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
```