Introducción al Python

> Página extraída de Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Palabras reservadas

Son palabras reservadas que tienen un significado especial para el compilador y que no podemos usar para poner nombres a variables o funciones. Todas las palabras, excepto True, False y None se escriben en minúsculas. A continuación se da un listado de todas las palabras reservadas o keywords:

> False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

El listado al principio nos puede resultar abrumador, pero imaginemos un lenguaje con tan solo esas palabras y entenderemos que no resultará tan complejo familiarizarse, al menos con las mas usuales.

Identificadores

Los identificadores son los nombres que se dan a variables, clases, métodos, etc. No podemos usar palabras reservadas para estos nombres.

Algunas reglas que nos pueden resultar útiles para nombrar identificadores son:

- Los identificadores son sensibles a mayúsculas y minúsculas
- Los identificadores no pueden ser palabras reservadas
- Los espacios en blanco no están permitidos
- Un identificador puede ser una secuencia de letras y números. Siempre debe empezar por una letra o por el símbolo de subrayado "_".
- El primer carácter de un identificador no puede ser un número.
- No podemos utilizar caracteres especiales como la ñ, ¡, ¿ o letras con acentos.
- No podemos utilizar los símbolos como !, @, #, $, etc.

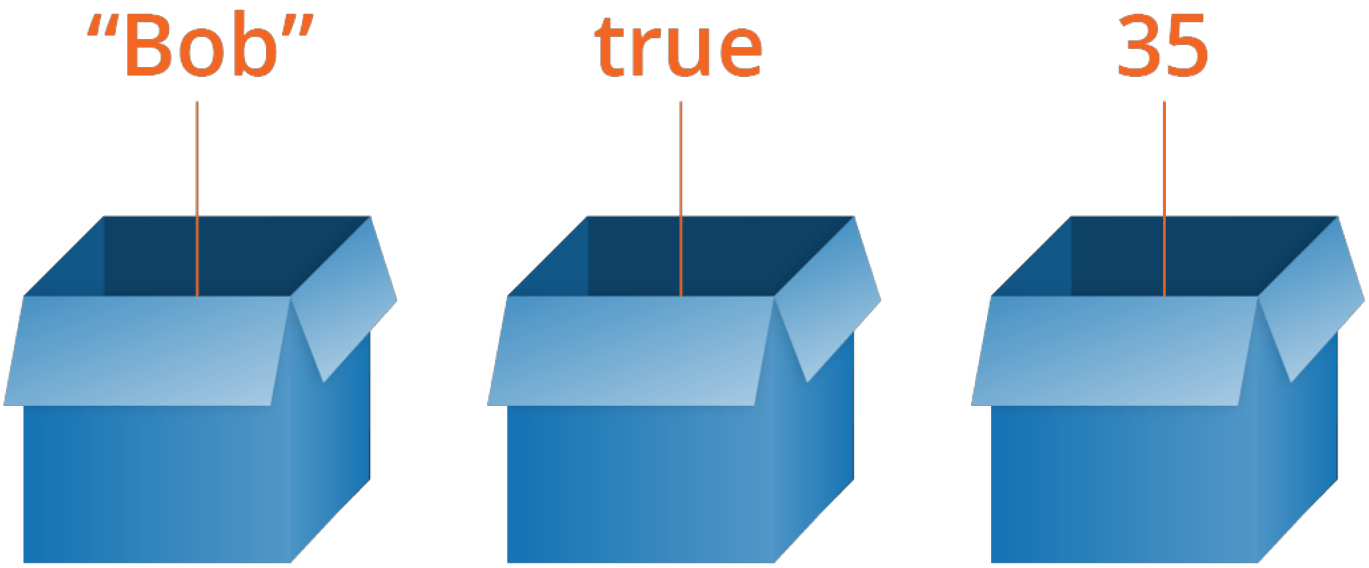
Nos va a resultar muy útil recordar lo siguiente:

- Python es un lenguaje que distingue entre mayúsculas y minúsculas. Esto significa que **Variable** y **variable** no son lo mismo
- Damos siempre a los identificadores un nombre que tenga sentido. Aunque que **c = 10** es un perfectamente válido, escribir **contador = 10** tendría más sentido, y sería más fácil averiguar lo que representa cuando miremo el código pasado un tiempo.
- Las palabras múltiples se pueden separar usando un guión bajo, como por ejemplo **esto_es_un_nombre_de_variable_muy_largo.**

Variables, constantes y literales

Variables

En programación, una variable es un nombre que se utiliza para referirse a una posición de memoria donde se almacena un valor. De forma más abstracta, puede considerarse como una caja que almacena un valor. El nombre de la caja es el nombre de la variable y el contenido su valor. Todas las variables constan de tres partes: un nombre, un tipo de dato y un valor. En la figura siguiente vemos tres variables de distintos tipos:



Licencia CC-BY-SA fuente

La variable `name` contiene la cadena `Bob`, la variable `winner` es cierta y la variable `score` contiene el valor `35`.

Python no dispone de ningún comando para declarar variables. Una variable se crea cuando se le asigna valor por primera vez. La técnica de declarar variables es poner un nombre seguido del signo de asignación (=) y el valor asignado a la variable. En la declaración es importante tener claro que se distinguen mayúsculas de minúsculas y que no están permitidos los caracteres especiales.

En Python no se declara de forma explicita el tipo de la variable pues se trata de un lenguaje inferido. Las variables incluso pueden cambiar de tipo desde el que se establece al asignarle valor

la primera vez. Es decir, si declaro `valor = 5` inicialmente la variable será de tipo entero (int), pero si en el programa se realizan operaciones que al final hacen que `valor = 1.33` ahora valor es de tipo float. Automáticamente sabe que `valor` es un número entero y declara la variable `valor` como un `int`.

Aunque no es necesario si es posible especificar el tipo de dato de una variable, haciendo:

```
x = str(22) # x será la cadena '22'
y = int(22) # y será el entero 22
z = float(22) # z será el número de coma flotante 22.0
```

Algunas reglas para nombrar variables que podemos tener en cuenta son:

> - Los nombres pueden tener una combinación de letras minúsculas o mayúsculas o números o el símbolo de subrayado "_".
> - Crear nombres que tengan sentido, aunque sean largos.
> - Si usamos varias palabras para definir el nombre, estas las separamos por "_"
> - Python es sensible a mayúsculas y minúsculas.
> - Hay que evitar palabras reservadas en nombres de variables.

Constantes

Una constante no es mas que un tipo especial de variable cuyo valor no puede modificarse.

En Python, las constantes suelen declararse y asignarse en un **módulo** (un nuevo archivo que contiene variables, funciones, etc y que se importa al archivo principal).

Veamos cómo declaramos constantes en un archivo separado y lo usamos en el archivo principal,

Creamos un archivo que nombramos constantes.py y que contendrá:

```
PI = 3.141592
FUERZA_GRAVEDAD = 9.82
```

Creamos el archivo principal main.py, que contendrá:

```
import constantes


print(constantes.PI)
print(constantes.FUERZA_GRAVEDAD)
```

En el ejemplo creamos el archivo de módulo constantes.py y asignamos el valor constante a PI y FUERZA_GRAVEDAD.

Después, creamos el archivo main.py e importamos el módulo constantes. Finalmente, imprimimos el valor de cada constante.

La convención es nombrarlas en mayúsculas para distinguirlas de las variables.

Literales

Numéricos

Los literales son representaciones de valores fijos en un programa. Pueden ser números, caracteres, cadenas, etc. Por ejemplo, "¡Hola, mundo!", 12, 23.0, "C", etc.

Los literales numéricos son inmutables (no pueden cambiar) y pueden pertenecer a uno de los tres tipos de datos numéricos posibles: Entero, Coma flotante y Complejo. Los tipos son:

- **Decimal**. Números regulares. Por ejemplo: 5, 22, -40
- **Binario**. Deben comenzar por 0b. Por ejemplo: 0b110, 0b11
- **Octal**. Deben empezar con 0o. Por ejemplo: 0o13, 0o7
- **Hexadedicmal**. Deben empezar con 0x. Por ejemplo 0x13, 0xFF
- **Coma flotante**. Contienen el punto decimal. Por ejemplo 10.2, 3.14
- **Complejo**. Tienen la forma `a + bj`. Por ejemplo: 3 - 2j, -4 + j
- **Booleanos**

Solamente hay dos literales booleanos `True` y `False`

Cadenas de caracteres

Los literales de caracteres son caracteres unicode encerrados entre comillas, por ejemplo `S`. Los literales cadenas de caracteres son cadenas de csaracteres encerradas entre comillas, por ejemplo `Python es divertido`.

Especiales

En Python existe un literal especial, `None`. Podemos usarlo, por ejemplo, para especificar una variable nula, por ejemplo:

```
var = None
print(var)
# El resultado será: None
```

Tipos de datos en Python

En Python, al igual que en programación en general, los tipos de datos especifican el tipo de datos que puede almacenarse en una variable.

Numéricos

Contienen valores numéricos y sabemos que:

> - Los números enteros son de tipo int
> - Los fraccionarios son de tipo float
> - La división (/) siempre devuelve un número en coma flotante
> - Para obtener la parte entera de una división se usa el operador //
> - Para calcular el resto de una división se usa el operador %
> - Para calcular potencias podemos usar el operador **
> - Los paréntesis se pueden usar para agrupar expresiones
> - El signo igual (=) se utiliza para asignar un valor (números, booleanos, cadenas, ...) a una variable
> - El tipo de la variable será el del dato asignado, no se declara el tipo de la variable al crearla
> - Por convención el nombre comienza en minúscula y si son varias palabras se unen por guión bajo

Los tipos básicos de datos son:

> - **`int`**: números enteros con signo sin límite de tamaño, ejemplo: entero = 5
> - **`float`**: números reales, decimales o de coma flotante con precisión de hasta 15 decimales, ejemplo: real = 5.6
> - **`complex`**: números complejos, por ejemplo 5.5 - 5j
> - Para averiguar el tipo de dato usamos la función `type()`.

Podemos realizar conversión de tipos así:

- A entero int(variable)
- A real float(variable)

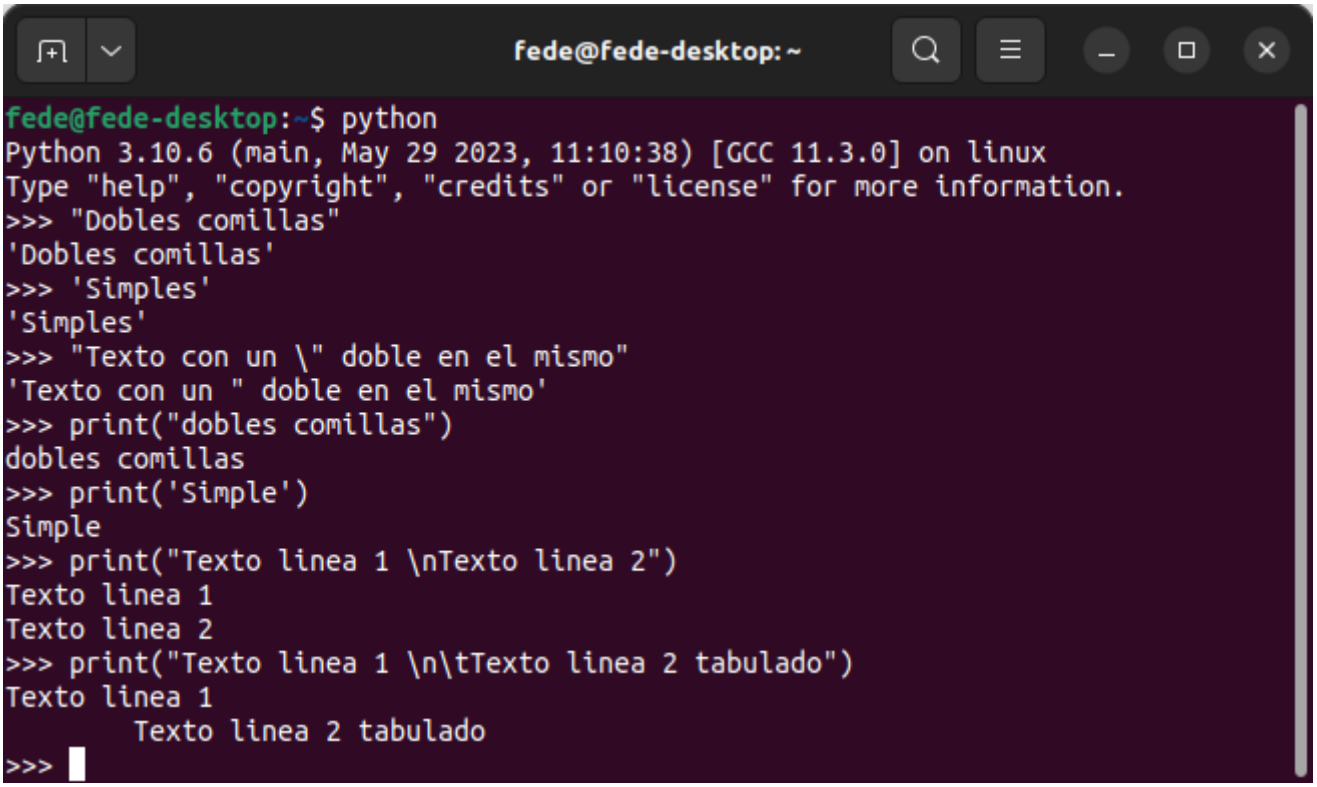
Cadenas

Contienen secuencias de caracteres. Una cadena es una secuencia de caracteres representada entre comillas simples o dobles.

Las cadenas pueden estar encerradas entre comillas simples ('...') o dobles ("...") con el mismo resultado. Podemos usar para incluir comillas en una cadena.

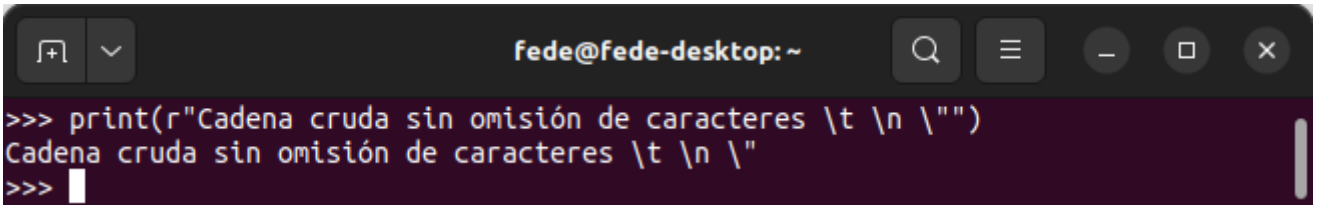
La función print() devuelve la cadena que encierra entre los paréntesis, omitiendo las comillas que la encierran.

En la imagen siguiente se ven varios ejemplos con cadenas utilizando como editor el IDLE que por defecto se instala con Python y que se abre desde una terminal simplemente invocando a Python.



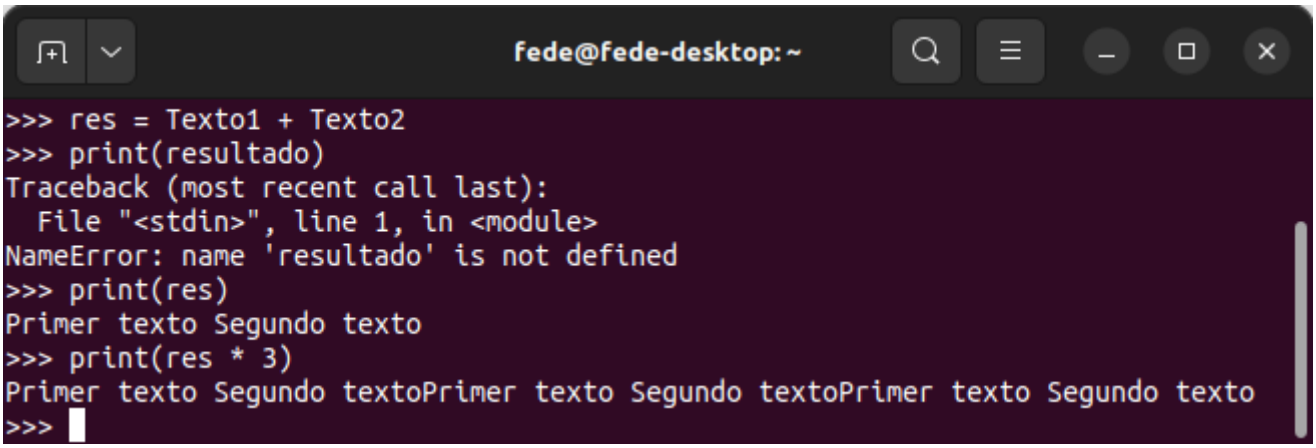
Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Una cadena raw (cruda) se interpreta tal como se escribe, es decir, se omiten los caracteres especiales expresados con . Las cadenas raw se escriben entrecomilladas y van precedidas del carácter 'r'. En la imagen vemos un ejemplo.



Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Es posible aplicar la operación de multiplicar a textos haciendo que estos se repitan. En la imagen siguientes vemos ejemplos de concatenación y multiplicación, así como un error cometido.



Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Secuencias

Contienen colecciones de datos, como las listas, las tuplas, las colecciones de datos (set) o los diccionarios.

Una lista es una colección ordenada de elementos similares o de distinto tipo separados por comas y encerrados entre corchetes [ ].

Tupla es una secuencia ordenada de elementos, igual que una lista. La única diferencia es que las tuplas son inmutables. Una vez creadas, las tuplas no pueden modificarse. En Python, se utilizan los paréntesis () para almacenar los elementos de una tupla.

Las colecciones de datos son un conjunto desordenada de elementos únicos. Una colección de datos se define por valores separados por comas dentro de llaves { }.

Un diccionario es una colección ordenada de elementos. Almacena los elementos como pares clave/valor. Siendo las claves identificadores únicos que se asocian a cada valor.

Estudiaremos estos últimos tipos mas extensamente cuando los necesitemos.

Los datos de tipo booleano solamente pueden contener `True` o `False`.

Dado que en programación Python todo es un objeto, los tipos de datos son en realidad clases y las variables son instancias(objeto) de estas clases.

Comentarios en Python

- Una sola linea : Escribiendo el símbolo almohadilla (#) delante del comentario.
- Multilinea: Escribiendo triple comillas dobles (“””) al principio y al final del comentario.

En los comentarios, pueden incluirse palabras que nos ayuden a identificar además, el subtipo de comentario:

```
# TODO esto es algo por hacer
# FIXME (arreglarme) esto es algo que debe corregirse
# XXX esto también, es algo que debe corregirse
```

Identation o sangría en Python¶

La sangría se refiere a los espacios al comienzo de una línea de código.

Mientras que en otros lenguajes de programación la sangría en el código es solo para facilitar la lectura, la sangría en Python es muy importante ya que se usa para indicar un bloque de código.

```
if 5 > 2:
    print("Cinco es mayor que 2")
```

Lo siguiente sería un error de sintaxis.

```
if 5 > 2:
print("Cinco es mayor que 2")
```

El número de espacios de la identation puede ser cualquiera siempre que al menos sea un espacio. **Siempre** hay que usar el mismo número de espacios en el mismo bloque de código.

Operadores en Python

Los operadores son símbolos especiales que realizan operaciones con variables y valores.

A continuación tenemos una lista de los diferentes tipos de operadores de Python:

- Operadores aritméticos
- Operadores de asignación
- Operadores de Comparación
- Operadores Lógicos
- Operadores Bitwise
- Operadores especiales

Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas como sumas, restas, multiplicaciones, etc.

Operador	Descripción	Ejemplo
+	Suma o concatenación en textos	`5+3=8`, `"Hola" + "Mundo" = "Hola Mundo`
-	Diferencia	`6-3=3`
*	Multiplicación	`3*3=9`
/	División	`6/2=3`
//	Parte entera de un cociente	`10//3=3`
%	Resto de un cociente	`10%3=1`
**	Potenciación	`5**2=25`

Operadores de asignación¶

Los operadores de asignación se utilizan para asignar valores a variables.

Operador	Descripción	Ejemplo
=	Asignación	`x=4`, `a = a + 1`
+=	Suma y asignación	`x+=1` equivale a `x = x + 1`
-=	Diferencia y asignación	`x-=1` equivale a `x = x - 1`
*=	Multiplicación y asignación	`x*=3` equivale a `x = x * 3`
/=	División y asignación	`x/=3` equivale a `x = x / 3`
%=	Asignación de restos	`x%=3` equivale a `x = x % 3`
**=	Asignación de exponentes	`x**=3` equivale a `x = x ** 3`

Operadores de Comparación

Los operadores de comparación comparan dos valores/variables y devuelven un resultado booleano: Verdadero o Falso `True` o `False`.

Operador	Descripción	Ejemplo
==	Igual a	`2==3` retorna `False`
!=	Distinto de	`2!=3` retorna `True`
<	Menor que	`2<3` retorna `True`
>	Mayor que	`2>3` retorna `False`
<=	Menor o igual que	`2<=3` retorna `True`
>=	Mayor o igual que	`2>=3` retorna `False`

Operadores Lógicos

Los operadores lógicos se utilizan para comprobar si una expresión es Verdadera o Falsa. Se utilizan en la toma de decisiones.

Operador	Descripción	Ejemplo
and	AND lógica	`a and b #True si a y b son ciertos`
or	OR lógica	`a or b #True si a o b son ciertos`
not	NOT lógica	`not a #True si el operador a es falso`

En la figura siguiente vemos un ejemplo con lo que devuelve en cada caso.

Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Operadores Bitwise¶

Los operadores bit a bit o bitwise actúan sobre los operandos como si fueran cadenas de dígitos binarios. Operan bit a bit, de ahí su nombre.

Operador	Descripción	Ejemplo
&	AND bit a bit	`5&6 # 101 & 110 = 110 = 4`
|	OR bit a bit	`5 | 6 # 101 | 110 = 111 = 7`
~	NOT bit a bit	`~3 # ~011 = 100 = -4`

Operador	Descripción	Ejemplo
^	XOR bit a bit	`5^3 # 101^011 = 110 = 6`
<<	Desplazamiento izquierda	`4<<1 # 100 << 1 = 1000 = 8`
>>	Desplazamiento derecha	`4 >> 1 # 100 >> 1 = 010 = 2`

Operadores especiales

El lenguaje Python ofrece algunos tipos especiales de operadores como el operador de identidad (`identity`) y el operador de pertenencia (`membership`).

- **Operadores `identity`**

En Python, `is` e `is not` se utilizan para comprobar si dos valores se encuentran en la misma parte de la memoria. Dos variables que son iguales no implica que sean idénticas. Algunos ejemplos aclaran mejor lo dicho.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'

print(x1 is not y1)  # False

print(x2 is y2)  # True
```

Vemos que x1 e y1 son enteros con los mismos valores, por lo que son iguales e idénticos. Lo mismo ocurre con x2 e y2 (cadenas).

- **Operadores `membership`**

En Python, `in` y `not in` son los operadores de pertenencia. Se utilizan para comprobar si un valor o variable se encuentra en una secuencia (cadena, lista, tupla, conjunto y diccionario).

En un diccionario sólo podemos comprobar la presencia de la clave, no del valor.

Revision #1
Created 25 September 2024 11:49:24 by Javier Quintana
Updated 25 September 2024 11:49:54 by Javier Quintana