Implementing Function Decorators

Mateo Prigl
Software Developer

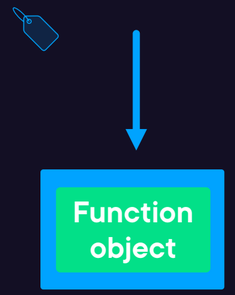
```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


@email_decorator
def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

email_decorator



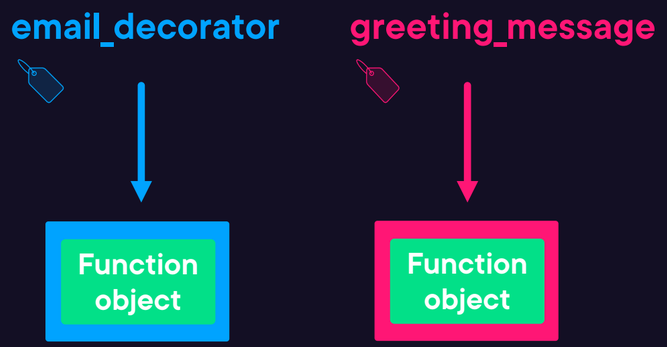

```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper



def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

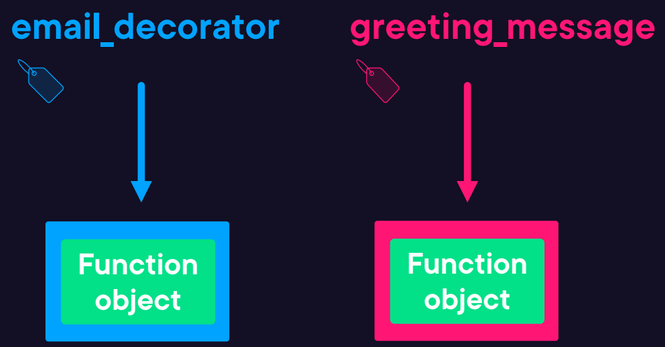


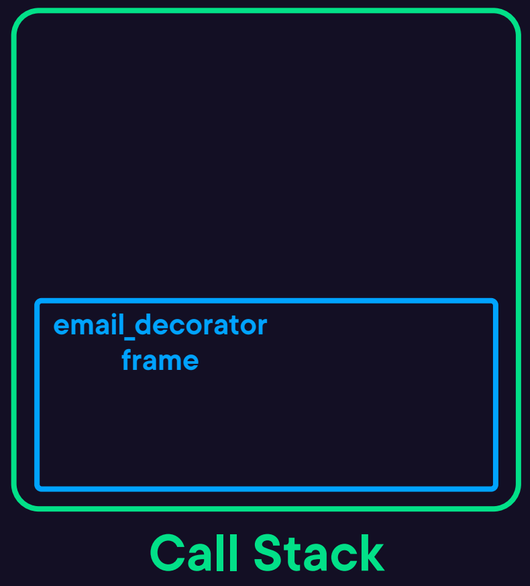

```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```



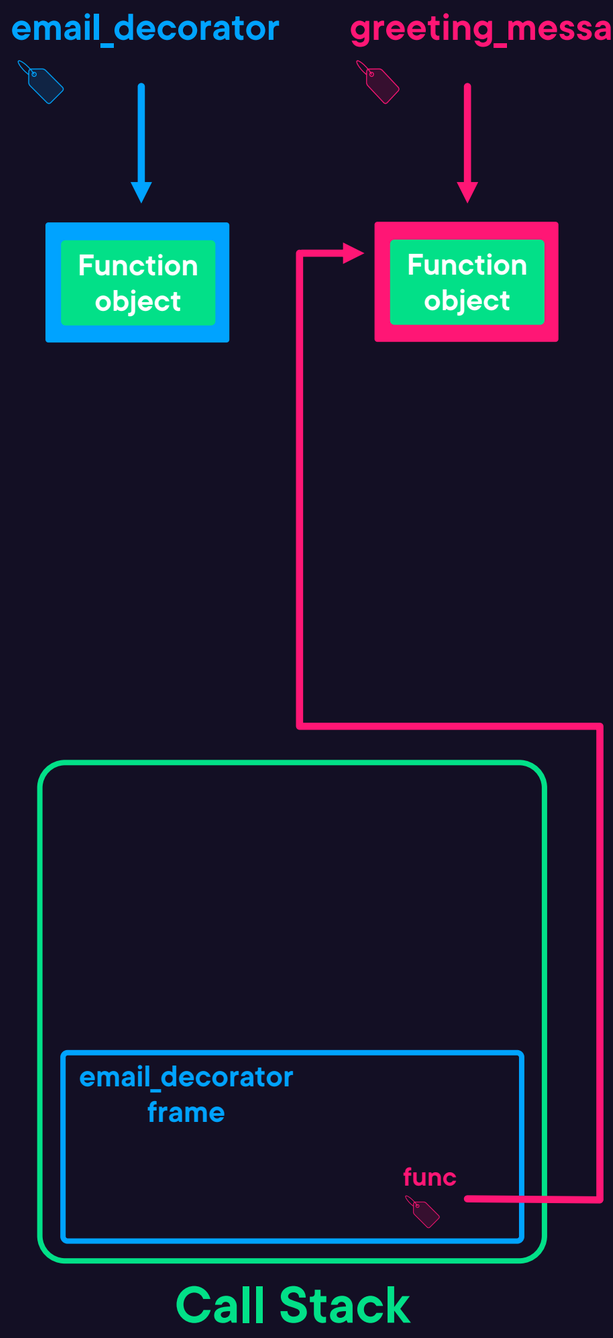



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

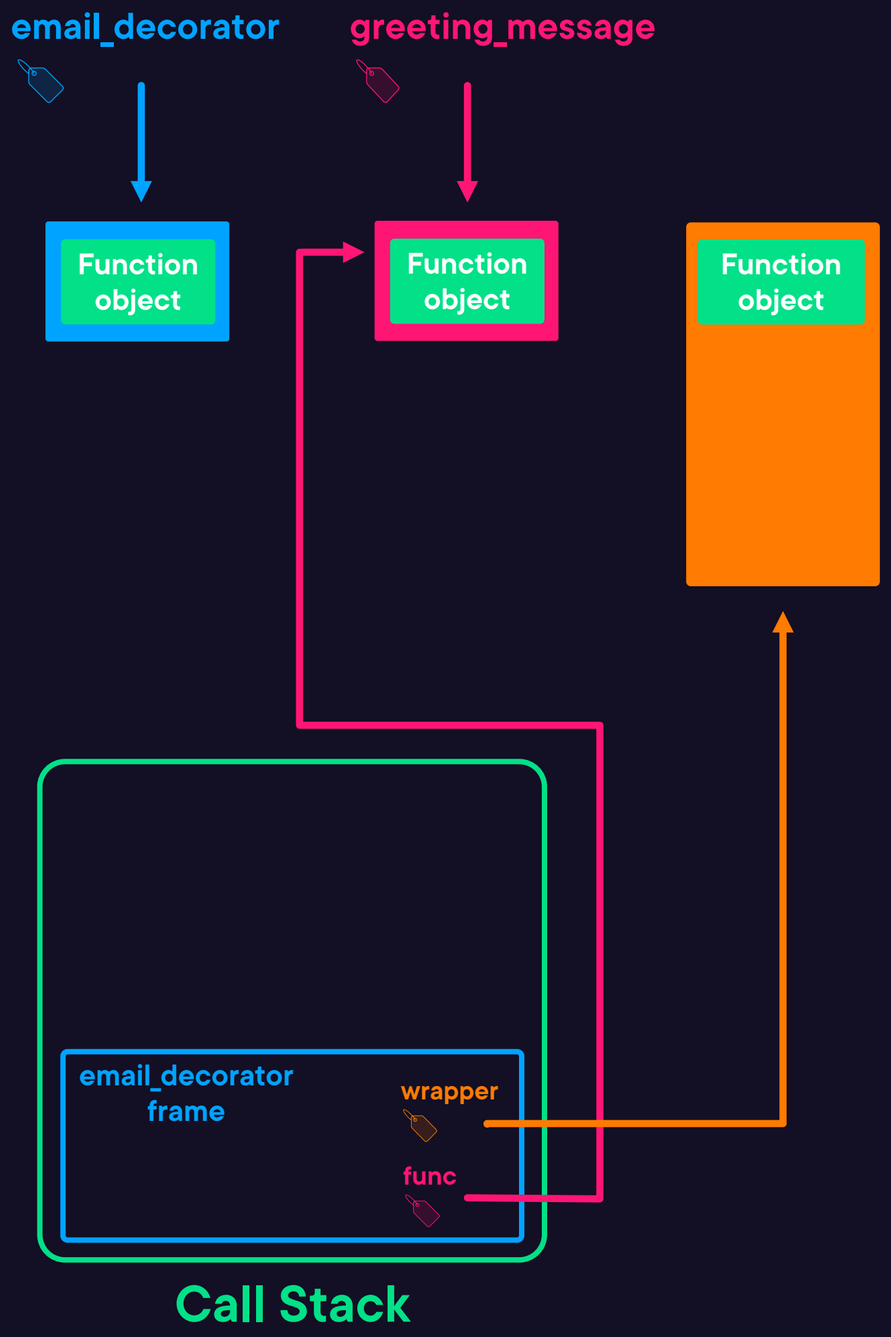



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

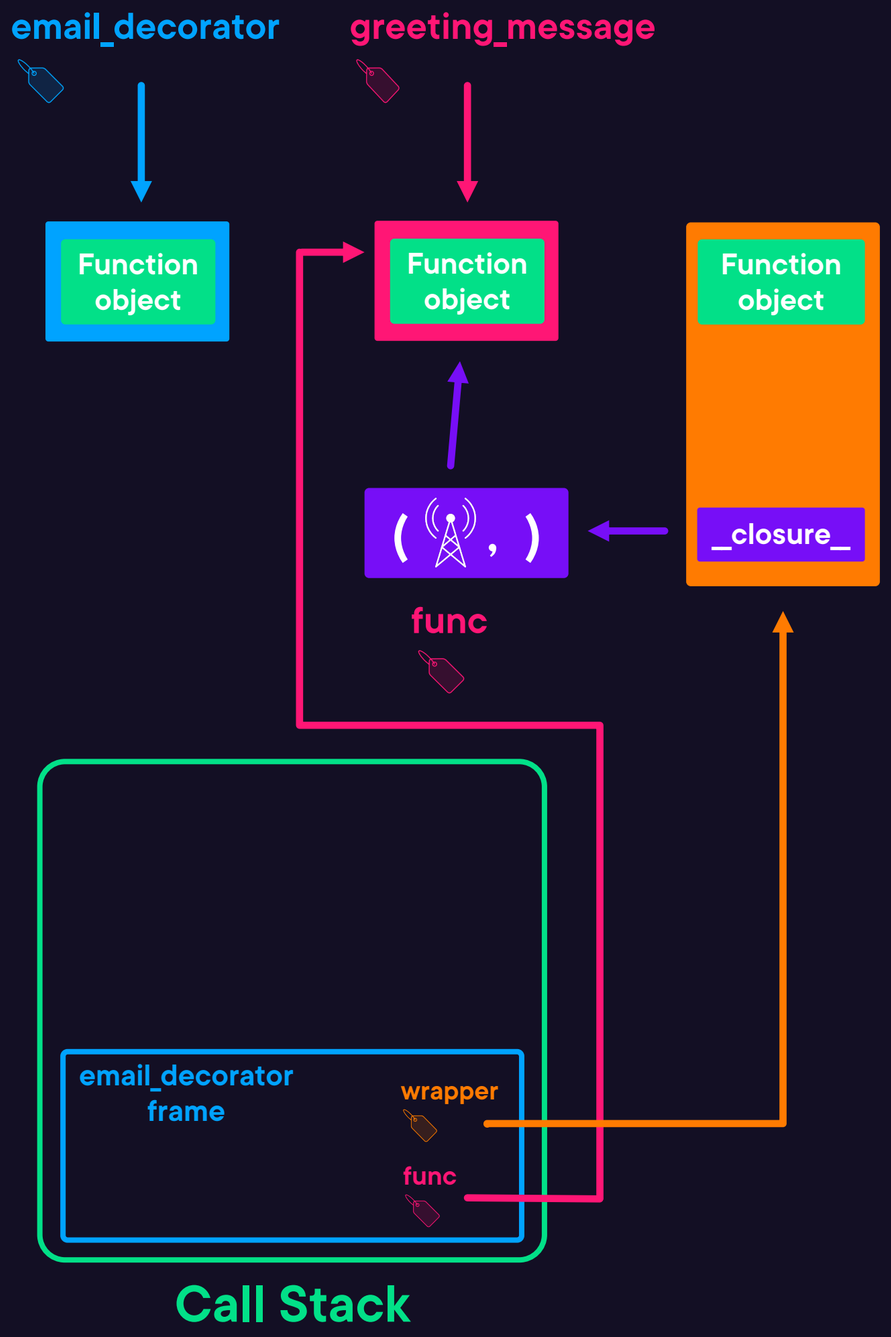



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

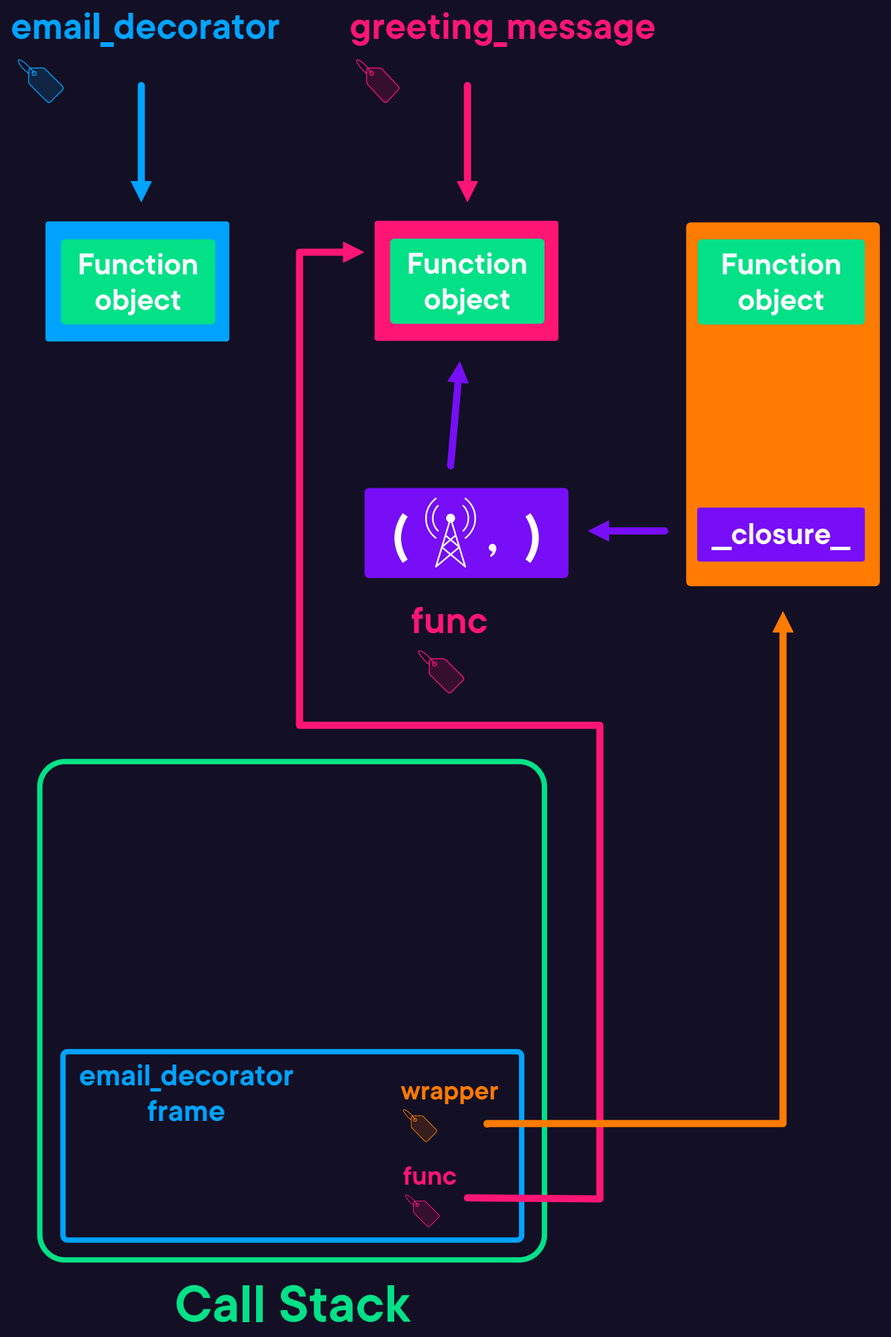



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

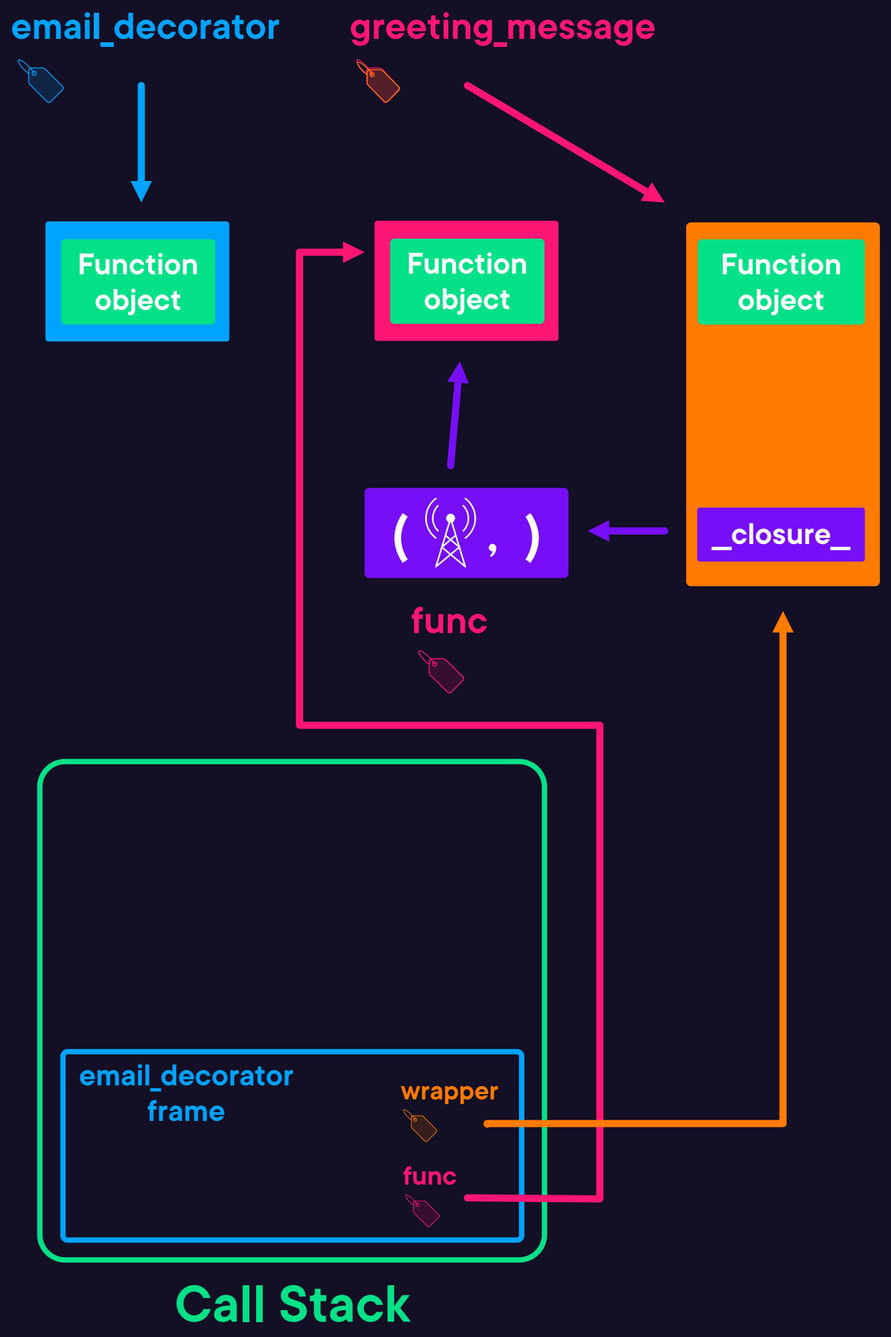



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

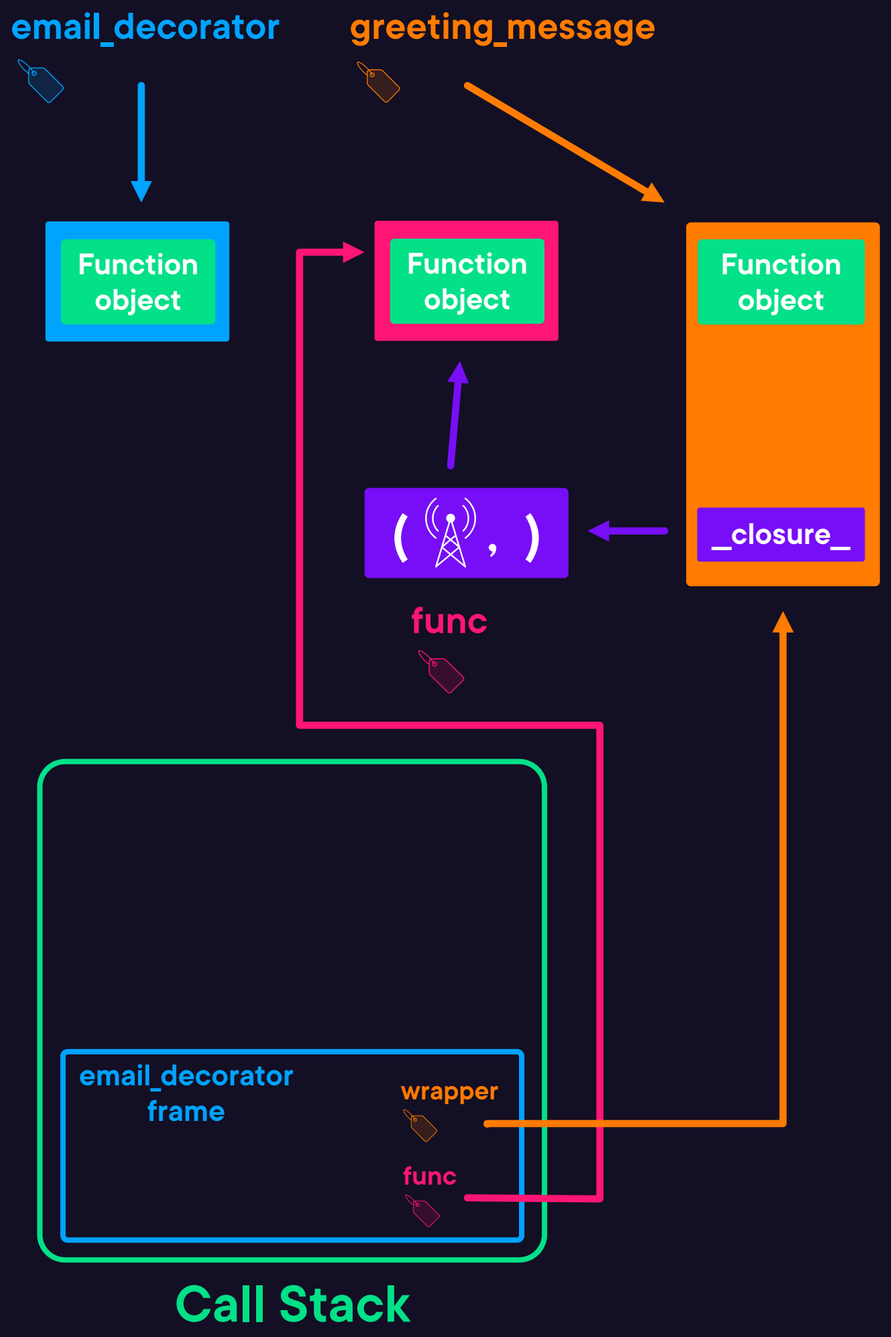



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

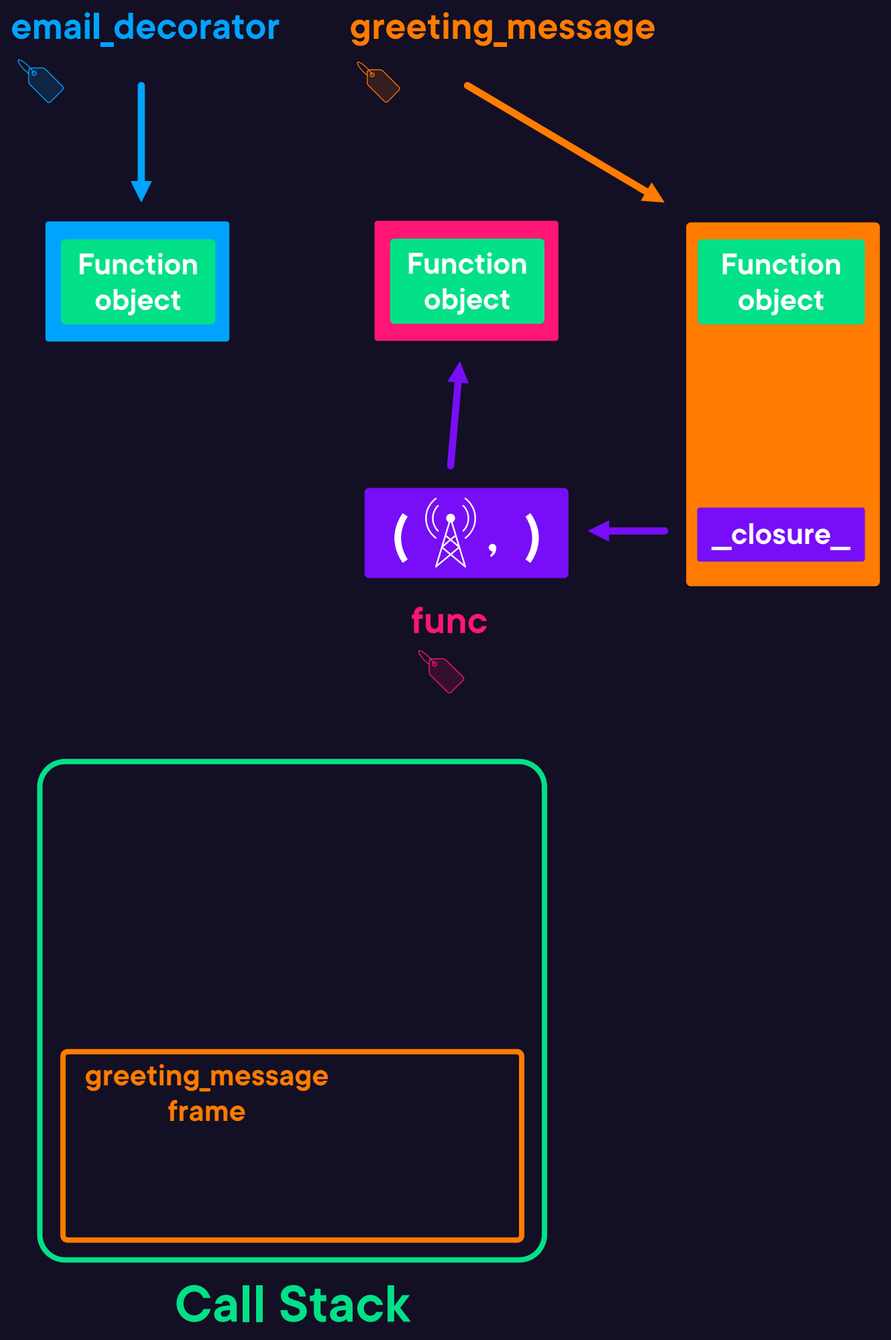



```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```




```
def email_decorator(func):
    def wrapper():
        print("Dear interns,")
        func()
        print("Best regards,")
        print("your new boss")
    return wrapper


def greeting_message():
    print("Welcome to your new job!")


greeting_message = email_decorator(greeting_message)

greeting_message()
```

Summary

Use *args and **kwargs to decorate functions with arguments

Functools.wraps decorator preserves the metadata of the decorated function

Some decorators from the standard library: @classmethod, @property, @lru_cache, @wraps

Decorator examples for timing and logging

Up Next:

Using Advanced Decorator Workflows