Python

Full stack Skills Bootcamp

Introducing Python Decorators

- What are Decorators?
  - Decorators are higher-order functions that modify or extend the behaviour of functions or methods. They accept a function as an argument and return a new function that adds functionality.
- Purpose:
  - Enhance code readability and reusability.
  - Commonly used for logging, enforcing access control, instrumentation, and caching.

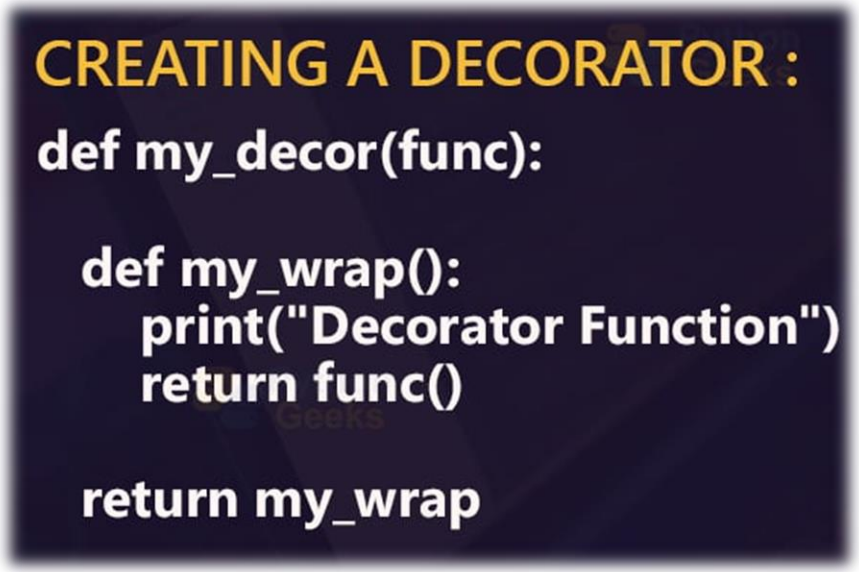
Defining a Decorator

- Creating a Decorator:

```python
def simple_decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper

@simple_decorator
def greet():
    print("Hello!")

greet()
```



- simple_decorator() wraps the greet function, adding behaviour before and after its execution.
- Output: When we call greet(), we get output as

  Before function call
  Hello!
  After function call

Understanding the Wrapper Function

What is a Wrapper ?

A function that "wraps" another function, allowing additional functionality to be added before or after the wrapped function executes.

Key Benefits:

- Keeps the original function intact.
- Allows for reusability and separation of concerns.

Decorator with Parameters Example

```python
from functools import wraps
import time

def timing_decorator(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Execution time: {end_time - start_time} seconds")
        return result
    return wrapper

@timing_decorator
def long_running_task():
    time.sleep(2)  # Simulates a long-running task
    print("Task complete")

long_running_task()
```

- The timing_decorator measures how long function takes to execute, printing the execution time.
- Output: Execution time: 2.002345 seconds
  Task complete

Use Cases for Decorators

- Logging: Track function usage and debug issues.
- Timing: Measure execution duration to optimize performance
- Access Control: Restrict access to certain functions based on conditions.
- Caching: Store results of expensive function calls and return cached results when the same inputs occur.

Conclusion

- So,
  - Decorators are a powerful feature in Python that enhance the functionality of functions and methods in a clean and maintainable way.
  - Experiment with decorators in your own projects to see their benefits firsthand.