Function Annotations

Function Annotations

- Definition:
  - Function annotations are a Python feature that allows you to add arbitrary metadata to function parameters and return values.
- Introduced in:
  - Python 3.0, with enhancements in later versions.
- Purpose:
  - Primarily for documentation and type hinting, but annotations can be used for other purposes as they do not affect the runtime behavior of the program.

Function Annotations

```
def func_name(param1: type, param2: type) -> return_type:
```

Example

```
def greet(name: str) -> str:
    return f"Hello, {name}"
```

Function Annotations

Annotations can be accessed through the function's __annotations__ attribute.

Example:

```
def multiply(x: int, y: int) -> int:
    return x * y

print(multiply.__annotations__)
```

```
{'x': <class 'int'>, 'y': <class 'int'>, 'return': <class 'int'>}
```

Why Use Function Annotations?

- Documentation:
  - Makes the code easier to understand.
- Type Checking:
  - Can be used by third-party tools, IDEs, or static type checkers like MyPy to catch type errors.
- Enforces a Coding Standard:
  - Helps in maintaining a consistent coding style, especially in large projects or teams.
- Flexibility:
  - Annotations can store any type of information, not just types.

Decorators

Decorators

- Definition:
  - Decorators are a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure.
- Purpose:
  - They are used to modify the behavior of function or class methods.
- How they work:
  - Decorators wrap another function, modifying its behavior in the process.

How Decorators Work

```
def my_decorator(func): # Decorator function definition:
    def wrapper():
        # Code before function execution
        func()
        # Code after function execution
    return wrapper


@my_decorator
def my_function():
    print("The function is called.")
```

Example

```
def simple_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

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


say_hello()
```

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