Closures in Python

A closure is a function that is created inside another function and remembers the values of the variables that were present when it was created — even after the outer function has finished running.
This means that:

- The inner function can still use the outer function's variables.
- Those variables don't disappear when the outer function ends — the inner function keeps a reference to them.

In simpler terms:
A closure allows an inner function to remember and access variables from the outer function — even after the outer function has stopped running.

Example 1: Returning and Immediately Executing the Inner Function

```
def outer_func():
  message = 'Hi'

  def inner_func():
    print(message)

  return inner_func()  # Notice the parentheses: we're calling inner_func

outer_func()
```

Output:
Hi

Example 2: Returning the Inner Function Without Executing It

```
def outer_func():
  message = 'Hi'

  def inner_func():
    print(message)

  return inner_func  # No parentheses = function returned, not called

my_func = outer_func()
print(my_func)
print(my_func.__name__)
```

Output:
<function outer_func.<locals>.inner_func at 0x...>
inner_func
Here, my_func becomes a reference to inner_func. It's not executed yet — just stored.

You can now use my_func() to run the inner function:

```
my_func()
my_func()
my_func()
```

Output:

Copy code

```
Hi
Hi
Hi
```

Key Insight:

Even though outer_func() has finished executing, inner_func() still has access to the message variable. That's the **power of closures**.

Example 3: Closure with Parameters

Now let's modify outer_func to accept an argument:

```
def outer_func(msg):
  message = msg  # Closure variable

  def inner_func():
    print(message)

  return inner_func
```

Usage:

```
hi_func = outer_func('Hi')
hello_func = outer_func('Hello')

hi_func()
hello_func()
```

Output:

```
Hi
Hello
```

Each inner function remembers the message from the outer function it was created in.

Summary: Key Properties of Closures

- A **closure** is created when:
  - A nested function **uses a variable** from its enclosing scope.
  - The **outer function returns the inner function**.
- Closures help:
  - Preserve state across calls without using global variables.
  - Implement decorators and callback functions.
  - Encapsulate logic and data elegantly.