ARGPARSE

📌 ¿Qué es `argparse` y por qué usarlo?

`argparse` es un módulo de Python que nos permite:

✅ Definir opciones de manera clara y organizada.

✅ Hacer que ciertos argumentos sean obligatorios o tengan valores predeterminados.

✅ Mostrar ayuda automática (`--help`).

✅ Validar automáticamente los tipos de datos (por ejemplo, asegurarse de que un argumento sea un número).

Ejemplo Basico:

```
import argparse

# Crear el parser
parser = argparse.ArgumentParser(description="Ejemplo básico con argparse

# Agregar argumentos
parser.add_argument("-n", "--nombre", type=str, help="Tu nombre")
parser.add_argument("-e", "--edad", type=int, help="Tu edad")
```

```
# Parsear los argumentos
args = parser.parse_args()

# Usar los argumentos
print(f"Hola {args.nombre}, tienes {args.edad} años.")
```

✅ ¿Cómo defines un argumento en `argparse`?

- Usando `parser.add_argument()`. Por ejemplo:

```
parser.add_argument("-n", "--nombre", type=str, help="Tu nombre")
```

Esto indica que `-n` o `--nombre` es un argumento que espera una cadena de texto.

✅ ¿Qué pasa si ejecutas el script con `--help`?

- Muestra automáticamente una descripción del script y los argumentos disponibles, como en este ejemplo:

```
usage: script.py [-h] [-n NOMBRE] [-e EDAD]

Ejemplo básico con argparse

optional arguments:
  -h, --help            show this help message and exit
  -n NOMBRE, --nombre NOMBRE
                        Tu nombre
  -e EDAD, --edad EDAD  Tu edad
```

1. ✅ **¿Por qué `argparse` es mejor que `getopt` en la mayoría de los casos?**
    - Porque es **más fácil de usar**, maneja `-help` automáticamente, **valida tipos de datos** y permite **valores predeterminados**.

📌 Manejo de tipos de datos con `argparse`

Argumentos con tipo específico

`argparse` permite validar tipos de datos automáticamente. Por ejemplo, si quieres asegurarte de que un argumento sea un número, puedes especificar el tipo como `int` o `float`.

Ejemplo: Argumento de tipo entero

```
import argparse

# Crear el parser
parser = argparse.ArgumentParser()

# Definir los argumentos
parser.add_argument("-n", "--numero", type=int, help="Un número entero")

# Parsear los argumentos
args = parser.parse_args()

# Usar los argumentos
print(f"El número ingresado es {args.numero}")
```

📌 Argumentos opcionales y obligatorios

Por defecto, los argumentos son opcionales. Si quieres que un argumento sea **obligatorio**, puedes hacerlo especificando el parámetro `required=True`.

Ejemplo: Argumento obligatorio

```
import argparse

# Crear el parser
parser = argparse.ArgumentParser()

# Definir el argumento obligatorio
parser.add_argument("-n", "--nombre", type=str, required=True, help="Tu
```

```
nombre")

# Parsear los argumentos
args = parser.parse_args()

# Usar el argumento
print(f"Hola, {args.nombre}")
```

📌 Argumentos con valores predeterminados

También puedes darle un valor predeterminado a un argumento, de modo que si no se proporciona, el valor predeterminado será usado.

Ejemplo: Argumento con valor predeterminado

```
import argparse

# Crear el parser
parser = argparse.ArgumentParser()

# Definir un argumento con valor predeterminado
parser.add_argument("-v", "--verbose", action="store_true", help="Habilitar m

# Parsear los argumentos
args = parser.parse_args()

# Usar el argumento
if args.verbose:
    print("Modo detallado activado")
else:
    print("Modo normal activado")
```

Desafío práctico con decimales y lista de números

Ahora, vamos a modificar el script para que acepte tanto enteros como decimales y calcule la suma.

Código modificado:

```
import argparse

# Crear el parser
parser = argparse.ArgumentParser()

# Definir el argumento para una lista de números (que pueden ser enteros o d
parser.add_argument("-n", "--numeros", type=float, nargs='+', help="Lista de

# Parsear los argumentos
args = parser.parse_args()

# Verificar si se pasaron números y hacer la suma
if args.numeros:
    suma = sum(args.numeros)
    print(f"La suma de los números es: {suma}")
else:
    print("No se proporcionaron números para sumar.")
```

Explicación:

1. `type=float`: Esto asegura que los valores ingresados sean convertidos a números decimales (flotantes).
2. `nargs='+'`: Esto sigue permitiendo múltiples valores como una lista.

Ejemplo de ejecución con decimales:

```
python3 script.py -n 1.5 2.3 3.7
```

Esto imprimirá:

```
La suma de los números es: 7.5
```