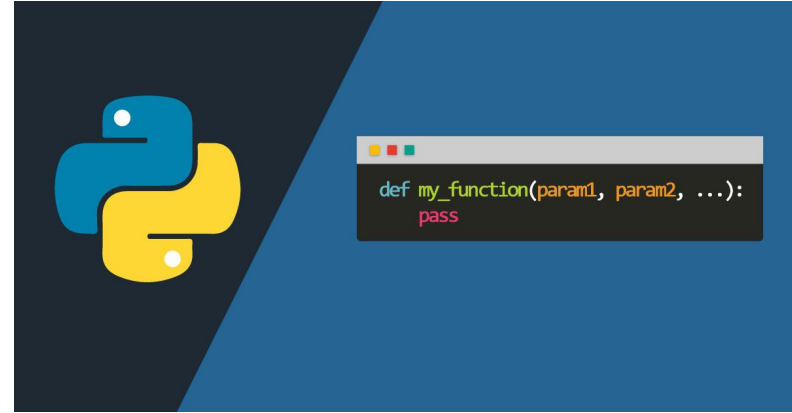


# Funciones

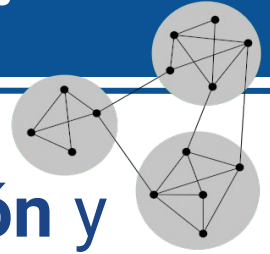
Programación y Laboratorio I



# 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.

# ¿Qué es una función?

- Una función es un bloque de código que se puede llamar en cualquier momento para realizar una tarea específica.
- Las funciones se utilizan para organizar y reutilizar código en programas más grandes.

# ¿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**.

# Objetivos de una función

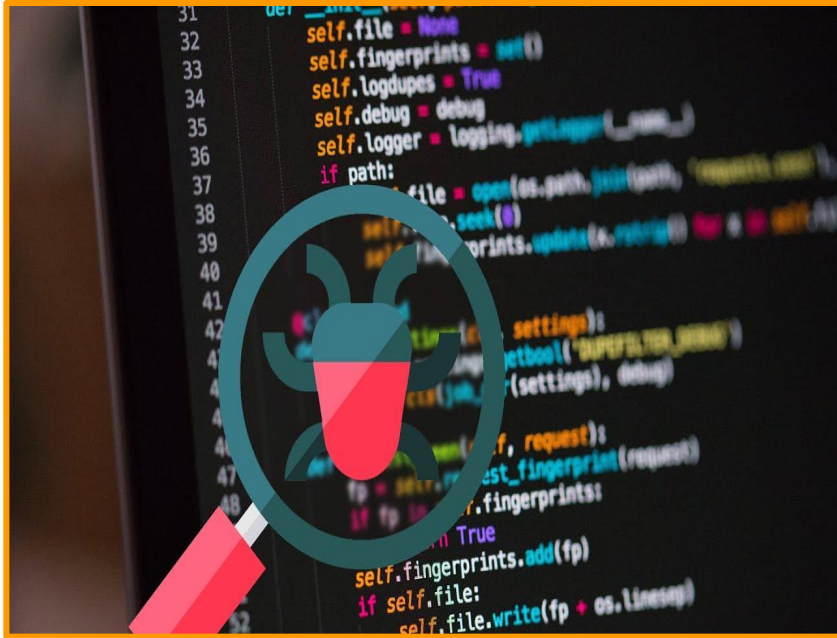


# Minificación



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

# Depuración



La depuración queda acotada a cada función.

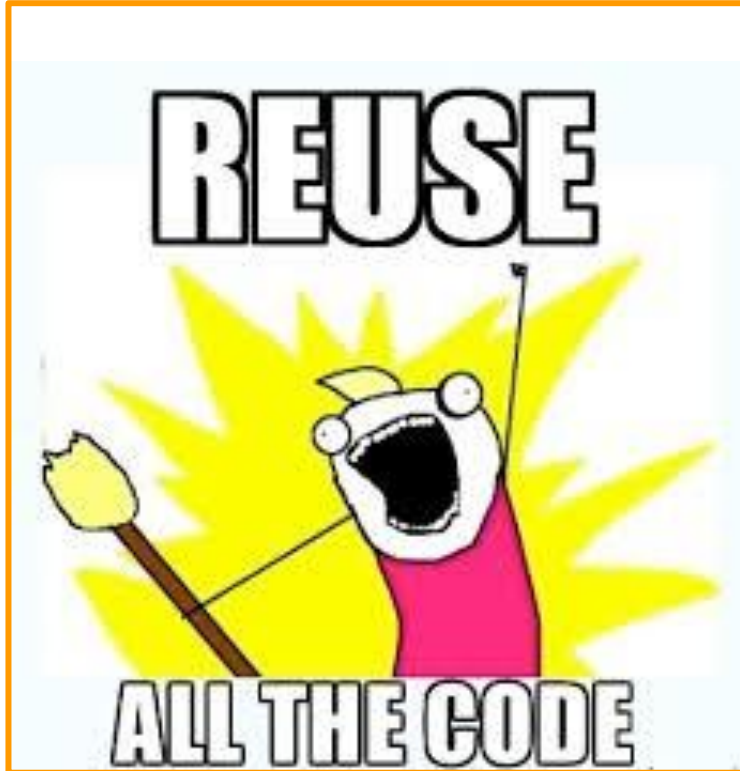


# Modificación



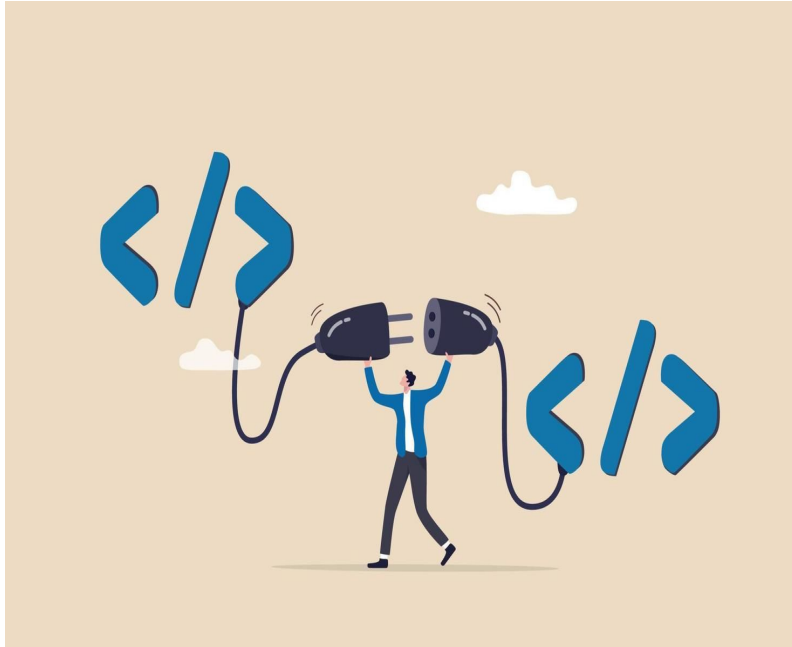
Las modificaciones al programa se reducen a cada módulo.

# Reutilización



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

# Independencia



Se obtiene una independencia del código (cada función es independiente de otra)

# Componentes de una función

Keyword

Nombre

Argumento

Argumento opcional

```
def calcular_precio_con_iva(valor_sin_iva, iva=21):  
    '''Documentación'''  
    resultado = valor_sin_iva * (1+(iva/100))  
    return resultado
```

Documentación

Variable local

Valor de retorno

# Componentes de una función

Para declarar una función en Python, utilizamos la palabra reservada **"def"** seguida del nombre de la función y los parámetros entre paréntesis.

La sintaxis básica para declarar una función es la siguiente:

```
def nombre_de_la_funcion(variable_a, variable_b):  
    # Cuerpo de la función  
    # Realizar tarea específica  
    return resultado  
    # El retorno puede no estar
```

# Keyword: **def** y **return**

El uso de **def** nos permite crear una función para que la función devuelva uno o varios valores, debemos usar **return**.

```
def sumar(a, b):  
    return a + b
```

```
suma = sumar(33,1)  
print("La suma es", suma)
```

Las funciones y las variables se deberán escribir en formato **snake\_case**.

Las palabras separadas por barra baja (underscore) en vez de espacios y con la primera letra de cada palabra en minúscula.

```
calcular_precio_con_iva
```

# Instancias de una función

## Desarrollo de la función:

```
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
```

## Llamada o invocación

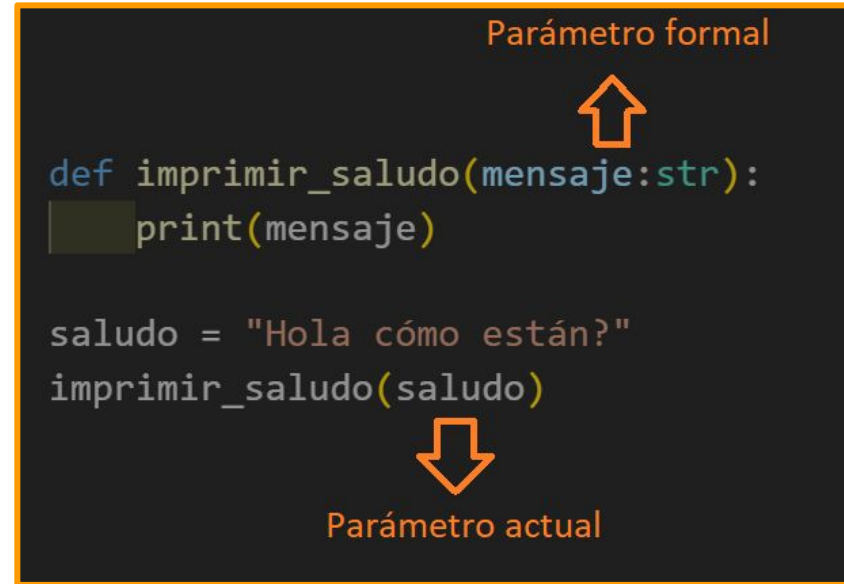
```
print(calcular_precio_con_iva(100))           # 121.0  
print(calcular_precio_con_iva(100,10.5))      # 110.5
```



# Tipos de parámetros

**Parámetros Formales:** Son las variables que le llegan a la función (estos son locales a la función, y son utilizados por ella)

**Parámetros Actuales:** Son las variables o datos que se les pasa a la función al momento de ser invocada.



# Argumento por posición

Los argumentos posicionales son la forma más básica e intuitiva de pasar parámetros.

```
def restar(variable_a, variable_b):  
    return variable_a-variable_b
```

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

# Argumento por nombre

Otra forma de llamar a una función, es usando el nombre del argumento con = y su valor.

```
def restar(variable_a, variable_b):  
    return variable_a - variable_b
```

```
print(restar(variable_b = 5, variable_a = 15)) #  
10
```

# Argumento opcional

En python, es posible tener una función con parámetros opcionales. Estos pueden ser utilizados o no dependiendo de diferentes circunstancias.

```
def restar(variable_a, variable_b = 5):  
    return variable_a - variable_b
```

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

**Nota:** los parámetros opcionales tienen que ser los últimos en la lista de parámetros.

# Parámetros con tipo

Es posible documentar los tipos de los parámetros que espera recibir una función.

```
def restar(variable_a:int, variable_b:int = 5):  
    return variable_a - variable_b  
  
print(restar(variable_a = 15)) # 10
```

# Retorno con tipo

Para indicar el tipo de dato del retorno de la función, utilizamos la -> (flecha).

```
def restar(variable_a:int) -> int:  
    return variable_a - 1
```

```
print(resta(15)) # 14
```

# Variables locales vs. 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.

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

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.

# Pasaje por valor y referencia

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**. 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 referencia

*pass by reference*

cup = 

fillCup(        )

*pass by value*

cup = 

fillCup(        )

**Nota: recuerden que en python, todos los parámetros se pasan por referencia!!!**

Es importante documentar las funciones.

```
def sumar(variable_a, variable_b):  
    #Indicar qué hace  
    #Qué parámetros acepta  
    #Qué devuelve  
    return variable_a + variable_b
```

# A tener en cuenta

- Los nombres de las funciones deben hacer referencia a acciones, por ejemplo nombres como **imprimir\_texto**, **mostrar\_informacion**, **pedir\_numero**, etc. Deben ser siempre **verbos en infinitivo acompañados de un sustantivo**.
- Las funciones deben estar **siempre documentadas**.
- Las funciones **pueden no devolver nada y no recibir nada** pero este tipo de funciones **deben ser evitadas** ya que no tienen un sentido específico y se vuelven demasiado complejas.