Lesson Plan

Decorators

PW SKILLS

Topics to be covered:

1. Introduction to Decorators
2. Basics of decorators.
3. Decorator syntax

Introduction to Decorators

A decorator is a strong and adaptable tool in Python that lets you improve or alter a function or method's behavior without altering the source code. Decorators offer a neat and sophisticated method of enclosing functions, facilitating code reuse and improving readability.

Suppose you had a magic pen that could change the color of everything it came into contact with. Python decorators are similar to the magic pen, but for functions. They let you alter a function's operation without altering the function itself.

To put it another way, decorators are unique functions that have the ability to encircle other functions. They assist you in enhancing functions with new features without disorganizing the existing ones. It's similar to adding a lovely frame to a photo to improve its appearance without changing the original image.

You can keep your functions small and focused by using decorators, and you can add new capabilities to them as needed. Decorators improve your functions without requiring you to rewrite them from scratch, like a helpful assistant.

Basics of decorators

1. Function Review:

Before diving into decorators, let's review functions in Python.

```
def simple_function():
     print("This is a simple function.")
```

This is a basic function that prints a message when called.

2. Understanding Decorators:

Decorators are functions themselves. They take another function as an argument and extend its behavior.

```
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

decorated_function = my_decorator(simple_function)
decorated_function()
```

Output:

```
Something is happening before the function is called.
This is a simple function.
Something is happening after the function is called.
```

In this example, 'my_decorator' wraps the 'simple_function' and modifies its behavior.

Decorator syntax

1. Using the '@' symbol

Python provides a syntactic sugar for decorators using the @ symbol.

```
@my_decorator
def simple_function():
    print("This is a simple function.")
```