Создание и использование декораторов в Python

По определению, декоратор - это функция, которая принимает другую функцию и расширяет поведение последней, не изменяя ее явно. На самом деле это не так. Декораторы предоставляют простой синтаксис для вызова функций высшего порядка.

Прежде чем понять как работают декораторы и начать создавать их, вспомним, как работают функции:

1. Функции являются объектами первого класса. Это означает, что функции можно передавать и использовать в качестве аргументов.
2. Можно определить функции внутри других функций.
3. Функции умеют возвращать другие функции в качестве результата.

Теперь владея этими знаниями, применим их, что бы создать простой декоратор:

```
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper

def say():
    print('Привет Мир.')

# Декорирование
say = sample_decorator(say)

# Выполняем функцию
say()

# Я родился...
# Привет Мир.
# Меня зовут Лунтик!
```

Декорирование происходит в следующей строке:

```
say = sample_decorator(say)
```

Здесь имя say указывает на внутреннюю функцию wrapper(). **Важно понимать** то, что при вызове функции sample_decorator(say) с переданной в качестве аргумента функцией say(), возвращается вложенная функция wrapper() в качестве результата.

```
>>> say
# <function sample_decorator.<locals>.wrapper at 0x7f591a0a42f0>
```

В свою очередь функция wrapper() имеет ссылку на переданную в качестве аргумента функцию say() и вызывает эту функцию между двумя вызовами встроенной функции print().

Проще говоря: **декораторы обертывают функцию, изменяя ее поведение.**

Так как wrapper() является обычной функцией Python, следовательно декоратор может изменять переданную функцию динамически.

Новое в Python 3.9:

Декораторы теперь можно хранить в словаре, списке или другом контейнере. Обращение осуществляется согласно синтаксису контейнера:

```
# декораторы хранятся в словаре
decorators = {'dec': dec}

@decorators['dec']
def foo(x):
    print('Hello')

# или списке
decorators = [dec1, dec2, dec3]

@decorators[0]
def foo(x):
    print('Hello')
```

В ранних версиях Python на такую конструкцию поднимется исключение SyntaxError.

Примеры использования декораторов в Python:

В этом примере в разное время будет выводится разные приветствия.

```
from datetime import datetime

def say():
    print('Привет Мир.')

def time_talk(func):
    def wrapper():
        if 0 <= datetime.now().hour < 9:
            talk1 = 'Я еще не родился'
            talk2 = ''
        elif 9 <= datetime.now().hour < 13:
            talk1 = 'Я родился...'
            talk2 = 'Меня зовут Лунтик!'
        elif 13 <= datetime.now().hour < 24:
            talk1 = 'Меня зовут Лунтик.'
            talk2 = 'Поиграй со мной.'

        print(talk1)
        if talk2:
            func()
        print(talk2)
    return wrapper

say = time_talk(say)

say()
# С 13 до 24 часов будет выведено
# Меня зовут Лунтик.
# Привет Мир.
# Поиграй со мной.
```

Способ, который декорирует функцию say() - многословен, приходится набирать имя say три раза. Кроме того, декорирование скрывается под определением функции. Вместо этого Python позволяет использовать декораторы более простым способом с символом "собака" @.

Следующий синтаксис декорирования @time_talk функции talk() будет делать с последней то же самое, что и в предыдущем примере с функцией say():

```
@time_talk
def talk():
    print('Как здесь интересно!')

talk()
# С 9 до 13 часов будет выведено
# Я родился...
# Как здесь интересно!
# Меня зовут Лунтик.
```

Повторное использование декораторов.

Декоратор - это всего лишь обычная функция Python. Как мы знаем, что модули в основном состоят из функций и классов. Давайте переместим декоратор time_talk в его собственный модуль, который может быть использован во многих других функциях. Для этого создадим файл с именем decorators.py и скопируем в него код строки импорта from datetime import datetime и код определения функции time_talk(). Теперь декоратор @time_talk можно использовать следующим образом:

```
from decorators import time_talk

@time_talk
def talk():
    print('Как здорово!')

talk()
# С 0 до 9 часов будет выведено
# Я еще не родился
```

Содержание раздела: