In Python, functions can be used as return values. This allows you to return a function from another function, enabling **higher-order functions** and **closures**.

**Example 1: Returning a Function**

You can return a function from another function, which can then be called later.

def outer\_function():

def inner\_function():

return "Hello from the inner function!"

return inner\_function # Returning the function itself, NOT calling it

# Get the function

func = outer\_function()

# Call the returned function

print(func()) # Output: Hello from the inner function!

**Example 2: Using Function Arguments**

Since functions are first-class citizens in Python, they can be passed around as values.

def multiplier(factor):

def multiply(n):

return n \* factor

return multiply # Returning the function

double = multiplier(2)

triple = multiplier(3)

print(double(5)) # Output: 10

print(triple(5)) # Output: 15

**Example 3: Closures**

A closure is a function that remembers variables from its enclosing scope even after the scope has finished execution.

def counter():

count = 0

def increment():

nonlocal count

count += 1

return count

return increment # Returning the inner function

counter\_func = counter()

print(counter\_func()) # Output: 1

print(counter\_func()) # Output: 2

print(counter\_func()) # Output: 3

**Use Cases**

* **Factory Functions:** Dynamically create functions
* **Decorators:** Functions that wrap other functions
* **Callback Functions:** Used in event-driven programming