Function Arguments and Types in Python

Function Arguments and Types in Python

Function Arguments

- Function arguments are the values or variables you pass into a function when you call it.
- They provide input data that the function uses to perform its task.
- When you define a function, you specify parameters — placeholders for the arguments that will be passed in.

Why are function arguments important?

- They let functions be flexible and reusable.
- Different arguments can produce different outputs or behaviors.

Syntax of a Function with Arguments

```
def function_name(parameter1, parameter2):
    # function body
    print(parameter1, parameter2)

function_name(argument1, argument2)
```

- **Parameters** → variables defined in the function definition (like placeholders).
- **Arguments** → actual values you pass when calling the function.

Types of Function Arguments in Python

Python supports **5 types of arguments**:

1. Positional Arguments

These are the most common.
The order of arguments **must match** the order of parameters.

```
def greet(name, age):
    print("Hello", name, "! You are", age, "years old.")

greet("XYZ", 25)
```

Output:
Hello XYZ ! You are 25 years old.

If you change the order:

```
greet(25, "XYZ")
```

Output:
Hello 25 ! You are XYZ years old.
❌ Wrong meaning but valid code — order matters!

Function Arguments and Types in Python

Explanation:
1. The function greet() is **defined** with two parameters: name and age.
These are like **placeholders** for values that will be given later.
2. When we call greet("XYZ", 25) –
- "XYZ" is assigned to name
- 25 is assigned to age

(because of the **position** in which they are written).
3. The function then prints:
"Hello XYZ! You are 25 years old."

2. Keyword Arguments
You can specify which parameter you're assigning a value to.
Order doesn't matter.

```
def greet(name, age):
    print("Hello", name, "! You are", age, "years old.")

greet(age=25, name="QWERTY")
```

Output:
Hello QWERTY ! You are 25 years old.
NOTE: Keyword arguments make code more readable and prevent mistakes in order.

Explanation:
1. Here, when calling the function, you specify which parameter you are assigning a value to.
2. Even though age is written before name, Python knows which value goes where because of the **keywords** (age= and name=).
3. This allows flexibility and better readability.

3. Default Arguments
You can assign default values to parameters.
If no value is passed, the default value is used.

```
def greet(name, age=18):
    print("Hello", name, "! You are", age, "years old.")

greet("ABC")
greet("Riya", 22)
```

Output:
Hello ABC ! You are 18 years old.
Hello Riya ! You are 22 years old.

Function Arguments and Types in Python

NOTE: Useful when you want some parameters to be optional.

Explanation:
1. The function greet() has a default value for age = 18.
2. When you call greet("Mehul"), only the name is given — so Python uses the **default age 18**.
3. When you call greet("Riya", 22), you provide both values, so the **default is ignored** and replaced by 22.

4. Variable-length Arguments
Sometimes you don't know how many arguments will be passed.
You can use:
- *args → for variable **positional** arguments.
- **kwargs → for variable **keyword** arguments.

***Using args (Non-keyword arguments)**

```
def add_numbers(*nums):
    total = 0
    for n in nums:
        total += n
    print("Sum is:", total)

add_numbers(2, 4, 6)
add_numbers(1, 3, 5, 7, 9)
```

Output:
Sum is: 12
Sum is: 25

Explanation:
1. The *nums parameter means "accept as many arguments as you want" — all are collected into a **tuple**.
2. For example,
   - In add_numbers(2, 4, 6) → nums = (2, 4, 6)
   - In add_numbers(1, 3, 5, 7, 9) → nums = (1, 3, 5, 7, 9)
3. Then, a for loop adds up all the numbers and prints the sum.

NOTE: The *args collects all extra positional arguments into a tuple.

****Using kwargs (Keyword arguments)**

```
def show_details(**info):
    for key, value in info.items():
        print(key, ":", value)
```

show_details(name="Music", age=25, city="Ambala")

Output:
name : Music
age : 25
city : Ambala

Explanation:
1. The **info collects all **keyword arguments** into a dictionary.
So inside the function:
info = {'name': 'Music', 'age': 25, 'city': 'Ambala'}
2. The loop goes through each key-value pair and prints them.

NOTE: The **kwargs collects all keyword arguments into a dictionary.

5. Required Arguments
If a parameter has no default value, it must be passed when calling the function — otherwise you get an error.

```
def greet(name):
    print("Hello", name)

greet("Music")    # ✅ Works
greet()           # ❌ Error: missing 1 required positional argument
```

Explanation:
1. The function expects **one argument (name)**.
2. The first call greet("Music") works because the argument is provided.
3. The second call greet() gives an error because the **required argument is missing**.

NOTE: Required arguments **must be given** — otherwise Python will throw an error.

Type	Symbol	Description	Example
Positional	—	Order matters	greet("Mahi", 25)
Keyword	key=value	Order doesn't matter	greet(age=25, name="Mahi")
Default	=value	Uses default if missing	def greet(name, age=18)
Variable-length	*args / **kwargs	Multiple arguments	add(1,2,3) / info(name="Mahi")
Required	—	Must provide	greet("Mahi")

Function Arguments and Types in Python

Assignment

Ques 1: Write a Python program that defines a function student_info(name, grade) to display a student's name and grade.
Call the function by passing both values.

Output:
Student Name: Riya
Grade: A

Ques 2: Create a function show_employee(name, salary) that prints employee details. Call it using **keyword arguments** (in any order).
Output:
Employee Name: Mansi
Salary: 40000

Ques 3: Write a function greet(name, message="Welcome!") that greets a person. Call it once with only the name and once with both name and message.

Output:
Hello Sanya, Welcome!
Hello Rohit, Good to see you!