

Python / Guía 2

Tipos de datos

primitivos

## 

## 

## OBJETIVOS DE LA GUÍA

En esta guía aprenderemos a:

* Definir variables en Python
* Reconocer los operadores: aritméticos, de asignación, de comparación, y lógicos
* Usar los tipos de datos numéricos, booleanos y de cadena
* Conocer algunas funciones propias de cada tipo de dato
* Convertir un tipo de dato a otro
* Buenas prácticas de programación en Python

## 



# Variables y números

## ¿Cómo crear variables en Python?

Se utiliza un operador de asignación para asignar valores a una variable. Usaremos el operador = para, en este caso, asignar un número entero a una variable. Prueba el siguiente ejemplo en el intérprete de Python, presionando la tecla 'enter' después de cada instrucción. No es necesario que al final de línea uses el punto y coma, como sucede en otros lenguajes de programación:

a = 10

b = 12

a + b

c = a \* b

c

| La comunidad de Python recomienda (para mayor legibilidad del código) separar las variables y el operador de asignación con un espacio, asimismo entre el operador de asignación y el valor, como se ve en el ejemplo de arriba. |
| --- |

**Consideraciones para los nombres de las variables:**

* Pueden contener caracteres del alfabeto inglés y el guión bajo.
* También puede contener caracteres con tildes, diéresis y la letra eñe, y otros pocos caracteres, pero se recomienda usar el alfabeto inglés y el guión bajo.
* No pueden contener espacios.
* Python es sensible a las minúsculas y mayúsculas, por lo que debes tener cuidado.

| La comunidad de Python recomienda (para mayor legibilidad del código) nombrar a las variables solamente con letras en minúsculas, y separar las palabras con un guión bajo (estilo snakecase). También aconseja declarar las variables constantes con mayúsculas: Ejemplos: un\_numero = 3 y NUMERO\_FAVORITO = 10. |
| --- |

**¡MANOS A LA OBRA!**

**Ejercicio 1**

Ejecuta el siguiente código en el intérprete de Python, e intenta explicarlo:

variable\_a, variable\_b = 1, 100

variable\_a

variable\_b

## ¿Cuáles son los tipos de datos para manejar números?

Python trabaja con tres tipos numéricos. En Python, el valor de un número no tiene límite: solo depende de la memoria disponible.

* **Enteros:** Representan todos los números enteros: positivos, negativos y cero. La sintaxis en Python es int, que viene del inglés integer. Ejemplo: 1000
* **Reales:** Representan los números reales. Tienen una parte entera y otra decimal. La sintaxis es float. Ejemplo: 1000.50
* **Complejos:** Representan los números complejos. Tienen una parte real y otra imaginaria. La sintaxis es complex, y se escribe situando la letra j al lado del número. Ejemplo: 5j

## ¿Alguna función que ayude a saber el tipo de dato?

La función type nos permite conocer qué tipo de datos es un valor. Por ejemplo, probemos en el intérprete:

type(100)

Python devuelve <class 'int'>

type(100.50)

Python devuelve <class 'float'>

type(5j)

Python devuelve <class 'complex'>

Esto significa que para Python todo es un objeto, incluso los números, y nos está respondiendo diciendo: tal objeto pertenece a la clase de los números enteros, de los números reales o de los números complejos.

## Veamos los operadores aritméticos

Un operador aritmético toma dos operandos como entrada, realiza un cálculo y devuelve el resultado. Todos los tipos numéricos (menos los complejos) soportan las siguientes operaciones:

| **Operador** | **Cálculo** | **Resultado** |
| --- | --- | --- |
| +x | valor de x, sin cambiar | +10 (será igual a 10) |
| -x | valor de x, negado | -10 (será igual a -10) |
| x + y | suma de x e y | 10 + 3 (será igual a 13) |
| x - y | resta de x e y | 10 - 3 (igual a 7) |
| x \* y | multiplicación de x por y | 10 \* 3 (igual a 30) |
| x \*\* y | x elevado a y | 10 \*\* 3 (igual a 1000) |
| x / y | división de x por y | 10 / 3 (igual a 3.333) |
| x // y | División entera de x por y (El resultado es un número entero en el sentido matemático) | 10 // 3 (igual a 3) |
| x % y | Resto o residuo de x dividido y | 10 % 3 (igual a 1) |

**¡MANOS A LA OBRA!**

**Ejercicio 2**

Ejecuta el siguiente código en el intérprete de Python:

abs(-10)

pow(10, 3)

divmod(10, 3)

capital = 10\_000\_000

balance = capital / 3

round(balance, 2)

| Prueba cambiar de lugar el guión bajo dentro del valor de un número. |
| --- |

## Ahora, los operadores de asignación

| **Operador** | **Significado** |
| --- | --- |
| = | a = 5 El valor 5 es asignado a la variable a |
| += | a += 5 es equivalente a: a = a + 5 |
| -= | a -= 5 es equivalente a: a = a - 5 |
| \*= | a \*= 3 es equivalente a: a = a \* 3 |
| /= | a /= 3 es equivalente a: a = a / 3 |
| %= | a %= 3 es equivalente a: a = a % 3 |
| \*\*= | a \*\*= 3 es equivalente a: a = a \*\* 3 |
| //= | a //= 3 es equivalente a: a = a // 3 |

**¡MANOS A LA OBRA!**

**Ejercicio 3**

Ejecuta el siguiente código en el intérprete de Python, e intenta explicar los resultados.

x = 2

x += 1

x

x += 1

x

x \*= 2

x

x /= 4

x

## ¿Algunas funciones matemáticas?

| **Funciones** | **Significado** |
| --- | --- |
| divmod(x, y) | El par de valores (x // y, x % y) |
| pow(x, y) | x elevado a y (Python define pow(0, 0) y 0 \*\* 0 igual a 1) |
| abs(x) | Valor absoluto de la magnitud de x |
| round(x[, n]) | El valor de x redondeado a n dígitos, redondeando la mitad al número par más cercano. Si no se especifica n, se asume 0. |

| Puedes ver más funciones matemáticas que vienen incorporadas en Python en la documentación oficial: <https://docs.python.org/es/3/library/math.html.> |
| --- |

## Revisemos lo aprendido hasta aquí

* Definir variables (con el estilo recomendado por la comunidad)
* Identificar tipos de datos int, float, complex y bool
* Usar funciones matemáticas integradas en Python

# Verdadero o falso

## ¿Existe un tipo booleano en Python?

Verdaderamente sí. La sintaxis es True y False. Tener en cuenta mayúsculas y minúsculas. Matemáticamente, corresponde al 1 y al 0 respectivamente. Se representa, en Python, con la palabra bool. Probemos en el intérprete: type(True)

## Veamos los operadores de comparación

| **Operador** | **Significado** | **Resultado** |
| --- | --- | --- |
| < | es menor que | 5 < 10 Python: True |
| <= | es menor que o igual a | 5 < 10 Python: True |
| > | es mayor que | 5 > 10 Python: False |
| >= | es mayor que o igual a | 10 >= 10 Python: False |
| == | igual a | 5 == 5 Python: False |
| != | no es igual a | 5 != 10 Python: True |
| is | identidad de objeto | ver ejercicio de abajo |
| is not | negación de identidad de objeto | ver ejercicio de abajo |

## Ahora los operadores lógicos

| **Operador** | **Significado** |
| --- | --- |
| and | Devuelve True si ambos operandos son True |
| or | Devuelve True si alguno de los operandos es True |
| not | Devuelve True si alguno de los operandos es False |

**¡MANOS A LA OBRA!**

**Ejercicio 5**

Ejecuta el siguiente código en el intérprete de Python e intenta explicar los resultados.

not False

10 > 1 or 10 > 100

10 > 1 and 10 > 100

a = 10 > 1

b = 5 == 5

a and b and True

Ejercicio 6

Ejecuta el siguiente código en el intérprete de Python e intenta explicar los resultados.

True \* 5 == 2.5 \* 2

numero\_1 = 10

numero\_2 = 10.0

numero\_1 == numero\_2

numero\_1 is numero\_2

type(numero\_1) is not int

True != 1.0

type(True) is bool

**Ejercicio 7**

Ejecuta el siguiente código en el intérprete de Python e intenta explicar los resultados.

True == 1 and True

True > False or True

True > (False or True)

prueba = (not 3 == 5, 33 // 3 == 11 and 5 > 2, True or False)

prueba

## Revisemos lo aprendido hasta aquí

* Reconocer los operadores de comparación
* Reconocer los operadores lógicos

# Caracteres

## ¿Cuál es el tipo de datos para manejar caracteres?

El tipo de datos se llama string, y su sintaxis es str. Representa cadenas de caracteres.

| Las cadenas de caracteres en Python están codificadas en Unicode. |
| --- |

Para crear cadenas tenemos que encerrar caracteres dentro de comillas simples 'texto', comillas dobles "texto", o comillas triples: """texto""" o '''texto'''.

'texto'   
"texto"  
'''texto'''  
"""texto"""

También podemos incluir comillas dentro de una cadena de la siguiente forma:

'"texto"'  
"'texto'"  
"""'texto'"""  
'''"texto"'''

O, si decidimos siempre usar comillas dobles, tenemos la opción de usar secuencias de escape, utilizando la barra invertida:

"este es un \"ejemplo\""

También podemos crear cadenas envolviéndolas en paréntesis:

("esta es una cadena")

Para hacer una cadena en varias líneas podemos optar por los paréntesis:

("una parte") # presiona 'enter'  
(" y otra parte") # Python devuelve: 'una parte y otra parte'

O usar las cadenas triples:

"""una parte # presiona 'enter'  
y otra parte""" # Python devuelve: 'una parte\n y otra parte'

Como vemos, Python nos muestra una secuencia de escape \n que indica que hay un salto de línea.

Finalmente, probemos type con una cadena:

type("123") # Python devuelve: <class 'str'>: es una cadena

## ¿Algunas funciones de cadena?

Todos los objetos que son clase cadena, heredan funciones que nos facilitan su manejo, es decir, todas las cadenas comparten las mismas funciones. Veamos algunas:

**¡MANOS A LA OBRA!**

**Ejercicio 8**

Ejecuta el siguiente código y explica lo que Python hace:

a = "esTa es una Prueba "

a.lower()

a.upper()

a.capitalize()

a.title()

a.count("a")

a.replace("a", "i")

a.strip()

a.index("T")

## Índices y slicing

Cada carácter de una cadena tiene una posición que se llama índice. En Python los índices comienzan desde cero.

"cadena"  
 012345

"cadena".index("a") # Python devolverá: 1

Podemos recorrer una cadena utilizando una técnica llamada slicing. Consiste en invocar la cadena seguida de corchetes, donde se puede colocar hasta tres números enteros separados por coma. La lógica del slicing es la siguiente:

nombre\_de\_variable[ desde : hasta : paso ]

**¡MANOS A LA OBRA!**

**Ejercicio 9**

Ejecuta el siguiente código y explica lo que Python hace:

cadena = "El 1 es mayor a -1"

cadena[0]

cadena[3]

cadena[0:2]

cadena[:2]

cadena[3:]

cadena[::2]

cadena[::-1]

## 

## Operaciones con cadenas

Algunos operadores funcionan con las cadenas:

**¡MANOS A LA OBRA!**

**Ejercicio 10**

Ejecuta el siguiente código y explica lo que Python hace:

texto\_1 = "hola"

texto\_2 = "adiós"

texto\_1 + texto\_2

a = "Había"

a += " una vez"

a

texto\_1 \* 3

"A" == "A"

"A" == "a"

## Función len

La función len devuelve un entero que me indica la cantidad de elementos que tiene un objeto. Los tipos de datos numéricos no son soportados por len, pero sí puedo medir cadenas:

**¡MANOS A LA OBRA!**

**Ejercicio 11**

Ejecuta el siguiente código y explica lo que Python hace:

len("")

len("mido")

len("1234")

len(1234)

## Función print

La función print devuelve una cadena en la terminal o consola. Puede traer el valor de una variable, puede mostrar cálculos, o dar formato a una cadena.

Puedes concatenar cadenas usando el operador + o puedes separar cadenas de números, cálculos y variables con la coma.

Si utilizas la letra efe 'f' delante de las comillas de una cadena, le estarás indicando a Python que dentro puedes utilizar llaves para traer variables o hacer cálculos. Este método se llama f-string.

Además, podemos crear saltos de línea con la secuencia de escape \n:

**¡MANOS A LA OBRA!**

**Ejercicio 12**

Ejecuta el siguiente código. ¿Qué forma prefieres usar y por qué?:

nombre = "María"

edad = "16"

print(nombre, edad)

print("Mi nombre es", nombre, "y tengo", edad, "años")

print("Mi nombre es " + nombre + " y tengo " + edad + " años")

print(f"Mi nombre es {nombre} y tengo {edad} años")

print(f"{nombre=}\n{edad=}")

## Función input

La función input devuelve una cadena. Obtiene caracteres escritos por teclado y por consola o terminal. La función se detiene esperando a que se pulse la tecla 'enter' para continuar.

**¡MANOS A LA OBRA!**

**Ejercicio 13**

Ejecuta el siguiente código.

nombre = input("Escribe tu nombre: ")

print(f"Hola {nombre.upper()}")

## Revisemos lo aprendido hasta aquí

* Crear cadenas y manejarlas con funciones
* Recorrer las cadenas con el método slicing
* Usar las funciones integradas len, print e input

# Conversión de tipos de datos

Python es un lenguaje de tipado dinámico, esto significa que no es necesario declarar qué tipo de datos es una variable, como sucede en otros lenguajes de programación. El siguiente código no dará ningún error, cosa qué sí puede suceder en otro lenguaje de programación de tipado estático:

a = 23  
a = "cadena"  
a = True

Esto permite que Python nos ofrezca cambiar un tipo de datos por otro:

**¡MANOS A LA OBRA!**

**Ejercicio 14**

Ejecuta el siguiente código.

int(100.50)  
 float(100)  
 str(2000)  
 str(2000) \* 10  
 a, b, c = int(10.0), float(100\_000), float("10.343")  
 a  
 b  
 c

saldo = float(input("Saldo: "))  
 print(f"Tu saldo es ${saldo:.2f}")  
 type(saldo) is str  
 type(saldo) is float  
 nacimiento = input("Año en que naciste: ")  
 nacimiento = int(nacimiento)  
 print(f"En el año 2050 tendrás {2050 - nacimiento} años")