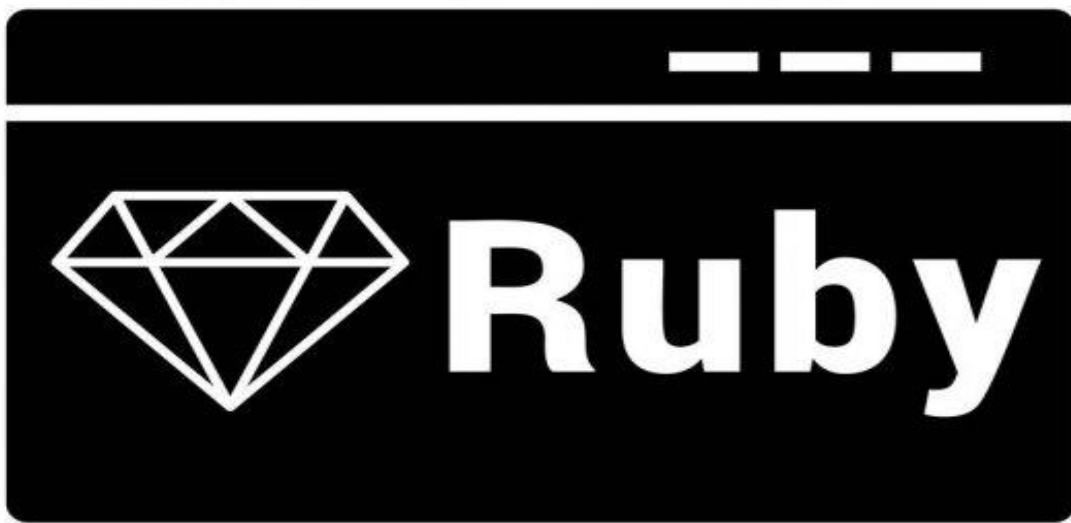


Ruby

Sintaxis básica



Introducción

En este documento se tratará la sintaxis básica del curso de Ruby, con algunos ejemplos.

SECCIONES

- **Getting started**

- **Método puts** : El método puts sirve para imprimir en consola cualquier tipo de objeto.
- **Método print** : De manera similar el método print imprime en consola cualquier tipo de objeto pero al final de cada impresión no genera un salto de línea como si lo hace 'puts' .
- **Método p** : Método que imprime en consola, pero con un formato diferente en algunos casos. El método p es similar al método inspect usado con un puts.
- **Comentarios** : Los comentarios en Ruby se representan de dos maneras:
 - **Comentario de línea** : El comentario de línea se representa por el símbolo '#' y después de esto el comentario a realizar.
 - **Comentario de bloque** : Comentario de bloque se representa por las palabras '=begin' Todo lo que está dentro de esta cláusula es un comentario de más de una línea o comentario de bloque '=end'.
- **Constantes** : Las constantes se representan con identificadores en MAYÚSCULA.
- **Interpolación** : La interpolación es la unión, concatenación de una cadena con cualquier tipo de variable y se representa de la siguiente manera, "Hello #{Juan} ", IMPORTANTE LA INTERPOLACIÓN SÓLO SE PUEDE REALIZAR USANDO COMILLAS DOBLES.
- **Método gets, chomp** : El método gets se usa para recibir datos escritos por consola, el método chomp elimina todo los saltos de línea creados en la consola para no alterar el resultado.

- **Numbers and Booleans**

- **Métodos de conversión de datos :**

- `to_i` : Método que convierte una cadena, o un valor numérico a entero.
 - `to_f` : Método que convierte una cadena, o un valor numérico a float.
 - `to_s` : Método que convierte una cadena, o un valor numérico a string.

- **Métodos `odd?`, `even?` :**

- `odd?` : Método boolean que retorna true si el número es impar de lo contrario retorna false.
 - `even?` : Método boolean que retorna true si el número es par de lo contrario retorna false.

- **Método `between?` :** Método boolean que retorna si un numero esta dentro de un rango ejemplo: `'20.between(10, 30)'` true.

- **Métodos float :**

- `floor` : El método floor aproxima al número más cercano por debajo de el, ejemplo `10.5 = 10`.
 - `ceil` : El método ceil aproxima al número más cercano por encima de el, ejemplo `10.5 = 11`
 - `round` : El método round redondea al número más cercano ya sea por encima o por debajo.
 - `abs` : retorna el valor absoluto de un número.

- **Método `times` :** El método times tiene la función de iterar n veces se le indique y poder realizar instrucciones en esas iteraciones.

- **Método `downto`, `upto` :**

- `downto` : El método `downto` es una iteración que empieza en cierto número y disminuye hasta otro valor indicado, durante esta iteración se puede realizar diferentes operaciones, ejemplo `5.downto(1)`.
 - `upto` : El método `upto` es una iteración que empieza en cierto número y aumenta hasta otro valor indicado, durante esta iteración se puede realizar diferentes operaciones, ejemplo `5.upto(12)`.

-
- **Método step** : El método step es otro tipo de iteración en rangos, pero en este caso con un patrón definido, ejemplo empezamos en el número 1 y vamos hasta el número 100 pero saltando de 5 en 5.
1.step(100, 5) en el transcurso de esta iteración se pueden realizar diferentes tipos de sentencias.

- **Strings I**

- **Metodo capitalize** : El metodo capitalize convierte la primer letra de una cadena en mayuscula.
- **Metodo upcase** : El metodo upcase convierte todo una cadena en mayuscula.
- **Metodo downcase** : El metodo downcase convierte toda una cadena en miniscula.
- **Metodo reverse** : El metodo severse invierte el sentido de una cadena.
- **Metodo include?** : El metodo include? verifica si una cadena incluye ciertos coaracteres o caracter que se le pasa por parametro, retorna un boolean.
- **Metodo empty?** : El metodo empty? retorna si una cadena esta vacia.
- **Metodo nil?** : El metodo nil? verifica si un objeto es null, o nil para este caso.
- **Metodo slice** : El metodo slice se utiliza para obtener un caracter o multiples caracteres de una cadena.
- **Metodo swapcase** : Invierte las minisculas por mayusculas y vice versa.
- **MLS** : Los MLS o multi line string, son Strings de mas de una linea que tiene la sintaxis siguiente : words = <<MLS this is a MLS MLS
- **Caracteres de escape** : Los caracteres de escape sirven para hacer diferentes acciones dentro de un String, por ejemplo:
 - \n : Genera un salto de linea
 - \t : Genera un tab en el string

Los saltos de linea solo se pueden usar con comillas dobles "".

- **Comilla doble vs Comilla simple** : Las comillas simples son procesadas mas rapido por el interprete de ruby ya que tienen que hacer menos validaciones que con las dobles, ejemplo: no tiene que validar secuencias de escape, interpolacion entre otras.
- **Metodos Bang!** : Los metodos Bang, son metodos que sobreescriben o mutan un objeto, por eso son metodos peligrosos se reconocen por que en la terminacion en su nombre tienen el signo de "!".

- **Strings II**

- **Metodo each_char** : Este metodo retorna un String caracter por caracter
- **Metodo split** : Este metodo retorna un array con cada uno de los caracteres de un string, segun su delimitador.
- **Metodo chars** : Este metodo retorna un array con cada uno de los caracteres de un String.
- **Metodo clear** : Este metodo limpia todo un String, es un metodo peligroso ya que muta el estado del String.]
- **Metodo count** : Este metodo cuenta cuantas veces esta un caracter dentro de un String.
- **Metodo delete** : Este metodo elimina uno o varios caracteres de una String, es un metodo peligroso ya que muta el estado del String.
- **Metodo index** : Este metodo retorna el indice de un caracter en una cadena.
- **Metodo insert** : Este metodo inserta un caracter en una poscion dada, es un metodo peligroso ya que muta el estado del String.
- **Metodo join** : Agrega un caracter en un arreglo despues de cada posicion.
- **Metodo squeeze** : Metodo que elimina todos los caracteres repetidos.

- **Metodos y Condicionales I**

- **Retornos** : Los retornos deben ser implicitos y siempre estan en la ultima linea del metodo, pero si lo que queremos es terminar con la ejecucion del metodo y retornar un valor debemos usar la palabra reservada return.
- **True - False** : Todo en Ruby es true excepto el : false y el nil.
- **Metodo respond_to?** : El metodo respond_to? recibe como parametro el nombre de un metodo en cadena de texto o simbolo, y retorna true o false si el objeto tiene como funcionalidad el metodo enviado por parametro.

- **Metodos y Condicionales II**

- **Condicional ||=** : Este condicional se usa para asignar un valor a variable que podria ser nil o false, ejemplo:

y = nil ; y ||= 5 ; p y

Esta condicion asigna el valor de 5 a 'y' porque 'y' es nil si y no fuera nil, no le asignaria el valor de 5.

- **Rangos**

- **Los rangos** se puede definir de manera explicita a si: 100..230 , a..d o con el metodo rand(100..200).
- **Metodo first** : El metodo first imprime en un arreglo los primeros n valores del rango, n es un parametro.
- **Metodo last** : El metodo last imprime en un arreglo los ultimos n valores del rango, n es un parametro.
- **Metodo include?** : El metodo include? nos devuelve si un valor que se le ingresa por parametro existe o no en ese rango.
- **Metodo size** : El metodo size nos da el tamano del rango.

Si el rango solo lleva dos puntos, incluye el ultimo valor del rango si lleva tres puntos no lo incluye.

- **Arrays I**

- **Un Arreglo** de String puede definirse con la siguiente sintaxis %w[], no se deben poner apostrofes ni separarlos con comas. solo se debe usar para crear arreglos de palabras no compuestas.
- **Metodo values_at** : El metodo values_at obtiene los datos del arreglo dado por parametro su indice.
- **Metodo to_a** : Convierte un objeto a un arreglo
- **Crear Arreglo** : Una manera de crear un arreglo es a si: Array.new(10, 0)

Donde el primer parametro es el tamano del arreglo y el segundo el valor por defecto con el cual se va a inicializar este.

- **Metodo fetch** : El metodo fetch obtiene un valor dado un indice.
- **Metodo is.a?** : El metodo is.a? retorna si un objeto es o no es un arreglo.
- **Metodo count** : El metodo count cuando se usa sin parametros es equivalente a los metodos size, length. Pero cuando count tiene parametros, cuenta cuantas veces esta en el arreglo el parametro enviado.
- **Metodo pop** : El metodo pop elimina el ultimo elemento del arreglo arreglo, cuando a pop se le envia un parametro elimina los ultimos n elementos del arreglo.

-
- **Metodo push** : El metodo push agrega elementos al final del arreglo una sintaxis un poco mas corta para agregar elementos al final de arreglo es la siguiente : "<<".
 - **Metodo insert** : El metodo insert sirve para agregar elementos en cualquier posicion de un arreglo, el primer parametro es la posicion del arreglo donde se van a insertar los elementos.
 - **Metodo shift** : Cuando al metodo shift no se le envian parametros elimina la primera posicion del arreglo, cuando se le envian elimina desde la primera poscion hasta el indice que se le envio por parametro.
 - **Metodo unshift** : El metodo unshift agrega en las primeras posciones los elementos que se le envian por parametro.
 - **Operador spaceship** : El operador spaceship se usa cuando se implementa el modulo de comparacion y los resultados son los siguientes:
if a < b then return -1
if a = b then return 0
if a > b then return 1
if a and b are not comparable then return nil

- **Arrays II**

- **Metodo each** : El Metodo each sirve para recorrer un arreglo con una sentencia de bloque.
- **Metodo each_with_index** : El metodo each_with_index sirve para recorrer un arreglo y poder saber su indice en todo momento.
- **Metodo select** : El metodo select retorna un arreglo con los elementos seleccionado que cumplan la condicion de seleccion.
- **Metodo reject** : El metodo reject retorna un arreglo con los elementos seleccionado que no cumplen la condicion de seleccion.
- **Metodo partition** : El metodo partition devuelve dos arreglos con los elementos aceptados y no aceptados segun la condicion.
- **Metodo map** : El metodo map evalua todos los elementos de un arreglo y devuelve otro arreglo con las respuestas de esas evaluaciones o operaciones.
- **Metodo concat** : El metodo concat concatena dos arreglos.

-
- **Metodo include?** : El metodo include? verifica si un elemento esta dentro de un arreglo.
 - **Metodo index** : El metodo index devuelve el indice de la primera ocurrencia del elemento buscado.
 - **Metodo max** : Retorna la ultima posicion de un arreglo.
 - **Metodo min** : Retorna la primera posicion de un arreglo.
 - **Metodo reverse** : El metodo reverse invierte un arreglo.
 - **Metodo sort** : El metodo sort ordena un arreglo de menor a mayor
- **Arrays III**
 - **Simbolo &** : El simbolo & retorna la interseccion de dos arreglos sin repetir elementos.
 - **Simbolo |** : El simbolo | retorna la union de dos arreglos sin repetir elementos.
 - **Metodo dub** : El metodo dub sirve para duplicar un arreglo.
 - **Metodo any?** : El metodo any? retorna si algun elemento cumple la condicion, retorna un boolean.
 - **Metodo all?** : El metodo all? retorna si todos los elementos cumplen la condicion, retorna un boolean.
 - **Metodo compact** : El metodo compact devuelve un arreglo descartando los valores nil.
 - **Metodo find** : El metodo find retorna el primer elemento que cumpla la condicion dada.
 - **Metodo flatten** : El metodo flatten convierte un arreglo multidimensional en unidimensional.
 - **Metodo reduce** : sirve para reducir algunas operaciones con arreglos que podemos obtener en el mismo instante el valor actual y el valor siguiente del arreglo.
 - **Metodo sample** : El metodo sample obtiene ya sea un valor o varios de manera aleatoria de un arreglo.
 - **Metodo uniq** : El metodo uniq elimina todos los valores repetidos en un arreglo.
 - **Metodo zip** : Transforma dos arreglos unidimensionales en uno bidimensional.
-

- **Block, procs y Lambdas**

- **Block** : Un bloque es una trozo de código, una función sin nombre que puede ser adherida a otra función, se puede crear usando llaves o con las palabras reservadas `do` `end`. No es un objeto
- **Lambdas** : Son sentencias de bloque, pero un poco más estrictas con relación a los parámetros que recibe.
- **Proc** : Es también una sentencia de bloque, muy parecido a una Lambda pero este no tan estricto con relación a los parámetros.
- **yield** : Yield es un tipo de retorno desde el método hacia el bloque.
- **Método a Proc** : Para poder llamar un método como si fuera un proc solo debemos usar el carácter `&`:nombre_método.
- **Método block_given** : Este método verifica si existe un bloque, para poder ejecutarlo a través del yield.

- **File input and output**

- **ARGV** : ARGV sirve para poder recibir datos por consola al mismo tiempo
- **File.open** : Sirve para abrir un archivo y poder leerlo o escribir en él.
- **Método write** : Sirve para escribir en un archivo.
- **Método exist?** : Verifica si un archivo existe.
- **Método delete** : Elimina un archivo.
- **load** : Sirve para cargar un archivo o clase desde otra.

- **Hashes I**

- **Símbolo** : Un símbolo es el objeto más básico que puedes crear en Ruby: es un nombre y una ID interna. Los símbolos son útiles por que dado un símbolo, se refiere al mismo objeto en todo el programa. ... Si el contenido del objeto es lo importante, usa un string. Si la identidad del objeto es importante, usa un símbolo.
- **Método to_s** : Convierte un objeto en un String.
- **Método to_sym** : Convierte un string en un símbolo.
- **Método keys** : Devuelve todas las claves del hash en un arreglo.
- **Método values** : Devuelve todos los valores del hash en un arreglo.

- **Hashes II**

- **Metodo to_h** : Convierte un arreglo en un hash.
- **Metodo default** : Crea un valor por defecto en un hash.
- **Metodo delete** : Elimina un item del hash.
- **Metodo key?** : Verifica si lo enviado por parametro es una key del hash.
- **Metodo value?** : Verifica si lo enviado por parametro es un value del hash.
- **Metodo merge** : Une dos hash si tiene una key repetida guarda el valor del segundo hash.
- **Metodo sort** : El metodo sort ordena un hash por su key.
- **Metodo sort_by** : El metodo sort_by ordena un hash por su key o por su value.

- **The Time Object**

- **Crear un time** : Para crear un time se puede hacer de las siguientes maneras:
 - **Time.new** : Crea una fecha con la actual.
 - **Time.new(1999, 05, 05)** Crea una fecha con la enviada por paramaetro.
 - **Time.now** : Crea una fecha con la actual
- **Metodo strftime** : Agrega un formato a una fecha ejemplo :
strftime("%d-%B-%j").

- **Expresiones regulares**

- **Metodo start_with?** : Metodo que retorna un valor boolean si la cadena inicia con los characters ingresados por parametro.
- **Metodo end_with?** : Metodo que retorna un valor boolean si la cadena inicia con los caracteres ingresador por parametro.
- **=~** : Esta sentencia sive para saber si una cadena incluye un valor enviado por parametro, retorna el indice de la primera ocurrencia.

-
- **Metodo scan** : Este metodo retorna todas las ocurrencias de la cadena con el respectivo filtro echo en el parametro con la expresion regular, esto lo retorna en un arreglo.
 - **Metodo sub** : Metodo que remplaza la primer ocurrencia de una cadena o expresion regular enviada por parametro por otra tambien enviada por parametro.
 - **Metodo gsub** : Metodo que remplaza todas las ocurrencias de una cadena o una expresion regular enviada por parametro por otra tambien enviada por parametro.
 - **web Rubular** : <https://rubular.com/>