Nested function

Functions in Python: Nested Functions

In Python, **nested functions** refer to functions that are defined inside another function. A nested function can access variables from its enclosing (outer) function, allowing for more complex behaviors like closures and the creation of helper functions within the outer function.

Here’s an overview of **nested functions** in Python, including examples and how they work:

Functions in Python: Nested Functions

In Python, **nested functions** refer to functions that are defined inside another function. A nested function can access variables from its enclosing (outer) function, allowing for more complex behaviors like closures and the creation of helper functions within the outer function.

Here’s an overview of **nested functions** in Python, including examples and how they work:

1. Defining Nested Functions

You can define a function inside another function just like any other function in Python.

Syntax:

```
def outer_function():
    # Outer function body
    def inner_function():
        # Inner function body
        pass
    # You can call the inner function from here
    inner_function()
```

Example of a Nested Function:

```
def outer_function():
    print("This is the outer function.")

    def inner_function():
        print("This is the inner function.")

    inner_function()  # Calling the inner function from within the outer function
```

```
outer_function()
# Output:
# This is the outer function.
# This is the inner function.
```

2. Accessing Variables in Nested Functions

A nested function has access to variables in its **enclosing scope**, i.e., the scope of the outer function. This is one of the key features of nested functions, as the inner function can use variables from its outer function.

Example of Nested Function with Variables:

```
def outer_function(a, b):
    result = a + b  # Outer function variable

    def inner_function():
        print(f"Result from outer function: {result}")

    inner_function()

outer_function(3, 5)

# Output:

# Result from outer function: 8
```

3. Returning Inner Function (Closures)

A powerful feature of nested functions is the ability to return an inner function, creating a **closure**. A closure occurs when the inner function retains access to variables from its enclosing function's scope even after the outer function has finished executing.

Example of a Closure:

```
def outer_function(x):
```

```
    def inner_function(y):
        return x + y  # inner function uses 'x' from the outer function
    return inner_function  # Return the inner function

closure = outer_function(10)  # outer_function is executed, returning the inner_function
print(closure(5))  # Output: 15
```

4. Function as an Argument (Using Nested Functions)

You can pass a nested function as an argument to other functions or use it in higher-order functions.

Example of Passing a Nested Function:

```
def outer_function():
    def inner_function(x):
        return x * 2
    return inner_function  # Return the nested function

def apply_function(func, value):
    return func(value)  # Apply the function passed as an argument

# Get the nested function from outer_function
doubler = outer_function()

# Pass the nested function to another function
result = apply_function(doubler, 5)
```

```
print(result)  # Output: 10
```