**Apuntes**

La programación no se trata de memorizar código sino de **aplicar conocimiento**.

Realmente no hay necesidad de memorizar nada. Todo vendrá de manera natural a medida que trabajes en nuevos proyectos.  
Mientras tanto puedes consultar los [recursos](https://rubyapi.org/2.6) para responder las preguntas que surjan..

¡Por suerte tenemos **toda la información clave documentada** para que recuerdes todos los **conceptos de hoy**

**Tipos de datos**

Todo en Ruby es un object. Los objetos tienen métodos integrados a los que puedes llamar.

**Integers**

* Representan a los números enteros
* Nos permiten hacer operaciones aritméticas

4.**class** *# => Integer*

1 **+** 2 *# => 3*

2 **\*** 4 *# => 8*

4 **/** 2 *# => 2*

* También tienen métodos integrados personalizados

20.**even?** *# => true*

20.**odd?** *# => false*

* Puedes convertir números en strings

1984.**to\_s** *# => "1984"*

**Floats**

* Representan a los números decimales

3.14.**class** *# => Float*

1.23 **+** 2.1 *# => 3.33*

* Tienen sus propios métodos integrados

3.14.**round** *# => 3*

**Strings**

* Representan al texto
* Están definidos por comillas simples o dobles: 'wagon' o "wagon"

"wagon".**class** *# => String*

'wagon'.**class** *# => String*

'Hello world!'. **class** *# => String*

"wagon".**upcase** *# => "WAGON"*

"wagon".**capitalize** *# => "Wagon"*

* Puedes convertir strings en números

'1984'.**class** *# => String*

'1984'.**to\_i** *# => 1984*

'1984'.**to\_i**.**class** *# => Integer*

* Puedes unir dos strings usando la concatenación:

"Ruby" **+** " " **+** "workshop" *# => "Ruby workshop"*

* Podemos agregar código Ruby en un string usando interpolación. Sin embargo, esto sólo es posible con strings que tienen comillas dobles. Si utilizas comillas simples, Ruby simplemente mostrará lo siguiente:

'two: #{1 + 1}' *# => "two: #{1 + 1}"*

"two: #{1 **+** 1}" *# => "two: 2"*

**Booleans**

* Representan algo que es verdadero o falso (se usan el término en ingles **true**, **false**):

18 **<** 25 *# => true*

10 **+** 5 **==** 20 *# => false*

**¿Cómo podemos hacer que nuestro código se muestre?**

Puts y print se utilizan para mostrarle información al usuario en la consola.  
Puts también muestra una línea después del contenido.

gets.chomp se utiliza para obtener datos de entrada del/de la usuario/a lo cuales podemos almacenar en una variable. El .chomp nos permite remover los espacios vacíos o caracteres que el/la usuario/a agregue como cuando presiona enter, por ejemplo.

print 'Hey, '

puts "what's your name?"

name **=** gets.**chomp**

puts "Welcome #{name}!"

devuelve:

*Hey, what’s your name?*

Lo que significa: “Hola, ¿Cómo te llamas?” El/la usuario/a introduce su nombre, ‘John’, y presiona enter. El programa almacenará dicho nombre y mostrará un mensaje de saludo para el/la usuario/a:

*Welcome John!*

Lo que significa “Bienvenido, John”  
️ gets.chomp no se puede usar en el editor de código. ¡No te preocupes ya que no lo necesitarás en los desafíos!

**Variables**

Veamos cómo podemos almacenar y reutilizar datos en nuestro código. Las variables:

* te permiten almacenar valores y usarlos luego
* pueden sobreescribirse e incrementarse

age **=** 21

puts "You are #{age} years old"

*# => You are 21 years old*

age **=** age **+** 1

puts "You are now #{age}!"

*# => You are now 22!*

* La convención nos indica que debemos escribir los nombres de las variables en snake\_case (palabras de letras minúsculas separadas por guiones bajos)
* Luego le **asignas** un valor a la variable

first\_name **=** "Alex"

last\_name **=** "Benoit"

puts "My name is #{first\_name} #{last\_name}"

**Control de flujo**

**Flujo básico**

El programa lee y ejecuta nuestro código línea por línea:

age **=** 21

puts "You are #{age} years old"

*# => You are 21 years old*

age **=** age **+** 1

puts "You are now #{age}!"

*# => You are now 22!*

**¿Cómo cambiar el flujo?**

Para cambiar el flujo, utilizamos declaraciones condicionales y bucles (también conocidos como loops)

* Las declaraciones condicionales con If nos permiten ejecutar pedazos de código si una condición se cumple (se utiliza el término “thruthy”).

puts "How old are you?"

print ">"

age **=** gets.**chomp**.**to\_i**

**if** age **>=** 18

puts "you can vote!"

**end**

El programa **solo** mostrará “You can vote” (lo que significa “Puedes votar”) si se cumple la condición i.e. si la edad suministrada por el/la usuario/a es mayor a 18.

* Las declaraciones condicionales If/Else nos permiten ejecutar una parte específica de un código determinado si se cumple una condición (se usa el término “thruthy” **u** otra parte de dicho código si la misma condición **no** se cumple (es decir, no es “truthy”).

**if** age **>=** 18

puts "you can vote!"

**else**

puts "too young to vote.."

**end**

El programa mostrará uno de los dos mensajes **de acuerdo** a la edad suministrada por el/la usuario/a.

¡Esperamos que disfrutes este desafío!

Yendo más lejos para aprender conceptos avanzados:

**Tu primer método**

Los métodos nos permiten almacenar **bloques de código que podemos reutilizar** llamándolos por sus nombres. Esto nos permite evitar repetir código (en inglés se utiliza la abreviación **DRY**: Don’t Repeat Yourself que significa “No te repitas a ti mismo/a”).

Debemos definir el método antes de llamarlo.

Los métodos toman **parámetros** y gracias a estos, podemos usar los métodos con entradas dinámicas cuando los necesitemos.  
Los parámetros funcionan como marcadores de posición (también conocidos como placeholders) para los **argumentos** que le asignes a tu método cuando lo llames.

**def** **say\_hi**(name)

**return** "Hi #{name}!"

**end**

puts say\_hi("Alex") *# => "Hi Alex!"*

puts say\_hi("Edward") *# => "Hi Edward!"*

Usamos el parámetro name en la definición del método para referirnos al string que será devuelto cuando llamemos al método.  
Después llamamos al método pasándole un **argumento**, el string ‘Alex’. Nuestro programa ejecutará el código dentro de la definición del método y reemplazará al parámetro ‘name’ con el argumento que le demos. Esto devolverá “Hi Alex!”

**Arrays (también conocidos como arreglos)**

* Pueden almacenar una **lista** de elementos.
* Los elementos pueden ser de cualquier tipo de datos. Es posible almacenar elementos de diferentes tipos de datos en el mismo array pero normalmente trabajamos con un solo tipo.
* Los elementos se separan con **comas** y el array se define con **corchetes** (también conocidos como square brackets).

array **=** [1, 4, 6, 7]

beatles **=** ['John', 'Ringo', 'Paul', 'George']

* Los arrays también tienen **métodos integrados** que podemos usar

['Sponge Bob', 12, 3.14].**size** *#=> 3*

['Huey', 'Dewey', 'Louie'].**sort** *#=> ["Dewey", "Huey", "Louie"]*

[3, 5, 1].**sort** *#=> [1, 3, 5]*

[1, 5, 7, 9, 12].**sample** *#=> 7 (random sample)*

* Podemos agregar y borrar elementos de un array:

cities **=** ['London', 'Paris']

cities **<<** 'Berlin' *#=> ["London", "Paris", "Berlin"]*

cities.**delete**('Paris') *#=> ["London", "Berlin"]*