemos escribir el código entre comillas simples. Por ejemplo:

```
ruby -e 'puts "hola mundo"'
```

3. CADENAS DE CARACTERES

Las cadenas de caracteres pueden estar entre comillas dobles ("...") o comillas simples (...)

Una cadena de comillas dobles permite la presencia de caracteres de escape precedidos por una barra invertida (\) y evaluación de una expresión #{ }. Una cadena de comillas simples no realiza esta evaluación, lo que se escribe es exactamente lo que se muestra.

Ejemplos:

```
puts "Hola mundo\nAquí estoy,\tprogramando en ruby\n"
```

muestra

```
Hola mundo
Aquí estoy,    programando en ruby
```

mientra que

```
puts 'Hola mundo\nAquí estoy,\tprogramando en ruby\n'
```

muestra

```
Hola mundo\nAquí estoy,\tprogramando en ruby\n
```

Dentro de una cadena delimitada por comillas dobles se pueden incluir evaluaciones de expresiones, por ejemplo:

```
puts "Tres docenas de huevos son #{3*12} huevos"
```

o bien

```
base=40
altura=20
puts "El rectángulo tiene #{base}cm de base y #{altura}cm de altura, por lo tanto tiene #{base*altura}cm2 de área."
```

La extracción de subcadenas en ruby es trivial. Veamos algunos ejemplos.

```
frase = "Me encanta el lenguaje ruby."
puts frase[4,3]    # tres caracteres a partir de la posición 4
puts frase[8,1]    # el carácter de la posición 8
puts frase[-1,1]   # muestra el carácter de la primera posición empezando por el final
puts frase[-10,5] # muestra 5 caracteres a partir del décimo carácter empezando por el final
```

Hay que tener en cuenta que cuando se empiezan a contar caracteres por la izquierda, el primer carácter corresponde a la posición 0. Sin embargo, cuando se empiza contando por la derecha, el primer carácter corresponde a la posición 1.

Para comprobar si dos cadenas son iguales se utiliza el doble igual.

```
puts "¿Cuál es el lenguaje que más te gusta?"
lenguaje = gets
lenguaje.chop!
if lenguaje == "ruby"
        puts "Tú sí que sabes"
  else
        puts "No conozco ese lenguaje"
end
```

Hemos utilizado **gets** para leer una cadena de caracteres de teclado. Aplicando **chop** a una cadena, se devuelve esa cadena sin el último carácter (salto de línea).

En este caso **lenguaje.chop!** equivale a **lenguaje = lenguaje.chop**

Se puede hacer directamente **lenguaje = gets.chop**, de esa manera primero leemos una cadena por teclado, luego le quitamos el caráter de salto de línea y por último se asigna el valor a la variable **lenguaje**.

A continuación se describen algunos métodos que se pueden aplicar a las cadenas de caracteres:

capitalize capitalize!	Pone en mayúscula el primer carácter.
downcase downcase!	Pasa a minúscula todos los caracteres.
empty?	Devuelve verdadero si la cadena está vacía.
length	Devuelve la longitud de la cadena.
reverse reverse!	Invierte la cadena.
swapcase swapcase!	Cambia mayúsculas por minúsculas y viceversa.
to_f	Convierte una cadena en un número en punto flotante.
to_i	Convierte una cadena en un número entero.
upcase upcase!	Pasa a mayúscula todos los caracteres

En lugar de utilizar **puts**, también podemos utilizar **print**. Esta última instrucción no salta de línea de forma automática después de mostrar una cadena.

4. ESTRUCTURAS DE CONTROL IF Y CASE

La estructura **if** tiene el siguiente formato:

```
if expr [then]
           expr...
       [elsif expr [then]
           expr...]...
       [else
           expr...]
end
```

Ya hemos visto un ejemplo de un **if** en el apartado anterior.

La sintaxis del case es la siguiente:

```
case expr
     [when expr [, expr]...[then]
       expr..]..
     [else
       expr..]
end
```

Veamos un ejemplo.

```
puts "¿Cuál es el lenguaje que más te gusta?"
lenguaje = gets.chop

case lenguaje
        when "ruby"
                puts "Tú sí que sabes"
        when "html", "php", "java"
                puts "Te dedicas a hacer páginas web, ¿verdad?"
        when "cobol"
                puts "Eres un carcamal"
        else
                puts "No conozco ese lenguaje"
end
```

Ejercicios

1. Escribe un programa que pida un día de la semana y diga qué asignatura toca a primera hora ese día.
2. Escribe un programa que calcule el área de un rectángulo.
3. Escribe un programa que calcule el área de un triángulo.
4. Realiza un conversor de euros a pesetas.
5. Realiza un conversor de pesetas a euros.
6. Realiza un programa que pida por teclado una frase y que luego la muestre partida en dos trozos, cada uno de ellos en una línea diferente.
7. Escribe un programa que calcule el salario semanal de un trabajador en base a las horas trabajadas y el pago por hora trabajada. Horas ordinarias (40 primeras horas de trabajo) 12 euros/hora. A partir de la hora 41, se pagan 16 euros/hora.
8. Escribe un programa que una vez leída una hora en formato hh:mm:ss indique cual será la hora dentro de un segundo.