

Декораторы

План урока

- Декораторы

Главное предназначение
декораторов – расширять
поведение объектов

ДЕКОРАТОР

это структурный шаблон проектирования или подход, предназначенный для того, чтобы динамически добавлять дополнительное поведение какому-то объекту

Пример с замыканиями

ooo | код

```
import functools

def outer(func): <-----  
  
    def inner():  
        print('**start**')  
        func()  
        print('***end***')  
  
    return inner  
  
def hello():  
    print('Hello')  
  
decorated_hello = outer(hello)  
  
decorated_hello()
```

ooo | вывод

```
>>> **start**  
>>> Hello  
>>> ***end***
```

Функция "outer" принимает в качестве аргумента параметр "func" - это будет функция

Пример с замыканиями

ooo | код

```
import functools

def outer(func):

    def inner(): ←
        print('**start**')
        func()
        print('***end***')

    return inner

def hello():
    print('Hello')

decorated_hello = outer(hello)

decorated_hello()
```

ooo | вывод

```
>>> **start**
>>> Hello
>>> ***end***
```

Определяем
функцию "inner"

Пример с замыканиями

ooo | код

```
import functools

def outer(func):

    def inner():
        print('**start**')
        func() ←
        print('***end***')

    return inner

def hello():
    print('Hello')

decorated_hello = outer(hello)

decorated_hello()
```

ooo | вывод

```
>>> **start**
>>> Hello
>>> ***end***
```

“func” - это нелокальная
переменная

Пример с замыканиями

ooo | код

```
import functools

def outer(func):

    def inner():
        print('**start**')
        func()
        print('***end***')

    return inner

def hello():
    print('Hello')

decorated_hello = outer(hello)

decorated_hello()
```

ooo | вывод

```
>>> **start**
>>> Hello
>>> ***end***
```

Вводим переменную “decorated_hello” и присваиваем ей значение вызова функции “outer” от функции “hello”

Пример с замыканиями

ooo | код

```
import functools

def outer(func):

    def inner():
        print('**start**')
        func()
        print('***end***')

    return inner

def hello():
    print('Hello')

decorated_hello = outer(hello)

decorated_hello()
```

ooo | вывод

```
>>> **start**
>>> Hello
>>> ***end***
```

Вызываем объект
“decorated_hello”

Пример с замыканиями

ooo | код

```
import functools

def outer(func):

    def inner():
        print('**start**')
        func()
        print('**end**')

    return inner

def hello():
    print('Hello')

decorated_hello = outer(hello)

decorated_hello()
```

ooo | вывод

```
>>> **start**
>>> Hello
>>> **end**
```

Выводим текст

Пример с синтаксисом декоратора

ooo | код

```
import functools

def outer(func):

    def inner():
        print('**start**')
        func()
        print('***end***')

    return inner

@outer
def hello1():
    print('Hello1')

hello1()
```

ooo | вывод

```
>>> **start**
>>> Hello1
>>> ***end***
```

Перед созданием функции,
которая будет декорироваться
методом “outer”, мы пишем
сочетание “@outer”

Пример с синтаксисом декоратора

ооо

код

```
import functools

def outer(func):

    def inner():
        print('**start**')
        func()
        print('***end***')

    return inner

@outer
def hello1():
    print('Hello1')

hello1()
```

ооо

вывод

```
>>> **start**
>>> Hello1
>>> ***end***
```

Выводим текст

Пример с аргументами

ooo | КОД

```
import functools

def start_end_decor(func): ←  
  
    def inner(name):  
        print('**start**')  
        func(name)  
        print('***end***')  
  
    return inner  
  
def hello_name(name):  
    print(f'Hello, {name}!')  
  
decorated_hello_name = start_end_decor(hello_name)  
  
decorated_hello_name('Peter')
```

Вводим функцию
“start_end_decor”

ooo | ВЫВОД

```
>>> **start**  
>>> Hello, Peter!  
>>> ***end***
```

Пример с аргументами

ooo | код

```
import functools

def start_end_decor(func):
    def inner(name):
        print('**start**')
        func(name)
        print('**end**')
    return inner

def hello_name(name):
    print(f'Hello, {name}!')

decorated_hello_name = start_end_decor(hello_name)

decorated_hello_name('Peter')
```

Функция “inner”
принимает
параметр “name”

ooo | вывод

```
>>> **start**
>>> Hello, Peter!
>>> **end**
```

Пример с аргументами

ooo | Код

```
import functools

def start_end_decor(func):

    def inner(name):
        print('**start**')
        func(name)
        print('***end***')

    return inner

def hello_name(name): ←
    print(f'Hello, {name}!')

decorated_hello_name = start_end_decor(hello_name)

decorated_hello_name('Peter')
```

Определяем
функцию "hello_name"

ooo | Вывод

```
>>> **start**
>>> Hello, Peter!
>>> ***end***
```

Пример с аргументами

ooo | КОД

```
import functools

def start_end_decor(func):

    def inner(name):
        print('**start**')
        func(name)
        print('***end***')

    return inner

def hello_name(name):
    print(f'Hello, {name}!')

decorated_hello_name = start_end_decor(hello_name)

decorated_hello_name('Peter')
```

Передаем
параметр "Peter"

ooo | ВЫВОД

```
>>> **start**
>>> Hello, Peter!
>>> ***end***
```

Пример с аргументами

ooo | Код

```
import functools

def start_end_decor(func):

    def inner(name):
        print('**start**')
        func(name)
        print('***end***')

    return inner

def hello_name(name):
    print(f'Hello, {name}!')

decorated_hello_name = start_end_decor(hello_name)

decorated_hello_name('Peter')
```

Выводим текст

ooo | ВЫВОД

```
>>> **start**      ←
>>> Hello, Peter! ←
>>> ***end***      ←
```

Пример с синтаксисом декоратора

ooo | код

```
import functools

def start_end_decor(func):

    def inner(name):
        print('**start**')
        func(name)
        print('***end***')

    return inner

@start_end_decor
def hello1_name(name):
    print(f'Hello1, {name}!')

hello1_name('Garry')
```

ooo | вывод

```
>>> **start**
>>> Hello1, Garry!
>>> ***end***
```

Функция “hello1_name”, которая вызывается параметром “Garry”

Пример с синтаксисом декоратора

ooo | код

```
import functools

def start_end_decor(func):

    def inner(name):
        print('**start**')
        func(name)
        print('***end***')

    return inner

@start_end_decor
def hello2_name(name):
    print(f'Hello2, {name}!')

hello2_name('Steven') ←
```

ooo | вывод

```
>>> **start**
>>> Hello2, Steven!
>>> ***end***
```

Функция “hello2_name”

Пример с синтаксисом декоратора

ооо | код

```
import functools

def start_end_decor(func):

    def inner(name):
        print('**start**')
        func(name)
        print('***end***')

    return inner

@start_end_decor
def hello2_name(name):
    print(f'Hello2, {name}!')

hello2_name('Steven')
```

ооо | вывод

```
>>> **start**
>>> Hello2, Steven!
>>> ***end***
```

Более универсальный декоратор

ooo | КОД

```
import functools

def stars(func):    ←

    def inner(*args, **kwargs):
        print('*' * 10)
        result = func(*args, **kwargs)
        print('*' * 10)
        return result

    return inner

@stars
def bye_name(name):
    print(f'Bye, {name}!')

bye_name('Richard')
```

Вводим функцию "stars"

ooo | ВЫВОД

```
>>> *****
>>> Bye, Richard!
>>> *****
```

Более универсальный декоратор

ooo | код

```
import functools

def stars(func):

    def inner(*args, **kwargs):
        print('*' * 10)
        result = func(*args, **kwargs)
        print('*' * 10)
        return result

    return inner

@stars
def bye_name(name):
    print(f'Bye, {name}!')
```

bye_name('Richard')

ooo | вывод

```
>>> *****
>>> Bye, Richard!
>>> *****
```

Более универсальный декоратор

о о о

КОД

```
import functools

def stars(func):

    def inner(*args, **kwargs):
        print('*' * 10)
        result = func(*args, **kwargs)
        print('*' * 10)
        return result

    return inner

@stars
def volume(a, b, c):
    print('start calculating... ')
    vol = a * b * c
    print('success')
    return vol

v = volume(1, 2, 3)

print(f'v = {v}')
```

о о о

ВЫВОД

```
>>> *****
>>> start calculating...
>>> success
>>> *****
>>> v = 6
```

Декоратор с аргументами

ooo | КОД

```
import functools

def stars_times(times):

    def inner_decorator(func):

        def inner(*args, **kwargs):
            result = []
            for i in range(times):
                print('*' * 10)
                result.append(func(*args, **kwargs))
                print('*' * 10)
            return result

        return inner

    return inner_decorator

@stars_times(times=2)
def hello3_name(name):
    print(f'Hello3, {name}!')

hello3_name('Claire')
```

ooo | ВЫВОД

```
>>> *****
>>> Hello3, Claire
>>> *****
>>> *****
>>> Hello3, Claire
>>> *****
```

Проблема идентификации функций

ooo

КОД

```
import functools

def stars_times(times):

    def inner_decorator(func):

        def inner(*args, **kwargs):
            result = []
            for i in range(times):
                print('*' * 10)
                result.append(func(*args, **kwargs))
                print('*' * 10)
            return result

        return inner

    return inner_decorator

@stars_times(times=2)
def hello3_name(name):
    print(f'Hello3, {name}')
```

hello3_name('Claire')

ooo

ВЫВОД

```
>>> *****
>>> Hello3, Claire
>>> *****
>>> *****
>>> Hello3, Claire
>>> *****
>>> inner
```

Проблема идентификации функций

о о о

КОД

```
import functools

def stars_times_fix(times):

    def inner_decorator(func):

        @functools.wraps(func)
        def inner(*args, **kwargs):
            result = []
            for i in range(times):
                print('*' * 10)
                result.append(func(*args, **kwargs))
                print('*' * 10)
            return result
        return inner
    return inner_decorator

@stars_times_fix(times=3)
def hello4_name(name):
    print(f'Hello4, {name}')

hello4_name('David')

print(hello4_name.__name__)
```

о о о

ВЫВОД

```
>>> *****
>>> Hello4, David
>>> >>> *****
***** 
>>> Hello4, David
>>> *****
>>> *****
>>> Hello4, David
>>> *****
>>> hello4_name
```