¡Hola, Python!

Diego Morales

2024-03-19

Tabla de contenidos

Prefacio **4**

1 Introducción **5**
1.1 Mi primer programa 5

2 Variables **6**
2.1 Características 6

3 Tipos de datos **8**
3.1 Entero (int) 8
3.2 Flotante (float) 8
3.3 Texto (str) 8
3.4 Función type 9
3.5 Función input 9
3.6 Conversión de datos 10

4 Operadores Aritméticos **12**
4.1 Adición 12
4.2 Sustracción 12
4.3 Multiplicación 13
4.4 División 13
4.5 Exponenciación 13
4.6 Módulo 14
4.7 División de piso 14
4.8 Jerarquía de operaciones 14

5 Operadores de asignación **16**
5.1 Sustracción 16
5.2 Operadores de asignación y aritméticos 18

6 Operadores de comparación **20**
6.1 Igualdad 20
6.2 Desigualdad 21
6.3 Menor que 22
6.4 Mayor que 22
6.5 Menor o igual que 23

6.6 Mayor o igual que 23

Prefacio

Bienvenido a mi curso de programación introductoria a Python.

He creado este libro con el objetivo de compartir mis conocimientos con todo el mundo de la manera más efectiva posible.

And what if instead of it being in the right hands, it was in everyone's hands? -Steve Jobs [Film] (2015)

1 Introducción

En este capítulo creamos nuestro primer código de programación en Python, el cual mostrará un saludo en pantalla al ejecutar el programa.

1.1 Mi primer programa

La función `print` permite *desplegar mensajes* en la pantalla. Escribe el siguiente código teniendo cuidado de escribir correctamente el nombre de la función, luego, dentro de paréntesis deberás colocar tu mensaje utilizando comillas simples, `'`, o dobles, `"`, al inicio y al final.

```
print('¡Hola, Python! 2024')
```

```
¡Hola, Python! 2024
```

Si Python interpretó correctamente tu instrucción, entonces, deberías ver el mensaje anterior en la salida.

> **Nota**
>
> Es una práctica común crear el código para desplegar un mensaje de `Hello, world!` cuando se aprende un lenguaje nuevo de programación para comprobar que todo se encuentra configurado correctamente.

2 Variables

Una variable es un espacio de memoria en el cual podemos almacenar valores.

Por ejemplo, en el siguiente código. Se asigna el valor `4` a una variable llamada `x`.

```
x = 4
```

2.1 Características

Las variables poseen las siguientes características: 1. Puede almacenar distintos tipos de datos. 2. Su valor se puede modificar. 3. Se referencian por un nombre único (idealmente, significativo).

Observaciones sobre nombres de variables: 1. Python diferencia entre mayúsculas y minúsculas. 2. No pueden iniciar con números. 3. No pueden contener espacios. 4. No se pueden utilizar **palabras reservadas** (más adelante aprenderemos qué significan).

```
a = 100
print(a)
```

```
100
```

Si asignamos un nuevo valor a la misma variable, el valor anterior es reemplazado.

```
a = 100
a = 200
print(a)
```

```
200
```

Si se desean crear variables utilizando nombres significativos conformados por varias palabras, se recomienda utilizar la "notación de camello" (*camelCase*):

```
correoElectronico = 'juan_lopez2020@gmail.com'
```

También pueden separarse las palabras utilizando **guion bajo**, sin embargo, este tipo de notación suele utilizarse para funciones.

```
correo_electronico = 'juan_lopez2020@gmail.com'
```

3 Tipos de datos

Existen una gran cantidad de tipos de datos en Python: *Texto, numéricos, secuencia, mapeos, conjuntos, booleanos, binarios.*

3.1 Entero (int)

Dato numérico utilizado para representar números enteros.

```
cantidadAlumnos = 256
print(cantidadAlumnos)
```

```
256
```

3.2 Flotante (float)

Dato numérico utilizado para representar números reales, es decir, números con cifras decimales.

```
alturaMetros = 1.75
print(alturaMetros)
```

```
1.75
```

3.3 Texto (str)

Cadena de caracteres: letras, números y símbolos. El valor **se debe colocar dentro de comillas simples o dobles** para indicar que es un `str`.

```
movie = "The Lord of the Rings: The Return of the King (2003)"
cantidadAlumnos2 = "256"
alturaMetros2 = "1.75"
print(movie)
print(cantidadAlumnos2)
print(alturaMetros2)
```

```
The Lord of the Rings: The Return of the King (2003)
256
1.75
```

> **Advertencia**
>
> Se debe tener especial cuidado al manejar números representados como cadenas de caracteres ya que Python no los interpretará como valores numéricos (`int` o `float`).

3.4 Función type

La función `type()` se puede utilizar para conocer el tipo de dato de una variable.

```
print(type(cantidadAlumnos))
print(type(cantidadAlumnos2))
```

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

3.5 Función input

La función `input()` se utiliza para solicitar al usuario una entrada. El programa se detiene, hasta que el usuario presiona la tecla **Enter** en su teclado (luego de escribir el ingreso). Esta función, retorna una cadena de caracteres (`str`) con el ingreso del usuario.

```
nom = input("¿Cuál es su nombre? ")
```

Por lo tanto, puedes utilizar esta función para asignar valores a variables al iniciar la ejecución de tu programa.

Ahora veamos qué sucede si solicitamos al usuario un número que utilizaremos para realizar una operación más adelante.

```
age = input("Ingrese su edad: ")
```

Esa línea de código sería equivalente a que tomemos el valor devuelto por la función `input` y lo asignemos en la variable `age`. Sería igual al siguiente código.

```
age = '28'
birthYear = 2024 - age
print("Usted nació en el año ", birthYear)
```

Python nos muestra un mensaje de error (TypeError) debido a que la operación de resta (`-`) no está soportada entre variables numéricas (`int` o `float`) y cadenas de caracteres (`str`).

> **!** Importante
>
> Es completamente normal (y esperado) obtener errores al escribir código, incluso para programadores experimentados. Suelen aparecer cuando Python no es capaz de interpretar una instrucción (SintaxError) o porque se está realizando una acción inválida (como la anterior). Lo importante es comprender el error y corregirlo, lo cual será cada vez más sencillo con la práctica.

En este caso, podríamos realizar una conversión de datos para que la variable `age` se interprete como un entero.

```
age = '28'
age = int(age) # conversion a entero (casting)
birthYear = 2024 - age
print("Usted nació en el año ", birthYear)
```

```
Usted nació en el año  1996
```

3.6 Conversión de datos

Para convertir entre distintos tipos de datos, simplemente debemos especificar el tipo de dato y colocar el valor dentro de paréntesis.

1. **int()**
2. **float()**
3. **str()**

Por ejemplo, si queremos convertir una variable **`x = "99"`** a un valor numérico.

```
x = "99"
print(type(x))
x = int(x)
print(type(x))
```

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

4 Operadores Aritméticos

Los operadores ariméticos se utilizan para realizar operaciones matemáticas sencillas sobre valores numéricos:

1. Adición
2. Sustracción
3. Multiplicación
4. División
5. Exponenciación
6. División de piso
7. Módulo

4.1 Adición

La operación de suma entre un valor `a` y un valor `b` se realiza utilizando el operador `+`: $a + b$

> **Tip**
>
> Se pueden sumar múltiples valores en una misma expresión. Por ejemplo: $a + b + c + d$

```
100 + 717.345 + 0
```

```
817.345
```

4.2 Sustracción

La operación de resta entre un valor `a` y un valor `b` se realiza utilizando el operador `-`: $a - b$

> **Precaución**
>
> Nótese que la siguiente operación debería dar un resultado exacto, sin embargo, las computadoras manejan "números de punto flotante" los cuales no poseen una representación binaria de valores limitada y podrían mostrar valores poco precisos.

```
55.54 - 105.54
```

```
-50.00000000000001
```

4.3 Multiplicación

Para multiplicar un número **a** y un número **b**, se utiliza el operador ***.

$a * b$

```
7*2
```

```
14
```

4.4 División

Para dividir un número **a** y un número **b**, se utiliza el operador /.

$\frac{a}{b}$

```
15/2
```

```
7.5
```

4.5 Exponenciación

Para elevar un número **a** al exponente de un número **b**, se utiliza el operador ****.

a^b

```
2**3
```

```
8
```

4.6 Módulo

Para obtener el residuo de la división de un número **a** y un número **b**, se utiliza el operador %.

```
6%6
```

```
0
```

```
7%6
```

```
1
```

4.7 División de piso

Para obtener la división de piso de un número **a** y un número **b**, se utiliza el operador //.

Funciona de forma similar a la división normal, con la diferencia que retorna un número entero e ignora la parte decimal.

```
10//3
```

```
3
```

4.8 Jerarquía de operaciones

Es importante tener en cuenta la jerarquía de operaciones al utilizar varios operadores aritméticos en una misma expresión.

Las operaciones se efectúan en el siguiente orden: 1. Expresiones dentro de paréntesis: () 2. Exponente: ** 3. Multiplicación, División, División entera, Módulo: *, /, //, % 4. Suma, Resta: +, -

Si quisieramos convertir 100°F a °C. Se utilizaría la siguiente expresión:

$(100 - 32) * 5/9$

Siguiendo la jerarquía de operaciones, se realizan de la siguiente manera:

- $(100 - 32) * 5/9$
- $(68) * 5/9$

- 37.78

```
(100 - 32) * 5/9
```

```
37.77777777777778
```

Si no se hubieran colocado los paréntesis, obtendríamos un resultado erróneo: * $100 - 32 * 5/9$ * $100 - 32 * 0.5556$ * $100 - 17.78$ * 82.22

```
100 - 32 * 5/9
```

```
82.22222222222223
```

5 Operadores de asignación

Los operadores de asignación se utilizan para asignar un valor a una variable.

Por ejemplo, si queremos asignar un valor `200.0` a una variable `x` utilizamos el operador `=` de la siguienta manera:

```
x = 200
```

> **Nota**
>
> Nótese que la asignación se realiza de la derecha de la igualdad a la izquierda: $x \leftarrow 200.0$

```
100 + 717.345 + 0
```

```
817.345
```

5.1 Sustracción

La operación de resta entre un valor `a` y un valor `b` se realiza utilizando el operador `-`: $a - b$

> **Precaución**
>
> Nótese que la siguiente operación debería dar un resultado exacto, sin embargo, las computadoras manejan "números de punto flotante" los cuales no poseen una representación binaria de valores limitada y podrían mostrar valores poco precisos.

```
x = 200.0
print(x)
```

```
200.0
```

Los resultados de expresiones se pueden almacenar en variables.

Por ejemplo, si quisieramos sumar el costo de tres productos de una compra se suma el precio de cada uno (`249.99 + 9.99 + 50`) y se almacena el resultado final en una variable (`costoTotal`).

```
costoTotal = 249.99 + 9.99 + 50
print(costoTotal)
```

```
309.98
```

Esto nos permite almacenar resultados que podríamos utilizar posteriormente en el código.

Por ejemplo, asumamos que se le aplica al cliente un descuento del 20% sobre el total de la compra. Ya que tenemos una variable con el costo total (`costoTotal`) solo habría que multiplicar (`*`) por el porcentaje a pagar (`1-descuento`) y almacenar el resultado final en otra variable (`costoFinal`).

```
descuento = 0.20
costoFinal = costoTotal*(1-descuento)
print(costoFinal)
```

```
247.98400000000004
```

Otra ventaja de implementar variables en lugar de colocar valores fijos es que podemos modificar su valor para obtener distintos resultados.

Por ejemplo, si queremos calcular el área de un rectángulo de 20 metros por 40 metros, la fórmula sería:

$rea_{rectngulo} = 20 * 40$

El resultado siempre retornará 800.

$rea_{rectngulo} = base * altura$

Podríamos definir una variable para la base y otra para la altura.

De esta manera, si queremos calcular el área de otro rectángulo, solo debemos cambiar los valores de esas variables.

```
base = 35
altura = 40
areaRectangulo = base*altura
print(areaRectangulo)
```

```
1400
```

5.2 Operadores de asignación y aritméticos

Además del operador de asignación =, existen otros operadores de asignación que realizan operaciones aritméticas y asignan el resultado a la vez.

Si tenemos una variable **x = 3** y queremos aumentar 2 su valor. Utilizando el operador =. Haríamos lo siguiente:

```
x = 3
print(x)
x = x + 2
print(x)
```

```
3
5
```

Si tenemos una variable **cuotasRestantes = 24** y queremos restarle 6. El código sería:

```
cuotasRestantes = 24
print(cuotasRestantes)
cuotasRestantes = cuotasRestantes - 6
cuotasRestantes = cuotasRestantes - 3
print(cuotasRestantes)
```

```
24
15
```

En los ejemplos anteriores, para aumentar o disminuir el valor de una misma variable. Primero, se lee el valor actual de la variable, se realiza la operación y, finalmente, se asigna el resultado a la misma variable.

```
tareas = tareas + 2
```

En estos casos especiales cuando queremos modificar el valor de una variable y almacenar el resultado en la misma variable. Podemos hacer uso de los operadores de aritméticos en cojunto con la asignación. De la siguiente forma:

```
tareas += 2
```

Primero, colocamos la variable que queremos aumentar, luego, el operador aritmético seguido del operador de asignación, y finalmente, colocamos el valor.

Los dos códigos anteriores son equivalentes.

```
horasEstudio = 1
print(horasEstudio)
horasEstudio = horasEstudio + 2
print(horasEstudio)
```

```
1
3
```

```
horasEstudio = 1
print(horasEstudio)
horasEstudio += 2
print(horasEstudio)
```

```
1
3
```

Se puede utilizar cualquier operador aritmético y también podríamos sumar valores de variables en lugar de constantes.

```
calificacion = 90
print("Calificación inicial", calificacion, "pts")

extras = 10
calificacion += extras
print("Calificación con extra", calificacion, "pts")

bono = 1.1
calificacion *= bono
print("La calificación final con el extra y el bono es", calificacion)
```

```
Calificación inicial 90 pts
Calificación con extra 100 pts
La calificación final con el extra y el bono es 110.00000000000001
```

6 Operadores de comparación

Los operadores de comparación se utilizan para comparar dos operandos. El resultado de la operación retornará un valor booleano (`True` o `False`).

Las comparaciones que se pueden realizar son las siguientes:

1. Igualdad: `==`
2. Desigualdad: `!=`
3. Menor que: `<`
4. Mayor que: `>`
5. Menor o igual que: `<=`
6. Mayor o igual que: `>=`

6.1 Igualdad

Si el operando de la **izquierda** es igual al operando de la **derecha**, entonces, el resultado es `True`. De lo contrario, es `False`.

Por ejemplo, podríamos crear las siguientes variables.

```
a = 5
b = 7
c = 5
```

Al evaluar la igualdad entre `a` y `b`, lo escribiríamos de la siguiente manera y obtendríamos un resultado `True`.

```
a == b
```

```
False
```

> Nótese que el resultado es el mismo si hubiéramos escrito los operandos al revés, es decir, `b == a`.

Si ahora evaluamos la igualdad entre `a` y `c`, el resultado será `False`.

```
a == c
```

```
True
```

Finalmente, creamos una variable de tipo `str` con el valor de `'5'` y comparamos si es igual a la variable `a`.

```
d = '5'
d == a
```

```
False
```

A simple vista, los valores de cada variable son idénticos. Sin embargo, Python no considera iguales los valores numéricos al compararse con un número representado como `str`.

> Importante asegurarse que al realizar comparaciones las condiciones de cada variable sean lo más similares posibles. Por ejemplo, el código anterior se podría comparar perfectamente si realizamos un *casting* a entero: `int(d)`

```
int(d) == a
```

```
True
```

6.2 Desigualdad

Si el operando de la **izquierda** no es igual al operando de la **derecha**, entonces, el resultado es `True`. De lo contrario, es `False`.

Ejemplifiquemos este operador comparando directamente valores literales en lugar de variables.

```
'Diego' != 'Alberto'
```

```
True
```

```
'Diego' != 'Diego'
```

```
False
```

Ahora evalúemos la igualdad entre dos cadenas aparentemente idénticas pero que difieren por el uso de mayúsculas.

```
'Alberto' != 'alberto'
```

```
True
```

Python no considera igual el caracter `A` y la `a`. Por lo tanto, las cadenas anteriores son diferentes. Nuevamente, es importante tener cuidado al realizar comparaciones que los operandos se encuentren en condiciones similares.

También tomar en consideración que cualquier caracterer adicional, podría afectar nuestra comparación. Por ejemplo, veamos que si comparamos dos cadenas iguales pero colocamos un espacio adicional en una de ellas, entonces, los `str` se considerarían **no iguales**.

```
'Diego' != 'Diego '
```

```
True
```

6.3 Menor que

Si el operador de la **izquierda** es menor que el de la **derecha**, entonces, el resultado es `True`. De lo contrario, es `False`.

6.4 Mayor que

Si el operador de la **izquierda** es mayor que el de la **derecha**, entonces, el resultado es `True`. De lo contrario, es `False`.

Para los siguientes ejemplos, definiremos variables que contienen la edad de tres personas.

```
edad_1 = 7
edad_2 = 16
edad_3 = 33
```

Ahora, comparemos las edades de las personas. Primero, evaluamos si la edad de la persona 1 es menor yor que la edad de la persona 2.

```
edad_1 < edad_2
```

```
True
```

Luego, evaluemos si la edad de la persona 3 es mayor que la de la persona 2.

```
edad_3 > edad_2
```

```
True
```

6.5 Menor o igual que

Si el operador de la **izquierda** es menor o igual que el de la **derecha**, entonces, la condición es `True`. De lo contrario, es `False`.

6.6 Mayor o igual que

Si el operador de la **izquierda** es mayor o igual que el de la **derecha**, entonces, la condición es `True`. De lo contrario, es `False`.

A continuación, crearemos variables con notas de estudiantes.

```
notaLuis = 5.9
notaMaria = 7.8
notaIsa = 6.0
```

Utilizaremos los operadores para evaluar si los estudiantes aprobaron el curso. Se sabe que la nota mínima debe ser de 6.0

```
notaLuis >= 6.0
```

```
False
```

```
notaMaria>= 6.0
```

```
True
```

```
notaIsa>= 6.0
```

```
True
```

> Los comparadores anteriores se utilizan para realizar evaluar rangos de valores numéricos. No se recomienda su uso para comparar cadenas de caracteres.

Veamos la siguiente comparación.

```
'100' > '42'
```

```
False
```

Del ejemplo anterior, podemos observar que las comparaciones de números representados por `str` no funcionan de la misma forma que con valores numéricos en `int` o `float`. Incluso podríamos intentar hacer comparaciones entre textos.

```
'pequeño' > 'grande'
```

```
True
```

El resultado parece no tener ningún sentido lógico, sin embargo, Python no es capaz de reconocer que el significado de un `str`. Realmente, al comparar cadenas de caracteres de esta manera lo que se realiza es una comparación lexicográfica. Es decir, se compara la representación numérica de cada caracter.

Por ejemplo, el caracter `a` se representa por el valor decimal 97, mientras que el caracter `b` se representa por el valor decimal 98. Por eso al realizar la siguiente comparación obtendremos `True`. Ya que es equivalente a comparar `97 < 98`.

```
'a' < 'b'
```

```
True
```

¿Qué sucede cuando tenemos una cadena con más de un caracter? En este caso, se realiza la misma comparación mencionada anteriormente pero, caracter por caracter, hasta encontrar un caracter distinto. Por eso, al comparar si `caza` es mayor a `casa`, obtendremos `True`.

```
'caza' > 'casa'
```

```
True
```