`Python` básico

Variables

Variable. Consta de un espacio en el sistema de almacenaje (memoria principal de un ordenador) y un nombre simbólico (un identificador) que está asociado a dicho espacio.

Dicho de otro modo, una variable es la relación que hay entre un nombre y un objeto ubicado en algún lugar de la memoria del ordenador.

En `Python` asignamos valores a las variables siguiendo el siguiente formato:
`nombre_variable = valor`

```
x = 1
x
```

```
1
```

```
y = "hola"
y
```

```
'hola'
```

Restricciones sobre los nombres de las variables

- No pueden empezar ni contener carácteres especiales
- No pueden empezar por números
- No pueden ser llamadas igual que las palabras claves reservadas en Python
- No pueden contener espacios

Observación. Conviene que al darle nombre a una variable, éste tenga sentido en cuanto al dato que guarde, para que así resulte mucho más fácil la comprensión por parte de quien lea el código.

Observación. A día de hoy, si los nombres de las variables están compuestos por múltiples palabras, hay 4 formas de escribir dichos nombres:

- camelCase: `nombreMascota`
- PascalCase: `NombreMascota`
- snake_case: `nombre_mascota`
- kebab-case: `nombre-mascota`

Palabras clave en `Python`

Las palabras clave en `Python` son las que se muestran con el siguiente chunk de código.

```
import keyword
keyword.kwlist
```

```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 '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']
```

Declarando múltiples variables en una sola línea

Se hace del siguiente modo:

[ ] ↳ 3 celdas ocultas

Operando con una variable numérica

Una vez hemos guardado un valor numérico en una variable, podemos operar con él:

```
x = 3
```

```
x + 1
```

```
4
```

Incluso podemos guardar ese valor en una nueva variable

```
y = x + 1
y
```

```
4
```

O bien, podemos sobreescribir la variable que teníamos originalmente

```
x = x + 1
x
```

```
4
```

En `Python`, si queremos sobreescribir una variable numérica sumándole a esta una cantidad, lo podemos hacer del siguiente modo:

```
x = 7
x
```

```
7
```

```
x += 2
x
```

```
9
```

Observación. Al igual que existe `+=`, también tenemos `-=`, `*=`, `/=`, `//=`, `%=` y `**=`, que son el equivalente a `+=` con el resto de operaciones aritméticas existentes en `Python` y que trataremos en el siguiente tema.

Observación. Todos los tipos de variable en `Python` los iremos viendo en profundidad a lo largo de este curso.

Comentarios

Dado un bloque de código, a veces puede ser útil explicar qué hace o en qué consiste, o bien hacer que una línea no se ejecute por algún motivo, pero que siga presente en dicho código. Aquí entran en juego los comentarios, que son parte del código, pero no se ejecutan.

Si queremos hacer un comentario en una línea, utilizamos `#`

```
# Vamos a definir la variable x y asignarle el valor 1
x = 1 # Aquí está nuestra variable x
# Esto es otro comentario
# Y ninguno de los comentarios se ejecutará
x
```

```
1
```

Observación. Los comentarios, como ya se ha dicho, pueden ser muy útiles a la hora de entender nuestro código. Sin embargo, no conviene abusar de estos, como se ha hecho en el ejemplo anterior, pues el código pasa de ser claro a ser excesivamente largo y confuso.

La función `import`

Antes de explicar en qué consiste la función `import`, hay que definir los siguientes conceptos:

- **Algoritmo.** Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un problema.
- **Función.** Bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea.
- **Script.** Archivo diseñado para ser ejecutado. Puede contener funciones, programas, etc.
- **Módulo.** Script que contiene colecciones de funciones, definiciones y declaraciones de `Python`.

Las funciones de un módulo pueden ser importadas. Es aquí donde entra en juego la función `import`.

Por ejemplo, vamos a importar el módulo `math`, del cual hablaremos en futuras secciones de este curso.

De momento, lo que nos interesa saber acerca de este módulo es que es de utilidad a la hora de usar funciones matemáticas (definidas según los estándares de C).

```
import math
```

Con la línea de código anterior, hemos cargado el módulo de `math`, permitiéndonos así poder trabajar con las funciones que contiene, haciendo uso de la sintaxis `math.funcion()` o `math.variable`.

A la hora de usar funciones de un modulo, puede resultar tedioso tener que poner siempre el nombre del módulo previo a la función. Es por ello que la función `import` nos permite hacer lo siguiente:

```
import math as mt
```

Con la línea de código anterior, no solo hemos cargado el módulo de `math`, sino que a la hora de usar alguna de sus funiones, ahora podremos usar la sintaxis `mt.funcion()` o `mt.variable`. Es decir, en vez de tener que poner el nombre del módulo, podemos cambiarle el nombre o, como en este caso, usar la abreviatura `mt` (o la que quereamos utilizar), cosa que resulta de mucha utilidad para módulos con nombres muy largos.

Si por el contrario no queremos cargar todo el módulo, sino que simplemente queremos cargar una función o una variable, lo podemos hacer de la siguiente forma:

```
from math import pi
```

La línea de código anterior nos permite acceder a la variable `pi` del módulo `math` directamente haciendo uso de la sintaxis `pi`, en vez de tener que usar la sintaxis `math.pi`.

Al igual que podíamos modificar el nombre del módulo a la hora de llamarlo, también lo podemos hacer con las funciones y las variables.

```
from math import pi as numero_pi
```

De modo que ahora ya no solo podemos acceder directamente a la variable `pi` sin necesidad de indicar el módulo del que procede, sino que le hemos cambiado el nombre a `numero_pi`.

Si por el contrario quisiésemos cargar más de una función o variable, pero sin necesidad de cargar todo el módulo, lo podríamos hacer del siguiente modo:

```
from math import pi, log, exp
```

En este caso, la línea de código anterior nos permite acceder a la variable `pi` y las funciones `log()` y `exp()`, todas del módulo `math`, directamente haciendo uso de la sintaxis `pi`, `log()` y `exp()` en vez de tener que usar la sintaxis `math.pi`, `math.log()` o `math.exp()`, respectivamente.

Por último, si quisiésemos cargar todas las funciones del módulo `math` y evitar tener que usar la sintaxis `math.funcion()` o `math.variable`, entonces lo podríamos hacer con la

siguiente instrucción

```
from math import *
```

No obstante, esto no es para nada recomendable puesto que puede que dos o más funciones de diferentes módulos tengan el mismo nombre aunque lleven a cabo cosas distintas, lo cual supondría un grave problema, pues no tendríamos forma de diferenciar qué función queremos utilizar.