Funciones de usuario II

Contenidos

Return

al indice

Continuamos con las funciones definidas por el usuario. Recuerda que la sintaxis que vimos en el anterior notebook era:

```
def nombre_funcion(entrada):
    operaciones varias
    return output
```

Hicimos un ejemplo de función que nos devolvía un resultado (output) que podíamos almacenar en una variable... pero las funciones **tampoco tienen por qué llevar un** `return`. No siempre es necesario un output. En tal caso, devuelve `None`

In [1]:
```
from datetime import datetime

def que_hora_es():
    now = datetime.now().time()
    print(now)
```

In [2]:
```
hora = que_hora_es()
print(hora, type(hora))
```

```
10:31:37.174450
None <class 'NoneType'>
```

In [3]:
```
print(now)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 print(now)

NameError: name 'now' is not defined
```

También **puedes poner varias salidas en el return**. En ese caso, si no se especifica nada más la salida de la función será de tipo "tupla" (tuple). Pero generalmente los agrupamos en una colección.

In [5]:
```
def conversor_km_millas(distancia):
    millas = 0.62 * distancia
    metros = 1000 * distancia
    return round(millas,1), millas, metros
```

In [8]:
```
salida = conversor_km_millas(2221)
print(salida)
print(type(salida))
```

```
(1377.0, 1377.02, 2221000)
<class 'tuple'>
```

Tipos de datos de los argumentos

al indice

Lo que quieras: numeros, texto, listas, tuplas, diccionarios, objetos de clases que hayas definido...

In [13]:
```
def recibe_mix(tupla, lista, diccionario):
    print("tupla contiene:")
    print(tupla)
    print("y es de tipo:", type(tupla))
    print(lista)
    print("y es de tipo:", type(lista))
    print(diccionario)
    print("y es de tipo:", type(diccionario))
    return [type(tupla), type(lista), type(diccionario)]
```

In [14]:
```
recibe_mix((12,34,23), ["Esto", "es", "una", "lista"], {"key1": "valor1
```

```
tupla contiene:
(12, 34, 23)
y es de tipo: <class 'tuple'>
['Esto', 'es', 'una', 'lista']
y es de tipo: <class 'list'>
{'key1': 'valor1'}
y es de tipo: <class 'dict'>
```

Out[14]: [tuple, list, dict]

In [15]:
```
recibe_mix(1, "Hola", True)
```

```
tupla contiene:
1
y es de tipo: <class 'int'>
Hola
y es de tipo: <class 'str'>
True
y es de tipo: <class 'bool'>
```

Out[15]: [int, str, bool]

Errores típicos con funciones

al indice

ERRORES variables de la función

In [18]:
```
# Todo lo que declaremos dentro de la función se crea UNICAMENTE para l
# Fuera de la misma, esas variables no existen

def km2millas(dist):
    millas = dist * 0.62
    return round(millas,2)
print(km2millas(200))
print(millas)
```

```
124.0
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call la
st)
Cell In[18], line 8
      6     return round(millas,2)
      7 print(km2millas(200))
----> 8 print(millas)

NameError: name 'millas' is not defined
```

Se crea un namespace interno dentro de las funciones, es decir, que lo que declaremos dentro, se queda dentro. No lo podremos usar fuera. Además, ten en cuenta que todo lo que introduzcamos dentro de flujos de control (`if/else` , bucles...), nos vale para el resto de la función

In [19]:
```
def numero_ifs(numero):
    if numero == 1:
        out = 1
    return out

numero_ifs(1)
```

Out[19]: 1

In [20]:
```
numero_ifs(2)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[20], line 1
----> 1 numero_ifs(2)

Cell In[19], line 4, in numero_ifs(numero)
      2 if numero == 1:
      3     out = 1
----> 4 return out

UnboundLocalError: local variable 'out' referenced before assignment
```

In [21]:
```
# Si no introducimos argumentos en una función que SI tiene argumentos,
millas = km2millas()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 2
      1 # Si no introducimos argumentos en una función que SI tiene argumentos, salta un error de este estilo
----> 2 millas = km2millas()

TypeError: km2millas() missing 1 required positional argument: 'dist'
```

Cuidado también con la sintaxis de línea. Después de dos puntos `:` , viene todo el bloque de código tabulado, de la función

In [22]:
```
def mala_funcion(otro_argumento):
print(otro_argumento)
retur [otro_argumento]
```

```
  Cell In[22], line 2
    print(otro_argumento)
    ^
IndentationError: expected an indented block
```

In [ ]: