Step-by-Step Explanation:

1. **Function Decorator Creation:**

   - First, we create a decorator function, which will take another function (like `say_hello()`) as an argument. We usually name this argument `func` to show that it's a function.

```python
def simple_decorator(func):
    def wrapper():
        print("Function se pehle kuch ho raha hai")
        func()  # This is where the original function (like say_hello) is called
        print("Function ke baad kuch ho raha hai")
    return wrapper
```

- `func` yahan par `say_hello()` jaisa koi bhi function ho sakta hai. Jab `func()` likha gaya hai, tab yeh `say_hello()` ko call karega.

2. **Applying the Decorator:**

   - Now, when you apply the decorator to a function using `@simple_decorator`, Python automatically passes the `say_hello()` function as the `func` parameter to `simple_decorator`.

```python
@simple_decorator
def say_hello():
    print("Hello!")
```

- This means, `@simple_decorator` se `say_hello` ko decorate karte hi, Python `say_hello` function ko `simple_decorator` function mein `func` ke naam se pass kar deta hai.

3. **Wrapper Function Returns:**
   - The `simple_decorator` function returns the `wrapper` function, which now becomes the new version of `say_hello()`. Essentially, `say_hello()` ab `wrapper()` function ban gaya hai.
4. **Calling the Function:**
   - Finally, when you call `say_hello()`, what you're really calling is the `wrapper()` function.

```python
say_hello()
```

- When you run `say_hello()`, the following happens:
  - `wrapper()` function executes.
  - Inside `wrapper()`, `func()` is called, which is the original `say_hello()` function.
  - The extra code before and after `func()` runs as well.

Final Code in Summary:

python Copy code

```python
def simple_decorator(func):  # 'func' is 'say_hello' here
    def wrapper():  # 'wrapper' is the new version of 'say_hello'
        print("Function se pehle kuch ho raha hai")
        func()  # Call the original 'say_hello'
        print("Function ke baad kuch ho raha hai")
    return wrapper  # Return 'wrapper' as the new 'say_hello'

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

say_hello()  # This actually calls 'wrapper()'
```

So, when you call `say_hello()`, you're actually running the `wrapper()` function, which includes both the original `say_hello()` (referred to as `func()`) and the extra code that the decorator adds.