Una Ambiciosa Introducción a Python
Parte II

Enumerate

La función enumerate permite recorrer una secuencia (lista, tupla, diccionario, etc) y obtener tanto un indice como el valor de cada elemento.

Sintaxis

```
enumerate(iterable, start=0)
```

La secuencia a recorrer

Opcional

```
frutas = ["Manzana", "Banana", "Cereza"]

for indice, fruta in enumerate(frutas):
    print(f"Índice: {indice}, Fruta: {fruta}")
```

→

```
Índice: 0, Fruta: Manzana
Índice: 1, Fruta: Banana
Índice: 2, Fruta: Cereza
```

```
frutas = ["Manzana", "Banana", "Cereza"]

for indice, fruta in enumerate(frutas, start = 1):
    print(f"Índice: {indice}, Fruta: {fruta}")
```

→

```
Índice: 1, Fruta: Manzana
Índice: 2, Fruta: Banana
Índice: 3, Fruta: Cereza
```

Podemos usar enumerate() para modificar elementos de una lista.

```
numeros = [10, 20, 30, 40]

# Multiplicamos cada número por 2

for i, num in enumerate(numeros):
    numeros[i] = num * 2

print(numeros)
```

→ [20, 40, 60, 80]

```
notas = [
[7, 8, 9],
[6, 7, 5],
[9, 9, 10]
]

for i, fila in enumerate(notas, start=1):
    print(f"Alumno {i}: {fila}")
```

→

Alumno 1: [7, 8, 9]
Alumno 2: [6, 7, 5]
Alumno 3: [9, 9, 10]

Uso indirecto en diccionarios mediante items()

```
estudiantes = {"Juan": 18, "Ana": 20, "Luis": 19}

for i, (nombre, edad) in enumerate(estudiantes.items(), start=1):
    print(f"{i}. {nombre} tiene {edad} años.")
```

→

1. Juan tiene 18 años.
2. Ana tiene 20 años.
3. Luis tiene 19 años.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Parámetros solo posicionales

La barra diagonal, /, indica a Python que todos los argumentos anteriores al mismo serán únicamente posicionales.

#invocaciones válidas

saludar('Franco')
saludar('Franco', 'Buenos días')

#invocaciones inválidas

saludar(nombre='Franco')
saludar(nombre='Franco', saludo='Buenos días')

```
def saludar(nombre, /, saludo="Hola"):
    print(f"{saludo}, {nombre}!")
```

```
def dividir(a, b, /, precision=2):
    return round(a / b, precision)
```

 # Invocaciones válidas

print(dividir(10, 3))
print(dividir(10, 3, precision=4))

#Invocaciones inválidas

dividir(a=10, b=3)

Parámetros solo de palabras claves

El asterisco, *, indica a Python que todo los argumentos posteriores al mismo serán únicamente de palabras claves.

#Llamadas válidas

crear_usuario("Franco", edad=30, ciudad="Madrid")

#Llamada inválida

crear_usuario("Franco", 30, "Madrid")

```
def crear_usuario(nombre, *, edad, ciudad):
    print(f"Usuario: {nombre}, Edad: {edad}, Ciudad: {ciudad}")
```

```
def procesar_datos(a, b, /, *, precision=2, formato="float"):
    if formato == "float":
        return round(a / b, precision)
    elif formato == "int":
        return a // b
    else:
        raise ValueError("Formato no soportado")
```

 # Invocaciones válidas

print(procesar_datos(10, 3, precision=3, formato="float"))
print(procesar_datos(10, 3, formato="int"))

 # Invocación inválida

procesar_datos(a=10, b=3, precision=2)

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Python nos ofrece los conceptos de ***"Número arbitrario de argumentos o de palabras claves"*** esto le da mas flexibilidad a las funciones respecto de la manera que trabaja con los argumentos de entrada.

Número arbitrario de argumentos (*args)

*args

```
def suma(*args):
    total = sum(args)
    print(f"Suma total: {total}")

suma(1, 2, 3)
suma(4, 5, 6, 7)
suma()
```

Estos argumentos se almacena en la tupla (args) dentro de la función

suma(1,2,3) pasará tres argumentos, mientras que suma() no pasará ninguno.

El asterisco seguido de un nombre en la definición de una función indica que se puede recibir un número indeterminado de argumentos posicionales. Estos elementos son empaquetados en una tupla.

Número arbitrario de argumentos de palabra claves (**kwargs)

El doble asterisco (**) seguido de un nombre permite que la función acepte cualquier número de argumentos de palabras claves. Estos elementos son empaquetados en un diccionario

**kwargs

```
def mostrar_info(**kwargs):
    for clave, valor in kwargs.items():
        print(f"{clave}: {valor}")

mostrar_info(nombre="Alejandro", edad=30, ciudad="Madrid")
```

**kwargs captura todos los argumento de palabras clave que se pasan a la función y los almacena en el diccionario (kwargs)

Python nos permite combinar ambos tipos de argumentos en una misma función.

Orden en la definición de la función

1. Argumentos posicionales.
2. *args
3. Argumentos de palabras claves
4. **kwargs

```
def funcion_completa(a, b, *args, c=5, **kwargs):
    print(a, b, args, c, kwargs)
```

Lic. Franco Herrera