Decorating Classes and Class Decorators

Mateo Prigl
Software Developer

Using Classes as Decorators

Function Decorator

```
def decorator_factory(some_arg):
    def decorator_function(func):
        def wrapper():
            print("Decorator functionality")
            print(f"{some_arg}")
            return func()
        return wrapper
    return decorator_function



@decorator_factory(some_arg="value")
def some_func():
    pass
```

Class Decorator

```
class DecoratorClass:
    def __init__(self, some_arg):
        self.some_arg = some_arg

    def __call__(self, func):
        def wrapper():
            print("Decorator functionality")
            print(f"{self.some_arg}")
            return func()
        return wrapper

def some_func():
    pass

obj = DecoratorClass(some_arg="value")
some_func = obj.__call__(some_func)
```

Using Classes as Decorators

Function Decorator

```
def decorator_factory(some_arg):
    def decorator_function(func):
        def wrapper():
            print("Decorator functionality")
            print(f"{some_arg}")
            return func()
        return wrapper
    return decorator_function



@decorator_factory(some_arg="value")
def some_func():
    pass
```

Class Decorator

```
class DecoratorClass:
    def __init__(self, some_arg):
        self.some_arg = some_arg

    def __call__(self, func):
        def wrapper():
            print("Decorator functionality")
            print(f"{self.some_arg}")
            return func()
        return wrapper

def some_func():
    pass

obj = DecoratorClass(some_arg="value")
some_func = obj(some_func)
```

Using Classes as Decorators

Function Decorator

```
def decorator_factory(some_arg):
    def decorator_function(func):
        def wrapper():
            print("Decorator functionality")
            print(f"{some_arg}")
            return func()
        return wrapper
    return decorator_function



@decorator_factory(some_arg="value")
def some_func():
    pass
```

Class Decorator

```
class DecoratorClass:
    def __init__(self, some_arg):
        self.some_arg = some_arg

    def __call__(self, func):
        def wrapper():
            print("Decorator functionality")
            print(f"{self.some_arg}")
            return func()
        return wrapper
@obj
def some_func():
    pass

obj = DecoratorClass(some_arg="value")
```

Using Classes as Decorators

Function Decorator

```
def decorator_factory(some_arg):
    def decorator_function(func):
        def wrapper():
            print("Decorator functionality")
            print(f"{some_arg}")
            return func()
        return wrapper
    return decorator_function



@decorator_factory(some_arg="value")
def some_func():
    pass
```

Class Decorator

```
class DecoratorClass:
    def __init__(self, some_arg):
        self.some_arg = some_arg

    def __call__(self, func):
        def wrapper():
            print("Decorator functionality")
            print(f"{self.some_arg}")
            return func()
        return wrapper



@DecoratorClass(some_arg="value")
def some_func():
    pass
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    def get_x(self):
        return self._x

    def set_x(self, val):
        self._x = val

    # property(fget=None, fset=None, fdel=None, doc=None)
    x = property(get_x, set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    def get_x(self):
        return self._x

    def set_x(self, val):
        self._x = val


    x = property(get_x, set_x)


obj = MyClass()
obj.x = 2
print(MyClass.x.__get__(obj, MyClass))
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    def get_x(self):
        return self._x

    def set_x(self, val):
        self._x = val


    x = property(get_x, set_x)


obj = MyClass()
MyClass.x.__set__(obj, 2)
print(MyClass.x.__get__(obj, MyClass))
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    def get_x(self):
        return self._x

    def set_x(self, val):
        self._x = val


    x = property(get_x, set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    def get_x(self):
        return self._x

    def set_x(self, val):
        self._x = val


    x = property(get_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    def get_x(self):
        return self._x

    def set_x(self, val):
        self._x = val

    x = property(get_x)
    x = x.setter(set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    def set_x(self, val):
        self._x = val

    x = property(get_x)
    x = x.setter(set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    def set_x(self, val):
        self._x = val

    x = property(x)
    x = x.setter(set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    def set_x(self, val):
        self._x = val

    x = x.setter(set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, val):
        self._x = val

    x = x.setter(set_x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, val):
        self._x = val

    x = x.setter(x)


obj = MyClass()
obj.x = 2
print(obj.x)
```

Property is a Class

```
class MyClass:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, val):
        self._x = val


obj = MyClass()
obj.x = 2
print(obj.x)
```

Summary

Classes can be decorators

Property decorator is a class

Decorator syntax can also be used to decorate classes

Thank you for watching!