In Python, **keyword-only arguments** are function parameters that must be specified using their names in function calls. These arguments are defined after a \* in the function signature, which prevents them from being passed as positional arguments.

**How to Define Keyword-Only Arguments**

To enforce keyword-only arguments, you use \* in the function parameter list:

def func(a, \*, b, c):

print(f"a: {a}, b: {b}, c: {c}")

# Correct usage (using keyword arguments)

func(1, b=2, c=3)

# Incorrect usage (trying to pass keyword-only arguments positionally)

func(1, 2, 3) # TypeError

**Using \* to Force Keyword-Only Arguments**

* Any argument after \* **must** be provided as a keyword argument.
* This prevents accidental misordering of arguments.

Example:

def greet(name, \*, message="Hello"):

print(f"{message}, {name}!")

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

greet("Alice", message="Hi") # Output: Hi, Alice!

greet("Alice", "Hi") # TypeError: greet() takes 1 positional argument but 2 were given

**Keyword-Only Arguments with \*args**

If a function has \*args, any argument after \*args is keyword-only.

def func(\*args, b, c):

print(f"args: {args}, b: {b}, c: {c}")

func(1, 2, 3, b=4, c=5) # Works fine

func(1, 2, 3, 4, 5) # TypeError: missing keyword arguments

**Keyword-Only Arguments with Default Values**

Keyword-only arguments can have default values.

def example(\*, x=10, y=20):

print(x + y)

example() # Output: 30

example(x=5) # Output: 25

example(y=15) # Output: 25

example(5, 15) # TypeError

**Advantages of Keyword-Only Arguments**

1. **Improves readability** – It forces function calls to be more explicit.
2. **Avoids argument misplacement** – Positional confusion is avoided.
3. **Enhances API clarity** – Useful in functions with many optional parameters.