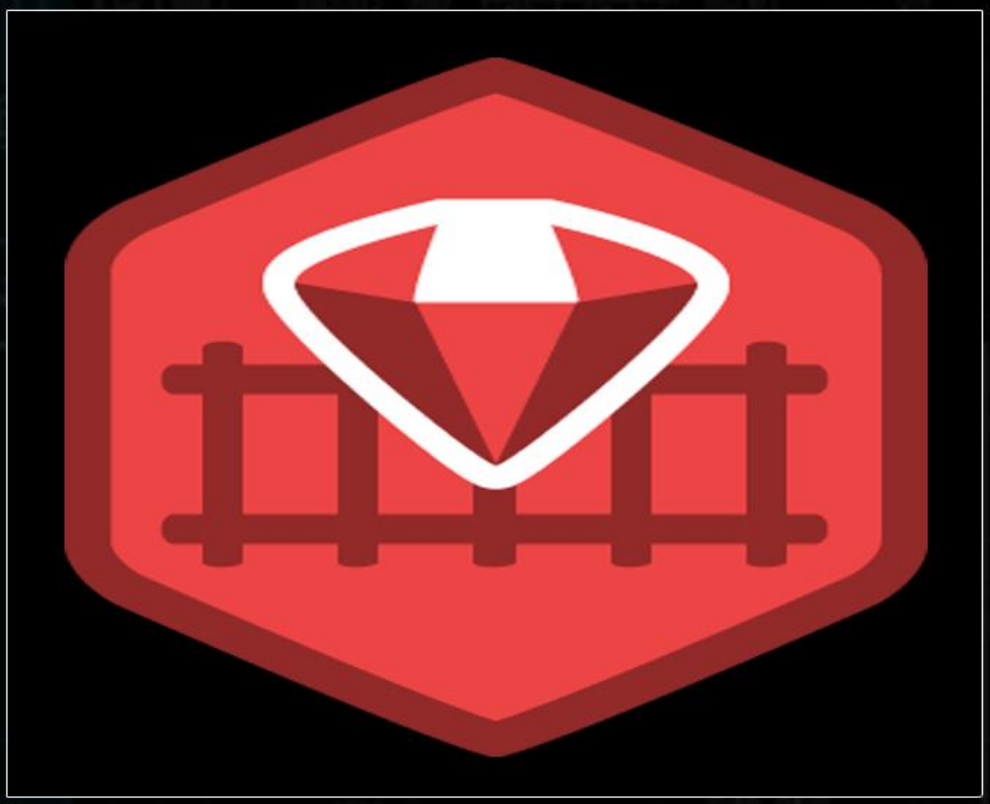
#1

UNDERCODE

TALLER DE PROGRAMACIÓN EN RUBY



TEMAS

¿QUÉ ES RUBY?
INSTALACIÓN
PRIMER PROGRAMA
VARIABLES Y CONSTANTES
EJERCICIOS
Y MÁS!

TUTOR

EXPERMICID

Antes de empezar

Hola a todos y bienvenidos a esta serie de talleres sobre RUBY, en donde iremos aprendiendo por etapas cada uno de los detalles sobre este lenguaje.

Particularmente, en esta primera edición, empezaremos describiendo un poco a RUBY y viendo su instalación, para luego poder entrar a lo que es el código propiamente dicho.

Al final de cada edición se propondrán diferentes ejercicios para practicar lo aprendido, y en el transcurso de unos días, a partir de su lanzamiento, proporcionaremos un video con una posible solución.

Empecemos…

¿Qué es RUBY?

RUBY es un lenguaje interpretado, es decir no necesita ser procesado mediante un compilador. Flexible, permite a los usuarios modificar cualquier aspecto de RUBY y al estar completamente orientado a objetos, ve a todo como un objeto.

Además, RUBY termina siendo un lenguaje multiplataforma y con una sintaxis muy simple.

Instalación

Para obtener RUBY, puedes descargarlo desde la página oficial:

https://www.ruby-lang.org/es/

En el enlace anterior, podrán encontrar el código fuente y asi compilarlo uno mismo, con la posibilidad de hacer configuraciones específicas que sean necesarias en el entorno.

En el caso de utilizar WINDOWS, existe un proyecto para ayudarte a instalar RUBY y que se llama RUBYINSTALLER. Se descargan desde http://rubyinstaller.org/downloads/ la versión que deseen (hoy en día la última versión es la 2.1.3). Luego ejecutan el instalador, y siguen los pasos de instalación hasta finalizar.

Por último, si utilizan LINUX pueden utilizar el gestor de paquetes para instalarlo.

Desde DEBIAN o UBUNTU ponen en la terminal:

```
:~$ sudo apt-get install ruby2.1.3
```

Y obtendrán la última versión.

IRB

IRB es el intérprete de RUBY, permite ingresar comandos o bloques de código que posteriormente serán ejecutados por RUBY.

Para ejecutarlo basta con ir a la terminal e introducir irb.

```
expermicid@Expuesto:~$ irb
irb(main):001:0>
```

¿Qué sucederá si introducimos una cadena de caracteres?

```
expermicid@Expuesto:~$ irb
irb(main):001:0> "Hola Underc0de"
=> "Hola Underc0de"
irb(main):002:0>
```

¿Y si es un número?

```
irb(main):002:0> 10
=> 10
irb(main):003:0>
```

En ambas ocasiones el resultado es similar, nos devolvió lo que habíamos ingresado. Y esa es la forma que tiene el intérprete para decirnos el resultado de la última expresión evaluada.

Podemos usar el IRB para evaluar operaciones aritméticas:

```
irb(main):003:0> 5 + 2
=> 7
irb(main):004:0> 2 * 9
=> 18
irb(main):005:0>
```

También podemos usar variables y operar con ellas:

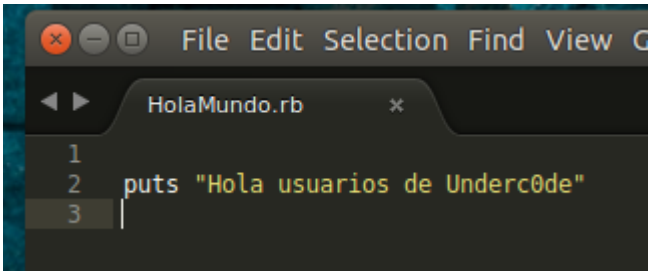
```
irb(main):005:0> a = 5
=> 5
irb(main):006:0> b = 20
=> 20
irb(main):007:0> a + b
=> 25
irb(main):008:0> a = b
=> 20
irb(main):009:0> a
=> 20
irb(main):010:0>
```

Como podrán ver, el uso del intérprete es muy sencillo. Sin embargo, para el transcurro de los talleres usaremos un editor de texto para generar los diferentes script con extensiones '.rb' y luego ejecutarlos a través del intérprete o con el comando RUBY en la terminal.

Primer Programa

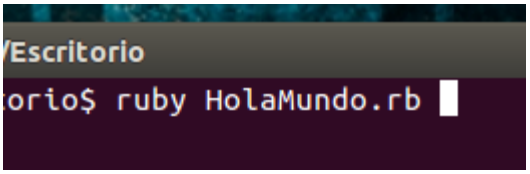
Para iniciarnos con el código haremos nuestro primer programa, que no es ni más ni menos que el clásico 'Hola Mundo'. Al mismo tiempo, es una excusa como para introducir el tema de salida de datos por pantalla.

Abrimos algún editor de texto y ponemos:

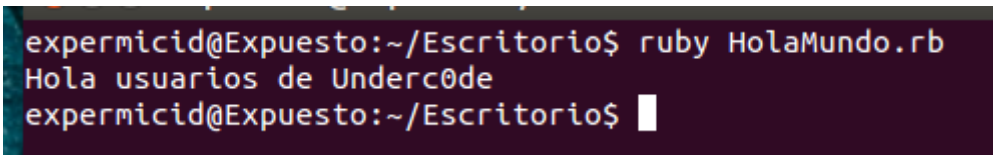


Guardamos -con algún nombre- seguido de la extensión '.rb'.

Abrimos la terminal, nos dirigimos al directorio donde guardamos el archivo y lo ejecutamos de la siguiente forma:



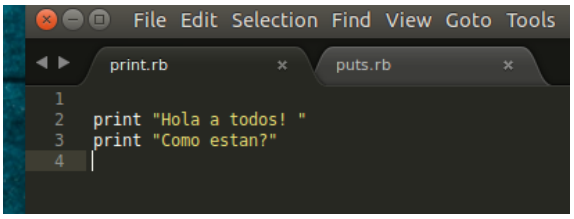
Y obtenemos nuestra frase como salida en pantalla.

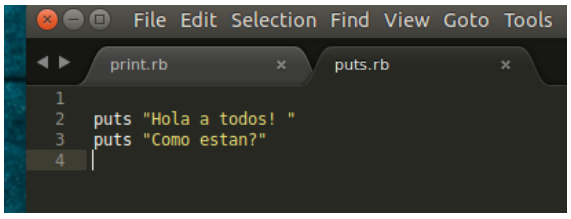


Por lo tanto, podemos determinar que la función de **puts** es imprimir en pantalla lo que se le pase como parámetro, en esta ocasión una cadena de caracteres.

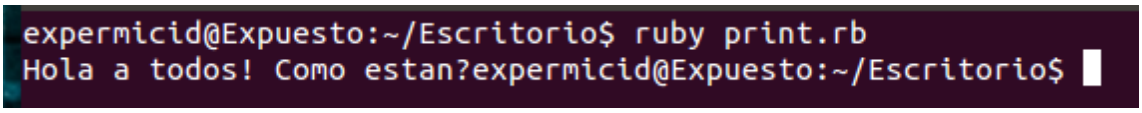
Además de **puts,** se puede utilizar **print** que tiene un resultado muy similar, pero con la siguiente principal diferencia:

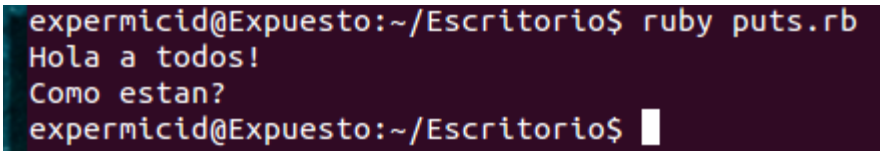
Veamos que obtenemos si ejecutamos estos dos códigos:





En ambos, estamos usando dos líneas para imprimir las mismas cadenas de caracteres, uno utilizando **puts** y el otro **print,** pero la salida es:





Como se puede observar, cuando se utilizó **print**, a pesar de haberlo usado dos veces las cadenas están una al lado del otro. Mientras que con **puts,** hay una nueva línea después de cada utilización.

Variables y Constantes

Variables

Una variable no es más que un espacio en memoria que se reserva para almacenar datos. En particular, en RUBY, a diferencia de muchos otros lenguajes no es necesario definir el tipo de datos.

Las variables son creadas en el momento de darles un valor y toma el 'tipo' del valor que se le asigna.

A la hora de darles un nombre hay que tener en cuenta algunas **reglas**:

- No puede contener espacios en blanco.
- No puede comenzar con un número.
- No puede ser una palabra reservada, tales como **end**, **if**, **def**, **class**, etc.
- Debe empezar con una letra en minúscula. Preferentemente todo en minúsculas.

Ahora, veremos algunos ejemplos:

```ruby
varCadena = "varCadena es una variable que almacena una cadena de caracteres"
varNumerica = 10

puts varCadena
print "varNumerica es una variable que almacena el numero "
puts varNumerica
```

En este programa, en la primera línea creamos una variable llamada **varCadena** y que almacena una cadena de caracteres. En la segunda línea en cambio creamos otra variable, **varNumerica**, pero esta vez almacena un número.

Luego en las líneas 4, 5 y 6 hacemos uso de **print** y **puts** para mostrar algunos mensajes en pantalla.

Veamos el resultado:

```
expermicid@Expuesto:~/Escritorio$ ruby variables.rb
varCadena es una variable que almacena una cadena de caracteres
varNumerica es una variable que almacena el numero 10
expermicid@Expuesto:~/Escritorio$
```

Podemos observar como en la línea 4 y 6 se imprime el valor almacenado que tenían las variables.

Ahora, en este segundo ejemplo:

```ruby
var1 = 200
var2 = "Undec0de"

puts "var1 almacena #{var1}"
puts "var2 almacena #{var2}"

varAux = var2
var2 = var1
var1 = varAux

puts "var1 almacena #{var1}"
puts "var2 almacena #{var2}"
```

Nos va a dejar -bien en evidencia- la independencia que existe en las variables y el tipo de datos.

Ya que como podrán ver en la línea 1 creamos una variable que almacena un número y en la línea 2 creamos otra variable pero que contiene una cadena de caracteres.

Ya en las líneas 7, 8 y 9 usamos una variable auxiliar para hacer el intercambio de valores entre las otras dos variables creadas primero. Esta acción, en cualquier otro lenguaje en donde al momento de declarar una variable se le asigna un tipo de datos, sería imposible hacerlo directamente.

En las líneas 4 y 5 -que son iguales a la 11 y 12- imprimimos en pantalla los valores de las variables para comprobar el exitoso intercambio de valores. Podrán observar algo nuevo en estas líneas y que son los #{var}. Estos símbolos #{} dentro de unas dobles comillas realizan una evaluación de lo que se encuentra dentro de las llaves.

Veamos si funciona bien:

```
expermicid@Expuesto:~/Escritorio$ ruby variables2.rb
var1 almacena 200
var2 almacena Undec0de
var1 almacena Undec0de
var2 almacena 200
expermicid@Expuesto:~/Escritorio$
```

Todo sin problemas.

Constantes

Las constantes, son espacios en memoria para almacenar datos, pero con la particularidad de que su valor no debería cambiar. Se crean al igual que las variables, cuando se les asigna un valor y comúnmente se las nombra con la primera letra en mayúscula o mejor todo en mayúscula.

En la versión actual de RUBY intentar cambiar el valor de una variable genera una advertencia, pero no da un error.

Ejemplo:

constantes.rb

```
1  CONSTANTE = "Creacion de constante"
2  puts CONSTANTE
3
4  CONSTANTE = "Cambiando el valor de la constante"
5  puts CONSTANTE
```

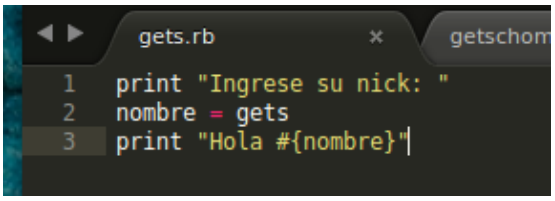
Creamos una constante con un valor inicial y luego cambiamos su valor. Si lo ejecutamos nos daría la advertencia pero igual reasignaría su valor.

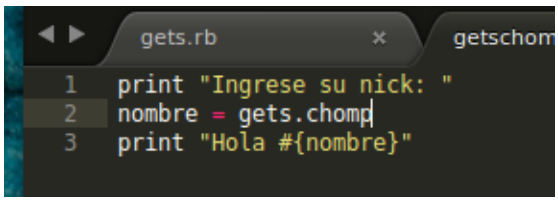
```
expermicid@Expuesto:~/Escritorio$ ruby constantes.rb
Creacion de constante
constantes.rb:4: warning: already initialized constant CONSTANTE
Cambiando el valor de la constante
expermicid@Expuesto:~/Escritorio$
```

Ejemplo integrador

Antes de ver un ejemplo que incluya todo lo visto, hablemos un poco sobre la entrada de datos. En RUBY está el método **gets** que realiza dicha función, obtener lo que se ingresa por teclado. El único inconveniente es que además almacena el carácter de retorno de carro (\n), para solucionar eso existe el método **chomp** que su función es borrar dicho carácter de la cadena.

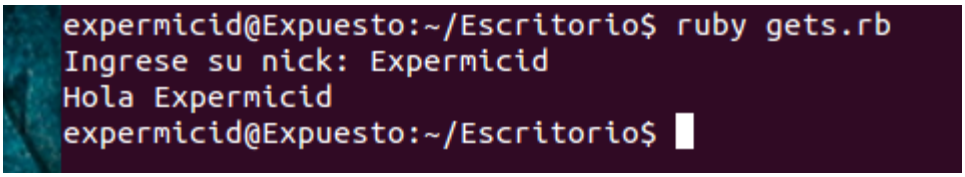
Veamos un ejemplo de su utilización:

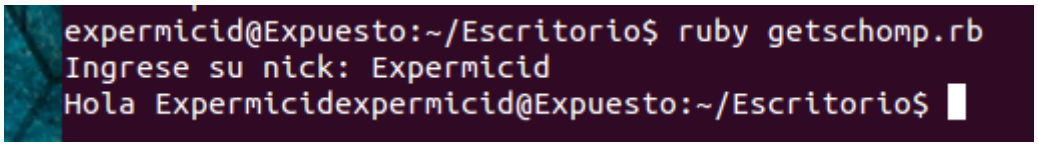




El mismo código, uno utilizando **gets** y el otro **gets.chomp**, ambos guardan lo que se ingresa en una variable. Las demás líneas son salidas en pantalla con **print** como vimos anteriormente.

Ejecutamos ambas y tenemos:





Como sabemos, **print** no agrega una nueva línea al final; por lo tanto, cuando utilizamos el **gets** existe una nueva línea en el último **print**, y que es de la cadena obtenida por teclado. Sin embargo, con el **gets.chomp** se ve claramente que la nueva línea ya no existe.

Ahora si, el enunciado del ejemplo integrador dice asi:

“Se desea hacer un programa que pida al usuario los siguientes datos: apellido, nombre, fecha de nacimiento, edad. Luego de que se hayan ingresado todos los datos se informan, mediante salidas en pantallas, lo obtenido.”

Si analizamos rápido el enunciado nos daremos cuenta que para el ingreso de datos vamos a necesitar del uso de **gets**, para las salidas en pantalla **puts** o **print** y además necesitaremos de diferentes variables para guardar los datos.

Una posible solución seria:

```ruby
print "Ingrese su Apellido: "
apellido = gets.chomp
print "Ingrese su Nombre: "
nombre = gets.chomp
print "Ingrese su Fecha de Nacimiento: "
fechaNacimiento = gets.chomp
print "Ingrese su edad: "
edad = gets.chomp

puts "SUS DATOS SON:"
puts "Apellido: #{apellido}"
puts "Nombre: #{nombre}"
puts "Fecha de Nacimiento: #{fechaNacimiento}"
puts "Edad: #{edad}"
```

El programa hace de forma iterada un **print**, en donde se detalla el dato que se desea, seguido de una línea en donde se almacena en una variable lo obtenido por teclado con **gets.chomp**. Se hace eso por cada dato que se quiere. Luego mediante diferentes **puts** se va informando lo que se guardó en cada variable.

Ejecutamos y...

```
expermicid@Expuesto:~/Escritorio$ ruby ejemploIntegrador.rb
Ingrese su Apellido: Rodriguez
Ingrese su Nombre: Juan Pepito
Ingrese su Fecha de Nacimiento: 9/12/1995
Ingrese su edad: 18
SUS DATOS SON:
Apellido: Rodriguez
Nombre: Juan Pepito
Fecha de Nacimiento: 9/12/1995
Edad: 18
expermicid@Expuesto:~/Escritorio$
```

Todo funciona a la perfección.

Ejercicios para practicar

1. Realice un programa que pida al usuario ingresar su nombre o Nick e imprima un saludo de la forma “Hola (nombreOnick). Espero que te haya gustado el primer taller de Ruby”
2. Dicho programa, imprima en una línea el titulo “Ejercicio para practicar nro 2”, deje una línea en blanco y luego pida al usuario que ingrese su Nick, deja nuevamente una línea en blanco y muestra la frase “Gracias por visitar Underc0de (Nick)”