# Curso de Desarrollo en Lenguaje Python para Inteligencia Artificial (Málaga)

M.374.001.001

## 19 de marzo 2020 09:30-13:30

Módulo 1 - Tema 6.1

Carmen Bartolomé

Gobierno de España · Ministerio de Industria, Comercio y Turismo · EOI Escuela de organización industrial

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

polo de contenidos digitales

Ayuntamiento de Málaga

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
Escuela de organización industrial

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

# Módulo 2: Aprendiendo Python. El core de Python

## Desarrollo en lenguaje Python

Año de realización: 2020

**PROFESORA**
Carmen Bartolomé Valentín-Gamazo

Escuela de organización industrial

www.eoi.es

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

# Tema 9-1

## Funciones

# Índice

1. Definición de funciones
2. Creación y llamada de funciones en Python
3. Pass
4. Variables locales y globales
5. Return
6. Envío de valores
7. Parámetros y argumentos
8. Tipos de argumentos
9. Argumentos por valor vs. argumentos por referencia
10. Errores con los argumentos
11. Estructuras con funciones
12. Argumentos indeterminados

# Funciones. Definición

Es un bloque de código que ejecuta una tarea determinada. Se define de forma unívoca para poder ser utilizado varias veces en un programa, al ser llamada la función.

Las principales ventajas son:

- No es necesario re-escribir código
- El programa queda más limpio, fácil de leer
- Es más sencillo encontrar errores

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

# Funciones en Python

## Cómo se crea: definición de una función

```
def nombre_de_la_función( ):
    instrucciones de la función
```

**Ejemplo:**

```
def contar_un_chiste( ):
    print("Iban dos y se cayó el del medio")
```

# Funciones en Python

## Cómo se usa: llamada a una función

*nombre_de_la_función( )*

**Ejemplo:**

contar_un_chiste( )

# Cosas a tener en cuenta

## Nombre de la función

- Nombres sencillos y descriptivos
- Sin espacios. Guiones bajos como separadores
- Todas las palabras en minúsculas

## Estructura

- Indentación de todo lo que vaya dentro de la función.

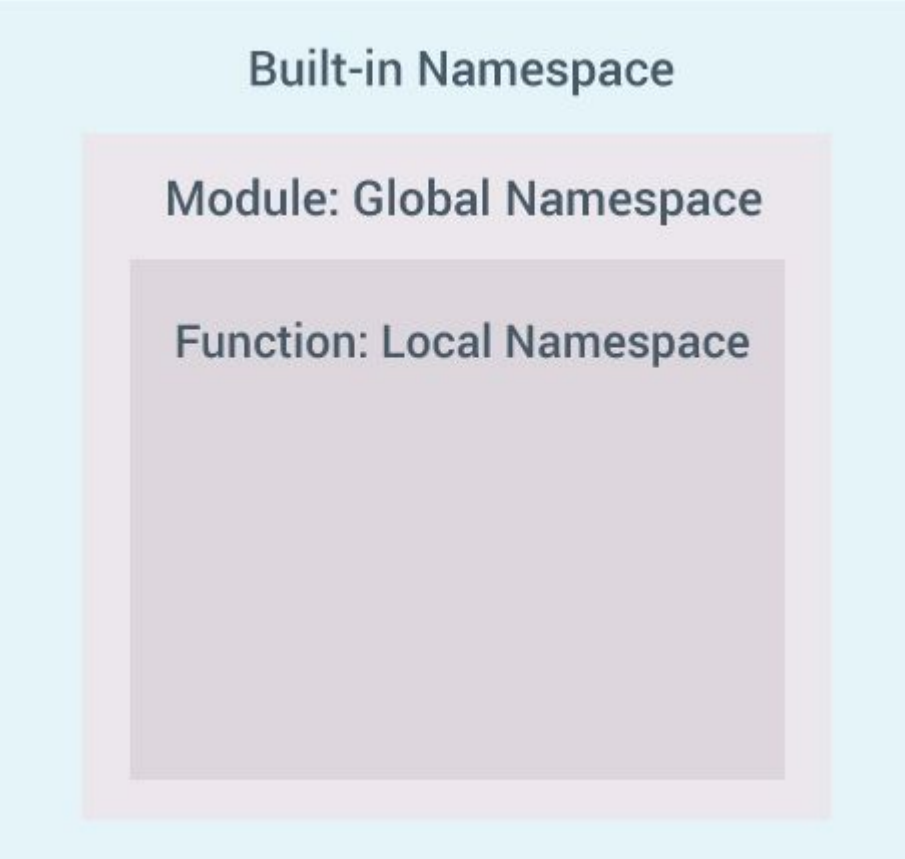
# Pass

- La instrucción **pass** es una operación nula. No hace nada.
- Se utiliza cuando es necesario poner una instrucción o se requiere sintácticamente, pero no se requiere de la ejecución de ningún código.
- Después de if, def, except, class,... debe ir algo de código, no puede haber una línea en blanco. Si no, obtenemos un error

# Funciones: variables locales y globales

- Una variable declarada dentro de una función no existe para el flujo general del programa.
- Una variable declarada fuera de la función, a su mismo nivel, sí es accesible para la función, además de para el resto del programa.
- Si una función necesita una variable global, ésta debe haber sido declarada antes de hacer la llamada a la función.
- No se podrá modificar una variable global dentro de la función. Para ello, hay que usar la instrucción global

# Namespaces



# Return

Utilizamos **return** para especificar el elemento o elementos que devuelve la función al programa general.

# Funciones. Envio de valores

Las funciones, análogamente a cómo devuelven valores, también interaccionan con el exterior admitiendo valores desde éste.

La forma más común de “enviar” valores a una función es a través de la definición de sus parámetros de entrada:

```
def saludo(nombre):
    print(f”Hola, soy {nombre}”)

saludo(Pepe)
```

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

# Parámetros y argumentos

Un parámetro es información que la función necesita para cumplir con su misión. Un argumento es el valor que se envía a la función.

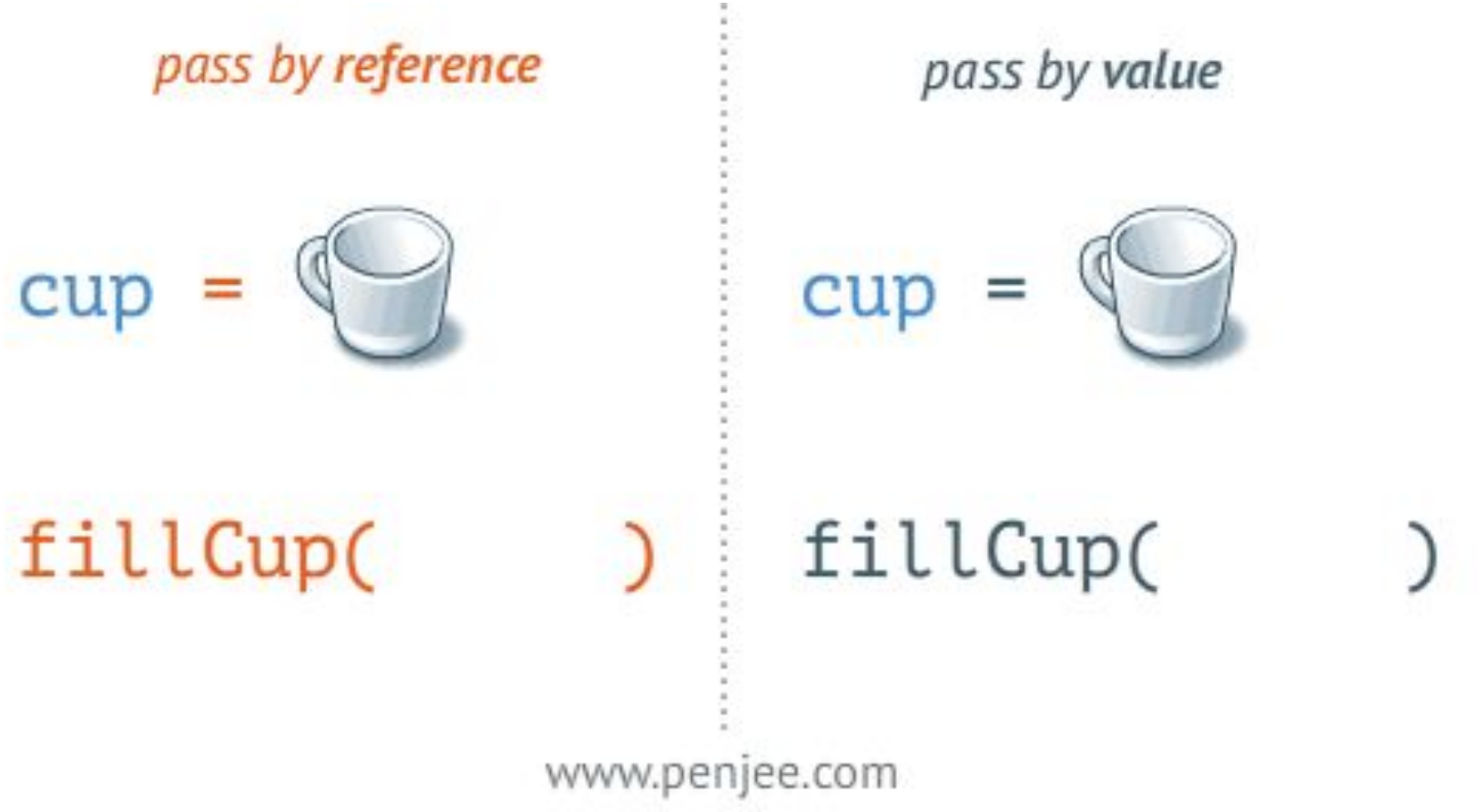
Habilitando un parámetro en la función, permitimos que la función pueda utilizar cualquier valor que se le pase a través del canal habilitado a través de ese parámetro.

En el ejemplo anterior, *nombre* es un **parámetro**, mientras que *"Pepe"* es un **argumento**.

# Tipos de argumentos

- **Argumentos por posición**: es necesario respetar el orden en el que están los parámetros para los valores se asignen correctamente. El primer valor se asignará al primer parámetro, el segundo valor al segundo parámetro y así sucesivamente.
- **Argumentos por palabra clave**: se vincula cada valor de argumento al correspondiente parámetro. En este caso, el orden ya no es relevante, pero se debe respetar el nombre de los parámetros declarados en la definición de la función.
- **Argumentos por defecto**: se asigna un valor por defecto a cada parámetro, que será el que se utilice a no ser que se envíe un valor diferente.

# Argumentos por valor vs. referencia



# Argumentos por valor vs. referencia

- Al enviar argumentos en forma de valores contenidos en una variable simple, en realidad, en la función se trabaja con una copia de dicho valor, sin modificar el valor originar (variable globlal)
- Las variables basadas en estructuras de datos, en las que los valores dependen de una referencia, como los índices en las listas, sí que son modificadas por la función.

# Errores con los argumentos

- Es relativamente común, al llamar a una función, dar más o menos argumentos de los que necesita.
- En ese caso, Python suele informar del tipo de error, la línea en la que se encuentra, e incluso, qué argumentos faltan.
- Por ésto, es importante dar nombres claros y descriptivos a los parámetros, para poder hacer una correcta llamada a la función a partir del mensaje de error, sin tener que ir a la definición de la función a ver qué parámetros se han definido.

```
def piso(habitaciones, precio):
  print(f"Este piso tiene {habitaciones} habitaciones y se vende por {precio} euros")

piso()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-4810c6c7a5fc> in <module>()
      2   print(f"Este piso tiene {habitaciones} habitaciones y se vende por {precio} euros")
      3
----> 4 piso()

TypeError: piso() missing 2 required positional arguments: 'habitaciones' and 'precio'
```

→ *notebook Tema9-3_Funciones*

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

# Utilizando estructuras con funciones

- Una función puede devolver estructuras complejas de datos, como una lista o un diccionario.
- Trabajar con estas estructuras de datos dentro de las funciones permite organizar mejor los parámetros: parámetros opcionales.
- Bucles while con una función. Break para salir.
- Pasar una lista a una función como argumento para hacer más eficiente el trabajo con listas.
- Los diccionarios también pueden ser argumentos de las funciones.

# Argumentos indeterminados

- Se define una función que puede ser llamada pasándole diferente cantidad de argumentos.
- Pueden ser argumentos por posición: *args
- Pueden ser argumentos por nombre: **kwargs
- El operador Splat * (asterisco) y sus particularidades: desempaqueta datos múltiples.
- La clave, en realidad es el operador splat. Podemos usar otras palabras para designar los argumentos indeterminados, pero siempre con el asterisco o doble asterisco delante.

**Siempre deben ir primero los *args y después los **kwargs**

# Funciones “void” y resultado “None”

- En función de lo que devuelve una función, tenemos funciones que devuelven elementos y funciones que ejecutan acciones pero no devuelven nada.
- Las primeras, conocidas como funciones fructíferas (fruitful), devuelven valores que, normalmente, asignaremos a variables
- Las segundas, llamadas “vacías” o “void”, como no devuelven nada predefinido, nos dan un valor especial de tipo None.

# V.I.P.

*"en Python todo es un objeto"*

*"las Funciones son ciudadanos de primera clase en Python"*

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

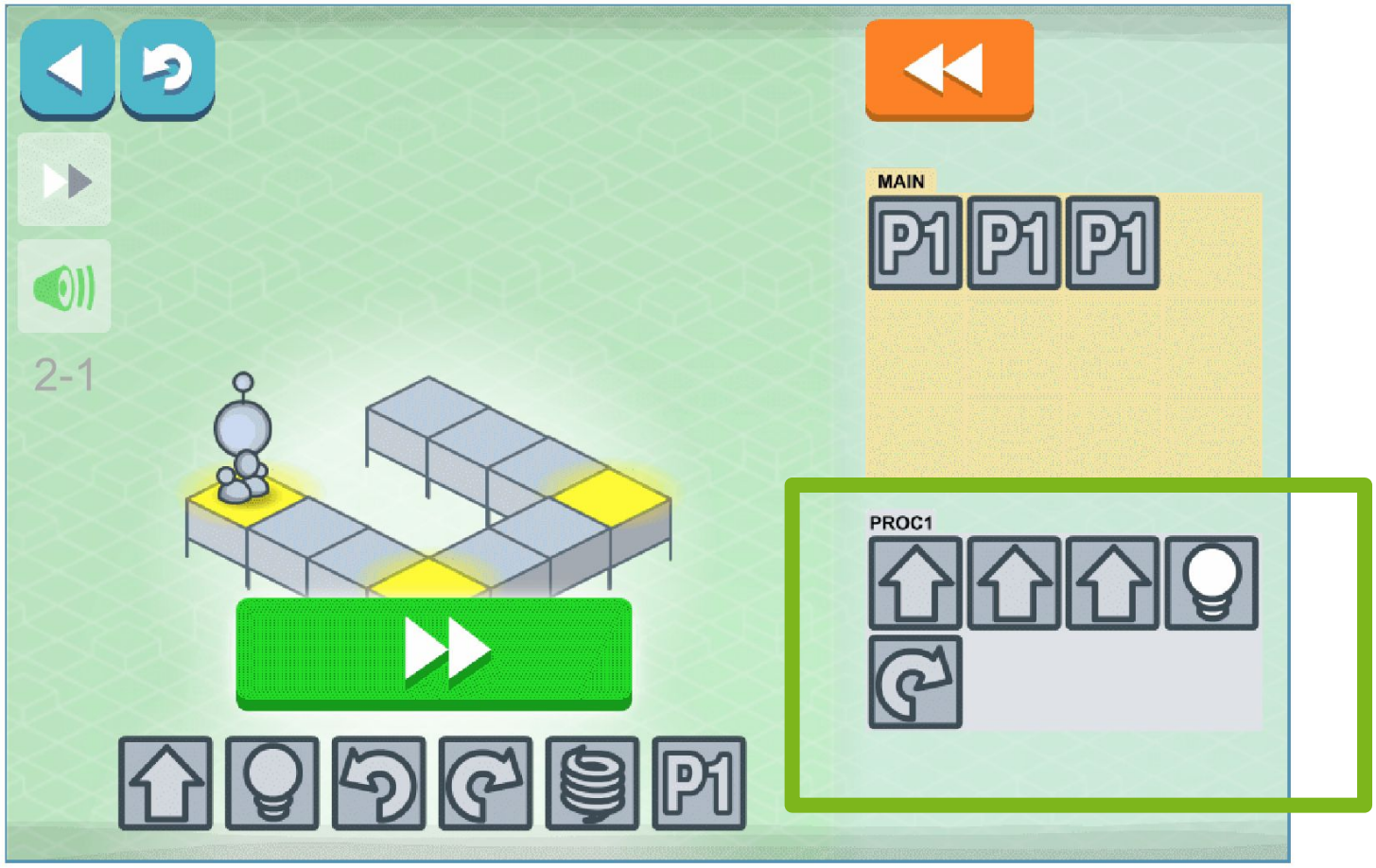
Escuela de organización industrial

# Funciones internas y de cierre

- Se puede definir una función dentro de otra función
- Es útil para evitar duplicidades de código en la definición de la función principal o cuando tenemos que modelizar tareas muy complejas
- Una función interna puede actuar como “cierre”
- Las funciones de “cierre” permiten cambiar y recordar los valores de variables que han sido creados fuera de éstas.

# Juguemos con el concepto

# En el Nivel 2, PROC1 es una función



carmenbvg@gmail.com