Curso de Python

5 – Funciones

Ramón Invarato Menéndez

Ricardo Moya García

Pseudo-código

```
def nombre_funcion(*args, **kwargs):
    print("Cuerpo de la función")
    return "Algo a devolver"
```

def función(parámetros)

Tabular
o
4 espacios

Cuerpo de la función que usa los parámetros

Pseudo-código

```
def mi_funcion(parametros...):
    print("Cuerpo de la función")
    return "Algo a devolver"
```

Funciones/Métodos

- Una función **es un fragmento de código** con un nombre asociado que realiza una serie de tareas y **devuelve un valor**
- **Una misma función se puede llamar múltiples veces**.
- En Python las funciones se **declaran** con la palabra reservada "**def**", seguida del **nombre de la función y sus parámetros entre paréntesis**
- Si la función **devuelve un valor** de forma explicita se utiliza la palabra reservada "**return**"
- Una función **se suele crear para evitar repetir código** (si hay que repetir código hay que pensar en encapsularlo dentro de una función)
- **Una función puede crearse para un solo uso** normalmente para **englobar código** bien definido (evita documentación innecesaria si la función hace algo concreto) o para **programar eventos** (Lambdas)
- "**return**", en el cuerpo de una función, **indica que "se ha terminado"** de procesar lo que se estuviera haciendo dentro de la función.
- Un cuerpo de **una única función puede tener un "return", varios "returns" o ninguno**.
- **Un "return" puede devolver un valor, varios valores o ninguno**.

https://jarroba.com/returns-en-python/

ReturnS

- Return que devuelve un **valor**

Código

```
def funcion():
    return "Algo a devolver"
```

- Return que devuelve varios valores en un Tupla

Código

```
def funcion():
    return "A", "C", 1, 2
```

- Return None

Código

```
def funcion():
    return None
```

- Return vacío (return **void**)

Código

```
def funcion():
    return
```

- Sin Return

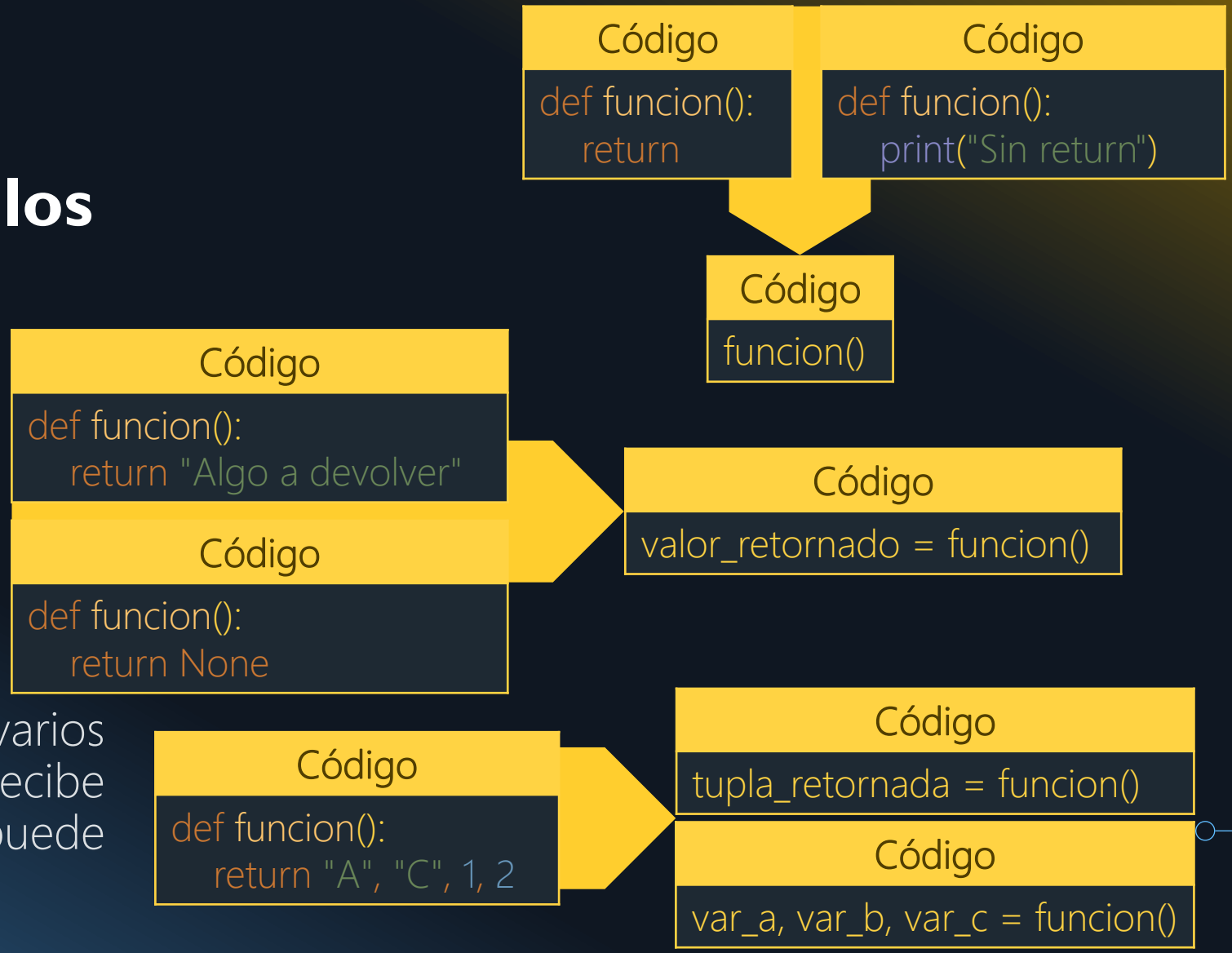
Código

```
def funcion():
    print("Sin return")
```

https://jarroba.com/returns-en-python/

Recoger valores de los ReturnS

- **Return** que devuelve un **valor** y **None:** en la llamada se recibe un **único valor**
- **Return** que devuelve varios **valores en un Tupla:** se recibe una **tupla** que se puede desempaquetar
- **Return** vacío y **Sin Return:** en la llamada **no se recibe nada**



Código

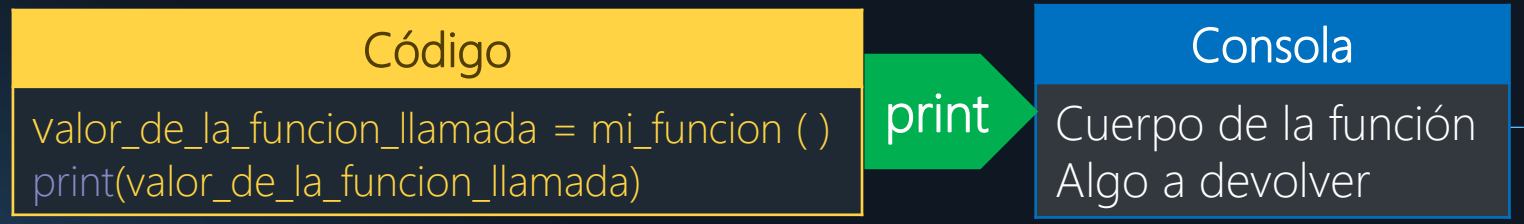
```
def mi_funcion ( ):
    print("Cuerpo de la función")
    return "Algo a devolver"
```

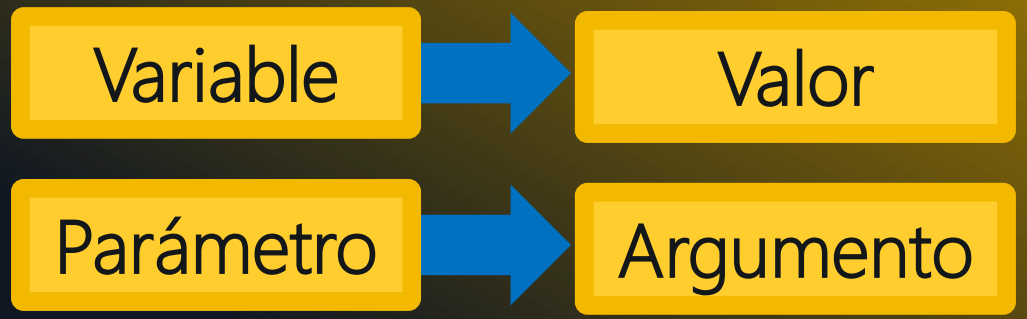
Llamar vs Referenciar

- **Llamar** (call): Después del nombre de la función hay escribir los paréntesis tenga o no argumentos a pasar, esto ejecuta el cuerpo de la función con los parámetros.

- **Referenciar**: Asignar la función a una variable sin paréntesis, esto no ejecuta la función (sirve para pasarla a "otro lado"; los parámetros que se pasen a ese "otro lado" tienen que ser los que pida la función).

- **Llamar y asociar:** Llama a la función y guarda el valor retornado en una variable.





Argumentos vs Parámetros

```
def  funcion(parametros...):
    # Cuerpo de la función
    return ...

variable = funcion(argumentos...):
```

Código

```
def mi_funcion(parametro):
    print("'parametro' tiene el argumento: {}".format(parametro))

variable_con_el_argumento = "Mi argumento"
mi_funcion(variable_con_el_argumento)
```

print

Consola

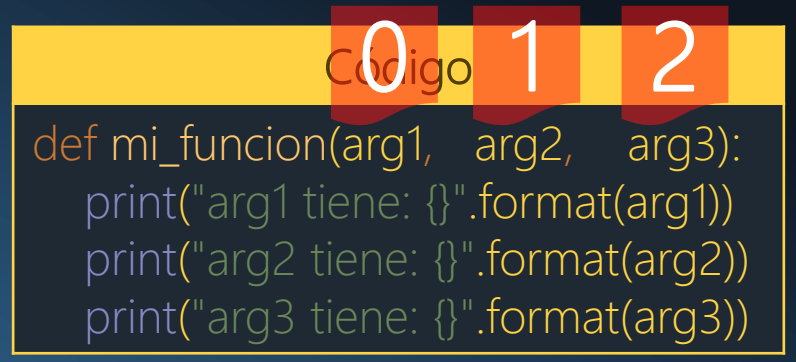
```
'parametro' tiene el argumento: Mi argumento
```

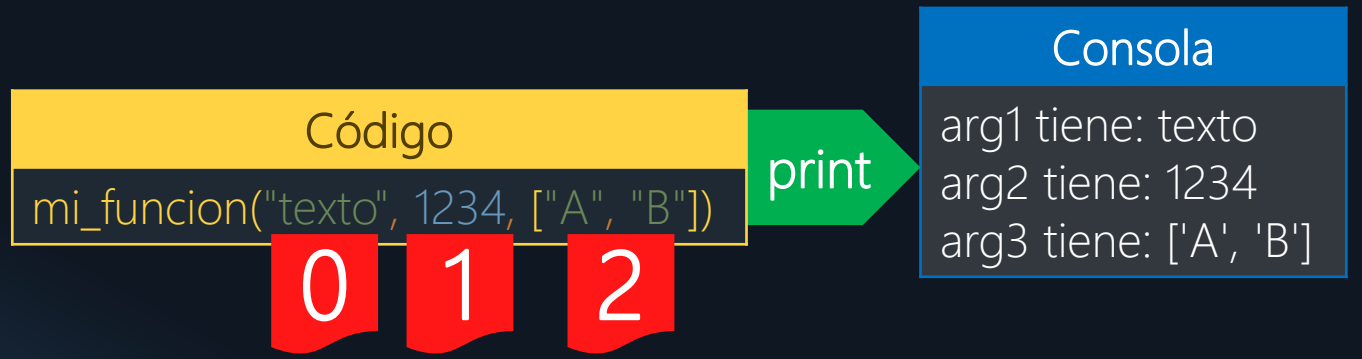
https://jarroba.com/args-kwargs-python-ejemplos/

Parámetros Args (Listado de parámetros)

```
def funcion(*args, **kwargs):
    cuerpo de la función
```

- **Args**: Hay que **respetar el orden** definido. Todos los Args definidos necesitan de un argumento **obligatorio** cuando se llama a la función.





https://jarroba.com/args-kwargs-python-ejemplos/

Parámetros Args (Listado de parámetros)

```
def funcion(*args, **kwargs):
    cuerpo de la función
```

- Los Args son un **listado** y se pueden extender con *.

Código

```
def mi_funcion(*args):
    for valor in args:
        print("arg tiene: {}".format(valor))
```

Código

```
mi_funcion("texto", 1234, ["A", "B"])
```

print

Consola

```
arg tiene: texto
arg tiene: 1234
arg tiene: ['A', 'B']
```

https://jarroba.com/args-kwargs-python-ejemplos/

Parámetros Kwargs (Parámetros por defecto)

```
def funcion(*args, **kwargs):
    cuerpo de la función
```

Código

```
def mi_funcion(kwarg1="-", kwarg2=0, kwarg3=None):
    print("Kwarg1 tiene: {}".format(kwarg1))
    print("Kwarg2 tiene: {}".format(kwarg2))
    print("Kwarg3 tiene: {}".format(kwarg3))
```

Código

```
mi_funcion()
```

print

Consola

```
Kwarg1 tiene: -
Kwarg2 tiene: 0
Kwarg3 tiene: None
```

Código

```
mi_funcion(kwarg1="texto")
```

print

Consola

```
Kwarg1 tiene: texto
Kwarg2 tiene: 0
Kwarg3 tiene: None
```

Código

```
mi_funcion(kwarg2=1234,
           kwarg1="texto")
```

print

Consola

```
Kwarg1 tiene: texto
Kwarg2 tiene: 1234
Kwarg3 tiene: None
```

Código

```
mi_funcion(kwarg2=1234,
           kwarg3=["A", "B"],
           kwarg1="texto")
```

print

Consola

```
Kwarg1 tiene: texto
Kwarg2 tiene: 1234
Kwarg3 tiene: ['A', 'B']
```


- **Kwargs**: Se escriben después de los Args si hay. **No es necesario respetar el orden definido si se utiliza su clave**, sino se usa su clave se respeta el orden de definición. Hay que añadirles un **valor por defecto**. Los Kwargs son **opcionales**, sino se le pasa un argumento se utilizará el valor por defecto.

https://jarroba.com/args-kwargs-python-ejemplos/

Parámetros Kwargs (Parámetros por defecto)

```
def funcion(*args, **kwargs):
    cuerpo de la función
```

Código

```
def mi_funcion(**kwargs):
    for clave, valor in kwargs.items():
        print("Clave '{}' tiene: {}".format(clave, valor))
```

Código

```
mi_funcion()
```

print

Consola

Código

```
mi_funcion(kwarg1="texto")
```

print

Consola

```
Clave 'kwarg1' tiene: texto
```

Código

```
mi_funcion(kwarg2=1234,
           kwarg1="texto")
```

print

Consola

```
Clave 'kwarg2' tiene: 1234
Clave 'kwarg1' tiene: texto
```

Código

```
mi_funcion(kwarg2=1234,
           kwarg3=["A", "B"],
           kwarg1="texto")
```

print

Consola

```
Clave 'kwarg2' tiene: 1234
Clave 'kwarg3' tiene: ['A', 'B']
Clave 'kwarg1' tiene: texto
```


- Los Kwargs son un **diccionario** y se pueden extender con **.

Resumen de parámetros

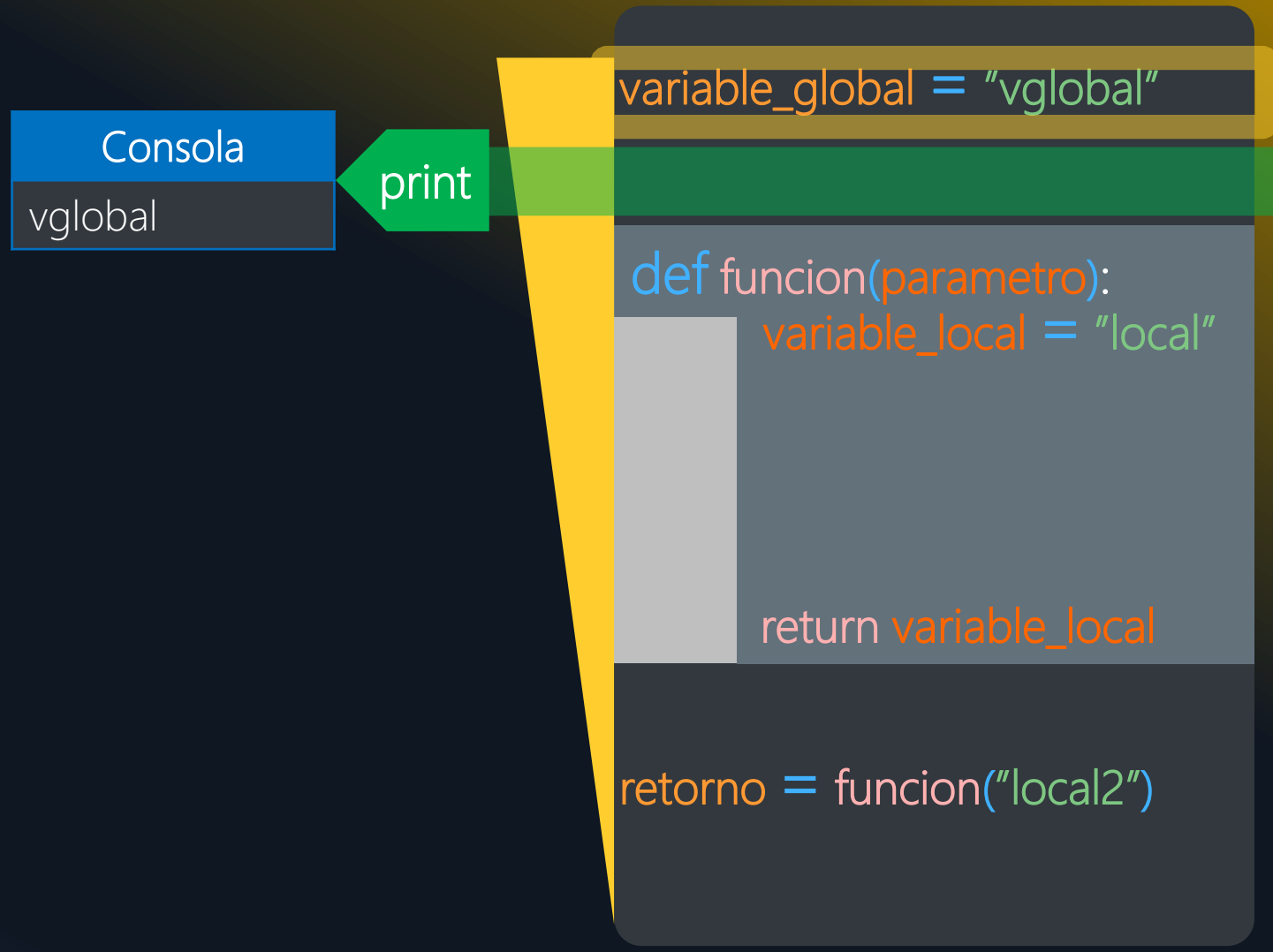
```
def funcion(*args, **kwargs):
    cuerpo de la función
```

- Argumentos en la **llamada** de la función:
  - **Posicional** ("ejemplo", 123, ["A", "B"], ...): **ordenados** con respecto a la cabecera de la función
  - **Clave y valor** (clave2="ejemplo", clave3=" 123, clave1="["A", "B"], ...): puede estar **desordenados** con respecto a la cabecera de la función

- **Parámetros** en la cabecera de la **declaración** de la función:
  - **Args: Listado de parámetros** (arg1, arg2, ...)
  - **Kwargs: Parámetros por defecto** (kwarg1="pordefecto", kwarg2=0, ...)
  - **Listado de parámetros seguido de parámetros por defecto** (arg1, ..., kwarg1="pordefecto", ...)

- **Parámetros indefinidos** en la cabecera de la **declaración** de la función:
  - ***args: tupla** de parámetros posicionales
  - ****kwargs: diccionario** de argumentos de acceso por clave

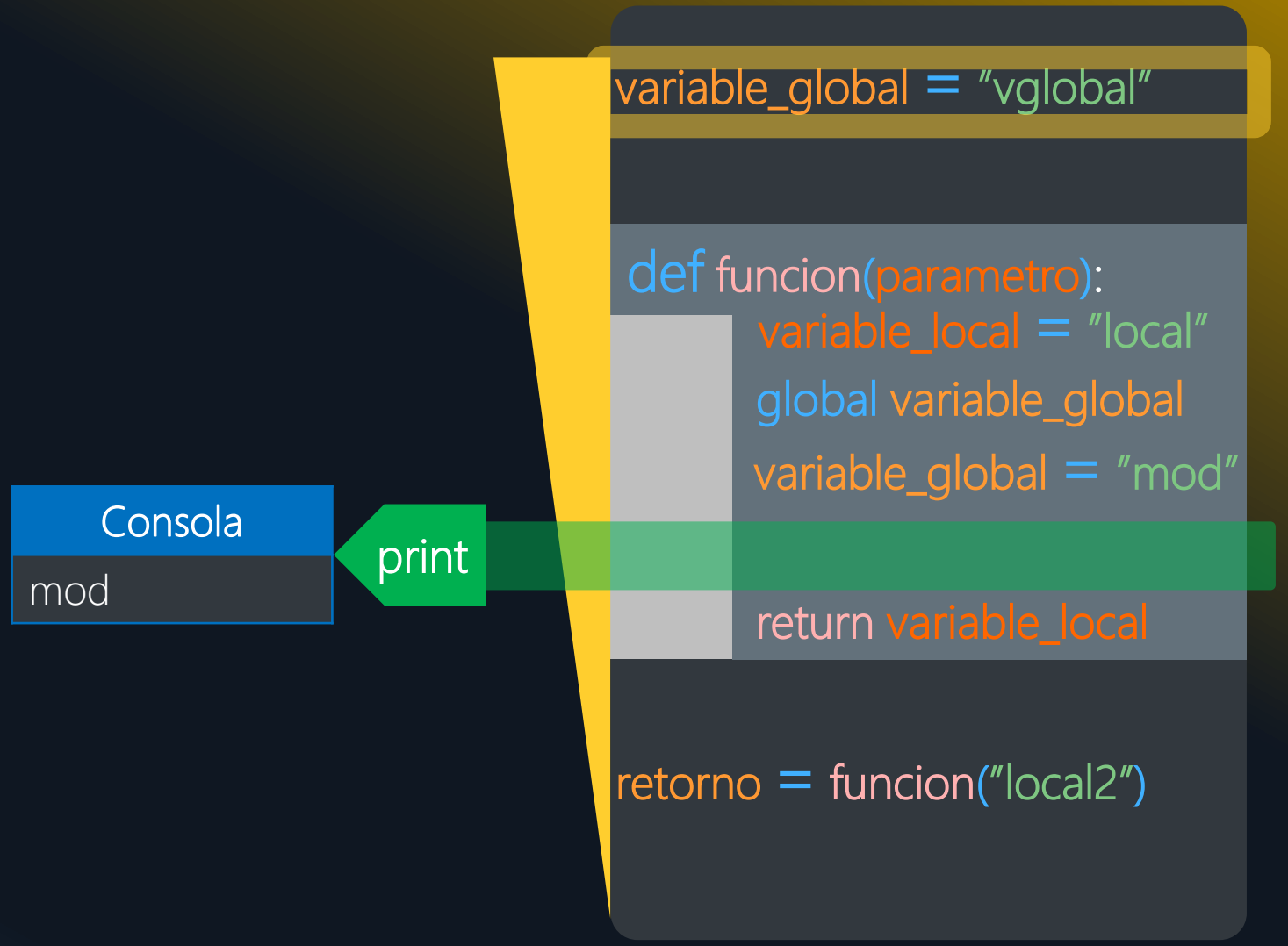
Alcance (Scope): Función (def)



Alcance (Scope): Función (def)

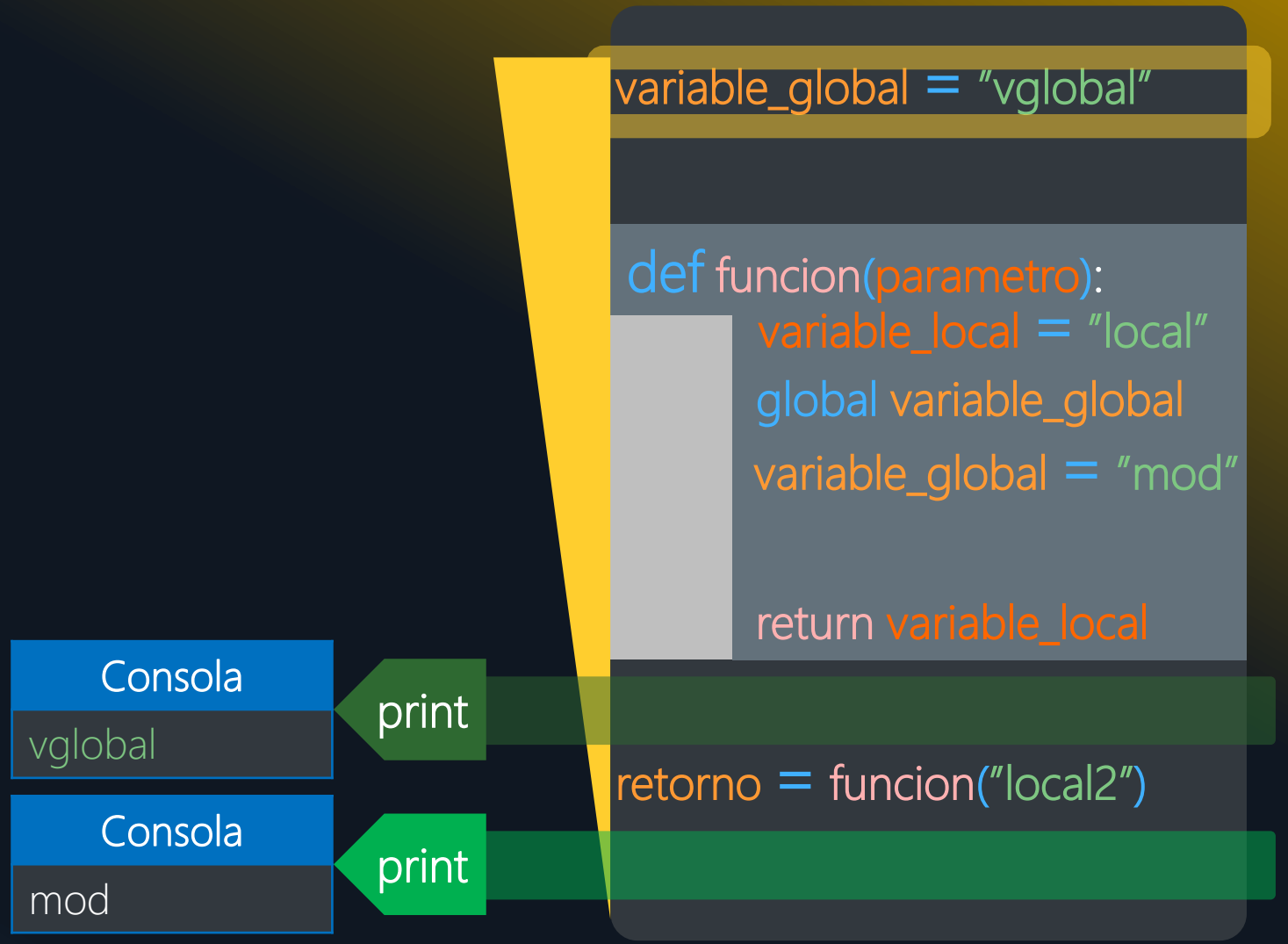
Dentro del **cuerpo de la función** si la palabra reservada "**global**":

- **Se usa**: permite **leer y escribir el contenido de una variable global**
- **NO se usa**: **se creará una variable local** solo disponible en el cuerpo de la función



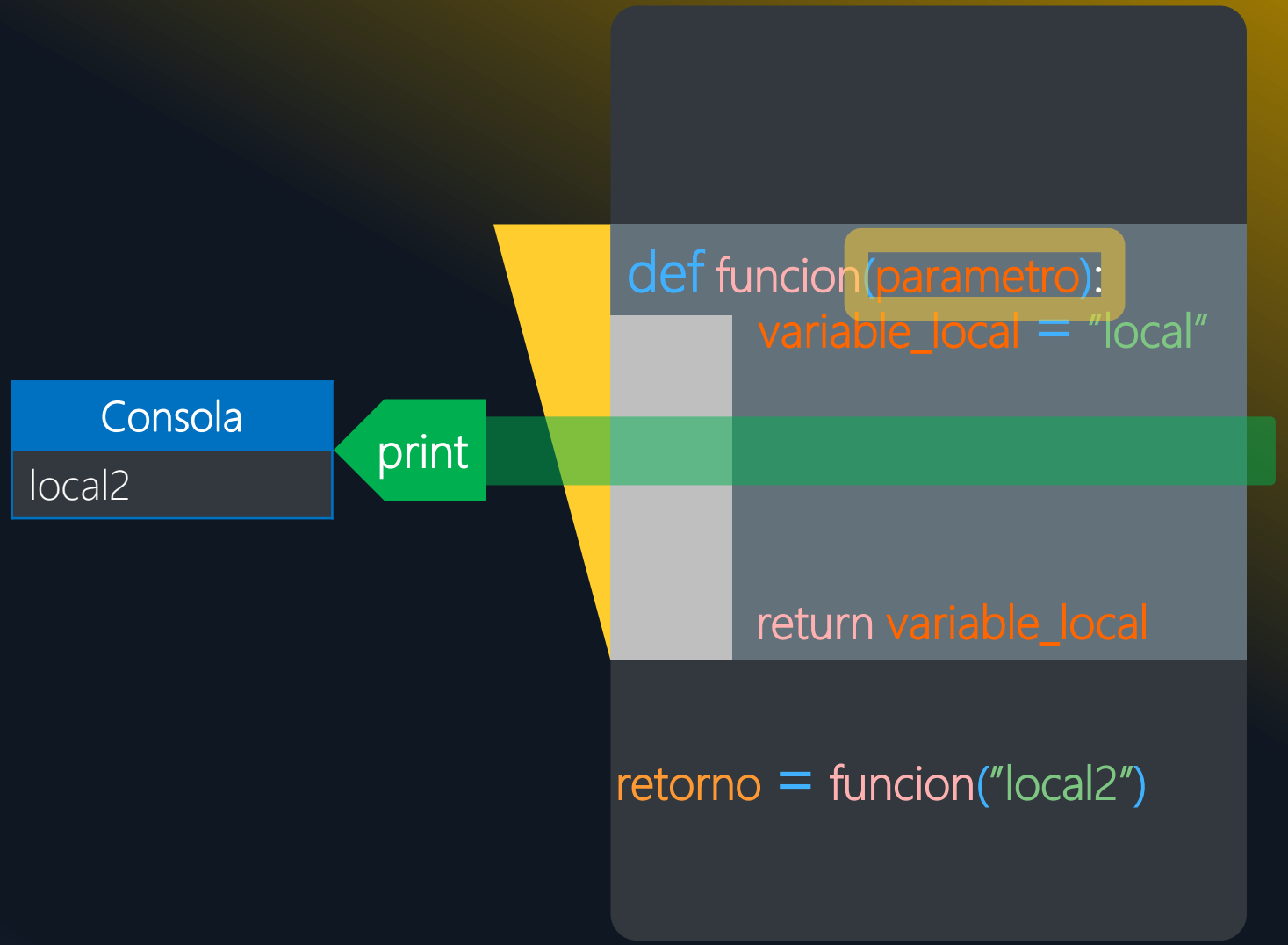
Alcance (Scope): Función (def)

Si se modifican con global las variables globales dentro de la función existe un antes y un después de su valor en el momento de llamar a la función



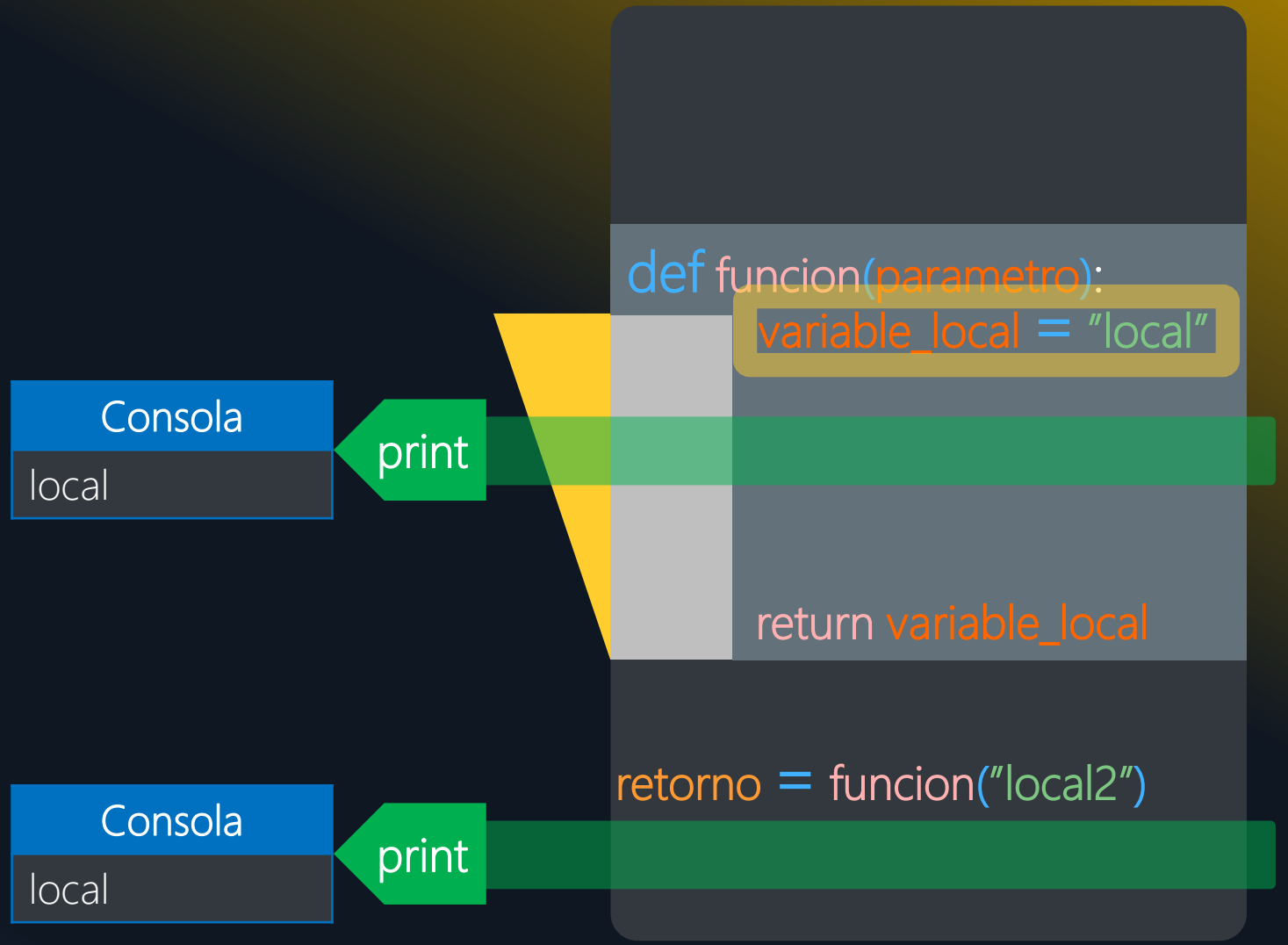
Alcance (Scope): Función (def)

Los parámetros son considerados “variables locales” que se inicializan desde la llamada a la función



Alcance (Scope): Función (def)

Utilizar “return” antes que “global” para extraer un “valor local” fuera de la función



Documentar funciones

```
def funcion(*args, **kwargs):
    """
    Documentación
    :param parametro: Explicación
    ...
    :param parametroN: Explicación
    :return: Explicación
    """
```

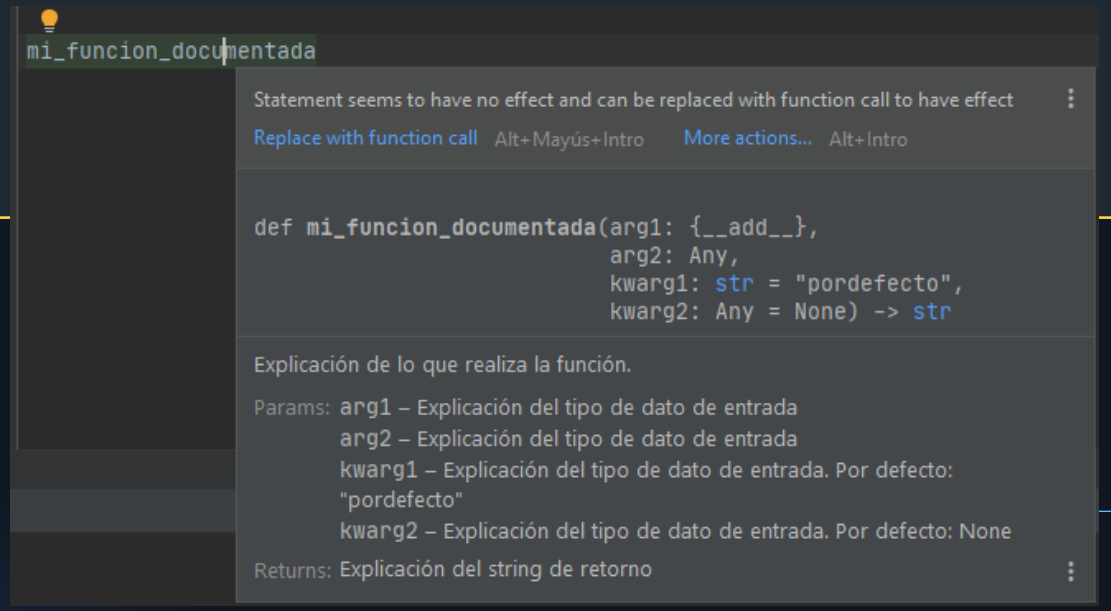
Código

```
def mi_funcion_documentada(arg1, arg2, kwarg1="pordefecto", kwarg2=None):
    """
    Explicación de lo que realiza la función.

    :param arg1: Explicación del tipo de dato de entrada
    :param arg2: Explicación del tipo de dato de entrada
    :param kwarg1: Explicación del tipo de dato de entrada. Por defecto: "pordefecto"
    :param kwarg2: Explicación del tipo de dato de entrada. Por defecto: None
    :return: Explicación del string de retorno
    """

    # Código de la función

    return "valor de retorno"
```



Documentar funciones

```
def funcion(*args, **kwargs):
    """
    Documentación
    :param ...
    :raise excepcion: Explicación
    ...
    :raise excepcionN: Explicación
    :return: Explicación
    """
```

Código

```
def mi_funcion_documentada(arg1, arg2, kwarg1="pordefecto", kwarg2=None):
    """
    Explicación de lo que realiza la función.

    :param arg1: Explicación del tipo de dato de entrada
    :param arg2: Explicación del tipo de dato de entrada
    :param kwarg1: Explicación del tipo de dato de entrada. Por defecto: "pordefecto"
    :param kwarg2: Explicación del tipo de dato de entrada. Por defecto: None
    :raise IOError: Explicación de la excepción lanzada
    :raise KeyError: Explicación de la excepción lanzada
    :return: Explicación del string de retorno
    """

    # Código de la función

    return "valor de retorno"
```

Tipado de funciones

```
def funcion(arg: tipo,
            kwarg: tipo = "pordefecto") -> tipo:
```

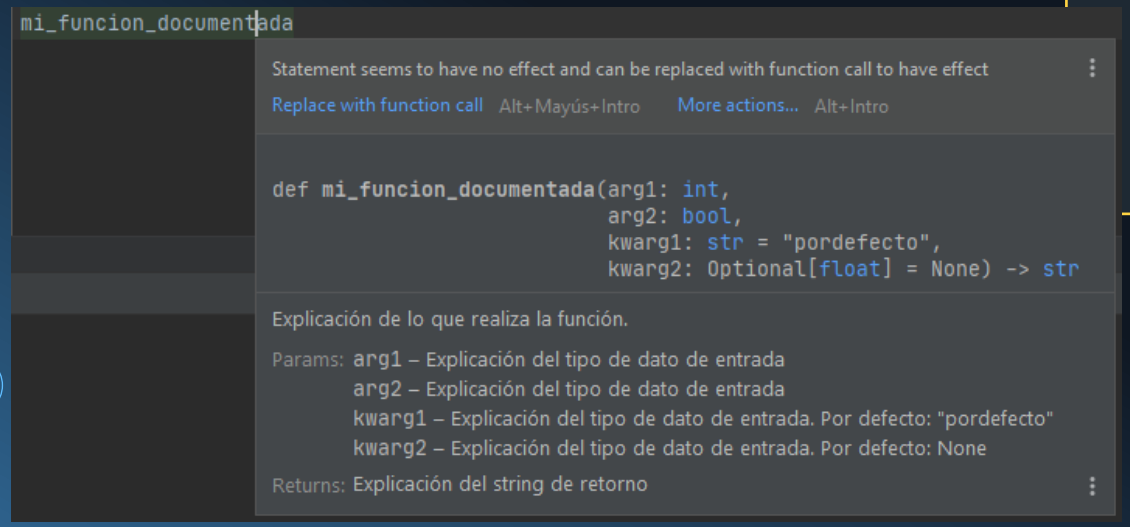
Código

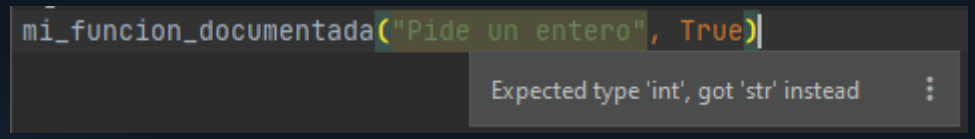
```
def mi_funcion_documentada(arg1: int,
                           arg2: bool,
                           kwarg1: str = "pordefecto",
                           kwarg2: float = None) -> str:
    """
    Explicación de lo que realiza la función.

    :param arg1: Explicación del tipo de dato de entrada
    :param arg2: Explicación del tipo de dato de entrada
    :param kwarg1: Explicación del tipo de dato de entrada
    :param kwarg2: Explicación del tipo de dato de entrada
    :return: Explicación del string de retorno
    """

    # Código de la función

    return "valor de retorno"
```





Lambda

lambda *args, **kwargs: valor_a_retornar

Pseudo-Código

```
def nombre_funcion(param1, param2="por defecto"):
    return "valor a retornar {} {}".format(param1, param2)
```

Pseudo-Código (lambda)

```
lambda param1, param2="por defecto": "valor a retornar {} {}".format(param1, param2)
```

¡GRACIAS!

Web: **https://jarroba.com/**

Ramón Invarato Menéndez

Linked-in
https://www.linkedin.com/in/rinvarato/

Github
https://github.com/Invarato

Ricardo Moya García

Linked-in
https://www.linkedin.com/in/phdricardomoya

Github
https://github.com/RicardoMoya