1. Tipos de Operadores

Un lenguaje de programación permite interactuar con elementos almacenados en memoria para resolver un

a tarea determinada. Los operadores permiten manipular esos elementos y ofrecen la versatilidad para obtener distintos resultados. En este capítulo se aborda los tipos de operadores que podemos utilizar con Python.

1.1 Operadores aritméticos

Todo lenguaje de programación permite trabajar con las operaciones básicas de la aritmética como lo son, la suma, la resta, la multiplicación y la división. Tendremos en cuenta que inicialmente los operadores aritméticos se aplicarán a números, sin embargo, veremos más adelante que estos operadores pueden ser aplicados a distintos tipos de elementos en Python.

- **Suma**

En el caso de la suma utilizamos el símbolo **+** para operar entre términos.

```
>>> 2 + 6
8
```

Un lenguaje de programación debe permitir tratar datos mediante operadores

También podemos operar la suma con términos que no son enteros:

```
>>> 2 + 4.3
6.3
```

- **Resta**

Para realizar una resta utilizamos el símbolo **-** que nos permite operar entre términos.

```
>>> 9 - 1
8
```

Igualmente podemos operar la resta con valores flotantes.

```
>>> 10 - 6.2
3.8
```

Otro ejemplo:

```
>>> 5 - 1.3
3.7
```

- **Multiplicación**

La multiplicación en Python se representa con el símbolo ***** para operar un par de términos.

```
>>> 2 * 7
14

>>> 2 * 5.2
10.4
```

Suma (+)
Resta (-)
Multiplicación (*)
División (/)
Potencia (**)
División piso (//)
Módulo (%)

Operadores Aritméticos en Python

- **División**

En el caso de la división el símbolo utilizado es **/**. Se debe tener en cuenta que el resultado de esta operación siempre es un valor flotante o con decimales así la división se visualice en consola entera.

```
>>> 12 / 4 # La división devuelve un resultado flotante
3.0
```

Se puede comprobar lo anterior si guardamos el resultado de una división en una variable y luego preguntamos por el tipo de dato que guarda la variable.

```
>>> x = 12 / 6 # Se guarda en la variable x

>>> type(x)
<class 'float'>
```

En los operadores aritméticos también tenemos la división piso, el módulo y la potencia.

- **División piso**

Realicemos primero una división entre dos números.

```
>>> 10 / 3
3.3333333333333335
```

Compruebe cuántos decimales presenta esta división

El operador para la división piso es una doble barra inclinada **//** entre dos términos.

```
>>> 10 // 3
```

3

Nos entrega el valor redondeado a 3. Para entenderlo mejor, revisemos otro ejemplo.

```
>>> 14.5 / 3
4.833333333333333
```

Realicemos la operación división piso entre los mismos números:

```
>>> 14.5 // 3
4.0
```

Como podemos ver al ejecutar la anterior instrucción tenemos un valor redondeado a 4.0. A pesar de que en la división tradicional el resultado se aproxime más a 5, la división piso devuelve el valor redondeado al entero inferior más próximo, en este caso 4.0.

Ejercicio propuesto

Prueba realizar la división piso **8//-3** y compárala con la división tradicional **8/-3**.

- **Módulo (Residuo)**

Esta operación devuelve el residuo de la división de dos términos. El símbolo utilizado es "%".

```
>>> 20 / 3
6.666666666666667
```

```
>>> 20 % 3
2
```

Podemos guardar el resultado de la operación en una variable.

```
>>> xy = 20 % 3
```

Prueba de la división:

Divisor * Cociente + Residuo

```
>>> type(xy)
<class 'int'>
```

```
>>> 3 * 6 + 2
20
```

La anterior instrucción muestra la prueba de la división. Al aplicar **20%3** encontramos como respuesta **2**, si aplicamos **3*6= 18** y **18** más **2** tenemos el resultado de **20** que es la prueba de la división.

```
>>> 24 % 8
0
```

```
>>> 10 / 4
2.5
```

```
>>> 10 % 4
2
```

- **Potencia**

Podemos utilizar dos asteriscos seguidos para representar la operación potencia entre una base y exponente.

```
>>> 2 ** 3
8
```

```
>>> 4 ** 2
16
```

```
>>> base = 10
```

```
>>> altura = 2
```

```
>>> base * altura
20
```

A pesar de que conocemos que el área de un rectángulo es igual a base por altura, no podemos llamar la variable "area"**.**

```
>>> area
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'area' is not defined
```

Este error aparece porque la variable no está definida

Python puede operar con otro tipo de valores numéricos como Decimales, Fraccionarios y los mismos números complejos.

1.2 Operadores de Comparación

Este tipo de operadores los usamos cuando deseamos comparar expresiones o variables. Python evalúa si se cumple la comparación y nos devuelve (retorna) un valor **True** o **False**

Los operadores de comparación se relacionan a continuación:

5

Definición	Símbolo
es igual a	==
es diferente a	!=
es mayor que	>
es menor que	<
es mayor o igual que	>=
es menor o igual que	<=

Tabla 1 Operadores de comparación en Python

Revisemos el uso de la comparación “es igual a”

Python distingue minúsculas de mayúsculas

```
>>> "palabra" == "Palabra"
False
```

Podemos aplicar esta misma comparación entre números

```
>>> 3 == 3
True
```

Comprobemos con un decimal cero después del punto.

```
>>> 3 == 3.0
True

>>> 3 == 3.0000000
True
```

Aunque puedan parecer lo mismo hay una diferencia en el siguiente ejemplo:

```
>>> 3 == 3.000000000000001
False
```

Existe un límite del número de cifras significativas que se pueden diferenciar.

Sin embargo, es necesario tener en cuenta que los lenguajes de programación tienen un límite de cifras decimales que pueden tratar. Compruebe esto definiendo dos variables de la siguiente manera:

```
>>> x = 4.123456789000001058946546

>>> y = 4.123456789000001

>>> x == y
True
```

Revisemos ahora el uso del operador de comparación "es diferente a". El símbolo que representa esta comparación es: **!=**

```
>>> 4 != 3
True
```

Aplicando el mismo operador entre números tenemos:

```
>>> 9!=9
False
```

Podemos aplicar el operador entre expresiones de tipo boolean.

```
>>> False != True
True
```

Podemos comparar entre expresiones booleanas

A continuación, revisemos el uso del operador de comparación "es mayor a". Utilicemos la función **print** para ejecutar la instrucción en el intérprete.

```
>>> print(5 > 8)
False
```

```
>>> print(10 > 3)
True
```

De manera análoga podemos utilizar el símbolo < para comprobar cuando el término o variable de la izquierda es menor al termino que tenemos a la derecha en la instrucción.

```
>>> 5 < 9
True
```

Si queremos incluir en la comparación la condición de igualdad podemos utilizar el símbolo >= como se muestra a continuación:

```
>>> 9 >= 5
True
```

De manera análoga podemos comprobar el uso de la expresión "es menor o igual a" para comparar dos términos o expresiones

```
>>> 2 <= 6
True
```

1.3 Operadores de asignación

Se considera el uso de operadores de asignación cuando estamos actualizando el valor de una variable utilizando siempre el símbolo = que representan igualdad.

CAPÍTULO 1. TIPOS DE OPERADORES

Los operadores de asignación son los relacionados en la siguiente tabla:

Definición	Símbolo
Igualdad	=
Incremento	+=
Decrecimiento	-=
Asignación multiplicación	*=
Asignación división	/=
Asignación división piso	//=
Asignación módulo	%=
Asignación exponente	**=

Partimos del valor inicial de una variable

Creemos primero una variable que queramos actualizar utilizando distintos operadores.

```
>>> var_1 = 5
```

```
>>> var_1 = var_1 + 1
```

- **Operador de incremento**

Con += podemos incrementar el valor actual de una variable indicando seguidamente en cuantas unidades queremos afectarla:

```
>>> var_1 += 1
```

```
>>> print("El resultado actual de la variables es: ",var_1)
El resultado actual de la variables es:  7
```

Definamos una nueva variable con un valor inicial de 8.

```
>>> var_2 = 8
```

Ejercicio propuesto

```
>>> var_1 += 6
```

Comprobemos los valores actuales en memoria de **var_1** y **var_2** utilizando la función **print**.

- **Operador de decrecimiento**

Podemos asignar un decrecimiento a una variable

Este operador se utiliza para hacer decrecer el valor actual de una variable, el símbolo utilizado es **-=** y debe ser acompañado de un valor que ejercerá el efecto de disminución.

PYTHON PURO

```
>>> var_3 = 8
```

```
>>> var_3 = var_3 - 2
```

Esto es lo mismo que expresar la instrucción **var_3-= 2**

```
>>> var_3 -= 2
```

```
>>> print("El resultado actual de la variable es: ", var_3)
El resultado actual de la variable es:  6
```

- **Operador asignación de multiplicación**

Este operador nos sirve para incrementar o decrecer un valor de una variable "**n**" veces utilizando el símbolo *= acompañado del número de veces a afectar la variable.

Las operaciones aritméticas básicas tienen operadores de asignación

Inicialicemos una variable:

```
>>> var_4 = 2
```

```
>>> print("el valor inicial de la variables es: ", var_4)
el valor inicial de la variables es:  2
```

Si deseamos tomar el valor actual y lo multiplicamos 2 veces podríamos expresarlo así:

```
>>> var_4 = var_4*2
```

```
>>> print("el valor actual de la variables es: ", var_4)
el valor incrementado de la variables es:  4
```

Alternativamente y con base en el último resultado, podríamos usar el operador para asignar un nuevo valor a la variable.

```
>>> var_4 *= 2
```

```
>>> print("el nuevo valor de la variables es: ", var_4)
el nuevo valor de la variables es:  8
```

Si utilizamos cifras decimales podríamos utilizar el operador para hacer decrecer una variable.

```
>>> var_5 = 10
```

```
>>> var_5 *= 0.5

>>> print(var_5)
5.0
```

- **Operador asignación división**

Este operador puede incrementar o decrecer el valor de una variable Normalmente al dividir lo que estamos haciendo es decrecer un valor, sin embargo, como lo vimos anteriormente, podríamos utilizar un valor mayor a cero y menor a uno para hacer crecer el valor actual.

```
>> 8/2
4.0

>>> var_6 = 20

>>> var_6 /= 2

>>> print(var_6)
10.0
```

Al dividir el valor entre un número >0 y <1 estaríamos incrementando el resultado.

Utilizar un número con decimales tiene el efecto inverso al divisor

```
>>> var_6 /= 0.5

>>> print(var_6)
20.0
```

- **Uso de operador de asignación división piso**

Este operador actualiza el valor de una variable redondeando el resultado al entero más próximo inferior.

Inicialicemos una variable

```
>>> var_7 = 5


>>> var_7//=3
```

El valor queda actualizado al entero más próximo inferior.

```
>>> print(var_7)
1
```

- **Uso de operador de asignación %=**

La operación módulo puede usarse en este caso para asignar el residuo de una división a una variable. Este operador **x %= 5** es equivalente a **x = x % 5**¸con lo cual se actualiza el valor de **x.**

```
>>> var_8 = 6

>>> var_8 %= 3

>>> print(var_8)
0
```

En este resultado no se encuentra residuo al hacer la operación

- **Uso de operador de asignación **=**

Este operador debe usarse cuando deseamos actualizar el valor de una variable posterior a la ejecución de la función potencia, especificando al lado derecho de la igualdad el exponente deseado.

```
>>> var_9 = 4

>>> var_9 **= 3

>>> print(var_9)
64
```

1.4 Operadores lógicos

En la sección anterior nos interesaba realizar una actualización a variable que se encuentran almacenadas en memoria. Es posible que para tratar dichas variables también nos interese consultar si cumplen condiciones determinadas. En Python podemos utilizar operadores lógicos para establecer condiciones entre variables. Los operadores lógicos son:

- and
- or
- not

Por reglas de sintaxis deben ir en minúsculas y en inglés.

11

Definamos un par de variables en primera instancia.

```
>>> x = 4

>>> y = 4
```

Las expresiones condicionales nos devuelven un resultado booleano.

```
>>> print(x == 4 and y == 4)
True
```

Según la instrucción anterior se cumplen las dos condiciones, por lo tanto, tenemos un **True** como resultado.

Los operadores:

and

or

not.

Devuelven un True o un False

- **Expresión or**

La expresión **or** valida simplemente que una de las dos condiciones se cumpla.

```
>>> print(x == 4 or y == 4)
True

>>> print(x == 2 or y == 2)
False
```

- **Expresión not**

En el caso del operador **not** se valida si una variable es **False** o **True**. Si la variable existe en memoria tenemos un valor por defecto True.

```
>>> print(not x)
False
```

El resultado de negar una variable que se encuentra en memoria es **False.**

```
>>> enunciado = False

>>> print(not enunciado)
True
```

De manera análoga, si creamos la variable asignando inicialmente un valor False, negar esta condición implica que tengamos un resultado **True**.

1.5 Operadores de identidad

Este tipo de operadores nos sirven para comparar la ubicación de las variables en memoria. Los tipos de operadores de identidad en Python son:

- is
- is not

A continuación, revisemos el uso de estas expresiones.

- **Uso del operador is**

Definamos inicialmente dos variables.

```
>>> x = -30

>>> y = -30

>>> print(x)
-30


>>> print(id(x))
2839315864304


>>> print(id(y))
2839315862128
```

Los valores de id son generados aleatoriamente.

Todas las variables en Python tienen un id asignado desde que se crean, enteros desde -5 a 256 tienen un id siempre único.

otros elementos tendrán siempre ubicaciones distintas

```
>>> w = 2

>>> z = 2

>>> print(id(w))
140731096110896

>>> print(id(z))
140731096110896
```

Comprueba si al realizar estos print se genera el mismo resultado.

13

Con el uso de id podemos saber por ejemplo no solo si un par de variables tienen el mismo valor, sino que si se encuentran en la misma ubicación de la memoria

```
>>> print(z is w)
True
```

En el caso de los strings también consideramos que se ubican en la misma área en memoria, podemos verificar esto con la expresión **is not**.

```
>>> texto_1 = "ejemplo"
>>> texto_2 = "ejemplo"
>>> print(texto_1 is not texto_2)
False
```

1.6 Operadores de contenido

Los operadores de contenido son útiles cuando queremos consultar si un elemento o variable se encuentra dentro de una secuencia, es decir en objetos como listas o **strings**. Revisemos el uso de la expresión **in**.

Podemos consultar por vacíos en una cadena de texto.

```
>>> var1 = "Python"
>>> '' in var1
False
```

De manera análoga Podemos utilizar **not in** para negar la existencia de un elemento de consulta en una secuencia.

This helps to deny the previous proposition

```
>>> '' not in var1
True
```

Ejercicios de repaso

Ejercicio 1.1

Escriba un programa que defina inicialmente x=4, utilice la función ***input*** para capturar del usuario un valor y. Calcule con ***x*** y con ***y*** la suma de los valores y guárdelo en una variable ***z***. Al ejecutar, el programa debe entregar un ***print*** con el siguiente texto: "El resultado es: ", y el valor resultante de ***z***.

Ejercicio 1.2

Escriba un programa que defina inicialmente x=4, utilice la función ***input*** para capturar del usuario un valor y. Calcule con ***x*** y con ***y*** la resta de los valores y guárdelo en una variable ***z***. Al ejecutar, el programa debe entregar un ***print*** con el siguiente texto: "El resultado de la resta es: ", y el valor resultante de ***z***.

Ejercicio 1.3

Escriba un programa que defina inicialmente x=8, utilice la función ***input*** para capturar del usuario un valor y. Calcule con ***x*** y con ***y*** la división de los valores y guárdelo en una variable ***z***. Al ejecutar, el programa debe entregar un ***print*** con el siguiente texto: "El resultado es: ", y el valor resultante de ***z***.

Ejercicio 1.4

Escriba un programa que capture del usuario dos valores ***a*** y ***b*** en dos inputs sucesivos***.*** Pida al usuario desde la función ***input*** que los valores a ingresar deben contener al menos un número decimal. Al ejecutar, el programa debe realizar la multiplicación entre los dos valores y entregar la respuesta en un ***formatted string*** que contenga una variable llamada resultado y el texto de su preferencia.

Ejercicio 1.5

Escriba un programa que realice la comprobación de la división. Recuerde:
`Divisor * Cociente + Residuo = Dividendo`
Tome como entrada dos números, realice la división entre ellos y entregue al usuario un texto descriptivo con la comprobación de la división.

Ejercicio 1.6

Teniendo en cuenta el ejercicio anterior calcule el residuo con el símbolo de módulo ***%*** y entregue la comprobación con los valores resultantes de dividir dos números entregados por el usuario del programa.

Ejercicio 1.7

Compruebe en un programa si el valor 29.0 es igual a 29 utilizando el operador de comparación correspondiente.

Ejercicio 1.8

Compruebe en un programa si la expresión "true" es igual a "True" mediante el operador de comparación correspondiente.

Ejercicio 1.9

Compruebe mediante el operador de identidad correspondiente si la letra ***w*** se encuentra en la expresión "Python es un lenguaje de programación muy popular". Utilice un input para consultar lo mismo y comprobar según la entrada que dé el usuario.