Tema 7: Funciones

Lectura fácil

En Python, una **función** es un bloque de código que se puede reutilizar. Las funciones nos permiten dividir un programa en pequeñas partes, cada una con una tarea específica. Así, el código se vuelve más organizado, fácil de leer y de mantener.

Definición de funciones

Para **definir** una función en Python, usamos la palabra clave **def**, seguida del nombre de la función, paréntesis **()**, y dos puntos **:**

Sintaxis básica:

```
def nombre_de_la_funcion():
    # Código que realiza la función
```

- **def**: Esta palabra clave le dice a Python que estamos definiendo una función.
- **nombre_de_la_funcion**: Es el nombre que le damos a la función, y puede ser cualquier nombre que describa lo que hace.
- Dentro del bloque de la función (debajo del **def**), escribimos el código que queremos que ejecute la función.

Ejemplo básico de una función:

```
def saludar():
    print("¡Hola!")
```

- Esta función se llama **saludar**. Cada vez que llamemos a esta función, Python imprimirá "¡Hola!" en la pantalla.
- Para usar (o **llamar**) la función, simplemente escribimos su nombre seguido de paréntesis:

```
saludar()
```

- Esto mostrará en pantalla:
  ¡Hola!

Parámetros y argumentos

Las funciones pueden tomar **parámetros** para trabajar con diferentes datos. Los parámetros son **variables** que se pasan a la función cuando la llamamos, y se utilizan dentro de la función para realizar alguna operación.

Sintaxis básica con parámetros:

```
def nombre_de_la_funcion(parametro1, parametro2):
    # Código que usa los parámetros
```

- **parametro1**, **parametro2**, etc., son los valores que pasamos a la función cuando la llamamos.

Ejemplo de función con parámetros:

```
def saludar(nombre):
    print("¡Hola, " + nombre + "!")
```

- En este ejemplo, la función **saludar** tiene un parámetro llamado **nombre**.
- Cuando llamamos a la función, le pasamos un valor para el parámetro:

```
saludar("Carlos")
```

- El programa mostrará:
  ¡Hola, Carlos!
- Si llamamos a la función con otro nombre, el resultado será diferente:

```
saludar("Ana")
```

- El programa mostrará:
  ¡Hola, Ana!

Retorno de valores

A veces, queremos que una función no solo ejecute un bloque de código, sino que también **devuelva** (retorne) un valor después de hacer una tarea. Para devolver un valor desde una función, usamos la palabra clave **return**.

Sintaxis básica con return:

```
def nombre_de_la_funcion():
    # Código
    return valor
```

- **return valor**: Devuelve un valor al lugar donde se llamó a la función.

Ejemplo de función que retorna un valor:

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

- Esta función toma dos parámetros **a** y **b**, los suma y devuelve el resultado.
- Para usar el valor que retorna la función, llamamos a la función y guardamos el resultado en una variable:

```
resultado = sumar(3, 5)
print(resultado)
```

- El programa mostrará:
  8

Ejemplos de funciones

Vamos a ver más ejemplos prácticos para entender mejor cómo usar las funciones.

Ejemplo 1: Función que calcula el área de un triángulo

```
def area_triangulo(base, altura):
     area = (base * altura) / 2
      return area
```

- Esta función toma dos parámetros: **base** y **altura**, y devuelve el área del triángulo.
- Para calcular el área de un triángulo con base 4 y altura 5, llamamos a la función así:

```
resultado = area_triangulo(4, 5)
print("El área del triángulo es", resultado)
```

- El programa mostrará:
  El área del triángulo es 10.0

Ejemplo 2: Función que verifica si un número es par

```
def es_par(numero):
    if numero % 2 == 0:
        return True
    else:
        return False
```

- Esta función toma un número como parámetro y devuelve **True** si el número es par, o **False** si es impar.
- Para verificar si el número 8 es par, llamamos a la función así:

```
resultado = es_par(8)
print("¿El número es par?", resultado)
```

- El programa mostrará:
  ¿El número es par? True

Ejemplo 3: Función que saluda de manera personalizada

```
def saludo_personalizado(nombre, hora_del_dia):
    if hora_del_dia == "mañana":
        print("¡Buenos días, " + nombre + "!")
    elif hora_del_dia == "tarde":
        print("¡Buenas tardes, " + nombre + "!")
    else:
        print("¡Buenas noches, " + nombre + "!")
```

- Esta función toma dos parámetros: **nombre** y **hora_del_dia** (que puede ser "mañana", "tarde" o "noche").
- Dependiendo de la hora del día, la función imprime un saludo personalizado.

```
saludo_personalizado("Laura", "tarde")
```

- El programa mostrará:
  ¡Buenas tardes, Laura!

Ventajas de usar funciones

Las funciones tienen muchas ventajas:

1. **Reutilización del código**: Una vez que defines una función, puedes usarla muchas veces en diferentes partes de tu programa sin tener que escribir el mismo código una y otra vez.
   - **Ejemplo**: Si necesitas calcular el área de varios triángulos en tu programa, puedes llamar a la función **area_triangulo()** cada vez que la necesites, en lugar de escribir la fórmula repetidamente.
2. **Organización**: Dividir un programa en funciones hace que el código sea más organizado y fácil de entender.

- **Ejemplo**: En lugar de tener un largo bloque de código, puedes separar las tareas en diferentes funciones, como **calcular_area()**, **verificar_par()**, **imprimir_saludo()**, etc.

3. **Facilidad de mantenimiento**: Si algo necesita cambiar en el programa, puedes modificar solo la función correspondiente sin afectar el resto del código.

Las **funciones** son una parte esencial de la programación en Python. Nos permiten organizar el código, reutilizarlo y hacerlo más eficiente. Ahora que entendemos cómo definir funciones, pasar parámetros y devolver valores, podemos empezar a escribir programas más complejos y mejor organizados.
