6. Funciones

📝 Resumen de Funciones en Python para Apuntes (Versión Mejorada)

Las funciones son bloques de código reutilizable que realizan una tarea específica1. Son esenciales para organizar el código, hacerlo más legible y seguir el principio **DRY** (*Don't Repeat Yourself*)2.

1. Estructura y Conceptos Básicos

Una función se define con la palabra clave `def`, seguida del nombre, paréntesis para los parámetros, y dos puntos (`:`)3.

Componente	Descripción
`def`	Palabra clave obligatoria para empezar la definición.
Nombre	Debe ser descriptivo e intuitivo.
Parámetros	Variables que la función espera recibir (opcional, pero los paréntesis son obligatorios).
Cuerpo	El código indentado que se ejecuta al llamar a la función4.
`return`	Opcional. Indica que la función ha terminado y **devuelve un valor**. Sin `return`, la función devuelve `None`.

Python

```
def mi_funcion(parametro1): # parametro1 es un parámetro
    resultado = parametro1 * 0.21
    return resultado

# Al llamar a la función:
impuesto = mi_funcion(100) # 100 es el argumento
# impuesto es 21.0
```

🎯 Énfasis: Diferencia entre `return` y `print()`

Característica	`return`	`print()`
Propósito	**Devuelve** un valor para usarlo en el código (almacenar, calcular).	**Muestra** información en la consola.
Flujo de Control	**Detiene** la ejecución de la función y sale de ella.	La ejecución de la función **continúa**5.
Valor de Retorno	Devuelve el valor especificado (ej. número, lista, string)6.	Siempre devuelve `None` (si no hay un `return` explícito).

---

2. Ámbito de Variables (Scope)

El ámbito define dónde es accesible una variable7.

- **Variable Local:** Creada **dentro** de la función8. Solo existe mientras la función se ejecuta y desaparece de la memoria al terminar.
- **Variable Global:** Creada **fuera** de cualquier función. Es accesible desde cualquier parte del código (solo lectura por defecto). Se necesita la palabra clave `global` para modificarla dentro de una función9.

---

3. Parámetros y Argumentos Especiales

A. Argumentos Posicionales vs. por Palabra Clave

Cuando llamas a una función, puedes pasar los argumentos de dos maneras:

Tipo	Descripción	Ejemplo
Posicionales	Se asignan por el **orden estricto** en que se definen los parámetros10.	`restar(10, 5)`  `a=10`, `b=5`
Palabra Clave (Keyword)	Se asignan especificando explícitamente el nombre (`nombre=valor`). El **orden se ignora** y mejora la legibilidad.	`restar(b=5, a=10)`  `a=10`, `b=5`

B. Parámetros por Defecto (Valores Predefinidos)

Permiten que una función use un valor predefinido si el argumento no es proporcionado al llamar a la función.

⚠️ **Regla de Orden:** Los parámetros con valores por defecto deben ir **siempre al final** de la lista de parámetros.

Python

```
def calcular_descuento(precio, tasa=0.10): # 0.10 es el valor por defecto
    return precio * (1 - tasa)

print(calcular_descuento(200))          # Usa tasa=0.10
print(calcular_descuento(200, 0.25))     # Sobrescribe tasa=0.25
```

C. Argumentos Arbitrarios (`*args` y `**kwargs`)

Se usan cuando el número de entradas es variable11.

Tipo	Sintaxis	Recibido como...	Uso
Posicionales		`*args` 12	
Palabra Clave		`**kwargs` 15	

Python

```
# Ejemplo de *args
def sumar_todo(*numeros):
    return sum(numeros)

print(sumar_todo(1, 2, 3, 4)) # Output: 10

# Ejemplo de **kwargs
def configurar(**opciones):
    # opciones = {'color': 'rojo', 'tamanio': 'L'}
    return opciones

print(configurar(color="rojo", tamanio="L"))
```

➡️ Orden de Argumentos Combinados (Regla de Oro)

Cuando se combinan todos, el orden debe ser estricto:

Python

```
def funcion(obligatorio1, obligatorio2, *args, **kwargs,
parametro_por_defecto=None):
```

```
    # ...
    pass
```

---

4. Funciones Lambda (Funciones Anónimas)

Son funciones **pequeñas, anónimas y de una sola línea** que se definen sin la palabra clave `def`. Se usan para operaciones sencillas o con funciones de orden superior como `map()`, `filter()`, o `apply()`18.

Sintaxis: `lambda argumentos: expresión`19.

Python

```
# Lambda que calcula el doble de un número
calcular_doble = lambda x: x * 2

valores = [1, 2, 3]
dobles = list(map(calcular_doble, valores))
print(dobles) # Output: [2, 4, 6]
```

---

5. Recursividad

Es una función que se **llama a sí misma** durante su propia ejecución.

- Requiere una **condición de parada (caso base)** clara para evitar bucles infinitos.

Python

```
def factorial(n):
    # Caso base (condición de parada)
    if n == 0 or n == 1:
        return 1
    # Caso recursivo
    return n * factorial(n - 1)

print(factorial(5)) # Output: 120 (5 * 4 * 3 * 2 * 1)
```

---

6. Ejercicios Resueltos (Ejemplos de Aplicación)

Ejercicio 1: Limpieza y Reutilización (DRY)

Utiliza una función para limpiar una lista de nombres de clientes (quitar espacios y poner formato Título).

Python

```
def limpiar_nombres(lista):
    lista_limpia = []
    for nombre in lista:
        # Lógica de limpieza encapsulada
        lista_limpia.append(nombre.strip().title())
    return lista_limpia

datos_sucios = [" ana ", " maRía lópez ", " luisS gArCía "]
clientes_limpios = limpiar_nombres(datos_sucios)
# Se almacena el resultado gracias al 'return'
print(clientes_limpios)
# Output: ['Ana', 'María López', 'Luiss García']
```

Ejercicio 2: Cálculo de Media con `*args`

Función que calcula la media de cualquier número de notas, redondeando el resultado.

Python

```
def media_examenes(*notas):
    # 'notas' se recibe como tupla
    if not notas: # Caso de tupla vacía
        return 0
    return round(sum(notas) / len(notas))

nota_final = media_examenes(4, 5, 6, 7) # 4 notas
print(f"Media 1: {nota_final}") # Output: 6

otra_media = media_examenes(9, 8, 10) # 3 notas
print(f"Media 2: {otra_media}") # Output: 9
```