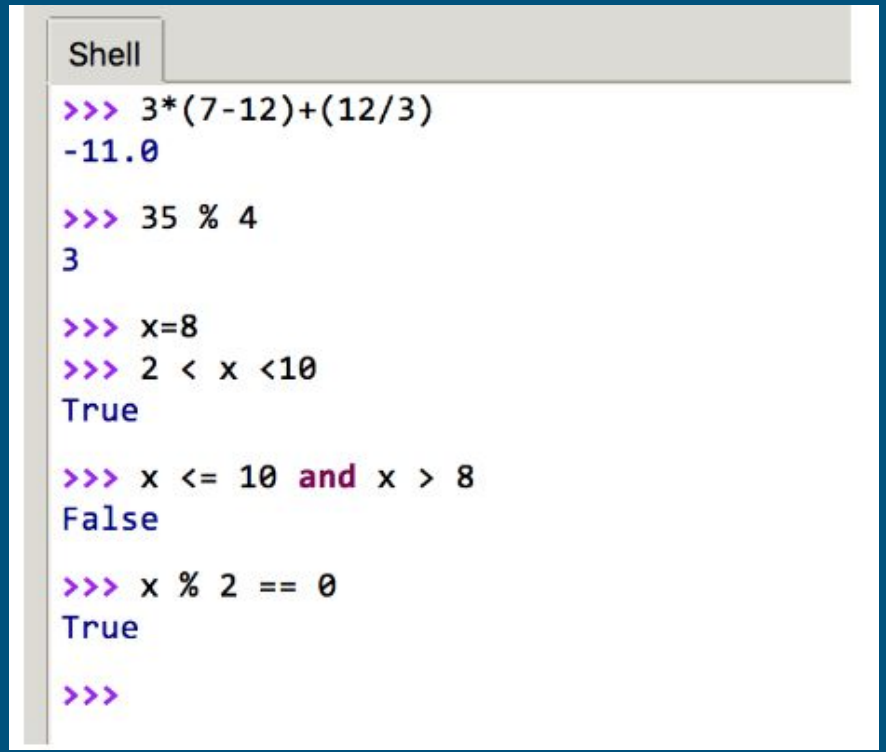
Python para principiantes

Capítulo 2: Tipos de datos. Operadores. Variables


Shell
>>> 3*(7-12)+(12/3)
-11.0
>>> 35 % 4
3
>>> x=8
>>> 2 < x <10
True
>>> x <= 10 and x > 8
False
>>> x % 2 == 0
True
>>>

Como una calculadora rápida y precisa

```
>>> 3 + 4 * (8 - 2)
27
>>> 45 / 7
6.428571428571429
>>> 45 // 7
6
>>> 7**2
49
>>> 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.

Como una calculadora rápida y precisa

```
>>> “Hola”

'Hola'

>>> “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).

>>> 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.

Como una calculadora rápida y precisa

```
>>> 8 > 3
True
>>> 5 < 2
False
>>> 6 >= 4 and 8 > 7
True
>>> 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.

De los ejemplos...

- Comprender cómo es la información que manipula un ordenador y qué tipo de operaciones se puede hacer con ellas.
- Hemos usado diferentes tipos de datos: numéricos (enteros y decimales), textos (strings) y datos lógicos (booleanos).
- 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.
- 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**

.

Operadores Matemáticos y de Textos

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

Operadores booleanos

Operador	Tipo	Operación que realizan
<	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

Operadores de asignación

Operador	Tipo	Operación que realizan
=	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

Operadores

Nota:

Dentro de los operadores de comparación, Python también permite operadores ternarios:

expresiones así: **1 <= x <= 10** es equivalente a: x>=1 and x<=10

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

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

Variable

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

Variable

El símbolo **=** debe entenderse como una **asignación**

La expresión: **x = x +1** 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.

Funciones de E/S

Instrucciones básicas en la elaboración de cualquier programa, son las **funciones de entrada y salida de información.**

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 función print se utiliza para mostrar información por pantalla y cuya sintaxis es:

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

Funciones de E/S

```
>>> 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.

Funciones de E/S

La función input es 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.

Desde la versión 3.x de Python la función input devuelve una cadena de texto, 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.

Funciones de E/S

```
>>> n = input('Escribe un número: ')
      Escribe un número: 12
>>> n
'12'
>>> v = int(n)
>>> 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

Funciones de E/S

```
>>> 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.

Es necesario ser cuidadoso al realizar operaciones con valores de diferentes tipos de datos.

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'

Funciones de E/S

```
>>> 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

Listas

Datos más elaborados

Son una colección de datos que resultan de gran importancia y ahorran muchísimo trabajo en la realización de un programa

```
>>> lista = ['Arturo', 11, 'Julio', 7.5, True]
>>> lista[0]
'Arturo'
>>> lista[4]
True
>>> 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.

Listas: modificar

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

Listas: Seleccionar

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

Cadenas de texto

```
>>> nombre = 'Arturo'
>>> nombre[0]
'A'
>>> nombre[1:4]
'rtu'
>>> nombre[:4]
'Artu'
>>> 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 y cunado indicamos [n:], le decimos que llegue hasta el final.

Listas

```
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.

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.