Python Programming

PYTHON DECORATORS

PYTHON DECORATORS

- In Python, a decorator is a special type of function that is used to modify or enhance the behavior of another function or method without changing its source code.
- Python decorators are a powerful and flexible way to modify or enhance the behavior of functions or methods without changing their actual code.
- Decorators allow you to "decorate" or wrap a function with additional functionality.
- They are often used for tasks like logging, caching, validation, access control, and more.
- Decorators make use of the fact that functions are first-class objects in Python, which means they can be passed around and manipulated like any other object.

PYTHON DECORATORS

Before diving deep into decorators let us understand some concepts that will come in handy in learning the decorators.

First Class Objects

- In Python, functions are first class objects which means that functions in Python can be used or passed as arguments.

Properties of first class functions:

- A function is an instance of the Object type.
- You can store the function in a variable.
- You can pass the function as a parameter to another function.
- You can return the function from a function.
- You can store them in data structures such as hash tables, lists, …

PYTHON DECORATORS

Here's a basic overview of how decorators work:

- A decorator is a function that takes another function (or method) as an argument.
- The decorator function defines an inner function (often named as ‘wrapper’ that adds the desired behavior around the original function.
- The inner function is returned by the decorator function.
- The original function is replaced with the inner function, effectively "decorating" it with the additional behavior.

We can apply a decorator to a function using the ‘@’ symbol followed by the decorator function's name just above the function definition. This is also known as "decorating" the function.

PYTHON DECORATORS

Here's a basic example of a decorator:

```
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()
```

PYTHON DECORATORS

In this example, the ‘my_decorator’ function is used to take another function ‘func’ as an argument and returns a new function ‘wrapper’. The ‘wrapper’ function adds some behavior before and after calling the original ‘func’. The ‘@my_decorator’ syntax is used to apply the decorator to the ‘say_hello’ function.

Decorators are a powerful and flexible feature in Python, often used to enhance code readability, reusability, and maintainability by separating concerns and applying cross-cutting concerns to functions or methods.