Lambda Function

**Python Lambda Functions** are anonymous functions means that the function is without a name.

As we already know the *def* keyword is used to define a normal function in Python. Similarly, the *lambda* keyword is used to define an anonymous function in [Python](https://www.geeksforgeeks.org/python-programming-language/).

def ----lambda

s1 = 'Hello Good Morning'

s2 = **lambda** func: func.upper()

print(s2(s1))

***Syntax:*** *lambda arguments : expression*

A lambda function can include conditions using if statements.

n = **lambda** x: "Positive" **if** x > 0 **else** "Negative" **if** x < 0 **else** "Zero"

print(n(5))

print(n(-3))

print(n(0))

Lamda1.py

Lambda2.py

Stringformat.py

**Python Inner Functions**

**def** fun1(msg): *# outer function*

**def** fun2(): *# inner function*

print(msg)

fun2()

fun1("Hello")

**Why Use Inner functions?**

**Inner functions** provide several advantages:

* **Encapsulation:** They help hide the inner logic from external access.
* **Code Organization:**They make the code cleaner by grouping related functionality.
* **Access to Enclosing Scope:** Inner functions can access variables of the outer function.
* **Closures:** They allow functions to retain the state of their enclosing function even after execution.

**Decorators in Python**

In [Python](https://www.geeksforgeeks.org/python-programming-language-tutorial/), decorators are a powerful and flexible way to modify or extend the behavior of functions or methods, without changing their actual code. A decorator is essentially a [function](https://www.geeksforgeeks.org/python-functions/) that takes another function as an argument and returns a new function with enhanced functionality.

Decorators are often used in scenarios such as logging, authentication and memorization, allowing us to add additional functionality to existing functions or methods in a clean, reusable way.

**def** decorator(func):

**def** wrapper():

print("Before calling the function.")

func()

print("After calling the function.")

**return** wrapper

*# Applying the decorator to a function*

@decorator

**def** greet():

print("Hello, World!")

greet()

decoratorEX.py

Function Signature

*import inspect*

*inspect.signature(decorator) ======🡺*