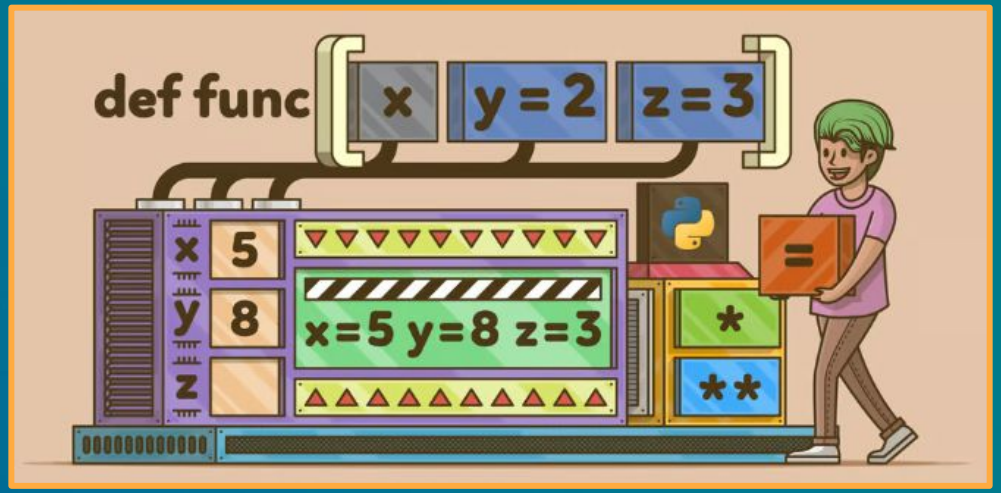
Scarafilo Germán - Scudero Yanina - Fernandez Mariano - Pavlov Valeriy

Funciones



Introducción

En Python, una función es un bloque de código que realiza una tarea específica.

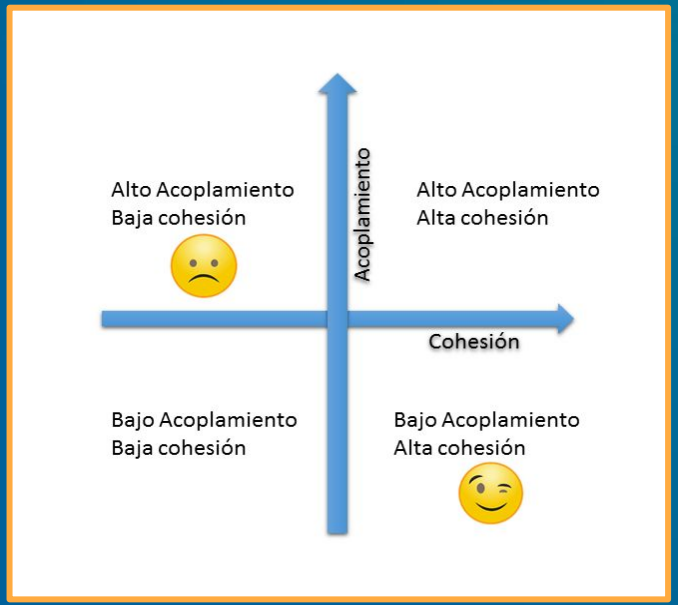
¿Qué es una función?

A medida que los programas crecen en extensión y complejidad la resolución se torna más **complicada** y su **depuración** y **modificaciones** futuras resultan casi imposibles. Para resolver este tipo de problemas lo que se hace es **dividir el programa** en módulos más pequeños que cumplan **una tarea simple y bien acotada** llamados funciones.

¿Para qué sirve una función?

- Las funciones sirven para **descomponer una tarea específica en un bloque de código** que se puede llamar varias veces desde cualquier parte del programa.
- Las funciones permiten que el código sea **modular, legible y fácil de mantener**.

Cohesión y Acoplamiento :



Cohesión: Implica una conexión o unión que asegura que las partes de un todo estén bien integradas y funcionen juntas de manera efectiva.

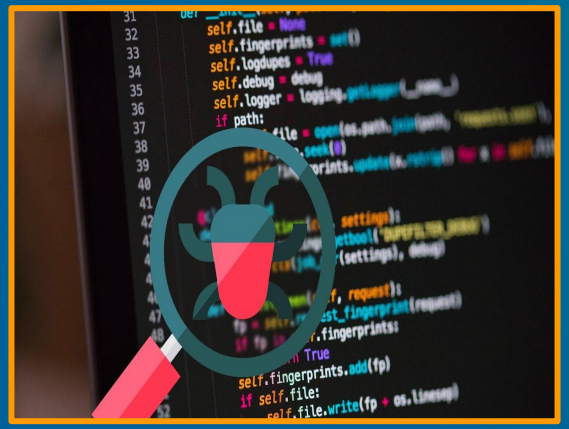
Acoplamiento: Es la dependencia que tienen las funciones entre sí.

Objetivos de una función:

Minificación: Logramos que el programa sea más simple de comprender ya que cada función se dedica a realizar una tarea en particular.

Scarafilo Germán - Scudero Yanina - Fernandez Mariano - Pavlov Valeriy

Objetivos de una función:

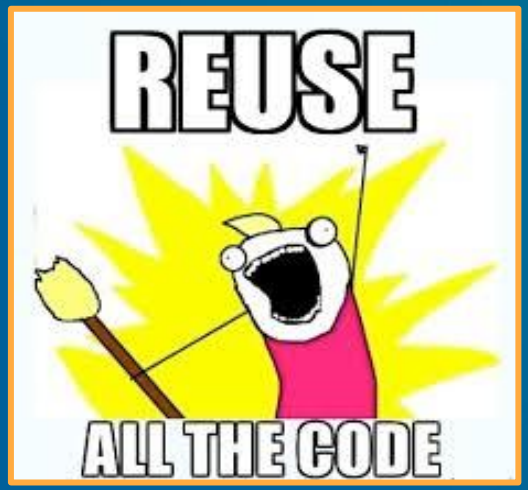


Depuración: La depuración queda acotada a cada función.

Objetivos de una función:

Modificación: Las modificaciones al programa se reducen a cada módulo.

Objetivos de una función:

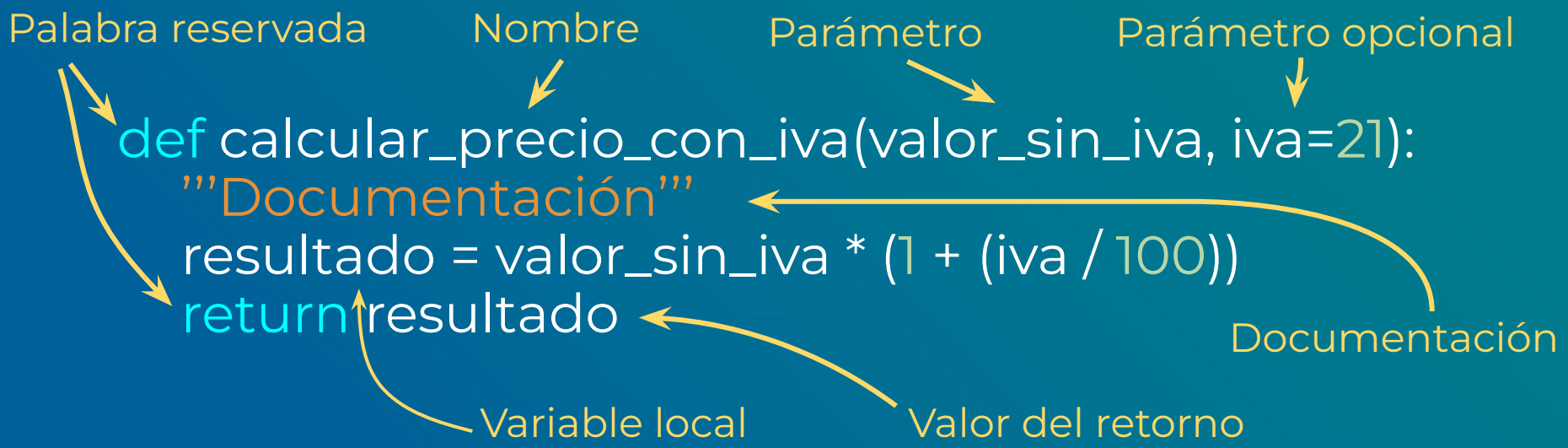


Reutilización: Cuando cada función está bien probada, se la puede usar las veces que se quiera y así reutilizar código.

Objetivos de una función:

Independencia: Se obtiene una autonomía del código donde cada función es independiente de otra.

Componentes de una función:



Desarrollo e invocación

```
def calcular_precio_con_iva(valor_sin_iva, iva=21):
    '''Suma el IVA al precio, por defecto toma 21%'''
    resultado = valor_sin_iva * (1 + (iva / 100)
    return resultado

print(calcular_precio_con_iva(100))   # 121.0
print(calcular_precio_con_iva(100, 10.5))  # 110.5
```

Definición y desarrollo

Invocación o llamada

Palabras reservadas: def y return

El uso de def nos permite crear una función.
Podemos utilizar return para devolver un valor.

```
def sumar(numero_a, numero_b):
    suma = numero_a + numero_b
    return suma

resultado = sumar(33, 2)
print("El resultado de la suma es", resultado)
```

Nombre

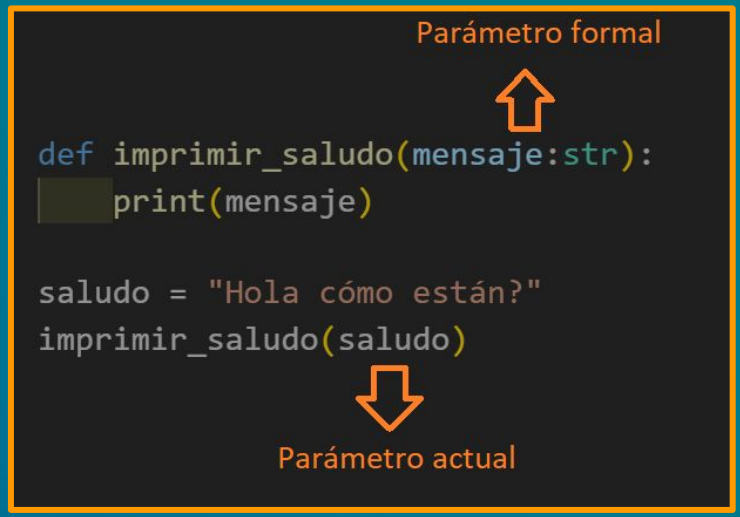
Las funciones y las variables se deberán escribir en formato **snake_case**.
Las palabras separadas entre sí por barra baja (underscore) en vez de espacios y las letras en minúscula.
Las funciones representan una acción, por lo que el nombre debe contener un verbo.

calcular_precio_con_iva()

Tipos de parámetros

Parámetros Formales: son variables locales declaradas en la definición de la función, que se utilizarán para recibir datos con los que va a operar la misma.

Parámetros Actuales: son las variables o datos que recibe la función al momento de ser invocada.



Parámetros por posición

La forma más intuitiva de trabajar con parámetros es pasandolos respetando la posición.

```
def restar(numero_a, numero_b):
    diferencia = numero_a - numero_b
    return diferencia

print(restar(15, 5)) # 10
```

Parámetros por nombre

Otra forma de pasar parámetros a una función es asignando valores a las variables utilizando sus nombres.

```
def restar(numero_a, numero_b):
    diferencia = numero_a - numero_b
    return diferencia

print(restar(numero_b = 5, numero_a = 15)) # 10
```

Parámetros opcionales

Python permite trabajar con parámetros opcionales o por defecto, estos deben ir siempre después de los obligatorios.

```
def restar(numero_a, numero_b = 5):
    diferencia = numero_a - numero_b
    return diferencia

print(restar(15)) # 10
print(restar(15, 7)) # 8
```

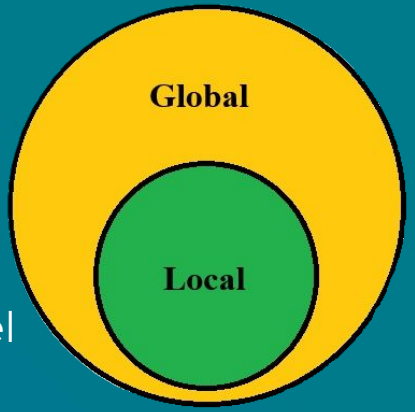
Parámetros y retorno con tipos

Se recomienda indicar que tipo de dato se espera que reciban los parámetros y que tipo de dato devuelve la función. * En Python estas indicaciones NO son restrictivas.

```
def restar(numero_a:int, numero_b:int = 5)-> int:
    diferencia = numero_a - numero_b
    return diferencia

print(restar(15)) # 10
```

Variables locales y globales



- Las variables **locales** son aquellas que se definen **dentro de una función** y solo son accesibles dentro de esa función.
- Las variables **globales** son aquellas que se definen **fuera de todas las funciones** y son accesibles desde cualquier parte del programa.

El **scope** (ámbito de la función) es donde las variables **locales** nacen y mueren, este mismo está separado por **sangría** al igual que las estructuras lógicas y repetitivas.

Nota: las variables globales, dentro de una función, pueden ser modificadas solo si se las marca como globales dentro del scope de la misma.

¿Qué es una Referencia?

Una referencia en programación es una forma de acceder a un valor mediante su **dirección de memoria** en lugar de interactuar directamente con el valor. Esto permite manipular el dato original sin crear copias adicionales, lo cual es eficiente en términos de recursos. Al utilizar una referencia, cualquier modificación realizada a través de ella afectará directamente al valor almacenado en esa dirección de memoria.

Pasaje por valor y por referencia en general :

En principio 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.

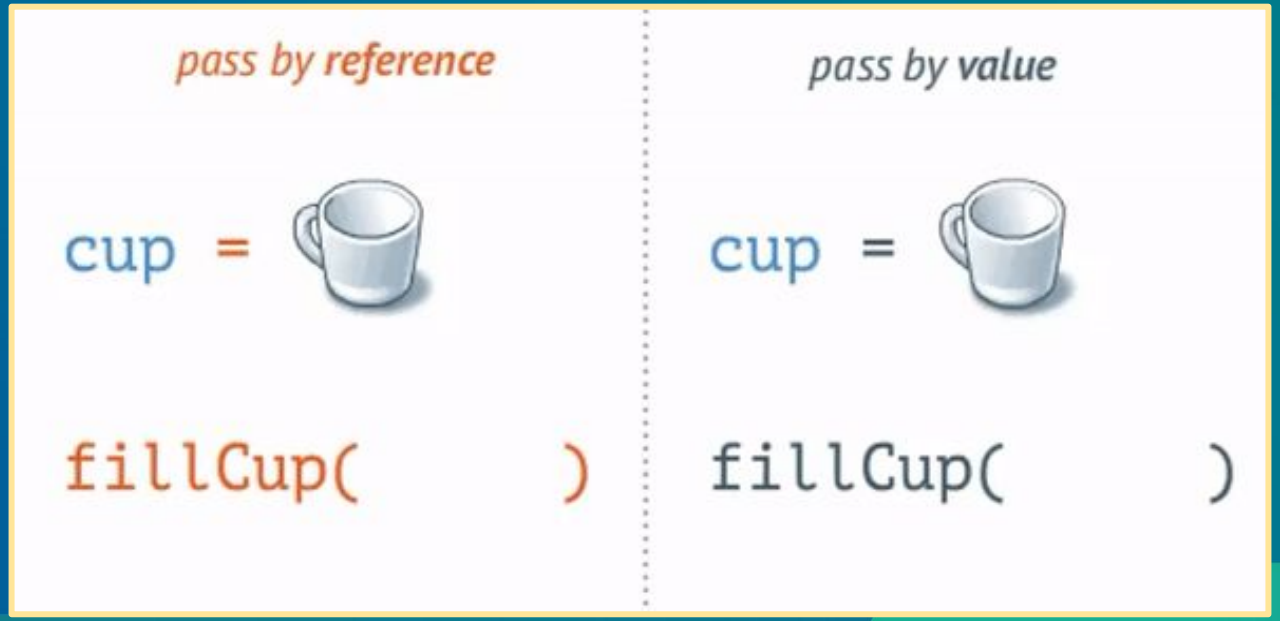
- Por VALOR: la función creará una copia local de la variable y cualquier modificación sobre ella no tendrá ningún efecto sobre la original.
- Por REFERENCIA: la función recibirá la dirección de memoria de la variable original y podrá realizar modificaciones directamente sobre ella.

Pasaje por valor y por referencia en Python :

En Python, **los parámetros de una función se pasan por referencia**, pero esta referencia es en realidad una referencia al objeto, no a la variable original. Esto significa que cuando se pasa una variable a una función, la función recibe una referencia al objeto al que apunta la variable.

Sin embargo, no puede modificar la variable original si el objeto al que apunta es **inmutable**, como números o cadenas de texto. En cambio, si el objeto es **mutable**, como una lista, un diccionario o una instancia de una clase, los cambios realizados dentro de la función afectarán al objeto original.

Pasaje por valor y por referencia



Documentación

La documentación de una función es fundamental para la claridad, comprensión, y mantenibilidad del código, facilitando tanto el trabajo individual como la colaboración en equipos de desarrollo.

```
def sumar(numero_a:int, numero_b:int)-> int:
    '''Indicar qué hace la función
    Qué parámetros acepta
    Qué devuelve'''
    suma = numero_a + numero_b
    return suma
```