Decorators in python

Python decorators are a powerful and versatile tool that allow to modify the behavior of functions and methods. They are a way to extend the functionality of a function or method without modifying its source code.

A decorator is a function that takes an argument and return a new function that modifiers the behavior of the original function. The new function is often referred to as a decorated function. The basis syntax for using a decorator is the following.

```
@greet
def hello():
    print("Hello world")
```

Practice use case

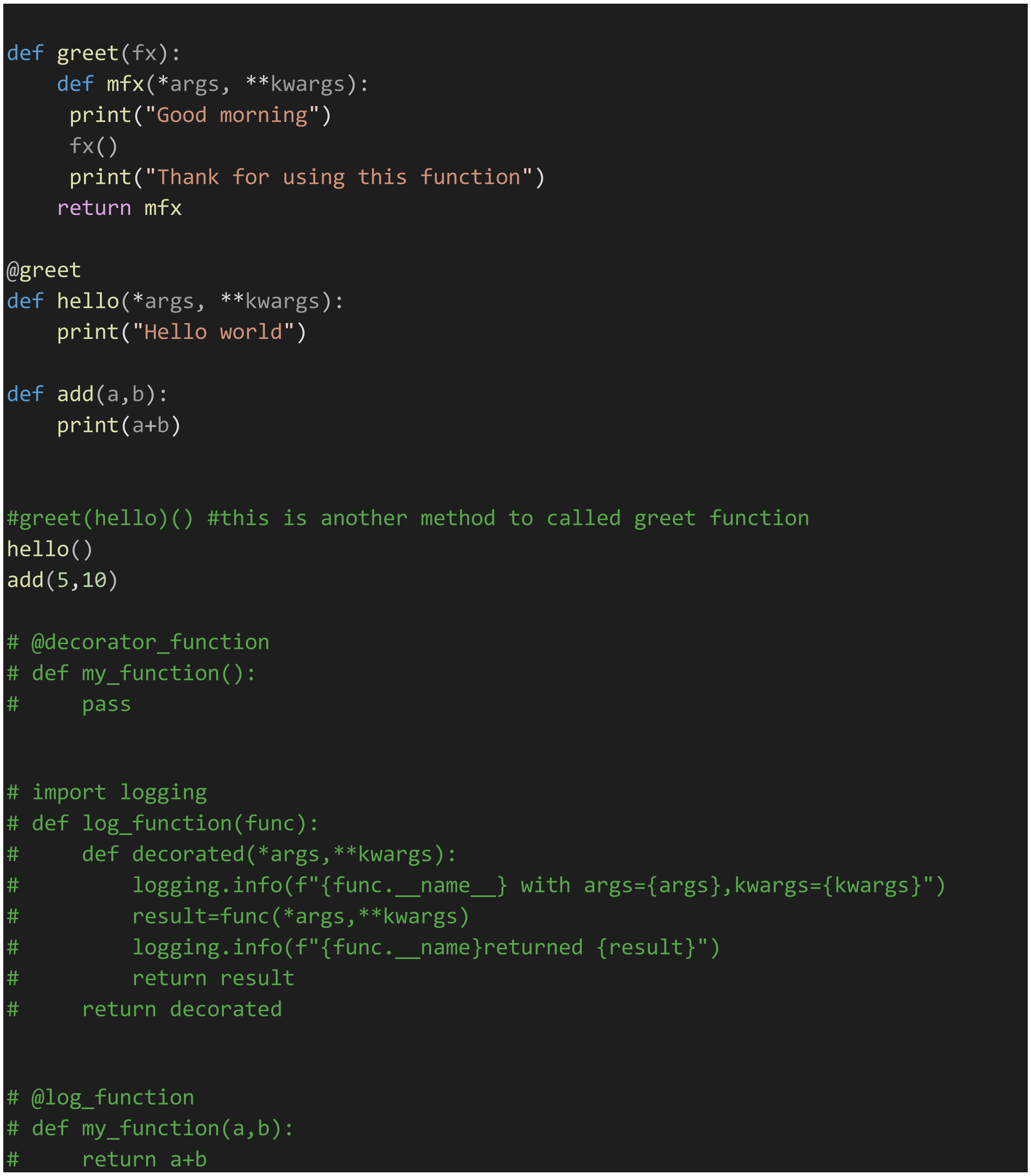
One common use of decorators is to add Logging to a function. For example you could use a decorator to log the arguments and return value of a function each time it is called:

```

import logging
def log_function(func):
    def decorated(*args,**kwargs):
        logging.info(f"{func.__name__} with args={args},kwargs={kwargs}")
        result=func(*args,**kwargs)
        logging.info(f"{func.__name}returned {result}")
        return result
    return decorated


@log_function
def my_function(a,b):
    return a+b
```

Source Code



```python
def greet(fx):
    def mfx(*args, **kwargs):
     print("Good morning")
     fx()
     print("Thank for using this function")
    return mfx

@greet
def hello(*args, **kwargs):
    print("Hello world")

def add(a,b):
    print(a+b)


#greet(hello)() #this is another method to called greet function
hello()
add(5,10)

# @decorator_function
# def my_function():
#     pass


# import logging
# def log_function(func):
#     def decorated(*args,**kwargs):
#         logging.info(f"{func.__name__} with args={args},kwargs={kwargs}")
#         result=func(*args,**kwargs)
#         logging.info(f"{func.__name}returned {result}")
#         return result
#     return decorated


# @log_function
# def my_function(a,b):
#     return a+b
```

Thank You