Funciones en Python

Construyendo código reutilizable y profesional

Python Full Stack - Clase 15

🎯 Objetivos de Hoy

¿Qué aprenderemos?

- 🧩 Comprender qué son las funciones y por qué son esenciales
- 📝 Dominar la sintaxis básica de funciones en Python
- 🔧 Entender parámetros, argumentos y valores de retorno
- ⚙️ Usar parámetros por defecto y argumentos con palabras clave
- ✍️ Documentar funciones con docstrings
- 🏷️ Aplicar type hints para código más profesional
- 🎨 Aplicar buenas prácticas al crear funciones

¿Qué son las Funciones?

🧩 ¿Qué son las Funciones?

¿Cuándo necesito esto?

Escenario: "Necesito calcular el área de un círculo 50 veces en mi programa"

Una función es un bloque de código reutilizable que:

- ✅ Tiene un nombre descriptivo
- ✅ Puede recibir datos (parámetros)
- ✅ Ejecuta una serie de instrucciones
- ✅ Puede devolver un resultado (return)

🍳 Analogía: Funciones como Recetas de Cocina

📋 RECETA: "Preparar Pizza"

```
Ingredientes (parámetros):
- Masa
- Queso
- Tomate

Instrucciones:
1. Extender la masa
2. Agregar ingredientes
3. Hornear

Resultado (return): Pizza lista
```

Cada vez que necesites pizza → llamas a la función "preparar_pizza"

🎯 Ventajas de las Funciones

¿Por qué usar funciones?

1. 🔄 Reducción de código duplicado

```
# ❌ Sin funciones: repetir código
area1 = 3.14159 * 5 * 5
area2 = 3.14159 * 10 * 10
area3 = 3.14159 * 15 * 15

# ✅ Con funciones: código reutilizable
def calcular_area(radio):
    return 3.14159 * radio * radio

area1 = calcular_area(5)
area2 = calcular_area(10)
area3 = calcular_area(15)
```

🎯 Ventajas de las Funciones

¿Por qué usar funciones?

2. 🧩 **Desglose de problemas complejos**
   - Dividir un problema grande en partes pequeñas
   - Cada función resuelve una tarea específica
3. 📖 **Mejor claridad y legibilidad**
   - Nombres descriptivos explican qué hace el código
   - Código más fácil de entender y mantener

📝 Sintaxis Básica de Funciones

📝 Sintaxis Básica de Funciones

```
def nombre_de_la_funcion(parametro1, parametro2):
    """
    Documentación de la función (docstring)
    """
    # Instrucciones aquí
    resultado = parametro1 + parametro2
    return resultado
```

Elementos clave:

- `def` → Palabra reservada para definir funciones
- `nombre_de_la_funcion` → Nombre descriptivo (snake_case)
- `(parametro1, parametro2)` → Parámetros (opcionales)
- `return` → Devuelve un valor (opcional)

💻 Ejemplo: Función de Multiplicación

Función completa paso a paso:

```
# Paso 1: Definir la función
def multiplicacion(num1, num2):
    """
    Multiplica dos números y devuelve el resultado.

    Args:
        num1: Primer número a multiplicar
        num2: Segundo número a multiplicar

    Returns:
        El producto de num1 y num2
    """
    resultado = num1 * num2  # Instrucción dentro de la función
    return resultado          # Devolver el resultado
```

⚠️ **Importante:** La función NO se ejecuta hasta que la llamamos

💻 Ejemplo: Llamar una Función

Cómo usar la función:

```
# Definimos la función
def multiplicacion(num1, num2):
    resultado = num1 * num2
    return resultado

# Llamamos (invocamos) la función
resultado_multiplicacion = multiplicacion(5, 5)
print(resultado_multiplicacion)  # Imprime: 25

# Podemos usar el resultado en otros cálculos
total = resultado_multiplicacion + 10
print(total)  # Imprime: 35
```

🎯 **La llamada a la función se reemplaza por su valor de retorno**

💻 Ejemplo: Función Simple sin Parámetros

Funciones que no necesitan entrada:

```
def saludar():
    """
    Imprime un saludo genérico.
    """
    print("¡Hola! Bienvenido al curso de Python")

# Llamar la función
saludar()  # Imprime: ¡Hola! Bienvenido al curso de Python
saludar()  # Podemos llamarla cuantas veces queramos
saludar()  # Siempre ejecuta las mismas instrucciones
```

✅ **No necesitan argumentos porque no tienen parámetros**

🔧 Parámetros y Argumentos

🔧 Parámetros vs Argumentos

¿Cuál es la diferencia?

📥 **Parámetros:** Son las "variables" que la función espera recibir

- Se definen en la declaración de la función
- Ejemplo: `def saludar(nombre):` → `nombre` es un parámetro

📤 **Argumentos:** Son los valores reales que enviamos a la función

- Se pasan cuando llamamos la función
- Ejemplo: `saludar("María")` → `"María"` es un argumento

💻 Ejemplo: Parámetros y Argumentos

Ejemplo práctico:

```
# Definimos la función con PARÁMETROS
def buenos_dias(nombre):  # 'nombre' es el PARÁMETRO
    print("Buenos días " + nombre)

# Llamamos la función con ARGUMENTOS
buenos_dias("alegría")     # "alegría" es el ARGUMENTO
buenos_dias("al amor")     # "al amor" es el ARGUMENTO
buenos_dias("a la vida")   # "a la vida" es el ARGUMENTO
buenos_dias("señor Sol")   # "señor Sol" es el ARGUMENTO
```

📝 **Resumen:**

- **Parámetro** = Lo que recibe la función (en la definición)
- **Argumento** = Lo que enviamos a la función (en la llamada)

💻 Ejemplo: Múltiples Parámetros

```
def calcular_promedio(num1, num2, num3):
    """
    Calcula el promedio de tres números.

    Args:
        num1: Primer número
        num2: Segundo número
        num3: Tercer número

    Returns:
        El promedio de los tres números
    """
    suma = num1 + num2 + num3
    promedio = suma / 3
    return promedio
# Llamar con múltiples argumentos
promedio = calcular_promedio(10, 20, 30)
print(promedio)  # Imprime: 20.0
```

⚠️ **El orden de los argumentos debe coincidir con el orden de los parámetros**

La Sentencia Return

🔄 La Sentencia Return

¿Qué hace return?

La sentencia `return` permite que la función devuelva un valor

```
def suma(a, b):
    resultado = a + b
    return resultado  # Devuelve el valor

# Sin return:
def suma_sin_return(a, b):
    resultado = a + b
    # No hay return → devuelve None implícitamente
```

🎯 **Regla importante:** La llamada a una función es igual a lo que la función retorna

💻 Ejemplo: Return vs Print

```
# Opción 1: Usar print (NO devuelve valor)
def saludar_print(nombre):
    print("Buenos días " + nombre)
    # No hay return → devuelve None

resultado = saludar_print("Python")
print(resultado)  # Imprime: None

# Opción 2: Usar return (devuelve valor)
def saludar_return(nombre):
    return "Buenos días " + nombre

mensaje = saludar_return("Python")
print(mensaje)  # Imprime: Buenos días Python
```

✅ Usa `return` cuando necesites usar el resultado fuera de la función

💻 Ejemplo: Return Múltiple

```python
# Devolver un número
def multiplicar(a, b):
    return a * b

# Devolver una cadena
def crear_saludo(nombre):
    return f"Hola, {nombre}"

# Devolver una lista
def crear_lista_numeros():
    return [1, 2, 3, 4, 5]

# Devolver un diccionario
def crear_perfil():
    return {"nombre": "Juan", "edad": 25}
```

✅ **Las funciones pueden devolver cualquier tipo de dato en Python**

⚠️ Errores Comunes con Return

Problema 1: Código después de return

```
def ejemplo():
    return 5
    print("Esto nunca se ejecuta")  # ❌ Código inalcanzable
    return 10  # ❌ Nunca se ejecuta

resultado = ejemplo()
print(resultado)  # Imprime: 5
```

⚠️ Todo el código después de `return` es ignorado

⚠️ Errores Comunes con Return

Problema 2: Intentar usar None como número

```
def suma_sin_return(a, b):
    print(a + b)  # Solo imprime, no devuelve

resultado = suma_sin_return(5, 3)  # resultado = None
total = resultado + 10  # ❌ Error: None + 10 no es válido
```

✅ **Solución:** Usar `return` cuando necesites el valor

```
def suma_con_return(a, b):
    return a + b  # Devuelve el valor

resultado = suma_con_return(5, 3)  # resultado = 8
total = resultado + 10  # ✅ Funciona: 8 + 10 = 18
```

💪 Ejercicio: Predicción de Salida

Intenta predecir qué se imprimirá en cada caso:

```
# Ejercicio 1
def cantidad_de_vocales():
    return 5

print(cantidad_de_vocales())
```

¿Qué imprime?

- A) 5
- B) None
- C) Error

💪 Ejercicio: Predicción de Salida

¡Hora de practicar!

```
# Ejercicio 2
def altura_machu_picchu():
    print(2438)

x = altura_machu_picchu()
print(x)
```

¿Qué imprime?

- A) 2438 y luego 2438
- B) 2438 y luego None
- C) Error

⏱️ **Tiempo:** 1 minuto

⚙ Parámetros por Defecto

⚙️ Parámetros por Defecto

¿Cuándo necesito esto?

Escenario: "Quiero una función que salude, pero si no doy nombre, que use un nombre por defecto"

Los parámetros por defecto permiten que algunos parámetros sean opcionales

```
def buenos_dias(nombre="alegría", cantidad=1):
    """
    Saluda con buenos días un número específico de veces.

    Args:
        nombre: Nombre a saludar (por defecto: "alegría")
        cantidad: Número de veces a repetir (por defecto: 1)
    """
    print(("Buenos días " + nombre) * cantidad)
```

💻 Ejemplo: Parámetros por Defecto

Casos de uso:

```
def buenos_dias(nombre="alegría", cantidad=1):
    print(("Buenos días " + nombre) * cantidad)

# Caso 1: Sin argumentos → usa valores por defecto
buenos_dias()  # Imprime: "Buenos días alegría"

# Caso 2: Solo primer argumento → segundo usa por defecto
buenos_dias("al amor")  # Imprime: "Buenos días al amor"

# Caso 3: Solo segundo argumento → especificamos con nombre
buenos_dias(cantidad=3)  # Imprime: "Buenos días alegría" 3 veces

# Caso 4: Ambos argumentos
buenos_dias("señor Sol", 2)  # Imprime: "Buenos días señor Sol" 2 veces
```

💻 Ejemplo: Argumentos con Palabras Clave

```
def crear_perfil(nombre, edad=25, ciudad="Madrid"):
    """
    Crea un perfil de usuario.

    Args:
        nombre: Nombre del usuario (obligatorio)
        edad: Edad del usuario (por defecto: 25)
        ciudad: Ciudad del usuario (por defecto: "Madrid")
    """
    return f"{nombre}, {edad} años, vive en {ciudad}"
# Con argumentos con palabras clave, el orden NO importa
perfil1 = crear_perfil("María", ciudad="Barcelona", edad=30)
perfil2 = crear_perfil(ciudad="Valencia", nombre="Juan", edad=28)
print(perfil1)  # Imprime: María, 30 años, vive en Barcelona
print(perfil2)  # Imprime: Juan, 28 años, vive en Valencia
```

✅ **Especificar por nombre hace el código más legible**

📋 Reglas de Parámetros por Defecto

Orden importante:

```
# ✅ CORRECTO: Parámetros con default al final
def funcion_correcta(a, b, c=10):
    return a + b + c

# ❌ INCORRECTO: Parámetro con default antes de uno sin default
def funcion_incorrecta(a, b=10, c):  # Error de sintaxis
    return a + b + c
```

Regla: Los parámetros con valores por defecto deben ir **después** de los parámetros sin valores por defecto

✍️ Documentación de Funciones (Docstrings)

✍️ Documentación de Funciones

Las docstrings explican qué hace tu función sin tener que leer el código

```
def calcular_area_circulo(radio):
    """
    Calcula el área de un círculo dado su radio.

    Args:
        radio (float): El radio del círculo en unidades

    Returns:
        float: El área del círculo calculada como π * radio²

    Example:
        >>> calcular_area_circulo(5)
        78.53975
    """
    pi = 3.14159
    return pi * radio * radio
```

✅ **Buena práctica:** Siempre documenta tus funciones

📝 Formato de Docstrings

Estructura recomendada:

```
def nombre_funcion(parametro1, parametro2):
    """
    Descripción breve de qué hace la función.

    Descripción más detallada si es necesario.
    Puede ocupar múltiples líneas.

    Args:
        parametro1 (tipo): Descripción del primer parámetro
        parametro2 (tipo): Descripción del segundo parámetro

    Returns:
        tipo: Descripción de lo que devuelve

    Raises:
        TypeError: Si los parámetros son del tipo incorrecto

    Example:
        >>> nombre_funcion(5, 10)
        15
    """
    # Código de la función
    pass
```

💻 Ejemplo: Función Bien Documentada

Comparación:

```python
# ❌ Sin documentación (difícil de entender)
def calc(a, b):
    return a * b + a

# ✅ Con documentación (clara y profesional)
def calcular_costo_total(precio_unitario, cantidad):
    """
    Calcula el costo total incluyendo el precio unitario.

    Esta función multiplica el precio unitario por la cantidad
    y suma el precio unitario adicional (como costo de procesamiento).

    Args:
        precio_unitario (float): Precio de cada unidad
        cantidad (int): Número de unidades a comprar

    Returns:
        float: Costo total calculado

    Example:
        >>> calcular_costo_total(10.5, 3)
        31.5
    """
    return precio_unitario * cantidad + precio_unitario
```

🏷️ Type Hints (Tipado)

🏷️ Type Hints

Los type hints indican qué tipo de datos espera y devuelve una función

```python
def suma(a, b): # Sin type hints (Python tradicional)
    return a + b

# Con type hints (más profesional)
def suma(a: int, b: int) -> int:
    """
    Suma dos números enteros.

    Args:
        a: Primer número entero
        b: Segundo número entero

    Returns:
        La suma de a y b
    """
    return a + b
```

✅ **Beneficios:** Código más claro, mejores herramientas de desarrollo

💻 Ejemplo: Type Hints Completos

```python
def saludar(nombre: str) -> str:
    """
    Crea un saludo personalizado.

    Args:
        nombre: Nombre de la persona a saludar

    Returns:
        Mensaje de saludo personalizado
    """
    return f"Hola, {nombre}"

def calcular_promedio(numeros: list[float]) -> float:
    """
    Calcula el promedio de una lista de números.

    Args:
        numeros: Lista de números

    Returns:
        El promedio de los números
    """
    return sum(numeros) / len(numeros)
```

💻 Ejemplo: Type Hints Avanzados

```python
from typing import Optional, List, Dict

def buscar_usuario(
    nombre: str,
    edad: Optional[int] = None
) -> Dict[str, any]:
    """
    Busca información de un usuario.

    Args:
        nombre: Nombre del usuario
        edad: Edad del usuario (opcional)

    Returns:
        Diccionario con información del usuario
    """
    usuario = {"nombre": nombre}
    if edad is not None:
        usuario["edad"] = edad
    return usuario
```

💡 **Tip:** Para principiantes, documentación es más importante que type hints

💻 Ejemplo: Función Completa con Buenas Prácticas

Todo junto:

```python
def calcular_descuento(
    precio: float,
    porcentaje: float = 10.0
) -> float:
    """
    Calcula el precio con descuento aplicado.

    Args:
        precio: Precio original del producto
        porcentaje: Porcentaje de descuento (por defecto: 10.0)

    Returns:
        Precio final después del descuento

    Example:
        >>> calcular_descuento(100, 20)
        80.0
    """
    descuento = precio * (porcentaje / 100)
    return precio - descuento

# Uso
precio_final = calcular_descuento(100, 20)
print(precio_final)  # Imprime: 80.0
```

💪 Ejercicio: Crear Funciones Básicas

¡Hora de practicar!

Crea un archivo `funciones_basicas.py` con:

1. **Multiplica por 2:** Crea una función que reciba un número y devuelva una lista con los números enteros multiplicados por dos, desde el 0 hasta el número proporcionado.
    - Ejemplo: `multiplica_por_2(5)` debe regresar `[0, 2, 4, 6, 8, 10]`
2. **Suma y resta:** Crea una función que reciba una lista con dos números. Imprime la suma y regresa la resta.
    - Ejemplo: `suma_y_resta([5, 4])` debe imprimir `9` y regresar `1`

Pistas:

- Usa un bucle `for` con `range()` para generar números
- Recuerda usar `return` para devolver valores

⏱️ **Tiempo:** 15 minutos

Buenas Prácticas

🎨 Buenas Prácticas para Funciones

1. Nombres descriptivos

```
# ❌ Nombres poco claros
def calc(x, y):
    return x * y

def f(a):
    return a + 1

# ✅ Nombres descriptivos
def calcular_multiplicacion(numero1, numero2):
    return numero1 * numero2

def incrementar_contador(contador):
    return contador + 1
```

✅ Usa nombres que expliquen qué hace la función

🎨 Buenas Prácticas para Funciones

2. Una función, una responsabilidad

```
# ❌ Función que hace demasiadas cosas
def procesar_usuario(nombre, edad, email):
    # Validar datos
    if len(nombre) < 3:
        return "Error"
    # Crear usuario
    usuario = {"nombre": nombre}
    # Enviar email
    print(f"Email enviado a {email}")
    return usuario
# ✅ Funciones separadas por responsabilidad
def validar_nombre(nombre: str) -> bool:
    return len(nombre) >= 3
def crear_usuario(nombre: str, edad: int) -> dict:
    return {"nombre": nombre, "edad": edad}
```

🎨 Buenas Prácticas para Funciones

3. Usar return en lugar de print (cuando corresponda)

```
# ❌ Usar print cuando necesitas el valor
def calcular_total(precio, cantidad):
    print(precio * cantidad)  # No puedo usar el resultado

total = calcular_total(10, 5)  # total = None
nuevo_total = total + 10  # ❌ Error

# ✅ Usar return cuando necesitas el valor
def calcular_total(precio, cantidad):
    return precio * cantidad  # Puedo usar el resultado

total = calcular_total(10, 5)  # total = 50
nuevo_total = total + 10  # ✅ Funciona: 60
```

🎨 Buenas Prácticas para Funciones: Documentar

```python
def proc(x): # ❌ Sin documentación
    return x * 2

def duplicar_numero(numero: float) -> float: # ✅ Con documentación
    """
    Duplica un número dado.

    Args:
        numero: Número a duplicar

    Returns:
        El número multiplicado por 2

    Example:
        >>> duplicar_numero(5)
        10
    """
    return numero * 2
```

¡Hora de practicar!

Crea un archivo `funciones_intermedias.py` con:

1. **Iterar diccionario:** Crea una función que reciba una lista de diccionarios e imprima cada llave y valor.
2. **Obtener valores:** Crea una función que reciba una llave y una lista de diccionarios, e imprima el valor de esa llave en cada diccionario.

Ejemplo de datos:

```
cantantes = [
    {"nombre": "Ricky Martin", "pais": "Puerto Rico"},
    {"nombre": "Chayanne", "pais": "Puerto Rico"}
]
```

Pistas:

- Usa un bucle `for` para iterar la lista

Resumen de Conceptos Clave

📚 Resumen de Conceptos Clave

1. Funciones básicas

- `def` para definir funciones
- Parámetros entre paréntesis
- `return` para devolver valores
- Funciones no se ejecutan hasta ser llamadas

2. Parámetros y argumentos

- **Parámetro:** Variable en la definición
- **Argumento:** Valor en la llamada
- Pueden ser múltiples y de cualquier tipo

📚 Resumen de Conceptos Clave

4. Parámetros por defecto

- Hacen parámetros opcionales
- Van después de parámetros sin default
- Se pueden especificar por nombre

3. Return

- Devuelve un valor que puede usarse fuera
- Sin `return`, la función devuelve `None`
- Código después de `return` no se ejecuta

📚 Resumen de Conceptos Clave

5. Documentación (Docstrings)

- Explican qué hace la función
- Van entre comillas triples `"""`
- Incluyen Args, Returns, Examples

6. Type Hints

- Indican tipos de parámetros y retorno
- Formato: `def func(param: tipo) -> tipo_retorno`
- Hacen el código más profesional

🎯 Lo Más Importante

Conceptos que debes recordar:

1. ✅ Las funciones encapsulan código reutilizable
2. ✅ Usa `return` cuando necesites usar el resultado
3. ✅ Documenta tus funciones con docstrings
4. ✅ Nombres descriptivos hacen el código más claro
5. ✅ Una función debe tener una responsabilidad clara
6. ✅ Parámetros con default van al final

📖 Recursos para Seguir Aprendiendo

Documentación oficial:

- Python Functions
- Type Hints
- PEP 257 - Docstring Conventions

Práctica interactiva:

- Python Tutor - Visualiza la ejecución de funciones
- Exercism Python Track - Ejercicios prácticos

🚀 Próximos Pasos

En la siguiente clase veremos:

- 🎯 Más sobre funciones avanzadas
- 🔁 Funciones lambda
- 📦 Módulos y paquetes
- 🎨 Programación orientada a objetos

¡Sigue practicando funciones antes de continuar!

📝 Práctica para Casa

Proyecto sugerido:

Crea un sistema de funciones para una calculadora básica:

1. `sumar(a, b)` - Suma dos números
2. `restar(a, b)` - Resta dos números
3. `multiplicar(a, b)` - Multiplica dos números
4. `dividir(a, b)` - Divide dos números (maneja división por cero)
5. `calcular_promedio(numeros)` - Calcula el promedio de una lista

Requisitos:

- ✅ Todas las funciones deben tener docstrings
- ✅ Usa type hints donde sea posible
- ✅ Usa parámetros por defecto cuando tenga sentido
- ✅ Prueba todas las funciones con diferentes valores

💡 Consejos Finales

Buenas prácticas para recordar:

1. 📝 **Siempre documenta:** Las docstrings son tu mejor amigo
2. 🎯 **Nombres claros:** Si necesitas comentar qué hace, el nombre es malo
3. 🔄 **Reutiliza código:** Si lo escribes 3 veces, hazlo función
4. ✨ **Mantén funciones pequeñas:** Más fácil de entender y depurar
5. 🧪 **Prueba tus funciones:** Verifica que funcionen con diferentes casos

¡El código limpio es código profesional!

🎉 ¡Felicidades!

Ya sabes crear funciones en Python

Dominas la base para escribir código profesional y reutilizable

? Preguntas

¿Alguna duda sobre funciones?

¡No te quedes con dudas!

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con más Python!

No olvides completar todos los ejercicios 📝