Decorators In Python

Introduction

Decorators are a powerful and versatile tool in Python that allows developers to modify or extend the behavior of functions or methods without changing their actual code. They are commonly used for logging, access control, instrumentation, and caching, among other purposes. Understanding decorators can help you write cleaner, more readable, and more maintainable code.

What are Decorators?

A decorator in Python is a function that takes another function as an argument and extends or alters its behavior. Decorators are applied using the `@decorator_name` syntax placed above the function definition.

Basic Syntax

Here's a simple example to illustrate the syntax of decorators:

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

Output:

```
Something is happening before the function is called.
Hello!
Something is happening after the function is called.
```

In this example:

- `my_decorator` is a decorator function that takes `say_hello` as its argument.
- The `wrapper` function is defined inside `my_decorator` and adds some behavior before and after the call to `say_hello`.
- The `@my_decorator` syntax applies the decorator to the `say_hello` function.

Code:

```
def test1(func1):
    def nowexec():
        print("progress now")
        func1()
        print("Completed")

    return nowexec
@test1

def pentest_diaries():
    print("Welcome to security world")

pentest_diaries()
```

Output:

```
C:\Users\attacker\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\attacker\PycharmProjects\pythonProject\30.py
progress now
Welcome to security world
Completed
```