Теоретический материал к занятию Декораторы 3 занятие.

В декоратор можно передать и сам параметр. В этом случае нужно добавить еще один слой абстракции, то есть — еще одну функцию-обертку.
Это обязательно, поскольку аргумент передается декоратору. Затем функция, которая вернулась, используется для декорации нужной. Проще разобраться на примере.

```
def decorator_with_args(name):
    print('> decorator_with_args:', name)
    def real_decorator(func):
        print('>> сам декоратор', func.__name__)
        def decorated(*args, **kwargs):
            print('>>> перед функцие', func.__name__)
            ret = func(*args, **kwargs)
            print('>>> после функции', func.__name__)
            return ret
        return decorated
    return real_decorator

@decorator_with_args('test')
def add(a, b):
    print('>>>> функция add')
    return a + b

print('старт программы')
r = add(10, 10)
print(r)
print('конец программы')
```

Результат

```
> decorator_with_args: test
>> сам декоратор add
старт программы
>>> перед функцие add
>>>> функция add
>>> после функции add
20
конец программы
```

В декораторах-классах выполняются такие же настройки. Теперь конструктор класса получает все аргументы декоратора. Метод __call__ должен возвращать функцию-обертку, которая, по сути, будет выполнять декорируемую функцию. Например:

```
class DecoratorArgs:
    def __init__(self, name):
        print('> Декоратор с аргументами __init__:', name)
        self.name = name

    def __call__(self, func):
        def wrapper(a, b):
            print('>>> до обернутой функции')
            func(a, b)
```

```
            print('>>> после обернутой функции')
        return wrapper

@DecoratorArgs("test")
def add(a, b):
    print('функция add:', a, b)

print('>> старт')
add(10, 20)
print('>> конец')
```

Результат

```
> Декоратор с аргументами __init__: teste
>> старт
>>> до обернутой функции
функция add: 10 20
>>> после обернутой функции
>> конец
```

Один из атрибутов функции — строка документации (docstring), доступ к которой можно получить с помощью __doc__. Это строковая константа, определяемая как первая инструкция в объявлении функции.
При декорации возвращается новая функция с другими атрибутами. Но они не изменяются.

```
def decorator(func):
    '''Декоратор'''
    def decorated():
        '''Функция Decorated'''
        func()
    return decorated

@decorator
def wrapped():
    '''Оборачиваемая функция'''
    print('функция wrapped')

print('старт программы...')
print(wrapped.__name__)
print(wrapped.__doc__)
print('конец программы')
```

В этом примере функция wrapped — это, по сути, функция decorated, которую она заменяет.
Результат

```
старт программы...
decorated
Функция Decorated
конец программы
```

Вот где на помощь приходит функция wraps из модуля functools. Она сохраняет атрибуты оригинальной функции. Нужно лишь декорировать функцию wrapper с ее помощью.

```
from functools import wraps

def decorator(func):
    '''Декоратор'''
    @wraps(func)
    def decorated():
        '''Функция Decorated'''
        func()
    return decorated

@decorator
def wrapped():
    '''Оборачиваемая функция'''
    print('функция wrapped')

print('старт программы...')
print(wrapped.__name__)
print(wrapped.__doc__)
print('конец программы')
```

Результат

```
старт программы...
wrapped
Оборачиваемая функция
конец программы
```

Декоратор можно использовать для декорирования класса. Отличие лишь в том, что декоратор получает класс, а не функцию.
Singleton — это класс с одним экземпляром. Его можно сохранить как атрибут функции-обертки и вернуть при запросе. Это полезно в тех случаях, когда, например, ведется работа с соединением с базой данных.

```
def singleton(cls):
    '''Класс Singleton (один экземпляр)'''
    def wrapper_singleton(*args, **kwargs):
        if not wrapper_singleton.instance:
            wrapper_singleton.instance = cls(*args, **kwargs)
        return wrapper_singleton.instance
    wrapper_singleton.instance = None
    return wrapper_singleton

@singleton
class TheOne:
    pass

print('старт')
first_one = TheOne()
second_one = TheOne()
print(id(first_one))
print(id(second_one))
print('конец')
```

Результат

старт
56909912
56909912
конец