**Understanding Python Functions Arguments**

Functions in Python allow us to reuse code and organize logic efficiently. Let's explore different types of function arguments.

**1. Function Arguments**

**Definition:**

Arguments are values **passed to a function** when calling it. Python supports multiple types of arguments.

**Example:**

def greet(name):

print("Hello,", name)

greet("Alice") # Output: Hello, Alice

Here, "Alice" is the argument passed to the greet function.

**2. Positional Arguments**

**Definition:**

Positional arguments are **matched to function parameters based on their position** in the function call.

**Example:**

def person\_info(name, age):

print(f"Name: {name}, Age: {age}")

person\_info("Alice", 25) # Correct

person\_info(25, "Alice") # Incorrect order

* The first argument ("Alice") is assigned to name, and the second (25) is assigned to age.
* Changing the order will result in incorrect data assignment.

**3. Default Arguments**

**Definition:**

Default arguments have **predefined values** that are used if no value is provided during the function call.

**Example:**

def greet(name="Guest"):

print("Hello,", name)

greet("Alice") # Output: Hello, Alice

greet() # Output: Hello, Guest

* If name is provided, it uses that value.
* If no argument is given, the function defaults to "Guest".

**4. \*args (Variable-Length Positional Arguments)**

**Definition:**

\*args allows a function to accept **multiple positional arguments** as a **tuple**.

**Example:**

def sum\_numbers(\*args):

total = sum(args)

print("Sum:", total)

sum\_numbers(1, 2, 3) # Output: Sum: 6

sum\_numbers(10, 20, 30, 40) # Output: Sum: 100

* \*args collects all extra arguments into a **tuple**.
* Useful when the number of arguments is **unknown**.

**.**

**5. \*\*kwargs (Variable-Length Keyword Arguments)**

**Definition:**

\*\*kwargs allows passing **multiple keyword arguments**, which are stored as a **dictionary**.

**Example:**

def person\_details(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key}: {value}")

person\_details(name="Alice", age=25, city="New York")

# Output:

# name: Alice

# age: 25

# city: New York

* \*\*kwargs collects keyword arguments into a **dictionary**.
* Useful when dealing with **flexible key-value arguments**.

**Combining \*args and \*\*kwargs in a Function**

Both \*args and \*\*kwargs can be used in the same function.

**Example:**

def complete\_info(greeting, \*args, \*\*kwargs):

print(greeting)

print("Args:", args)

print("Kwargs:", kwargs)

complete\_info("Hello!", "Alice", "Bob", age=25, city="NY")

# Output:

# Hello!

# Args: ('Alice', 'Bob')

# Kwargs: {'age': 25, 'city': 'NY'}

* greeting is a **regular argument**.
* \*args stores extra **positional arguments** ("Alice", "Bob").
* \*\*kwargs stores **keyword arguments** (age=25, city="NY").

**Summary Table**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Positional Arguments** | Matched by position | func("Alice", 25) |
| **Default Arguments** | Uses default if no value is given | func(name="Guest") |
| **\*args** | Collects multiple positional arguments | func(1, 2, 3) |
| **\*\*kwargs** | Collects multiple keyword arguments | func(name="Alice", age=25) |

**Sequence of Positional and Keyword Arguments in Python Functions**

In Python, **function arguments must follow a specific sequence** when defining and calling a function. The correct order is:

1. **Positional arguments**
2. **Default arguments**
3. **\*args (Variable-length positional arguments)**
4. **\*\*kwargs (Variable-length keyword arguments)**

**Correct Order of Arguments in Function Definition**

def my\_function(pos1, pos2, default1="default", \*args, \*\*kwargs):

print(f"Positional: {pos1}, {pos2}")

print(f"Default: {default1}")

print(f"Args: {args}")

print(f"Kwargs: {kwargs}")

my\_function("A", "B", "C", "D", "E", key1="value1", key2="value2")

**Execution Order:**

* "A" and "B" → Assigned to **positional arguments** (pos1, pos2)
* "C" → Assigned to **default argument** (default1)
* "D", "E" → Collected into **\*args**
* key1="value1", key2="value2" → Stored in **\*\*kwargs**