Capítulo 1

Tipos de datos, Operadores y Variables

Vamos a comenzar a usar Python desde el intérprete como si se tratase de una calculadora inteligente capaz de hacer cuentas y cálculos de forma rápida y precisa. Si queremos ver los resultados en nuestro entorno, deberemos usar **print,** tal y como aparece en el primer ejemplo

```
>>> print(3 + 4 * (8 - 2))
27
>>> print(45 / 7)
6.428571428571429
>>> print(45 // 7)
6
>>>print (7**2)
 49
>>> print(43 % 7)
1
```

Como puedes ver en los ejemplos Python puede realizar las operaciones básicas suma(+), resta(-), multiplicación(*) y división(/) y además, sabe aplicar correctamente la prioridad de unas operaciones sobre otras.

También habrás observado que hace algunas operaciones más como la potenciación(**), división entera(//) y el resto de la división(%, esta última también recibe el nombre de módulo o residuo).

Más adelante veremos como también es capaz de hacer muchísimas operaciones más, pero por el momento, es suficiente con estas.

Vamos a seguir dándole algunas órdenes más a Python, que aunque nos puedan parecer un poco extrañas, ya veréis como más tarde resultarán útiles.

```
>>> print("Hola")
'Hola'
>>> print("Hola " + "Arturo y Julio")
'Hola Arturo y Julio'
(Observa que en esta ocasión después de Hola se ha incluido un espacio para que el resultado se lea mejor y aparezca escrito correctamente).
>>> print(3 * 'Hola')
'HolaHolaHola'
```

En estos ejemplos hemos escrito texto entrecomillado, este tipo de información recibe el nombre de **cadenas de texto** (strings). En Python las cadenas de texto se pueden escribir entre comillas dobles o entre comillas simples.

En el segundo ejemplo hemos sumado 2 textos, visto así, esto te debería resultar un poco extraño. La explicación es que realmente no hemos sumado, lo que hemos hecho ha sido concatenar (unir) 2 cadenas de texto y hemos obtenido una nueva cadena texto con todo el texto.

Siguiendo la misma lógica anterior, multiplicar un número por un texto, es concatenar varias veces el texto consigo mismo.

Vamos a continuar dándole órdenes al intérprete de Python para conocer las respuesta que nos devuelve.

```
>>> print(8 > 3)
True
>>> print(5 < 2)
False
>>> print(6 >= 4 and 8 > 7)
True
>>> print(7 != 5)
True
```

Ahora nuestras órdenes lo que han hecho ha sido comparar valores numéricos y Python se ha encargado de decirnos si el resultado de las comparaciones es cierto (True) o falso (False).

A los valores True y False, se le llaman valore lógicos o booleanos.

Estos valores se utilizan para comprobar si se cumple alguna condición en un determinado momento. Estos valores en programación tienen una gran importancia porque sirve para tomar decisiones dentro del programa en función de las condiciones que se den.

En los ejemplos anteriores hemos realizado operaciones que seguramente te habrán resultado muy sencillas y fáciles de comprender. Es probable que pienses que no serán de mucha utilidad, pero no es así, comprender como es la información que manipula un ordenador y qué tipo de operaciones se puede hacer con ellas, es la base para conseguir darle instrucciones a un ordenador de forma correcta.

Antes de continuar, vamos a extraer algunos aspectos importantes de los ejemplos que hemos escrito antes.

Hemos usado diferentes tipos de datos: **numéricos** (enteros y decimales), **textos** (strings) y datos **lógicos** (booleanos). Python tiene más tipos de datos, pero por el momento, nos quedaremos con estos que son los más comunes cuando se empieza a programar.

Hemos realizado diferentes tipos de operaciones con los datos: operaciones matemáticas, operaciones con cadenas de texto y comparaciones cuyo resultado ha sido un valor lógico. A los símbolos que hemos utilizado para realizar estas operaciones, se les llama operadores, por lo tanto, hemos usado: **operadores matemáticos**, **operadores de cadenas** y **operadores de comparación**.

Operador	Tipo	Operación que realizan
+	Matemático	Sumas datos numéricos
-	Matemático	Restar datos numéricos
*	Matemático	Multiplicar datos numéricos
/	Matemático	División de 2 valores numéricos
//	Matemático	División entera, toma la parte entera del resultado
**	Matemático	Potenciación, para obtener un número elevado a un exponente
%	Matemático	Resto, módulo o residuo, se obtiene el resto de la división
+	Textos	Concatenación, unión de cadenas de texto
*	Textos	Repetir varias veces una misma cadena de texto
[]	Textos	Obtener partes de un texto
<	Comparación	Un valor menor que otro

<=	Comparación	Un valor menor o igual que otro
>	Comparación	Un valor mayor que otro
>=	Comparación	Un valor mayor o igual que otro
==	Comparación	Comprobar si dos valores son iguales
!=	Comparación	Comprobar si dos valores son diferentes
& (and)	Comparación	Comprobar si se cumplen varias condiciones
| (or)	Comparación	Comprobar si se cumple alguna de las condiciones
=	Asignación	Asignarle un valor a una variable. Ej.: x=5
+=	Asignación	Ej.: x+=1, es equivalente a, x=x+1
-=	Asignación	Ej.: x-=1, es equivalente a, x=x-1
*=	Asignación	Ej.: x*=2, es equivalente a, x=x*2
/=	Asignación	Ej.: x/=2, es equivalente a, x=x/2
%=	Asignación	Ej.: x%=2, es equivalente a, x=x%2

Nota:

Dentro de los operadores de comparación, Python también permite operadores ternarios, es decir, expresiones así: 1 <= x <= 10. Esta forma de escribirlo es similar a como se haría en matemáticas y es equivalente a: x>=1 and x<=10, sin embargo, resulta más simplificada y más intuitiva de leer.

Existen algunas variantes más de estos operadores y algunos conceptos más que poco a poco iremos conociendo, practicando y asimilando.

En los ejemplos anteriores hemos realizado operaciones y gestionado información que de poco serviría si nuestros programas no pueden guardar esos datos para volver a utilizarlos posteriormente. La forma de guardar en la memoria del ordenador estos datos es dándole un nombre y asignándole el valor a dicho nombre.

Este es el concepto de variable, se trata de un nombre al cual se le asigna un valor. Este valor puede ser directo o como resultado de una operación.

```
>>> x = 5
>>> y = x**2
print(y)
25
>>> print(x + y)
 30
>>> nombre1 = 'Arturo'
>>> nombre2 = 'Julio'
>>> print(nombre1 + ' ' +
nombre2)
 'Arturo y Julio'
```

Las variables son un almacén donde guardar una información y que en cualquier momento podemos recuperarla para volver a utilizarla.

El nombre de las variables debe cumplir ciertas reglas:

- No deben se palabras reservadas de Python
- Deben comenzar por una letra
- No pueden contener espacios ni caracteres extraños
- Python hace distinción entre mayúsculas y minúsculas

En los lenguajes de programación, **el símbolo = debe entenderse como una asignación**, no como una igualdad matemática. La expresión: **x = x +1**, en matemáticas no tendría mucho sentido puesto que sería afirmar que 1 = 0, sin embargo, en cualquier lenguaje de programación, es totalmente válida y **debe entenderse como asígnale a x el valor que tenga en este momento más una unidad**.

Importante: En Python, las variables no es necesario declararlas ni indicar qué tipo de información va a contener, basta con asignarles un valor y nada más.

A continuación, vamos a seguir avanzando dentro del intérprete de Python con algunas instrucciones que resultan básicas en la elaboración de cualquier programa. Estas funciones son las funciones de entrada y salida de información, es decir, las principales funciones para mostrar alguna información por pantalla y para introducir datos en el programa.

Las funciones están formadas por un nombre, seguidas de paréntesis, y dentro de estos paréntesis se escriben los argumentos o parámetros (valores que se le pasan a la función).

La primera de estas funciones que vamos a ver, es la función print(), que se utiliza para mostrar información por pantalla y cuya sintaxis es:

print(argumento1, argumento2, argumento3, ...)

Vamos a practicar en el intérprete las siguientes instrucciones:

```
>>> print('¡¡ Hola Mundo !!')
¡¡ Hola Mundo !!
>>> nombre1 = 'Arturo'
>>> nombre2 = 'Julio'
>>> print("Buenos días", nombre1,"y",nombre2)
Buenos días Arturo y Julio
>>> print("El resultado de", 5, "por", 9, "es",5*9)
El resultado de 5 por 9 es 45
>>> print("Buenos"+" "+"días")
Buenos días
```

La instrucción print la usaremos para mostrar mensajes por pantalla.

Observa que cuando se le pasan varios **argumentos separados por comas**, la función print al mostrar estos valores, **introduce un espacio** entre ellos para que se visualicen mejor.

Un buen uso de las cadenas de texto nos ayudará a mostrar bien la información.

Al igual que la función print sirve para mostrar información, disponemos de la función input para introducir datos en el programa para que puedan ser procesados. Estos datos se almacenan en una variable y se utilizan cuando el programa los necesita en función de las tareas que deba llevar a cabo.

variable = input('Mensaje a mostrar al solicitar el dato')

```
>>> n = input('Escribe un número: ')
      Escribe un número:
```

A partir de este momento la variable n tendrá como contenido el texto que escribamos hasta la pulsación de Intro.

Es importante resaltar que desde la versión 3.x de Python la función input devuelve una cadena de texto, en las versiones anteriores (2.x de Python), esta función devolvía el resultado de evaluar matemáticamente la expresión escrita.

Por lo tanto, si queremos usar numéricamente el valor introducido mediante input, tendremos que usar alguno de los métodos que Python facilita para convertir un texto en un valor

numérico.

```
>>> n = input('Escribe un número: ')
      Escribe un número: 12
>>>print( n)
'12'
>>> v = int(n)
>>> print (v)
12
>>> print(3*n,3*v)
121212 36
```

En estos ejemplos podemos comprobar que el valor de n es el texto '12', sin embargo, el valor de v es el número 12.

La conversión de n en un valor numérico podemos hacer de varias formas distintas:

```
v = int(n)
v = float(n)
v = eval(n)
```

eval permite que n contenga cualquier expresión válida en Python, por ejemplo: 3*(7-5)**2

Algunas aclaraciones sobre la función input y sus conversiones:

Los lenguajes de programación permiten realizar todo tipo de operaciones con la información que le vamos suministrando, sin embargo, es lógico que para determinadas operaciones, los datos deban ser de un determinado tipo, por ejemplo, no tiene mucho sentido hacer determinadas operaciones matemáticas con cadenas de textos

Si llevamos a cabo las instrucciones:

```
>>> n = input('Escribe un número: ')
      Escribe un número: 5.3
>>> v = float(n)
>>> print(n,' --- ',v)
5.3 ----- 5.3
>>> v2 = int(n)
```

Para que la conversión de una cadena de texto a entero o float, se realice correctamente, si usamos las funciones int() o float() los valores deben ser del tipo correspondiente.

La función eval() es más versátil que las funciones int() y float().

Se produce un error por ser n un valor inválido para convertirlo en un valor entero.

En este sentido, es necesario ser cuidadoso al realizar operaciones con valores de diferentes tipos de datos. Generalmente será necesario convertir alguno de los datos al tipo conveniente para que se puedan llevar a cabo las operaciones. Por ejemplo: "Hola" + 5, generará un error porque no se puede concatenar un texto con un valor entero entendido como tal, sin embargo, sí podemos convertir el número 5 a un texto y de esa forma poder concatenarlo con el texto 'Hola'. Esto podríamos hacerlos así: 'Hola' + str(5), se obtendría 'Hola5'.

```
>>> v3 = eval(n)
>>> print(v3)
5.3
>>> n = input('Escribe una expresión evaluable: ')
      Escribe una expresión evaluable: 4*(8-2*3)
>>> v = eval(n)
>>> print(n,' --- ',v)
4*(8-2*3) ----- 8
```

En la mayoría de ocasiones la función eval() es la más adecuada para convertir cadenas de texto en valores numéricos.

Incluso si tienes una variable x con un determinado valor, puedes usar la función eval, para evaluar expresiones del tipo: 2*(x-1)**2

Tipos de datos más elaborados

Ya conocemos básicos de Python, sin embargo, disponemos de otros tipos de datos que resultan de gran utilidad en la programación. Estos datos más elaborados suelen estar compuestos por colecciones de datos básicos (enteros, decimales, textos o lógicos).

Listas

Las listas en Python son una colección de datos que resultan de gran importancia y ahorran muchísimo trabajo en la realización de un programa. Por el momento vamos a indicar como se crean, como podemos hacer un uso básico de ellas y más adelante profundizaremos en su conocimiento.

```
>>> lista = ['Arturo', 11, 'Julio', 7.5, True]
>>> print(lista[0])

'Arturo'
>>> print(lista[4]
)True
>>> print(lista[3])
7.5
```

En las listas se almacenan varios datos que posteriormente pueden ser referidos mediante un índice.

Para indicar el elemento hay que utilizar corchetes [].

Las listas comienzan a numerarse por 0 (cero).

El uso de listas es muy común al programar con Python.

En Python tenemos muchas opciones disponibles que facilitan manipular el contenido de las listas. Vamos a ver algunas de estas posibilidades continuando con el ejemplo anterior:

```
>>> lista.append('Saludos')
>>>print(lista)
['Arturo', 11, 'Julio', 7.5, True, 'Saludos']
>>> lista.remove(11)
>>> print(lista)
['Arturo', 'Julio', 7.5, True, 'Saludos']
>>> lista.reverse()
>>> lista
['Saludos', True, 7.5, 'Julio', 'Arturo']
>>> lista.insert(1, 'Hola')
>>> print(lista)
['Saludos', 'Hola',True, 7.5, 'Julio', 'Arturo']
```

Observa que en las listas se puede almacenar información de diferentes tipos de datos.

En una misma lista puede haber texto, números, valores lógicos, etc.

Seleccionar uno o varios elementos de una lista, es muy fácil de hacer en Python, vamos a ver algunos ejemplos más que mostrarán algunas de estas posibilidades disponibles.

```
>>> a=[7, 'Hola', 12, True, 'Saludos', 4.3, False]
>>>print(a[1:4])
['Hola', 12, True]
>>> print(a[-3])
'Saludos'
>>> print(a[:3])
[7, 'Hola', 12]
>>> print(a[4:])
['Saludos', 4.3, False]
```

Nota importante: Las cadenas de texto, también permiten que se puedan acceder a sus caracteres de la misma forma que las listas. Esto no quiere decir, que las cadenas de texto sean listas, pero sí podemos acceder a letras de la siguiente forma:

```
>>> nombre = 'Arturo'
>>> print(nombre[0]
)'A'
>>>print( nombre[1:4])
'rtu'
>>>
print(nombre[:4])
'Artu'
>>> print(nombre[-3:])
'uro'
```

En varios de los ejemplos anteriores has podido observar que al indicar dos índices separados por 2 puntos, se obtienen todos los elementos cuyos índices se encuentran comprendidos entre ambos índices, incluido el primero de los índices, pero excluido el último, es decir, nombre[1:4] incluirá los caracteres que se correspondientes a los índices 1, 2 y 3.

Recuerda que los índices se comienzan a numerar por el 0.

Cuando indicamos [:n] estamos diciéndole que seleccione desde el principio hasta n y cuando indicamos [n:], le decimos que llegue desde n hasta el final.

Las listas son una herramienta muy poderosa en Python y las utilizaremos en multitud de ocasiones puesto que facilitan muchas operaciones con la información que gestionen nuestros programas.

```
lista1=[]          #Esta instrucción define una lista llamada lista1 pero vacía,
                   sin ningún elemento por el momento.
```

Las listas disponen de funciones (métodos) que permiten gestionar su contenido de forma bastante cómoda: append, clear, copy, count, extend, index, insert, popremove, reverse y sort. Cada una de estas funciones realiza una tarea específica con el contenido de una lista.

En Python disponemos de otros tipos de colecciones de datos: tuplas, diccionarios y conjuntos, los cuales tienen funcionalidades parecidas a las listas aunque con diferencias entre unas y otras.

Por el momento, nos vamos a quedar únicamente con las listas.

Ejercicios:

Ejercicio 1:

Evaluar las expresiones y escribir el resultado:

4*3-5*2+3*(-1)	5*(8-2*7+1)	4-6*(3+2*4)	4*(3-5)+(12+8)*3
2+4**3	(2+4)**3	2**10	36 % 8
36 % 7	36 % 6	16 / 5	16 // 5
3 < 7	5 <= 6	5 >= 5	6 < 8 and 7 > 9
5 == 4	7 != 8	6 == 4 or 5 < 7	6 == 4 and 5 < 7

Ejercicio 2:

Realizando previamente la asignación n=5, realiza en el orden que aparecen las siguientes operaciones y escribe el resultado obtenido (primero de izquierda a derecha y después hacia abajo):

y=2*n+2	y	n+=1	x=3*n
x	x**2	x%=5	x*=4
n**(1/2)	n / x	x // y	n /= 2

Ejercicio 3:

Realizando previamente la asignación palabra1='Informática' y palabra2='programación', obtén los resultados de estas expresiones:

palabra1+palabra2	palabra1+'---'+palabra2	palabra1[3]
palabra1[0]+palabra2[2]	palabra1[:4]	palabra2[-4]
palabra2[6:]	palabra1[:]	list(palabra1)