

U3.5

Funciones



Indice

Funciones en Python

- Las funciones son estructuras esenciales de código.
- Unidades lógicas del programa.
- Dividen y organizan el código para mayor legibilidad y reutilización.
- Agrupan instrucciones para resolver un problema concreto.
- Objetivos:
 - Dividir y organizar el código.
 - Encapsular código repetitivo.

Cómo definir una función en Python

- Utiliza la palabra reservada `def`.
- Nombre de la función seguido de paréntesis y lista de parámetros (opcional).
- Cabecera termina con dos puntos.
- Cuerpo de la función con sangrado mayor.
- Opcional: instrucción `return` para devolver un resultado.
- docstring se utiliza para documentar la función.

Cómo usar una función en Python

- Para invocarla, escribe su nombre como una instrucción.
- Pasa los argumentos según los parámetros definidos.

python [Copy code](#)

```
def multiplica_por_5(numero):  
    print(f'{numero} * 5 = {numero * 5}')    # 7 * 5 = 35  
  
multiplica_por_5(7)
```

Parámetros y argumentos

Diferencia entre parámetro y argumento

- Parámetro: definido en la función.
- Argumento: valor pasado a la función al ser invocada.

```
python Copy code
def multiplica_por_5(numero): # 'numero' es un parámetro
    print(f'{numero} * 5 = {numero * 5}')

multiplica_por_5(7) # '7' es un argumento
```

Note: Es importante distinguir entre ambos conceptos.

Parámetros

- Paso por valor: copia el valor de las variables.
- Paso por referencia: copia la dirección de memoria.

Note: En Python, se pasa por valor la referencia del objeto.

*args y **kwargs en Python

Significado de *args y **kwargs en Python

- Permiten funciones con un número variable de argumentos.
- Proporcionan flexibilidad en la cantidad y tipo de argumentos.

Note: `*args` y `**kwargs` permiten flexibilidad en las funciones.

Uso de *args

- `*args` permite argumentos sin palabras clave.


python  Copy code

```
def sumar(*args):  
    return sum(args) # sum es una función incorporada de Python  
  
print(sumar(3, 5, 10, 15)) # Imprime 33
```


Retorno de valores

Sentencia return

- El retorno de valores es opcional, puede devolver o no un valor.
- Termina la ejecución de la función y continúa el programa.

```
python  Copy code  
  
def cuadrado_de_par(numero):  
    if not numero % 2 == 0:  
        return  
    else:  
        print(numero ** 2)    # 64  
  
cuadrado_de_par(8)
```

Note: `return` puede usarse para finalizar una función y/o devolver un valor. Ten en cuenta que los ejemplos no muestran buenas prácticas de uso, sino que son para ilustrar el

Variables y ámbito

Ámbito y ciclo de vida de las variables

- Local: dentro de una función, no accesible fuera.
- Global: definidas fuera de funciones, visibles dentro.

```
python Copy code  
  
def muestra_x():  
    x = 10  
    print(f'x vale {x}') # x vale 10  
  
x = 20  
muestra_x()  
print(x) # 20
```

Note: Las variables tienen ámbitos locales o globales y ciclos de vida definidos.

Ámbito y ciclo de vida de las variables

Funciones Lambda en Python

- Funciones anónimas y pequeñas.
- No requieren definición con `def`.
- **Sintaxis:** `lambda argumentos: expresion.`

python  Copy code

```
doble = lambda x: x * 2
```

Resumen I

- Funciones en Python son unidades lógicas de código.
- Dividen y organizan el código, facilitando la reutilización.
- Se definen con `def`, seguido del nombre y parámetros (opcional).
- Se invocan escribiendo el nombre y pasando argumentos.
- `*args` para argumentos no clave.
- `*args` recoge en una tupla.

Resumen II

- `**kwargs` para argumentos clave.
- `**kwargs` recoge en un diccionario.
- Orden en la definición: `*args` primero, luego `**kwargs`.
- `return` opcional para devolver resultados.
- Ámbito de las variables: local y global.

¡Gracias por su atención!