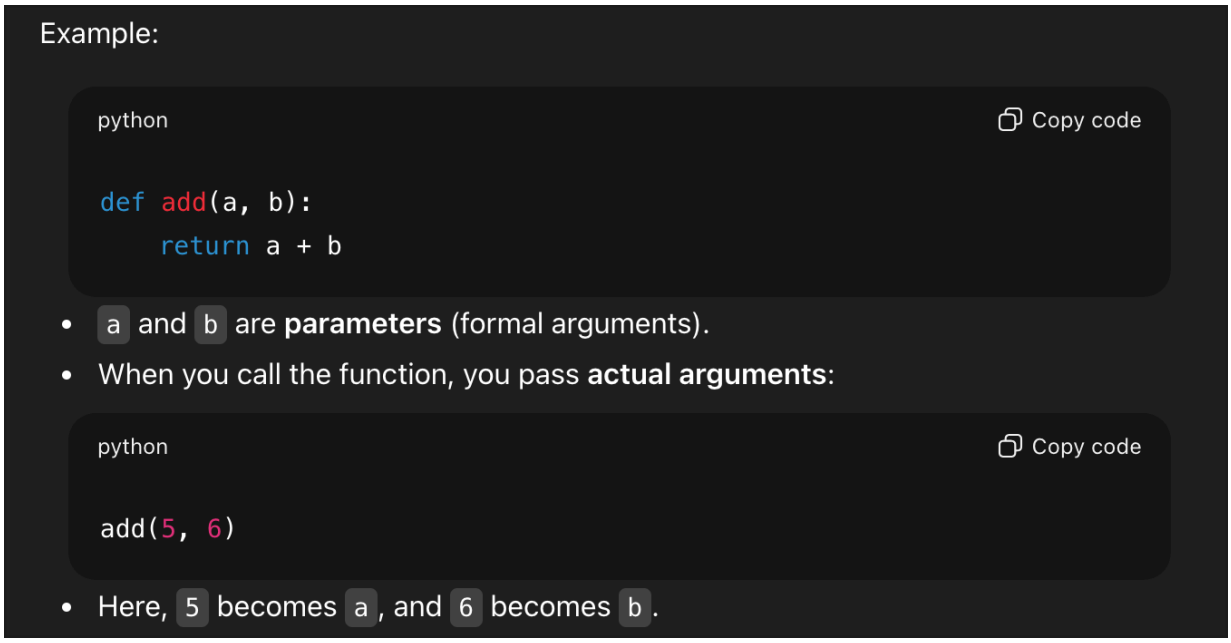
Python Function Arguments

1. What happens when you call a function with arguments?

When you define a function, you give it **parameters**. These are *names* that act like placeholders.



2. Positional Arguments

Positional arguments are the **normal** kind of arguments you've already used.

They are matched to parameters **based on order**.

Example:

```python
def person(name, age):
    print(name, age)

person("Naveen", 28)
```

How Python matches them:

- The first value `"Naveen"` goes to `name`
- The second value `28` goes to `age`

Why order matters

If you switch the order, you get nonsense:

```python
person(28, "Naveen")
```

Now:

- `name` = 28
- `age` = "Naveen"

This won't crash, but it's clearly wrong.

3. Keyword Arguments

Keyword arguments allow you to **name the parameter** when you call the function.

Example:

```python
person(age=28, name="Naveen")
```

Now order does **not** matter.

Python assigns values using the **names**, not the positions.

Benefits:

- Prevents mistakes from ordering.
- Makes the code easier to read.

4. Default Arguments

A default argument means a parameter already has a value unless the caller replaces it.

Example:

```python
def person(name, age=18):
    print(name, age)
```

Now:

```python
person("Naveen")
# age becomes 18 automatically
```

If you **do** give an age, it replaces the default:

```python
person("Naveen", 30)
```

Why use default arguments?

- They make function calls shorter and simpler.
- They let you define “common” or “typical” values once.

5. Mutability and Function Arguments

This is extremely important and students almost always get confused here.

Immutable types

int, float, and string **cannot be changed** inside a function.

Example:

```python
def increase(x):
    x = x + 1
    return x
```

```python
a = 5
increase(a)   # returns 6
print(a)      # still 5
```

Even though `x` was changed, the original `a` did not change.

Mutable types

A list **can** be changed inside a function.

Example:

```python
def add_one(lst):
    lst.append(1)

my_list = []
add_one(my_list)
print(my_list)   # [1]
```

6. Common mistakes beginners make

❌ Mistake: Wrong number of arguments

python Copy code

```python
def add(a, b):
    return a + b

add(5)      # ERROR: missing argument
add(5, 6, 7) # ERROR: too many arguments
```

❌ Mistake: Switching argument order

python Copy code

```python
def greet(name, age):
    print(name, age)

greet(25, "Sam")   # wrong!
```

❌ Mistake: Thinking strings are characters

Python has **no char type** — a character is just a string of length 1.

7. Helpful Examples

Example A — Normal positional arguments

python Copy code

```python
def multiply(x, y):
    return x * y

multiply(3, 4)   # 12
```

Example B — Keyword arguments

python Copy code

```python
def describe(name, age):
    print(name, age)

describe(age=20, name="Alex")
```

Example C — Default arguments

python Copy code

```python
def greet(name="Student"):
    print("Hello", name)

greet()            # "Hello Student"
greet("Naveen")    # "Hello Naveen"
```

Example D — Mutability inside a function

python Copy code

```python
def add_item(lst, item):
    lst.append(item)

my_list = ["a"]
add_item(my_list, "b")
print(my_list)    # ["a", "b"]
```