



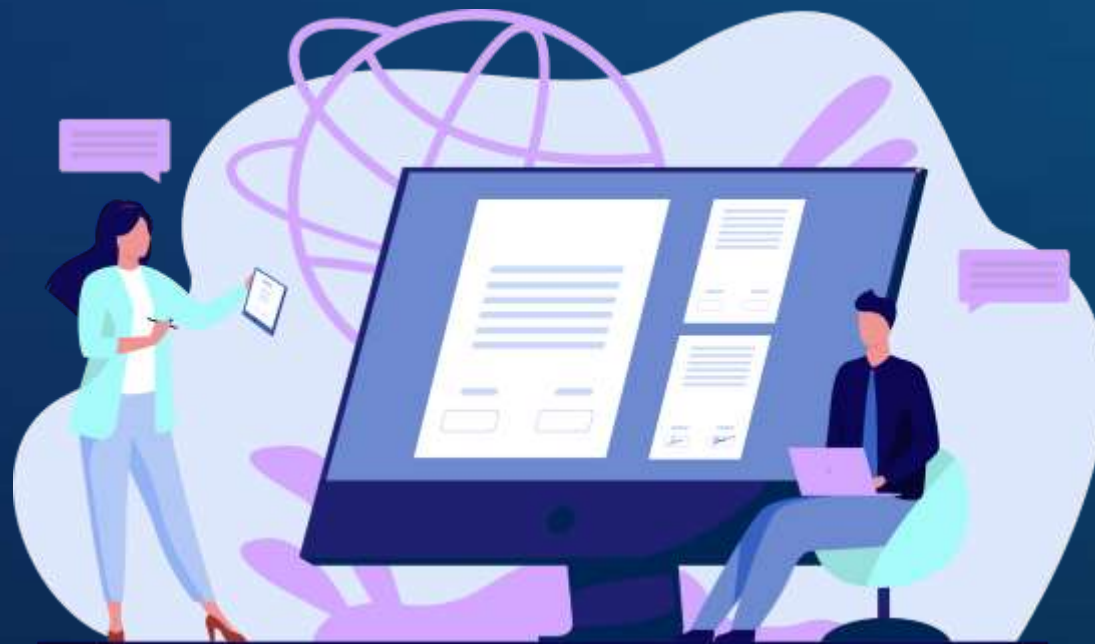
# Sesión 20

## Programación Nivel Básico



UNIVERSIDAD  
LIBRE®

eTraining®



**Sesión 20:**

# **Fundamentos del lenguaje Python**

**Declaración de funciones, parámetros y retorno de valores**

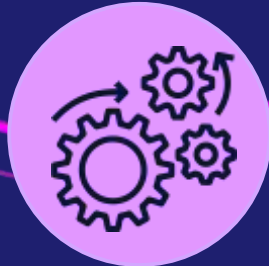
# Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:



## Comprender los conceptos básicos

El primer objetivo es entender qué son las funciones en Python. También aprenderemos sobre su sintaxis y estructura. Esto es fundamental para cualquier programador que quiera escribir código efectivo.



## Aplicar en ejercicios prácticos

El segundo objetivo es aplicar lo aprendido en ejercicios prácticos. Los estudiantes desarrollarán varias funciones en Python con diferentes tipos de parámetros. Esto les ayudará a consolidar sus conocimientos teóricos.

# Introducción a Funciones en Python

Las funciones son bloques de código reutilizables. Permiten organizar y simplificar el código en Python.

Mediante funciones, los programadores pueden definir tareas específicas. Así, pueden ser llamadas múltiples veces en diferentes partes del programa. Esto mejora la legibilidad y eficiencia del código.



# ¿Qué es una función en Python?

Una función en Python es un bloque de código que realiza una tarea específica. Se puede reutilizar en diferentes partes del programa para evitar la repetición de código.

Las funciones permiten organizar el código de manera estructurada y mejoran la legibilidad. También facilitan el mantenimiento del software y el trabajo en equipo.





# Definición de Función

## Concepto Básico

Una función en Python es un bloque de código. Este bloque realiza una tarea específica y puede ser reutilizado. Las funciones ayudan a organizar el código y mejorar su legibilidad.

## Importancia

Las funciones permiten ejecutar operaciones repetidamente sin duplicar código. Facilitan la gestión de grandes programas al encapsular lógica específica. Esto resulta en un código más limpio y eficiente.

## Ejemplo Simple

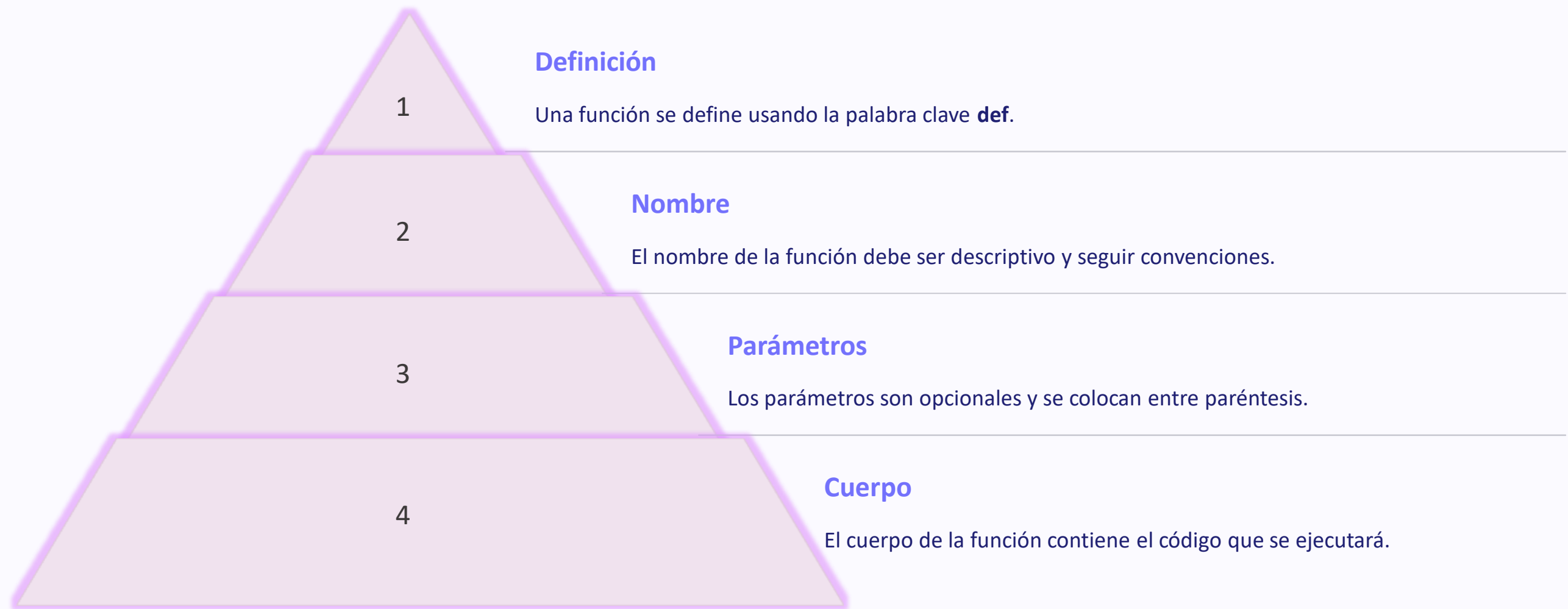
Puedes definir una función utilizando la palabra clave **def**. Luego, especificas el nombre y los parámetros necesarios. Esto establece la estructura para que puedas llamar a la función posteriormente.





# Sintaxis básica de una función

La sintaxis de una función en Python es esencial para su correcta declaración. Aquí, exploraremos los componentes fundamentales que permiten definir y usar funciones de manera efectiva. Entender esta estructura básica es clave para el aprendizaje de la programación en Python.



Estos son los elementos clave. Utilizar correctamente esta sintaxis permite crear funciones eficientes y organizadas. Con el tiempo, dominarás la creación de funciones en tus programas.



# Parámetros de una función



## Definición de Parámetros

Los parámetros son valores que se utilizan en las funciones para manipular datos. Permiten la entrada de información en la función. Esto hace que la función sea más versátil y reusable en diferentes contextos.

## Tipos de Parámetros

Existen varios tipos de parámetros en Python, como posicionales y por palabra clave. Los parámetros posicionales son obligatorios y deben ser proporcionados en el orden correcto. Los parámetros por palabra clave son opcionales y pueden ser nombrados al llamar la función.

## Uso de Parámetros

Los parámetros se definen en la declaración de la función. Al utilizar diferentes parámetros, se pueden realizar diversas operaciones. Esto permite a los programadores crear funciones más eficientes y adaptables a distintas situaciones.



# Tipos de parámetros



## Parámetros obligatorios

Son aquellos que deben ser proporcionados al invocar una función. Sin estos parámetros, la función no puede ejecutarse correctamente. Por esta razón, es crucial entender su necesidad en la declaración de funciones.



## Parámetros opcionales

Estos parámetros tienen valores predeterminados. Si no se proporcionan durante la llamada de función, se utilizan los valores por defecto. Esto permite a los desarrolladores tener más flexibilidad al trabajar con funciones.



## Parámetros variables

Permiten recibir un número variable de argumentos. Se utilizan en situaciones donde no se conoce de antemano el número de argumentos. Esto se logra usando `*args` y `**kwargs` para funciones flexibles.





# Argumentos Posicionales



## Definición

Los argumentos posicionales son aquellos que se pasan a una función en un orden específico. Este orden determina cómo se asignan los valores a los parámetros de la función. Por lo tanto, es crucial mantener el orden correcto cuando se llaman a las funciones.



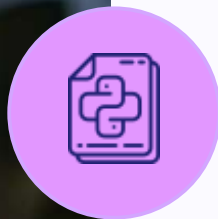
## Importancia

La utilización de argumentos posicionales simplifica la llamada a una función. También mantiene la claridad en el código, ya que cada argumento se asocia con un parámetro definido. Además, facilita el proceso de debugging, al asegurar que se reciben los valores esperados.



## Ejemplo

Considera una función que calcula el área de un rectángulo. Si la función se define como `area(length, width)`, al llamar `area(5, 10)`, el primero se asigna a **length** y el segundo a **width**.



# Argumentos por palabra clave



## Definición

Los argumentos por palabra clave permiten una mayor flexibilidad en la llamada de funciones. En lugar de depender del orden de los parámetros, se utilizan etiquetas para identificarlos.



## Ventajas

Facilitan la comprensión del código, haciendo que sea más legible. También permiten omitir argumentos opcionales, proporcionando un mayor control sobre la función.



## Ejemplo de uso

Al definir una función, puedes llamar a los parámetros por su nombre. Esto mejora la claridad y disminuye el riesgo de errores al llamar a la función.



# Retorno de valores en una función

## Concepto de Retorno

El retorno de valores es esencial en funciones. Permite que una función devuelva un resultado después de su ejecución. Esto facilita el uso de los datos de la función en otros procesos.

## Sintaxis de Retorno

Para devolver un valor en Python, se usa la palabra clave **return**. Al ejecutar **return**, la función termina y se envía el resultado. Este valor puede ser almacenado en una variable.

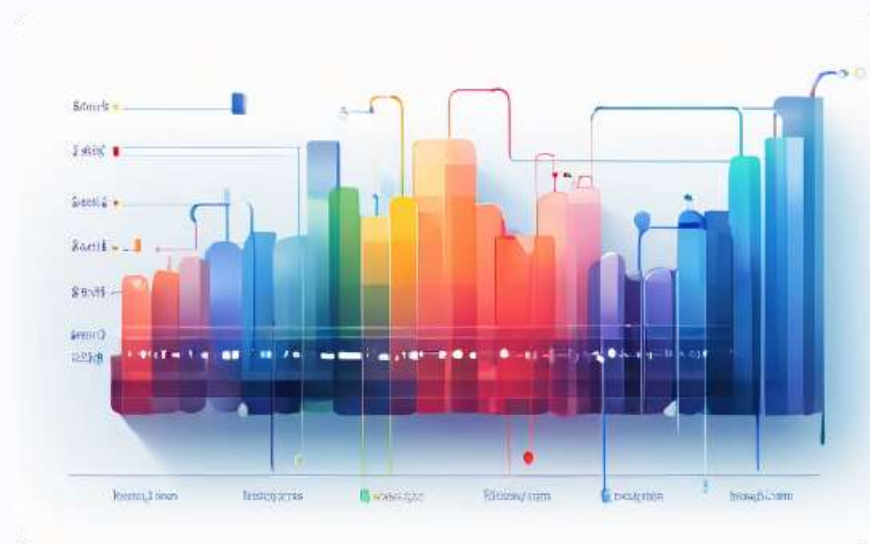
## Importancia del Retorno

Las funciones con retorno son más útiles y flexibles. Pueden ser reutilizadas en diferentes contextos. Esto ahorra tiempo y esfuerzo en la programación.



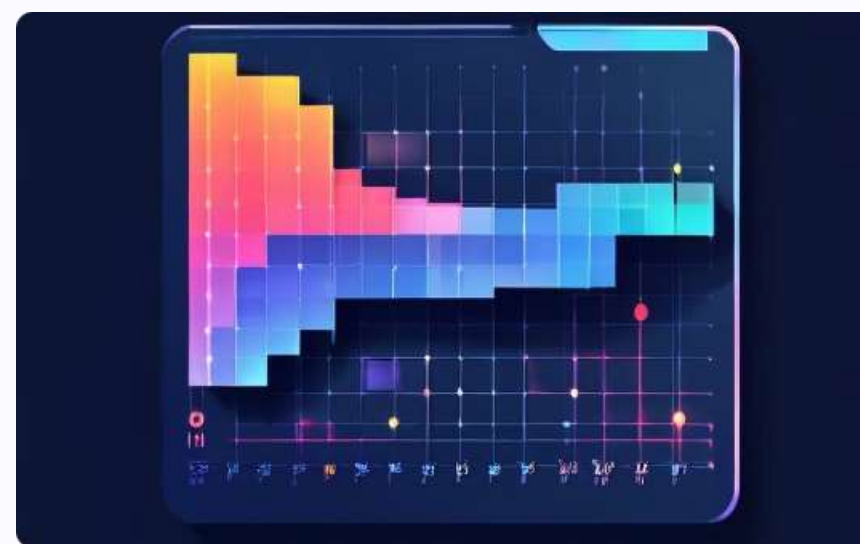


# Tipos de valores de retorno



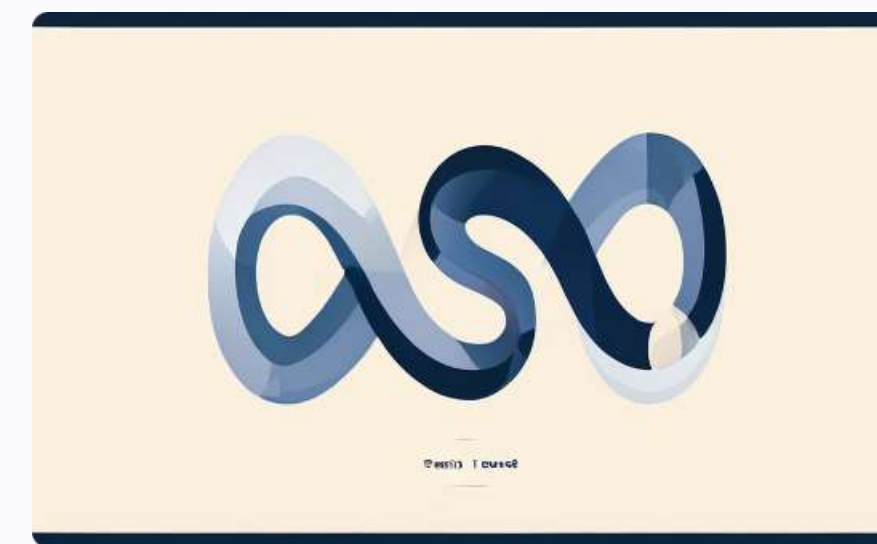
## Valores Simples

Las funciones en Python pueden retornar valores simples. Esto incluye tipos como enteros y cadenas. Este retorno es directo y sencillo.



## Listas y Colecciones

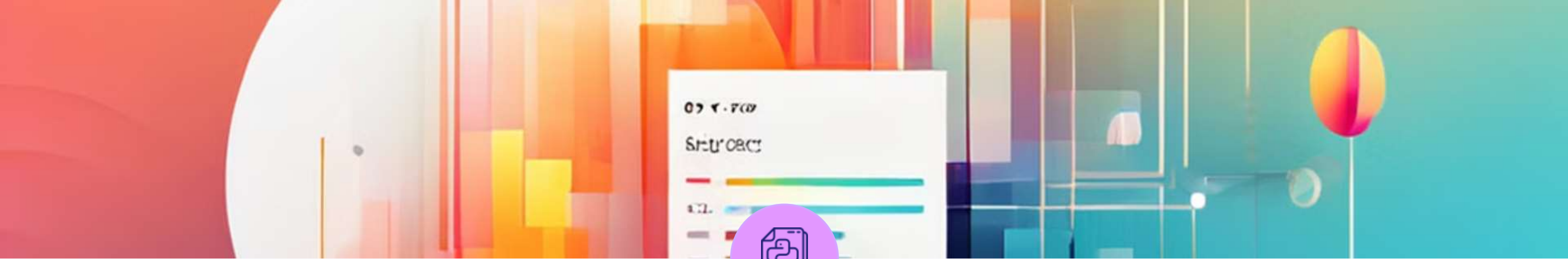
Las funciones también pueden devolver colecciones. Un ejemplo común es el retorno de listas. Estas permiten manejar múltiples datos juntos.



## Múltiples Valores

Python permite retornar múltiples valores usando tuplas. Esto ayuda a agrupar información relacionada. Es útil para funciones con resultados complejos.





# Funciones sin retorno



## ¿Qué son las funciones sin retorno?

Las funciones sin retorno son aquellas que no devuelven un valor después de ejecutarse. Su propósito principal es realizar una acción, como imprimir un resultado o modificar variables globales. A menudo se utilizan para realizar tareas útiles sin necesidad de devolver información.



## Usos comunes

Se utilizan comúnmente para realizar tareas repetitivas o de mantenimiento. También son prácticas en el manejo de entradas del usuario. Ayudan a organizar el código de manera efectiva y modular.



## Ejemplo de una función sin retorno

Un ejemplo sencillo podría ser una función que imprime un saludo en la consola. Al llamarla, simplemente ejecutará su bloque de código. No es necesario almacenar un valor después de su ejecución.



# Funciones con múltiples valores de retorno

## Definición

En Python, una función puede devolver varios valores al mismo tiempo. Esto se logra utilizando tuplas. Las tuplas permiten agrupar múltiples datos en un solo valor de retorno.

## Utilidad

Devolver múltiples valores mejora la eficacia de las funciones. Permite enviar información relacionada sin crear estructuras adicionales. Esto simplifica el código y mejora la legibilidad.

## Ejemplo clave

Un ejemplo común es una función que devuelve el área y perímetro de una figura. Esta función puede calcular ambos y devolverlos simultáneamente, facilitando su uso.

## Conclusión

Las funciones con múltiples valores de retorno son versátiles. Mejoran la eficiencia del código y son fáciles de implementar. Su uso es recomendado en programación de alto nivel.

# Alcance de Variables en Funciones



## Alcance Local

Las variables definidas dentro de una función tienen un alcance local. Esto significa que solo son accesibles dentro de esa función. Al salir de la función, estas variables son destruidas y no pueden ser usadas.



## Alcance Global

Las variables globales se pueden acceder desde cualquier parte del código. Esto incluye funciones en cualquier lugar. Sin embargo, es mejor limitar su uso para evitar conflictos.



## Variables Anidadas

Las funciones anidadas tienen acceso a variables de las funciones externas. Esto crea un contexto más amplio para el manejo de datos. Sin embargo, puede complicar el diseño del código si no se maneja adecuadamente.







# Variables locales y globales

## Variables Locales

Las variables locales son aquellas definidas dentro de una función. Su alcance se limita al bloque de la función donde se definen. Esto significa que no pueden ser accedidas desde fuera de esa función, lo que ayuda a evitar conflictos de nombres.

## Variables Globales

Las variables globales se definen fuera de cualquier función y pueden ser accesibles en cualquier parte del código. Su uso puede ser útil para compartir información entre funciones. Sin embargo, el abuso de variables globales puede dificultar la lectura y el mantenimiento del código.





# Funciones Anidadas

## Definición

Las funciones anidadas son funciones definidas dentro de otras funciones. Este concepto permite organizar el código y mejorar la legibilidad. Las funciones anidadas pueden acceder a las variables locales de su función externa.

## Ventajas

Una de las ventajas de usar funciones anidadas es el encapsulamiento. Esto ayuda a evitar conflictos de nombres y a mantener el código modular. Además, pueden ser útiles para tareas específicas dentro de la función externa.

## Ejemplo Simple

Considera una función que calcula un valor y llama a una función anidada para realizar una operación auxiliar. Esto evita la repetición de código y mejora la eficiencia. Mantiene el enfoque en la tarea principal.





# Ejercicio 1: Declaración de una función simple

1

## Concepto de función simple

Una función simple es un bloque de código que realiza una tarea específica. Se define utilizando la palabra clave `def` seguida del nombre de la función. Esto permite organizar mejor el código y hacer que sea más reutilizable.

2

## Creación de una función

Para declarar una función, primero elige un nombre descriptivo. Añade paréntesis para indicar los parámetros si es necesario. Así, puedes diseñar la función para que realice cálculos o manipule datos según la lógica que definas.

3

## Ejemplo práctico

Considera una función que calcule el cuadrado de un número. La función toma el número como argumento y retorna su cuadrado. Este ejercicio es útil para entender la sintaxis y estructura de las funciones en Python.

```
3  ### 1. Declaración de una función simple
4
5  ```python
6  def saludar():
7      print("¡Hola, mundo!")
8
9  # Llamada a la función
10 saludar()
11 ```
```

# Ejercicio 2: Función con parámetros

1

## Definir la función

Comienza escribiendo la palabra clave **def**, seguida del nombre de la función. Luego, abre paréntesis y define los parámetros que necesitará la función. Por ejemplo, si necesitas sumar dos números, los parámetros podrían ser **a** y **b**.

2

## Implementar la lógica

Dentro del cuerpo de la función, implementa la lógica que utilizarás. En el caso de la suma, simplemente debes retornar la suma de los parámetros **a** y **b**. Esta es la parte donde decides qué hace tu función.

3

## Ejecutar la función

Finalmente, llama a la función pasando los argumentos necesarios. Por ejemplo, si quieres sumar 3 y 5, escribirías **mi\_funcion(3, 5)**. Esto ejecutará la lógica que has definido.





# Ejercicio 3: Función con retorno de valores

1

## Definir la función

Inicia creando la función con la palabra clave def.

2

## Implementar el retorno

Usa la palabra clave return para devolver un valor.

3

## Probar la función

Ejecuta la función y verifica el valor devuelto.

En este ejercicio, aprenderás a declarar una función que retorne un valor. Primero, define la función utilizando la palabra clave "def" junto con su nombre y parámetros. Luego, implementa el retorno del valor deseado usando "return".

Finalmente, es crucial probar la función para asegurarte de que devuelve correctamente el valor esperado. Realizar estos pasos te permitirá comprender el concepto de retorno en Python.

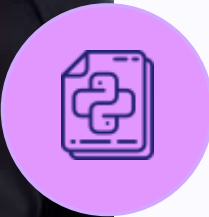


# Ejemplos: Función con parámetros, Retornos de valores

```
13  ### 2. Función con parámetros
14
15  ```python
16  def saludar(nombre):
17      print(f"¡Hola, {nombre}!")
18
19  # Llamada a la función con un argumento
20  saludar("Juan")
21  ```
22
23  ### 3. Función con múltiples parámetros
24
25  ```python
26  def sumar(a, b):
27      return a + b
28
29  # Llamada a la función con dos argumentos
30  resultado = sumar(3, 5)
31  print(f"El resultado de la suma es: {resultado}")
32  ```
33
34  ### 4. Función con valores por defecto para los parámetros
35
36  ```python
37  def saludar(nombre="mundo"):
38      print(f"¡Hola, {nombre}!")
39
40  # Llamada a la función sin argumento
41  saludar()
42
43  # Llamada a la función con un argumento
44  saludar("Ana")
45  ```
```

```
47  ### 5. Función que retorna multiples valores
48
49  ```python
50  def operaciones(a, b):
51      suma = a + b
52      resta = a - b
53      multiplicacion = a * b
54      division = a / b
55      return suma, resta, multiplicacion, division
56
57  # Llamada a la función y desempquetado de los valores retornados
58  resultado_suma, resultado_resta, resultado_multiplicacion, resultado_division = operaciones(10, 5)
59  print(f"Suma: {resultado_suma}")
60  print(f"Resta: {resultado_resta}")
61  print(f"Multiplicación: {resultado_multiplicacion}")
62  print(f"División: {resultado_division}")
63  ```
64
65  ### 6. Función con parámetros opcionales (args y kwargs)
66
67  ```python
68  def informacion_personal(nombre, edad, *args, **kwargs):
69      print(f"Nombre: {nombre}")
70      print(f"Edad: {edad}")
71      for arg in args:
72          print(f"Arg adicional: {arg}")
73      for clave, valor in kwargs.items():
74          print(f"{clave}: {valor}")
75
76  # Llamada a la función con argumentos adicionales
77  informacion_personal("Juan", 30, "Ingeniero", ciudad="Madrid", pais="España")
78  ```
```





# Casos de uso comunes de funciones



## Organización del Código

Las funciones permiten organizar el código en bloques manejables. Facilitan la lectura y el mantenimiento. Esto es esencial en proyectos grandes para evitar confusiones.



## Reutilización de Código

Las funciones promueven la reutilización de código, evitando duplicaciones. Puedes llamar a la misma función múltiples veces. Esto ahorra tiempo y recursos durante el desarrollo.



## Pruebas y Depuración

Las funciones hacen que las pruebas sean más simples. Puedes probar funciones independientes sin afectar el resto del código. Esto mejora la calidad del software.





# Ventajas de utilizar funciones



## Eficiencia

Las funciones permiten reutilizar código. Esto ahorra tiempo y esfuerzo al programar. Se pueden modificar y actualizar una sola vez, afectando todas las instancias.



## Claridad

Las funciones facilitan la comprensión del código. Dividen las tareas en partes más pequeñas y manejables. Esto hace que el código sea más fácil de leer y entender.



## Mantenimiento

El uso de funciones simplifica el mantenimiento del código. Los errores se pueden localizar y corregir más fácilmente. Además, se pueden agregar nuevas características sin reescribir todo el código.



# Buenas prácticas en la declaración de funciones

## Nombre descriptivo

Utiliza nombres claros para las funciones. Un buen nombre describe la acción que realiza. Esto facilita la lectura y mantenimiento del código.

## Cohesión

Las funciones deben tener un único propósito bien definido. Esto las hace más fáciles de probar y reutilizar. Mantén las funciones pequeñas y enfocadas.

## Documentación

Incluye comentarios y documentación en cada función. Esto ayuda a otros programadores a entender el propósito y funcionamiento. Una buena documentación es esencial para la colaboración efectiva.

## Manejo de excepciones

Implementa manejo de errores en tus funciones. Esto asegura que tu programa pueda responder adecuadamente a situaciones imprevistas. Un buen manejo de excepciones mejora la robustez del código.

# Resumen y Conclusiones

## Importancia de las funciones

Las funciones en Python permiten organizar y reutilizar el código. Facilitan la lectura y comprensión del programa. Al dividir tareas complejas en funciones más simples, se mejora la mantenibilidad.

## Flexibilidad y eficiencia

El uso de funciones hace que el código sea más flexible. Las funciones pueden recibir diferentes parámetros y valores de retorno. Esto optimiza la ejecución y reduce la redundancia.

## Mejores prácticas

Es esencial seguir buenas prácticas al declarar funciones. Nombres descriptivos y documentación son fundamentales. Esto asegura que tu código sea claro y fácil de entender por otros desarrolladores.







# Ejercicios de Práctica



# ¡Gracias

**Por ser parte de esta  
Experiencia de aprendizaje!**