Cápitulo 2. Variables, expresiones y sentencias

2. 1 EXPRESIONES

COSNTANTES

- **Los valores fijos, como números, letras y cadenas, se denominan "constantes" porque su valor no cambia.**
- Las constantes numéricas son como se espera.
- Las constantes de cadena se escriben entre comillas simples (') o dobles (").

PALABRAS RESERVADAS

NO SE PUEDEN USAR, SOLO ES PARA EL LENGUAJE PYTHON No se usan para :

- Nombrar variables
- Nombrar funciones
- etc..

VARIABLES

- **Una variable es un lugar con nombre en la memoria donde un programador puede almacenar datos y luego recuperar los datos usando el "nombre" de la variable.**
- Los programadores pueden elegir los nombres de las variables.
- Puedes cambiar el contenido de una variable en una instrucción posterior.

REGLAS PARA LOS NOMBRES VARIABLES

- Debe comenzar con una letra o un guion bajo _
- Debe consistir en letras, números y guiones bajos
- Sensible a mayúsculas y minúsculas
- **Uso de camel_case**. es decir si se usan dos palabras empezaran en minuscula y separadas por u guion bajo.
- Poner nombres a las variables con significado, nos hara la vida mas facil.

Sentencias o lineas

Expersiones numéricas

Orden delas expersiones numéricas

1. Parentesis
2. Potencia
3. División, Multiplicación, módulo y división entera
4. Suma, Resta
5. Y luego ejecutamos de izquierda a derecha

```
x = x = 1 + 2 ** 3 / 4 * 5
print(x)
```

Tipos de variables

Haz doble clic (o pulsa Intro) para editar

```
dd = 1 + 4
print(dd)
eee = 12
eee = 'Hola ' + 'Pepe'
print(eee)
```

```
vv = 'hola '
vv = vv + 'Mundo'
print(vv)
#vv = vv + 1
```

Funcion 'type' Nos dice de que tipo es la variables:

- Cadena de texto
- entero int
- decimal float
- boleano (bool) true/False

```
z = 8/4
type(z)
print(type(z))
print(z)
```

```
type(eee)
#type(x)
```

```
print(type(vv))
print(type(1))
print(type(3.5))
print(type(True))
type(vv)
```

Conversores de tipo de variable

1. Conversión de int a float (entero a decimal)

Para convertir un número entero (int) en un número decimal (float), simplemente puedes usar la función float().

```
# Ejemplo de conversión de int a float
x = 10  # entero
y = float(x)  # convierte 10 en 10.0 (float)
print(y)  # Salida: 10.0
```

2. Conversión de float a int (decimal a entero)

Para convertir un número decimal (float) a un número entero (int), puedes usar la función int(). Esta conversión trunca el número, es decir, elimina la parte decimal sin redondear.

```
# Ejemplo de conversión de float a int
x = 10.75  # decimal
y = int(x)  # convierte 10.75 en 10 (truncando)
print(y)  # Salida: 10
```

3. Conversión de str a int o float (cadena a entero o decimal)

Si tienes una cadena que representa un número, puedes convertirla a int o float usando int() o float().

```
# Ejemplo de conversión de str a int
x = "15"
y = int(x)  # convierte la cadena "15" en el número 15
print(y)  # Salida: 15

# Ejemplo de conversión de str a float
z = "15.5"
w = float(z)  # convierte la cadena "15.5" en el número 15.5
print(w)  # Salida: 15.5
```

4. Conversión de int o float a str (entero o decimal a cadena de texto)

Para convertir números en cadenas de texto, puedes usar la función str().

```
# Ejemplo de conversión de int a str
x = 100
y = str(x)  # convierte el número 100 en la cadena "100"
print(y)  # Salida: "100"

# Ejemplo de conversión de float a str
z = 99.99
w = str(z)  # convierte el número 99.99 en la cadena "99.99"
print(w)  # Salida: "99.99"
```

```
100
99.99
```

5. Conversión de bool a int o float

En Python, los valores booleanos también pueden ser convertidos a enteros o decimales. Los valores True y False se convierten de la siguiente manera:

True se convierte a 1

False se convierte a 0

```
# Ejemplo de conversión de bool a int
x = True
y = int(x)  # True se convierte en 1
print(y)  # Salida: 1

# Ejemplo de conversión de bool a float
z = False
w = float(z)  # False se convierte en 0.0
print(w)  # Salida: 0.0
```

6. Conversión de int a bool

La conversión de entero a booleano en Python sigue una regla sencilla:

Cualquier valor distinto de 0 se convierte en True

El valor 0 se convierte en False

```
# Ejemplo de conversión de int a bool
x = 5
y = bool(x)  # Cualquier número distinto de 0 es True
print(y)  # Salida: True

z = 0
w = bool(z)  # 0 se convierte en False
print(w)  # Salida: False
```

7. Conversión de str a bool

En Python, cualquier cadena no vacía se convierte en True, y una cadena vacía "" se convierte en False.

```
# Ejemplo de conversión de str a bool
x = "hello"
y = bool(x)  # Una cadena no vacía es True
print(y)  # Salida: True

z = ""
w = bool(z)  # Una cadena vacía es False
print(w)  # Salida: False
```

HAY TAMBIEN EN ENTRE COLECCIONES DICCIONARIOS ETC LO VEREMOS MAS TARDE

FUNCIÓN INPUT

Podemos indicar a Python que se detenga y lea datos del usuario usando la función input() La función input() devuelve una cadena

```
# name = input('Cual es tu nombre')
# print('Bienvenido', name)
```

```
# num = input("Dime un  número")
# print("La variable num es de tipo ", type(num))
```

Comentarios

- Pon comentarios por parrafo
- Que comentye lo que hace ese parrafo de codigo.

1. Comentario de una sola línea (comentarios simples)**

Para hacer un comentario de una sola línea, usas el símbolo #. Todo lo que esté después del # en esa línea se considera un comentario.

```
# Este es un comentario de una sola línea
x = 5  # Esta es una variable que almacena el valor 5
print(x)  # Imprime el valor de x
```

2. Docstrings (cadenas de documentación)**

Se usan tres comillas dobles """ o simples '''. No son exactamente comentarios, sino cadenas de texto que se usan para documentar funciones, clases o módulos.

```
def suma(a, b):
    """
    Esta función recibe dos números
    y devuelve su suma.
    """
    return a + b
```

```
#x = int(input("Dame un numero"))
#print(x)
```

```
# Programa: Nos indica el piso en el q estamos si es EEUU.
#convertimos la entrada del usuario a entero
#piso_res_mund = int(input("Dime en que piso estás"))
#print(type(piso_res_mund))
#piso_usa = piso_res_mund + 1
#print(f"El piso en el q estas si estubieras en EEUU es {piso_usa}")
```

Strings

Un **string** se corresponde con un conjunto de caracteres que forman una cadena de texto.

La sintaxis que debemos utilizar para definir strings en Python consiste en situar los caracteres entre " o '

```
var = "Esto es un string"
var2 = 'Esto es otro string'
```

Un string se corresponde con un conjunto de caracteres que forman una cadena de texto.

La sintaxis que debemos utilizar para definir strings en Python consiste en situar los caracteres entre " o '

```
var1 = "Los strings pueden definirse con el caracter ''"
```

```
var2 = 'Los strings pueden definirse con el caracter ""'
```

```
print(var1)
print(var2)
```

```
Los strings pueden definirse con el caracter ''
Los strings pueden definirse con el caracter ""
```

1. Indexación

En muchos tipos de datos en Python se puede **acceder a elementos individuales de un conjunto ordenado de datos directamente mediante un índice numérico o un valor clave. Este proceso se denomina indexación.**

En Python, las cadenas son secuencias ordenadas de caracteres, y por lo tanto pueden ser indexadas de esta manera. Se puede **acceder a los caracteres individuales** de una cadena especificando el nombre de la cadena seguido de un **número entre corchetes** `[]`.

El primer carácter de la cadena tiene el índice 0, el siguiente tiene el índice 1, y así sucesivamente. El índice del último carácter será la longitud de la cadena menos uno.

```
nombre = "Segundo"
nombre[2] # dara la letra g
print(nombre[2])
"----*----"[4]
```

```
g
'*'
```

```
"segundo"[2]
```

```
'g'
```

Tambien podemos utilizar números negativos para extraer caracteres por el final de la cadena de texto

```
nombre[-1] # deberia pintar la "o"
print(nombre[-1])
```

2 Slicing Extraer cadenas

Python también permite una sintaxis específica de indexación que extrae **subcadenas de una cadena de texto**, a esto se denomina **slicing**.

La sintaxis que se utiliza para extraer una subcadena de una cadena:

cadena[INICIO:FIN]

inicio: posición inicial (incluida).

fin: posición final (excluida). (la posicion es esa -1)

```
nombre = "Segundo"
nombre[1:3]
```

```
'eg'
```

```
nombre[-7:-1]
```

```
'Segund'
```

```
nombre[-9:]
```

```
'Segundo'
```

Cuando es negativo es igua m es el principio y n hasta el final. **OJO EJEMPLO**. Si queremos que muestre la ultima letra NO PONEMOS NADA.

```
nombre[-7:]
```

Si no indicamos uno de los números, lee hasta el final.

```
nombre[:7]
```

```
'Segundo'
```

```
nombre[4:]
```

```
'ndo'
```

3. Stride

El stride es otra variante más del slicing. **Si se añade un :** adicional y un tercer índice, se designa una stride, que indica **cuantos caracteres saltar hasta obtener el siguiente caracter.**

```
nombre1 = "Santiago Hernández"
```

```
nombre1[0:8:2]
```

```
'Snig'
```

```
nombre1[::-1]
```

```
'zednánreH ogaitnaS'
```

4. Modificación de strings

Un string es un tipo de dato que Python considera **inmutable**, esto quiere decir que **no podemos modificar** una parte de un string asociada a una variable

```
ur = "Pepe"
```

```
ur[1] = y
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[24], line 1
----> 1 ur[1] = y

TypeError: 'str' object does not support item assignment
```

Strings de múltiples líneas

En algunas ocasiones es posible que queramos definir un **string que tenga múltiples líneas**. Existen varias formas de definir esto en Python. La forma más sencilla es **introducir el caracter `\n`**en la posición de la cadena de texto donde queremos que se produzca el salto de línea.

```
var33 = "Pepe\nJuan\nDavid"
print(var33)
```

```
Pepe
Juan
David
```

Otra opción interesante es situar nuestra cadena de texto entre los caracteres """.

```
nombre222 = """Santiago
Hernandez
Ramos
"""
'
```

```
Empieza a programar o a crear código con IA.
```

```
Santiago
Hernandez
Ramos
```

```
nombre222 = ''' Santiago
Hernandez
Ramos'''
print(nombre222)
```

```
 Santiago
Hernandez
Ramos
```

Operadores Aritméticos

1. ¿Cuáles son los operadores aritméticos?

A continuación se presentan los operadores aritméticos soportados por Python 3

Operator	Example	Meaning
+ (unario)	+a	**Unario Positivo**
+ (binario)	a + b	**Suma**
- (unario)	-a	**Unario Negativo**
- (binario)	a - b	**Resta**
*	a * b	**Multiplicación**
/	a / b	**División**
%	a % b	**Módulo**
//	a // b	**División de enteros** (también denominado **Floor Division**)
**	a ** b	**Exponencial**

2. Operadores Unarios

Los operadores unarios se caracterizan porque se aplican sobre un único operando. En Python se soportan el operador **Unario Positivo** y **Unario Negativo**. Este tipo de operadores se aplican sobre tipos numéricos en Python.

```
num = 10
```

```
# Operador Unario Positivo
+num
```

```
# Operador Unario Negativo
-num
```

```
texto = "Hola mundo"
```

```
-texto
```

3. Suma y Resta

Los operadores **Suma** y **Resta** son operadores binarios que pueden aplicarse sobre distintos tipos de datos.

3.1. Tipos de datos Numéricos

```
num1 = 10
num2 = 5
```

```
num1 + num2
```

```
num1 - num2
```

```
num2 - num1
```

```
num3 = 1.5
num4 = 0.5
```

```
num3 + num4
```

```
num3 - num4
```

```
x = 1
y = 1.0
x + y
```

3.2. Strings

```
text1 = "Hola"
text2 = "mundo"
```

```
text1 + text2
```

```
text1 + " " + text2
```

```
text1 - text2
```

4. Multiplicación y División

Los operadores **Multiplicación** y **División** son operadores binarios que pueden aplicarse sobre distintos tipos de datos.

4.1. Tipos de datos numéricos

```
num1 = 10
num2 = 5
```

```
num1 * num2
```

```
# IMPORTANTE: El resultado de una division siempre es un float
num1 / num2
```

4.2. Strings

```
text1 = "Hola"
text2 = "mundo"
```

```
text1 * text2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 text1 * text2

TypeError: can't multiply sequence by non-int of type 'str'
```

```
text1 / text2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[37], line 1
----> 1 text1 / text2

TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

```
text1 * 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 text1 * 3.2

TypeError: can't multiply sequence by non-int of type 'float'
```

5. Módulo

El operador **Módulo** es un operador binario que devuelve el resto de una división entre tipos de datos numéricos.

```
num1 = 10
num2 = 7
```

```
```

```
num1 % num2
```

```
3.3
```

```
num2 % num1
```

```
7
```

```
num1 = 10.5
num2 = 7.2
```

```
num1 % num2
```

```
3.3
```

6. Exponencial

El operador **Exponencial** es un operador binario que se aplica sobre tipos de datos numéricos.

```
num1 = 10
```

```
num1 ** 2
```

```
100
```

```
num1 ** 3
```

```
1000
```

```
text1 = "Hola mundo"
```

```
text1 ** 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[45], line 1
----> 1 text1 ** 2

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

7. Floor Division

El operador **Floor Division** es un operador binario que se aplica sobre tipos de datos numéricos y **devuelve la parte entera del resultado. Sin redondear**

```
num1 = 10
num2 = 7
```

```
num1 / num2
```

```
num1 // num2
```

```
num1 = 13
num2 = 7
```

```
num1 / num2
```

```
num1 // num2
```

```
num1 = 10.3
num2 = 8.1
```

```
num1 // num2
```

```
1.0
```

Operadores de asignación

¿Cuáles son los operadores de asignación?

En secciones anteriores ya hemos presentado el operador de asignación que utilizabamos para asignar un valor a una variable, concretamente se corresponde con `=`.

Sin embargo, una de las cosas interesantes que nos proporciona Python es un mecanismo para combinar operadores aritméticos y operadores de asignación simplificando el código que escribimos. A la combinación de ambos operadores se les denomina ***Augmented Assignment***.

Augmented Assignment		Standard Assignment
`a += 5`	es equivalente a	`a = a + 5`
`a /= 10`	es equivalente a	`a = a / 10`
`a *= b`	es equivalente a	`a = a * b`

```
num = 10
```

```
num = num * 2
```

```
num
```

```
20
```

```
num = 10
```

```
num *= 2
```

```
num
```

```
20
```

Operadores de comparación

¿Cuáles son los operadores de comparación?

Los operadores de comparación evalúan la relación que existe entre dos valores en Python. Existen diferentes tipos de operadores de comparación.

Operador	Ejemplo	Significado
`==`	`a == b`	**Igual a**
`!=`	`a != b`	**No igual a**
`<`	`a < b`	**Menor que**
`<=`	`a <= b`	**Menor que o igual a**
`>`	`a > b`	**Mayor que**
`>=`	`a >= b`	**Mayor que o igual a**

1. Igualdad y desigualdad

Los operadores de comparación más utilizados son el de igualdad `==` y el de desigualdad `!=`. Estos operadores pueden aplicarse a varios tipos de datos en Python.

1.1. Tipos de datos numéricos

```
num1 = 2
num2 = 5
```

```
num1 == 2
```

```
True
```

```
num1 == num2
```

```
False
```

```
num1 == 2.0
```

```
True
```

```
num1 != 2
```

```
False
```

```
num1 != num2
```

```
True
```

1.2. Strings

```
text1 = "cadena de texto"
text2 = "cadena de texto 2"
```

```
text1 == "cadena de texto"
```

```
True
```

```
text1 == text2
```

```
False
```

```
text1 != text2
```

```
True
```

2. Otros operadores de comparación

2.1. Tipos de datos numéricos

```
num1 = 2
num2 = 5
```

```
num1 < num2
```

```
True
```

```
num2 < num1
```

```
False
```

```
num1 <= 2.0
```

```
True
```

2.2. Strings

```
text1 = "Cadena de texto"
```

```
text1 < "texto"
```

```
True
```

La comparación utiliza un orden lexicográfico: primero se comparan los dos primeros elementos, y si son diferentes, esto determina el resultado de la comparación; si son iguales, se comparan los dos siguientes elementos, y así sucesivamente, hasta que se agote cualquiera de las dos secuencias.

Esto quiere decir que la comparación se realiza utilizando los equivalentes numéricos (el resultado de la función por defecto `ord()`) de sus caracteres.

```
help(ord)
```

```
Help on built-in function ord in module builtins:

ord(character, /)
    Return the ordinal value of a character.

    If the argument is a one-character string, return the Unicode code
    point of that character.

    If the argument is a bytes or bytearray object of length 1, return its
    single byte value.
```

```
ord('a')
```

```
97
```

```
"Cadena" < "Texto"
```

```
True
```

```
ord('C')
```

```
67
```

```
ord('T')
```

```
84
```

```
help(chr)
```

```
Help on built-in function chr in module builtins:

chr(i, /)
    Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
```

```
chr(67)
```

```
'C'
```

```
chr(84)
```

```
'T'
```

```
help(print)
```

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

Operadores de identidad

¿Cuáles son los operadores de identidad?

Los operadores de identidad se utilizan para comparar objetos. Sin embargo, no comparan si los objetos son iguales, en su lugar, comparan si son el mismo objeto:

Operador	Ejemplo	Significado
`is`	`x is y`	`True` si las dos variables son el mismo objeto
`is not`	`x is not y`	`True` si las dos variables no son el mismo objeto

```
text1 = "Hola mundo"
text2 = "Hola mundo"
```

```
text1 is text2
```

```
False
```

```
help(id)
```

```
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```

```
id(text1)
```

```
2483667968368
```

```
id(text2)
```

```
2483667968496
```

```
text1 is text2
```

```
False
```

```
text1 == text2
```

```
True
```

```
text1 is not text2
```

```
True
```

```
text3 = text1
```

```
text3 is text1
```

```
True
```

```
id(text1)
```

```
2607307762480
```

```
id(text3)
```

```
2607307762480
```

```
id(text1) == id(text3)
```

```
True
```

Operadores de pertenencia

¿Cuáles son los operadores de pertenencia?

Los operadores de pertenencia se utilizan para evaluar sin una sequencia se encuentra presente en un objeto.

- una secuencia es un string
- los numeros no son secuencias(iterable)

Operador	Ejemplo	Significado
`in`	`x in y`	`True` si la secuencia `y` se encuentra presente en `x`
`not in`	`x not in y`	`True` si la secuencia `y` no se encuentra presente en `x`

```
text = "Cadena de texto"
```

```
"texto" in text
```

```
True
```

```
"hola" in text
```

```
False
```

```
"a" in text
```

```
True
```

```
num = 1345
```

```
1 in num
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-882699b1f88c> in <module>
----> 1 1 in num

TypeError: argument of type 'int' is not iterable
```

Operadores lógicos

1. ¿Cuáles son los operadores lógicos?

Los operadores lógicos modifican y unen expresiones evaluadas en contexto booleano para crear condiciones más complejas.

Operador	Ejemplo	Significado
`not`	`not x`	`True` if `x` is `False` `False` if `x` is `True`
`or`	`x or y`	`True` if either `x` or `y` is `True` `False` otherwise
`and`	`x and y`	`True` if both `x` and `y` are `True` `False` otherwise

2. Operador `not`

```
num = 5
```

```
num < 10
```

```
True
```

```
not num < 10
```

```
False
```

3. Operador `or`

```
num1 = 5
num2 = 10
```

```
num1 < 4
```

```
False
```

```
num1 < 4 or num2 > 5
```

```
True
```

4. Operador `and`

```
num1 = 5
num2 = 10
```

```
num1 < 4 and num2 > 5
```

```
False
```

```
num1 < 10 and num2 > 3
```

```
True
```

Booleanos

1. ¿Qué son los Booleanos?

Los booleanos se corresponden con uno de los tipos de datos más importantes de cualquier lenguaje de programación. Este tipo de dato se utiliza para representar la certeza de una expresión en Python.

Los tipos de datos Boolenos en Python se representan con el tipo `bool` y reciben únicamente dos posibles valores:

1. True
2. False

```
# Podemos verificar el tipo de estas dos palabras en Python 3
type(True)
```

```
bool
```

```
type(False)
```

```
bool
```

```
"----x----"[4]
```

```
'x'
```

Una de las cosas interesantes a tener en cuenta es que las palabras `True` y `False` son palabras clave dentro de Python. Esto quiere decir que son palabras reservadas dentro del lenguaje de programación y no se les puede asignar ningún valor.

```
True = "Hola mundo"
```

```
  File "<ipython-input-7-31dbcec71c48>", line 1
    True = "Hola mundo"
    ^
SyntaxError: cannot assign to True
```

De manera intuitiva, este tipo de palabras se comportan como si fuesen un tipo de datos numérico o texto.

```
"Hola mundo" = 4
```

```
  File "<ipython-input-8-ae0ff38b1627>", line 1
    "Hola mundo" = 4
    ^
SyntaxError: cannot assign to literal
```

```
5 = "Hola mundo"
```

```
  File "<ipython-input-9-f0bcfef0ce78>", line 1
    5 = "Hola mundo"
    ^
SyntaxError: cannot assign to literal
```

```
print(True or False)
True and False
```

```
True
False
```