Python Programming

Módulo 1

Ámbito de las variables

Ámbito de las variables

Las variables en Python son locales por defecto. Esto quiere decir que las *variables definidas y utilizadas en el bloque de código de una función, sólo existen dentro de la misma*, y no interfieren con otras variables del resto del código.

A su vez, *las variables existentes fuera de una función, no son visibles dentro de la misma*.

En caso de que sea conveniente o necesario, *una variable local puede convertirse en una variable global declarándola explícitamente como tal* con la sentencia global.

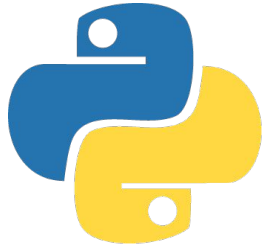
En Python NO es aconsejable hacer uso de variables globales, ya que disminuye la legibilidad de código (fundamental). Su uso puede producir efectos colaterales, produciendo alteraciones no previstas en una parte del programa y, por lo tanto, pueden afectar el uso del valor en otra zona. Entonces, concluimos que dificultan la forma del trabajo modular del código.

Ejemplo de uso de una **variable global**:

```
variable = 10

def funcion():
      global variable
      variable = 5

print(variable)
funcion()
print(variable)

out: 10
out: 5
```

La palabra reservada "`global`" es la que termina avisándole al interprete que esa variable es la variable que está fuera de la función. Si no se usa, el comportamiento de la variable dentro y fuera de la función es independiente.

Recordamos que es totalmente desaconsejable el uso de variables globales.



Paso por valor y por referencia

En muchos lenguajes de programación existen los conceptos de *paso por valor* y *por referencia* que aplican a la hora de cómo trata una función a los parámetros que se le pasan como entrada.

Si usamos un parámetro **pasado por valor,** se creará una copia local de la variable, lo que implica que cualquier modificación sobre la misma no tendrá efecto sobre la original.

Con una variable **pasada como referencia**, se actuará directamente sobre la variable pasada, por lo que las modificaciones afectarán a la variable original.

En Python las cosas son un poco distintas, y el comportamiento estará definido por el tipo de variable con la que estamos tratando.

Veamos un ejemplo de paso por valor.

```
def funcion(entrada):
    entrada = 0

dato = 5
funcion(dato)
print(dato)

out: 5
```

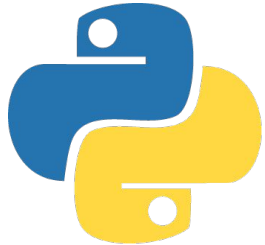
En el ejemplo anterior iniciamos “`dato`” con 5 y se lo pasamos a `funcion()`. Dentro de la función hacemos que la variable valga 0. Dado que Python trata a los `int` como *pasados por valor,* dentro de la función se crea una copia local de x, por lo que la variable original no es modificada.

No pasa lo mismo si por ejemplo “`dato`” es una lista. En este caso Python lo trata como si estuviese *pasada por referencia,* lo que hace que se modifique la variable original. La variable original “dato” ha sido modificada.

```
def funcion(entrada):
    entrada.append(40)

dato = [10, 20, 30]
funcion(dato)
print(dato)

out: [10, 20, 30, 40]
```

El ejemplo anterior nos podría llevar a pensar que si en vez de añadir un elemento a “`dato`”, hacemos `entrada=[]`, estaríamos destruyendo la lista original. Sin embargo, esto no sucede.

```
def funcion(entrada):
    entrada = []

x = [10, 20, 30]
funcion(x)
print(x)

out: [10, 20, 30]
```



¡Sigamos trabajando!