

## Informatorio Chaco I 2024 | Desarrollo web

Aprendamos

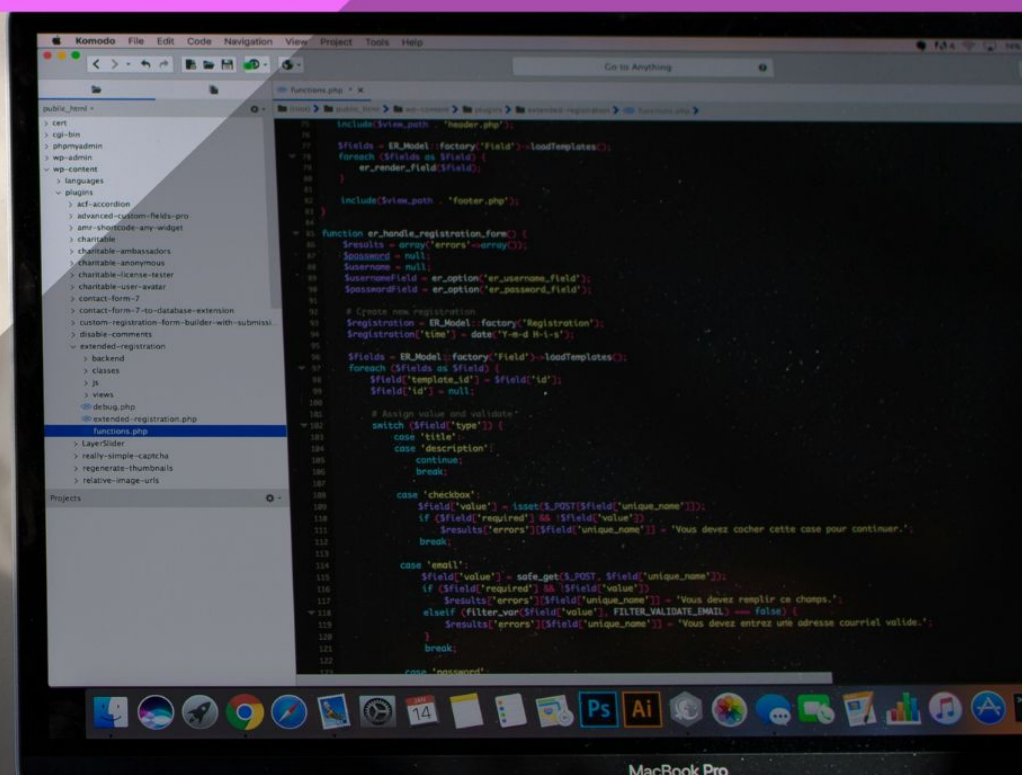
# Python

El lenguaje número uno - 2024

*Versátil y fácil de usar*

SEMANA

# 2



## &gt;&gt;&gt; ¿Qué es Python ?

# Hablemos sobre nuestro lenguaje de programación

SEMANA 2



La historia de Python se comienza en Amsterdam, cuando holandés Guido van Rossum y otros programadores trabajaban en la creación del lenguaje ABC en el CWI (Centro de Investigación de Ciencias Matemáticas e Informática), donde Lambert Meertens fue el mentor, el que le enseñó todo lo sobre diseño de lenguajes de programación.

“Era un entorno muy estimulante con investigadores que venían de todo el mundo, gente muy inteligente de Alemania, Grecia y por supuesto Estados Unidos” fueron las palabras de Guido.

En 1989, Van Rossum decidió crear un lenguaje de programación fácil de entender y simple de usar, lo que dio origen a Python.

El nombre del lenguaje fue inspirado en Monty Python Flying Circus, un programa de humor favorito de Guido van Rossum (no en la serpiente, como algunos piensan), y se convirtió en un lenguaje de programación popular gracias a su facilidad de uso y su gran comunidad.

Hoy en día, Python es utilizado en diversos campos como la creación de aplicaciones web, inteligencia artificial y hasta juegos. Cuenta con múltiples bibliotecas y, sobre todo, es de código abierto.

Este lenguaje de programación de propósito general es poderoso, versátil, fácil de usar y ha ganado una gran cantidad de seguidores en todo el mundo. Con una comunidad activa y una gran cantidad de recursos disponibles.

## Áreas fuertes de Python:

- **Inteligencia Artificial**
- **Big Data**
- **Data Science**
- **Desarrollo WEB**
- **Biología**

## ¿Quiénes usan Python?

- BitTorrent
- Instagram
- Battlefield 2
- Spotify
- Netflix
- Google usa Python
- YouTube es hecha en Python
- Nasa usa Python
- Google App Engine
- Pinterest
- Panda3D
- Facebook
- Ubuntu Software Center
- Dropbox
- Quora

## Características, cómo funciona y ventajas

- *Una excelente opción tanto para principiantes como para desarrolladores experimentados que buscan una herramienta poderosa y flexible para sus proyectos.*

Vamos a agrupar a los lenguajes de programación en INTERPRETADOS y COMPILADOS.

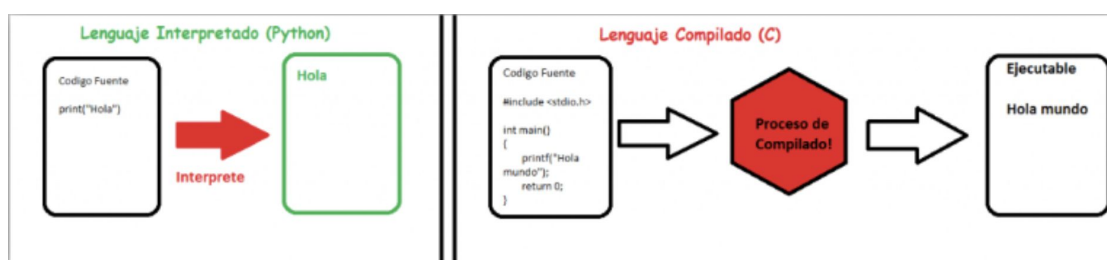
Cuando programamos, utilizamos un

lenguaje de programación para comunicarnos con la computadora y darle instrucciones sobre qué hacer.

Un lenguaje de programación compilado es como un traductor que toma lo que escribimos en un lenguaje de programación de alto nivel (como C, C++, C#, Java, Rust, Ada, Swift) y lo traduce a un lenguaje que la computadora puede entender directamente. Después de que el código se ha traducido, la computadora puede ejecutar el programa. Para hacer esto, debemos compilar el código antes de poder ejecutarlo.

Por otro lado, un lenguaje de programación interpretado (como Python, Ruby, JavaScript, Perl, PHP) no requiere una etapa de compilación previa. En lugar de esto, cada línea de código se traduce y ejecuta al mismo tiempo. En este caso, no es necesario compilar el código antes de ejecutarlo.

Por lo que vemos que la diferencia principal es que, en un lenguaje de programación compilado, el código debe compilarse antes de poder ser ejecutado, mientras que, en un lenguaje de programación interpretado, el código se traduce y ejecuta directamente sin necesidad de una etapa de compilación previa.



## Entonces, ¿qué ventajas nos da Python como lenguaje interpretado?

- **Mayor rapidez de desarrollo:** podemos realizar pruebas y cambios en el código de forma rápida sin tener que esperar la compilación.
- **Menor complejidad:** por su sintaxis sencilla, simple y fácil de aprender.
- **Mayor portabilidad:** el código puede ser ejecutado en distintas plataformas (multiplataforma).
- **Flexibilidad:** puede ser utilizado en una gran variedad de aplicaciones, desde el desarrollo web hasta la ciencia de datos, siendo así ideal para proyectos en distintos ámbitos y con diferentes requisitos.

**Multiplataforma** - Puede ser interpretado y ejecutado en Windows, Linux, MacOS, etc. Es decir, soporta varios sistemas operativos sin necesidad de recompilar para cada plataforma.

**Multiparadigma** - Los paradigmas de programación son formas de pensar y estructurar el código para resolver problemas y Python admite paradigmas de:

- Programación funcional
- Programación estructurada
- Programación orientada a objetos (POO)

**Tipado dinámico** - Con esto podemos hacer que una variable pueda tener diferentes valores y de distintos tipos durante la ejecución del programa.

**Fuertemente tipado** - Esto quiere decir que, dado un tipo de valor a una variable, por ejemplo, un número entero, este debe utilizarse como un número entero, no se lo podría utilizar como una letra, a menos que se haga una conversión.

No te preocupes si aún no comprendés muy bien todo. Esta es solo una pequeña introducción para darte un contexto de Python y algunos conceptos fundamentales, que más adelante, los vamos a volver a nombrar con ejemplos y, ahí sí, los vas a ¡comenzar a comprender mucho mejor!

Ahora es momento de comenzar a incorporar nuevos conceptos, y el primero que vamos a aprender es: **VARIABLE**.



>>> Variable

# Conociendo lo que es una variable

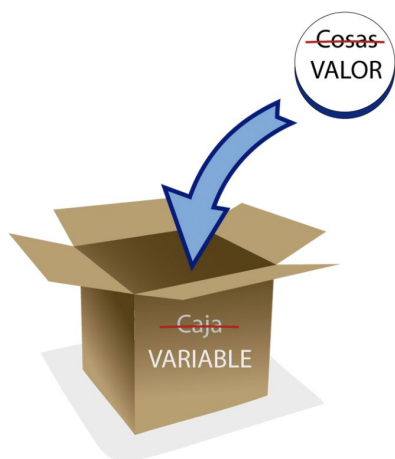
## SEMANA 2

### Primeros pasos

En palabras sencillas, una variable en programación, es una unidad de almacenamiento y recuperación de datos con valores que pueden cambiar durante la ejecución del programa. Esta se identifica con un nombre único en el código del programa, acorde al tipo de valor que almacenaremos.

Vamos a darte un ejemplo bien práctico para que comprendas mejor el concepto de variable.

Supongamos que nuestra variable es una caja y las cosas que vamos a poner adentro de esta caja, van a ser el valor.



Entonces, mi caja a la cual le di un nombre, en este caso como ves, se llama *variable*, y las cosas que voy a guardar y sacar de allí adentro van a ser valores, pero para simplificar vamos a decir, valor.

Con esto comprendido, vamos a trasladarlo a código y para esto es momento de tener abierto nuestro VSC.

Lo primero es crear un nuevo archivo, y para fines prácticos, vamos a poner el nombre del archivo como "01-variables.py" y en la primer línea de código trasladamos este ejemplo:

```
01-variables.py
01-variables.py > ...
1 variable = 'valor'
```

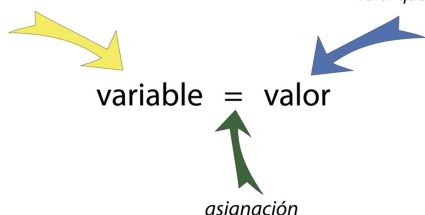
Y como verás, para asignar un valor a una variable, utilizamos el signo de igual (=), que aquí lo llamaremos asignación..

Además verás que la estructura consta de 3 partes:

nombre de la variable, signo de asignación y valor.

*elegimos un nombre para la variable con el que hagamos referencia a lo que vamos a almacenar*

*valor que va a tener la variable*



Algo más que hay que comprender para poder asignar un valor a una variable, es que hay varios tipos de datos que puede almacenar una variable. Algunos de los más importantes son:

```
int      #Números enteros
float    #Números decimales

str      #Strings o cadena de caracteres
bool     #Booleanos
```

Entonces, siguiendo con nuestro ejemplo, ahora vamos a hacerlo más práctico y vamos a imaginarnos que estamos de mudanza, y, porque somos muy organizados, tenemos cajas etiquetadas, donde cada caja contiene solo las cosas que dice el nombre de la caja, por ejemplo 'ropa de invierno'.



Y si esto lo trasladamos a código:

```
01-variables.py
01-variables.py > ...
1  variable = 'valor'
2
3  caja1 = 'ropa de invierno'
```

¿Lo vas comprendiendo mejor? ¿Y si ahora cargamos más cajas? ¡Pero en código!

```
1  caja1 = 'ropa de invierno'
2  caja2 = 'ropa de verano'
3  caja3 = 'elementos de la cocina'
```

Como se puede ver, tenemos 3 cajas o en este caso, 3 variables, y cada variable tiene un tipo de elemento o dato, que en este caso, es una cadena de caracteres. Si te alcanzas a dar cuenta, cada valor tiene comillas simples ( ' ' ) y por más que el valor sea distinto en cada variable, el tipo de dato sigue siendo el mismo: **STRING** o como se declara en Python **str**.

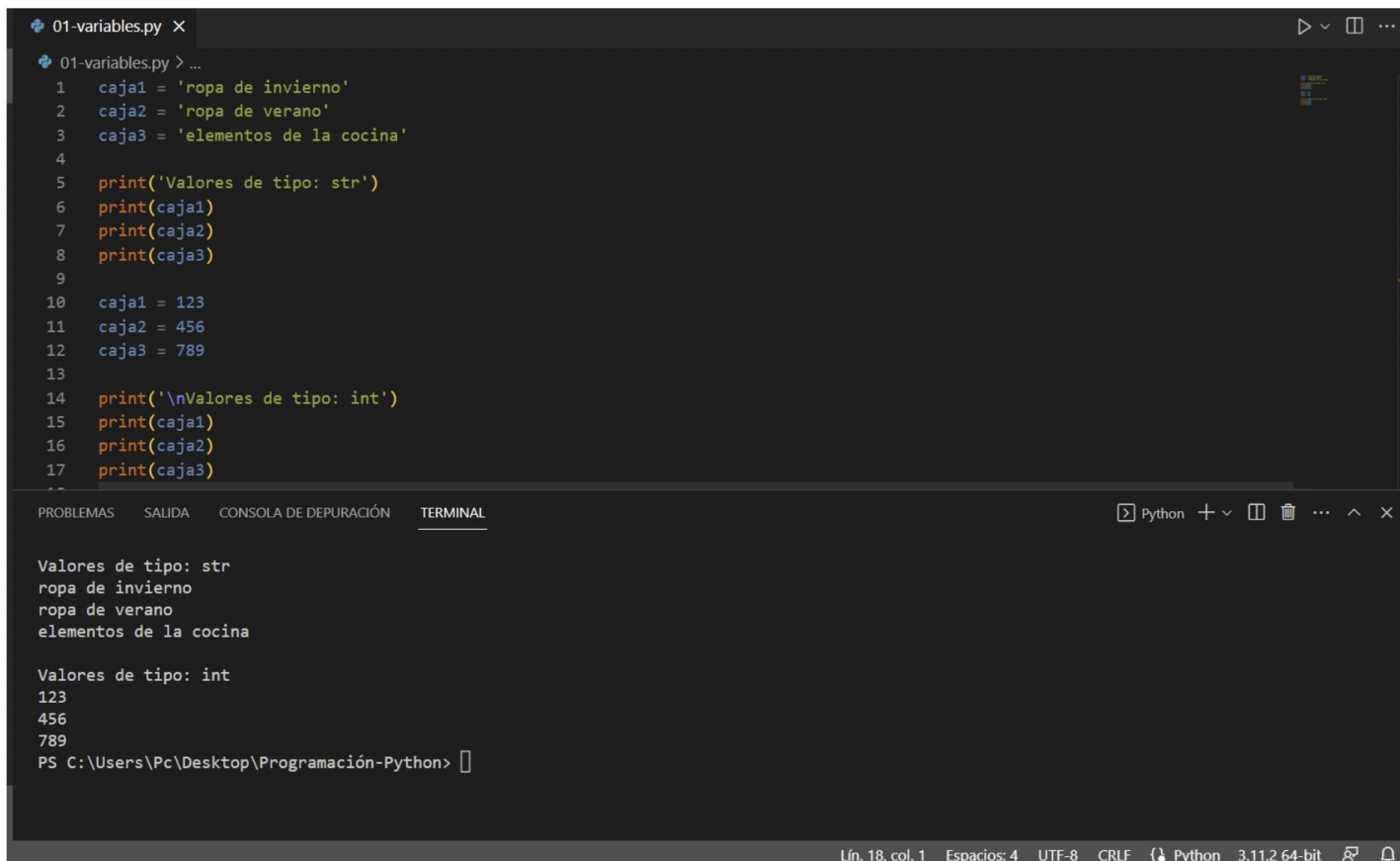
¿Pero qué pasa si ahora guardamos números enteros?

¡Esa respuesta fácil! Al ser Python de tipado dinámico, el valor de nuestra variable puede cambiar en tiempo de ejecución, y esto quiere decir, que solo debemos asignar un nuevo valor a la variable.

```
caja1 = 123
caja2 = 456
caja3 = 789
```

Como puedes ver, la función `print()` (imprimir en inglés) sirve para mostrar en pantalla el valor que tienen las variables.

Para abrir la consola en la parte inferior de VSC, debemos presionar la tecla `ctrl + ñ` o desde la parte superior, en “Ver” - >



```
01-variables.py x
01-variables.py > ...
1  caja1 = 'ropa de invierno'
2  caja2 = 'ropa de verano'
3  caja3 = 'elementos de la cocina'
4
5  print('Valores de tipo: str')
6  print(caja1)
7  print(caja2)
8  print(caja3)
9
10 caja1 = 123
11 caja2 = 456
12 caja3 = 789
13
14 print('\nValores de tipo: int')
15 print(caja1)
16 print(caja2)
17 print(caja3)
18
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
Python + - [ ] [x] ... ^ x

Valores de tipo: str
ropa de invierno
ropa de verano
elementos de la cocina

Valores de tipo: int
123
456
789
PS C:\Users\Pc\Desktop\Programación-Python>

Lín. 18, col. 1  Espacios: 4  UTF-8  CRLF  Python  3.11.2 64-bit
```

Lo podemos ir viendo en la parte de abajo, que es la consola, donde podemos ir ejecutando nuestro código (enseguida te mostramos cómo abrir la consola).

También, seguro que observaste que agregamos una pequeña oración para visualizar antes de mostrar los valores de las variables, esto solo fue para que quede más ordenado y puedas comprender mejor cómo podemos ir mostrando los valores de las variables.

“Terminal”. Ahí aparecerá una terminal (que es la que estamos viendo) en la parte inferior, pero para que se pueda ejecutar cómodamente los comandos de Python haremos click en la flechita hacia abajo al lado del + donde dice “powershell” y elegiremos “Command Prompt”.

```

01-variables.py > ...
1  caja1 = 'ropa de invierno'
2  caja2 = 'ropa de verano'
3  caja3 = 'elementos de la cocina'
4
5  print('Valores de tipo: str')
6  print(caja1)
7  print(caja2)
8  print(caja3)
9
10 caja1 = 123
11 caja2 = 456
12 caja3 = 789
13
14 print('\nValores de tipo: int')
15 print(caja1)
16 print(caja2)
17 print(caja3)

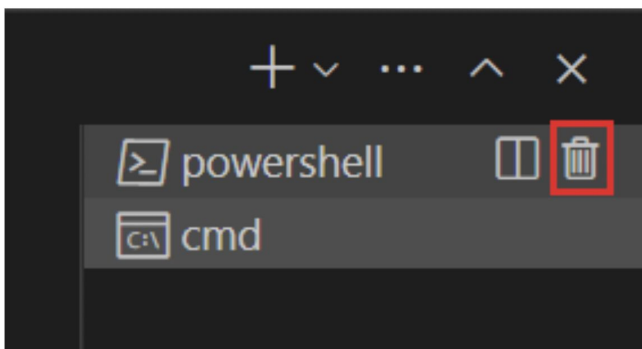
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\Pc\Desktop\Programación-Python>

PowerShell  
Git Bash  
**Command Prompt**

Se abrirá de la siguiente forma, por lo que solo acercando el cursor a hasta “powershell” verás que se habilita la opción de un tacho de basura, el cual si lo presionas, quedará solo el “cmd”.



Habrás notado que luego de la función `print()` van paréntesis de apertura y cierre, y dentro de los paréntesis, el nombre de la variable que queremos mostrar su valor o, el valor que queremos mostrar directamente, como la oración que pusimos antes de mostrar el valor de la variable.

Este valor que mostramos directamente, es una cadena de caracteres o string, y como todo string, va entre comillas (recuerda que pueden ser comillas simples o dobles, pero también recuerda que por convención usamos comillas simples).

Volviendo con el concepto de tipado dinámico. En la primeras líneas de código de nuestro programa, las variables tenían valores de tipo `str`, luego (en las siguientes líneas de código), cambiamos esos valores por números enteros o, mejor dicho, valores de tipo `integer`, o como se escribe en Python `int`.

<pre> 1  caja1 = 'ropa de invierno' 2  caja2 = 'ropa de verano' 3  caja3 = 'elementos de la cocina' </pre>	→	<pre> caja1 = 123 caja2 = 456 caja3 = 789 </pre>
--	---	--

Para estar seguros de que tipo de dato contiene nuestra variable, podemos usar otra función con la cual podemos identificar esto. Esta función es: `type()`



Y para ver los tipos de datos más comunes que vamos a utilizar en Python, vamos a crear nuevas variables con nuevos valores.

```
01-variables.py > ...
20 variable1 = 'Esto es un contenido de tipo str'
21 print(type(variable1))
22
23 variable2 = 123456
24 print(type(variable2))
25
26 variable3 = 4.351
27 print(type(variable3))
28
29 variable4 = True
30 print(type(variable4))
31
32 variable5 = False
33 print(type(variable5))
```

PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'bool'>
```

Analicemos la imagen anterior:

\* En variable1 (recordá que las variables pueden llevar cualquier nombre, pero siempre tenemos que intentar que éstos sean significativos con el valor que tenemos dentro de nuestra variable), tenemos un valor de tipo str y lo podemos ver en la consola cuando ejecutamos mediante la función type().

```
variable1 = 'Esto es un contenido de tipo str'
print(type(variable1))
```

```
<class 'str'>
```

\* En variable2 tenemos un valor de tipo int y lo vemos en la consola.

```
variable2 = 123456
print(type(variable2))
```

```
<class 'int'>
```

\* En variable3 tenemos un valor de tipo float y lo vemos en la consola.

```
variable3 = 4.351
print(type(variable3))
```

```
<class 'float'>
```

\* En variable4 y variable5 tenemos un valor de tipo bool. Los valores que puede tomar un tipo bool van a ser verdadero o falso. En Python se debe escribir en inglés y con la primera en mayúscula: True o False

```
variable4 = True
print(type(variable4))

variable5 = False
print(type(variable5))
```

```
<class 'bool'>
<class 'bool'>
```

Ahora hablaremos de algunas reglas generales.

1) Para definir nombres de variables hay que tener en cuenta las siguientes cinco simples reglas:

a) El nombre de una variable puede empezar con una letra o guión bajo, y, aunque es permitido usar letras mayúsculas, por convención no lo hacemos.

b) Pueden contener letras, números y se puede usar el guión bajo (`_`) .

c) Python distingue mayúsculas de minúsculas, por lo que hay que tener muy en cuenta esto a la hora de escribir los nombre. El que Python pueda distinguir entre mayúsculas y minúsculas, significa que es un lenguaje case sensitive.

d) No se pueden utilizar palabras clave. Por ejemplo, no se puede poner como nombre a una variable `"elif"`, ya que esta palabra reservada se usa en condicionales.

A continuación te mostramos una lista con algunos ejemplos de palabras reservadas, las cuales no se pueden usar como nombres de variables.

```
False      break
None       class
True       continue
__peg_parser__
and        del
as         elif
assert    else
async     except
await     finally
```

```
for        not
from       or
global     pass
if         raise
import     return
in         try
is         while
lambda     with
nonlocal   yield
```

2) Para modificar una variable, basta con asignarle un nuevo valor en cualquier momento y lugar en el programa, luego de definirla.

3) A una variable se le puede asignar un valor literal (de tipo `str`, `int`, `float`, `bool`), o se le puede asignar una expresión, o una llamada a una función o, la combinación de todos ellos (más adelante se verán ejemplos).

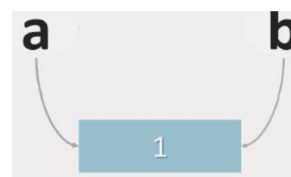
4) Se puede asignar un mismo valor a múltiples variables a la vez.

Por ejemplo: `a = b = c = 1`

5) En Python todo es un objeto. Entonces, si le asigno a la *variable* `"a"` el *valor* `"1"`, realmente la *variable* `"a"` hace referencia al objeto que representa al número entero con valor `"1"`.

Si creáramos una nueva *variable* `"b"` y le asignamos también el *valor* `"1"`, la *variable* `"b"` estará haciendo referencia al mismo objeto que la *variable* `"a"`.

En otros lenguajes de programación existen las variables y constantes, pero en Python no. Aquí todos son objetos, pero de manera tradicional vamos a referirnos como variables a estos objetos (como lo venimos haciendo hasta ahora).



En la siguiente página hablaremos de operadores.

>>> **Conceptos**

# Uso de operadores

## SEMANA 1

```

        .on("click", function() {
            if (a === b) {
                // ...
            }
        });
    });

```

Los operadores nos permiten manipular datos, sean variables, constantes, otras expresiones, objetos, atributos del objeto, entre otros, de manera que podamos:

- Transformarlos.
- Usarlos para decisiones para controlar el flujo de ejecución de un programa.
- Formar valores para asignarlos a otros datos.

El tipo de dato involucrado en una expresión se relaciona con los operadores utilizados. Vamos a ver algunos de éstos operadores:

### OPERADORES MATEMÁTICOS

SUMA	RESTA
$2 + 2$ Resultado: 4	$50 - 10$ Resultado: 40
DIVISIÓN	MULTIPLICACIÓN
$25 / 3$ Resultado: 8.333 Siempre retorna un punto flotante	$25 * 2$ Resultado: 50
$25 // 5$ Resultado: 5 Siempre retorna un número entero	
MÓDULO	POTENCIA
$25 \% 3$ Resultado: 1 Retorna el resto de la división	$8 ** 2$ Resultado: 64

### OPERADORES DE COMPARACIÓN

> MAYOR QUE	>= MAYOR O IGUAL QUE
$a > b$ Resultado: True si el operando de la izquierda es estrictamente mayor que el de la derecha: False en caso contrario.	$a >= b$ Resultado: True si el operando de la izquierda es mayor o igual que el de la derecha: False en caso contrario.
< MENOR QUE	<= MENOR O IGUAL QUE
$a < b$ Resultado: True si el operando de la izquierda es estrictamente menor que el de la derecha: False en caso contrario.	$a <= b$ Resultado: True si el operando de la izquierda es menor o igual que el de la derecha: False en caso contrario.
== IGUAL	!= DISTINTO
$a == b$ Resultado: True si el operando de la izquierda es igual al de la derecha: False en caso contrario.	$a != b$ Resultado: True si los operandos son distintos: False en caso contrario.

## OPERADORES LÓGICOS

AND	OR	
$a = \text{true}$ $b = \text{true}$ $x = a \text{ AND } b$ Resultado: <b>True</b> Devuelve True solo si ambos valores son True, en cualquier otro caso es False.	$a = \text{true}$ $b = \text{true}$ $x = a \text{ OR } b$ Resultado: <b>True</b>	$a = \text{true}$ $b = \text{false}$ $x = a \text{ OR } b$ Resultado: <b>True</b>
NOT	Cambia el valor de verdad de la variable a la que se aplica la operación.	
$a = \text{true}$ $x = \text{NOT } a$ Resultado: <b>False</b> cambia el valor de verdad de la variable a la que se aplica la operación.		

## OPERADORES LÓGICOS

CONCATENACIÓN	MULTIPLICACIÓN
$\text{'Hola'} + \text{' mundo'}$ Resultado: <b>Hola mundo</b> si tenemos dos strings o más entre comillas una al lado de la otra, se concatenan automáticamente.	$3 * \text{'hola'}$ Resultado: <b>HolaHolaHola</b>
MEZCLA	
Se pueden mezclar las operaciones de concatenación y multiplicación $3 * \text{'Hola'} + \text{' mundo'}$ Resultado: <b>HolaHolaHola mundo</b>	



&gt;&gt;&gt; Dato

# Tipos de datos

SEMANA 1



## DATO, CLASIFICACIÓN Y TIPOS

### Definición de dato

Es una representación simbólica (numéricas, alfanuméricas, algorítmicas, etc.) de un atributo o cualidad de una entidad. Los datos, aisladamente, pueden no contener información relevante. Una vez procesados los datos, bajo un enfoque o contexto, se puede apreciar información que puede resultar útil, ya sea para entender algo referido o para tomar decisiones en un contexto, o para la realización de cálculos.

Un dato puede ser una entrada leída desde un dispositivo de entrada como un teclado, o puede ser leída desde el almacenamiento de un disco, un número que se encuentra en memoria, etc.

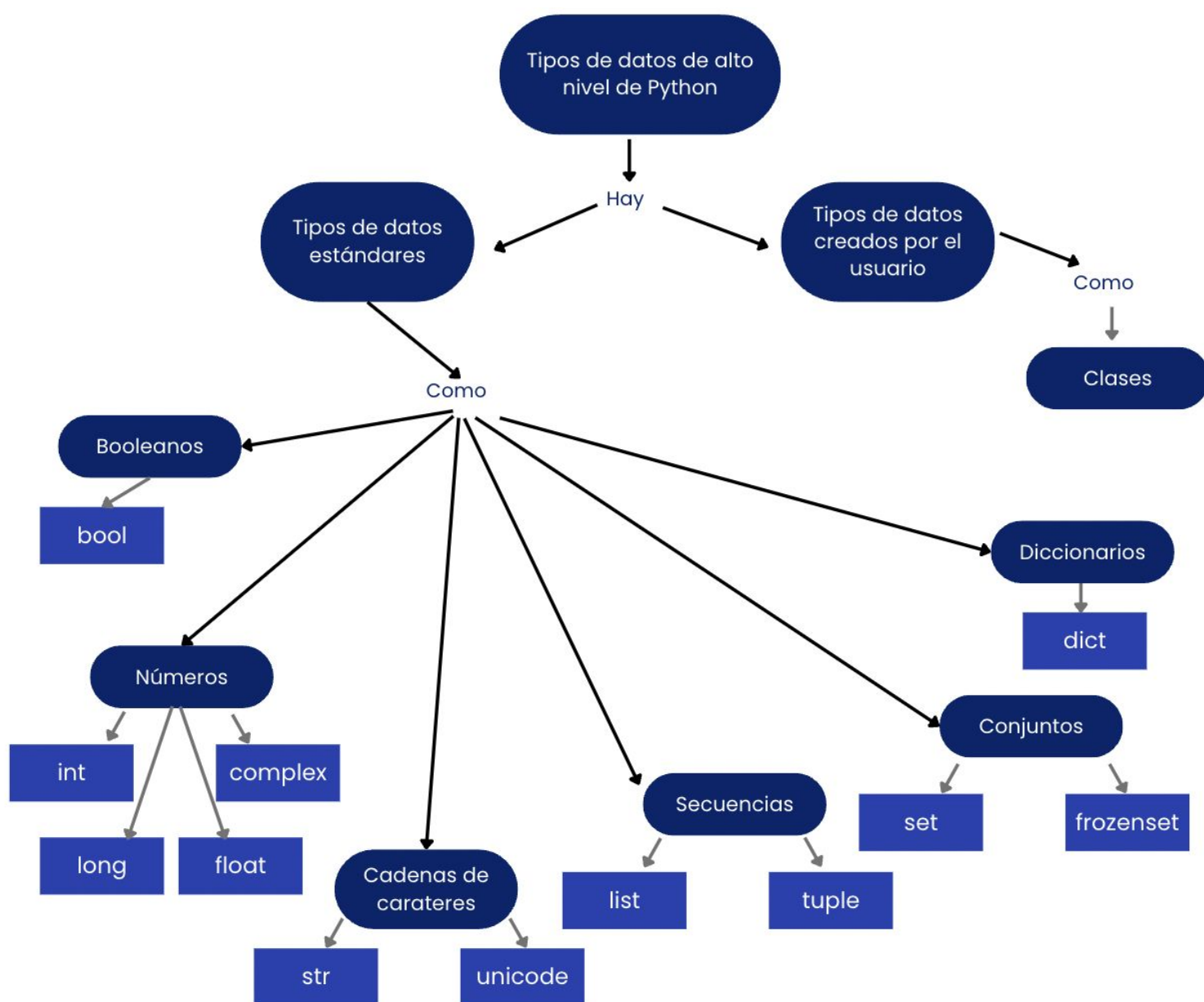
### Tipos de datos

El tipo de un dato está definido por el conjunto de valores que puede tomar a lo largo de un programa.

En la siguiente página encontrarás un esquema con los tipos de datos que maneja Python específicamente

En este curso no usaremos absolutamente todos, pero es bueno que sepas los tipos de datos que maneja este lenguaje por si quieres seguir desarrollandote en el amplio horizonte que brinda Python para todos los campos en los que se encuentra.





## NÚMEROS:

**Enteros:** son números positivos o negativos que no tienen decimales. Estos números se conocen como de tipo “int” (entero) o “long” (entero largo para más precisión).

Por ejemplo:  $x = 2$

**Reales:** son números de tipo decimal y en Python se conocen como de tipo “float”.

Por ejemplo:  $x = 2.5$

**Complejos:** son números que tienen una parte real y una parte imaginaria, en Python se conocen como de tipo “complex”.

Por ejemplo:  $x = 2,1 + 6j$

## CADENAS O STRINGS:

Se conocen como de tipo “str” y el texto va encerrado entre comillas (simples o dobles).

Las cadenas admiten operadores como la suma o la multiplicación.

Por ejemplo: `x = 'Hola mundo'` o `x = "Hola mundo"`

## BOOLEANOS O BOOLEAN:

Pueden tomar únicamente los valores de Verdadero o Falso. Se define usando el tipo “bool”.

Por ejemplo: `x = True` `z = False`

## CONJUNTOS O SET:

Es una colección de datos desordenada que no contiene elementos que se repiten. Se conoce como de tipo “set”.

Por ejemplo: `conjunto = {'naranja', 1, 'c', 2.5, True, 'manzana'}`

## LISTAS O LIST:

Contienen vectores (o como se los conoce en otros lenguajes: arrays), es decir, contienen un conjunto de valores que pueden contener distintos tipos de dato. Tienen índice. Se conocen como de tipo “list”.

Por ejemplo: `z = [0.5, 'bananas', 1500, 'b', True, False]`

## TUPLAS O TUPLES:

Es una lista que no se puede modificar después de su creación, es inmodificable o inmutable. Se puede anidar (o unir) una

tupla dentro de otra. Se conocen como de tipo “tuple”.

Por ejemplo: `numero = 1, 25, 3200, 'Hola mundo'`  
`anidada = numero, 'Nombre', 'Apellido', 20`

## DICCIONARIOS O DICT:

Es un tipo de dato similar a las listas, pero trabaja con clave y valor en vez de índices. Cada valor que está almacenado en un diccionario puede ser accedido usando la clave y obtener el valor en vez de usar un índice para referirse a un elemento (practicaremos sobre esto para que lo puedas entender mejor) Se conocen como de tipo “dict”.

Por ejemplo:

<code>agenda_telefonica = {</code>	<code>turnos = {</code>
<code>'Juan' : 3624123456,</code>	<code>1 : 'Alejandro',</code>
<code>'Ana' : 3624789012,</code>	<code>2 : 'Belén',</code>
<code>'María' : 3624345678</code>	<code>3 : 'Carlos'</code>
<code>}</code>	<code>}</code>

## NONE:

Python incorpora un quinto tipo de dato que, estrictamente hablando, se llama `NoneType` y cuyo único valor posible es `None`.

A menudo, `None` es utilizado cuando se quiere crear una variable (recuerda que es un objeto en realidad, porque en Python, todo es un objeto) pero aún no se le quiere asignar ningún valor en particular. Sin embargo, `None` es también un valor, pero en este caso, un valor nulo.

*En clases se hará el respectivo repaso y la ejercitación de todos estos conceptos y conocimientos adquiridos hasta esta parte del apunte. Sin embargo, en programación, la práctica constante y la búsqueda de nuevos conocimientos de forma autodidacta, hacen que toda la información que vamos incorporando, se establezca sólidamente en nuestra mente.*