

# ДЕКОРАТОРЫ



КИРИЛЛ ТАБЕЛЬСКИЙ



**КИРИЛЛ ТАБЕЛЬСКИЙ**

Lightmap



# План занятия

1. [Паттерны проектирования](#)
2. [Как изменить функцию](#)
3. [Декоратор](#)
4. [Область видимости](#)
5. [Параметризованный декоратор](#)

# Паттерны проектирования

C Абстрактная фабрика

S Адаптер

S Мост

C Строитель

B Цепочка обязанностей

B Команда

S Компоновщик

S Декоратор

S Фасад

C Фабричный метод

S Приспособленец

B Интерпретатор

B Итератор

B Посредник

B Хранитель

C Прототип

S Прокси

B Наблюдатель

C Одиночка

B Состояние

B Стратегия

B Шаблонный метод

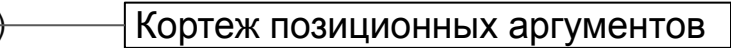

B Посетитель

# Как изменить поведение функций?

```
def foo(*args, **kwargs):  
    ...  
    return something  
  
def foo_2(*args, **kwargs):  
    ...  
    return something_2  
  
def foo_3(*args, **kwargs):  
    ...  
    return something_3
```

Предположим мы хотим принтить время вызова и return каждой функции

# args и kwargs

```
def foo(*args, **kwargs):  
    print(args)    
    print(kwargs)   
  
    return ...
```

```
foo('pos_1', 'pos_2', named_1='n1', named_2='n2')
```

**pos\_1** и **pos\_2** попадут в кортеж **args**

**args** = ('pos\_1', 'pos\_2')

**named\_1** и **named\_2** попадут в словарь **kwargs**

**kwargs** = {'named\_1': 'n1', 'named\_2': 'n2'}

## args и kwargs наоборот

```
def foo(pos_1, pos_2, named_1='n1', named_2='n2'):  
    return ...  
  
args = ('arg_1', 'arg_2')  
kwargs = {  
    'named_1': '1',  
    'named_2': '2'  
}  
  
foo(*args, **kwargs)
```

The diagram illustrates the mapping of arguments from the function call `foo(*args, **kwargs)` to the function definition `def foo(pos_1, pos_2, named_1='n1', named_2='n2')`. Dashed blue arrows show the following connections: `arg_1` from the `args` tuple to `pos_1`, `arg_2` from `args` to `pos_2`, `'named_1': '1'` from the `kwargs` dictionary to `named_1`, and `'named_2': '2'` from `kwargs` to `named_2`.

## args и kwargs пример

```
def get_average(*numbers, round_to=2):  
    result = sum(numbers) / len(numbers)  
  
    if round_to:  
        result = round(result, round_to)  
  
    return result
```

```
nums = range(12)
```

```
rounds = {'round_to': 0}
```

```
average = get_average(*nums, **rounds)
```



# Функция - это объект

```
def foo(*args, **kwargs):  
    ...  
    return
```

`foo(...)` — Вызов функции

`print(foo)` — Принт объекта без вызова

`some_variable = foo` — Передаем функцию в переменную без вызова

`foo_return = some_variable(...)` — вызываем функцию из переменной

# Функция - это объект

```
def foo(*args, **kwargs):  
    ...  
    return
```

```
def foo_2(*args, **kwargs):  
    ...  
    return
```

```
def foo_3(*args, **kwargs):  
    ...  
    return
```

```
list_of_functions = []  
list_of_functions.append(foo)  
list_of_functions.append(foo_2)  
list_of_functions.append(foo_3)
```

Положили функцию в список

```
list_of_functions[1](...)
```

Обратились к списку по индексу  
Вытащили функцию  
Вызвали

# Функцию можно передавать как аргумент

```
def foo(*args, **kwargs):
```

```
    ...
```

```
    return 42
```

Любая функция

Ее аргументы

```
def wrapper(some_function, *args, **kwargs):
```

```
    print(f'''
```

```
    Сейчас будут вызвана {some_function}
```

```
    с аргументам {args} и {kwargs}''')
```

```
    )
```

```
    result = some_function(*args, **kwargs)
```

```
    print(f'результат: {result}')
```

```
    return result
```

Вызов функции

```
result = wrapper(foo, ...)
```

```
sum_of_4_and_2 = wrapper(sum, (4, 2))
```

# Функции можно возвращать

```
def wrapper(...):  
    ...  
  
    def new_function(*args, **kwargs):  
        ...  
        return ...  
  
    return new_function
```

**wrapper** вернет функцию **new\_function**  
созданную внутри **wrapper**

**wrapper** - это фабрика функций

# Создаем функции на базу существующих

```
def wrapper(...):  
    ...  
  
    def new_function(*args, **kwargs):  
        ...  
        return ...  
  
    return new_function
```

**wrapper** вернет функцию **new\_function**  
созданную внутри **wrapper**

**wrapper** - это фабрика функций

# Декоратор

```
def print_decor(old_function):  
  
    def new_function(*args, **kwargs):  
        print(f'''  
Сейчас будет вызвана {old_function.__name__}  
с аргументами {args} и {kwargs}''')  
  
        result = old_function(*args, **kwargs)  
  
        print(f'Результат: {result}')        return result  
  
    return new_function
```

```
def summator(a, b):  
    return a + b
```

```
def div(a, b):  
    return a / b
```

```
summator = print_decor(summator)  
div = print_decor(div)
```

# Синтаксический сахар

```
def print_decor(old_function):  
    ...
```

```
@print_decor
```

```
def summator(a, b):  
    return a + b
```

аналогично

```
summator = print_decor(summator)
```

```
@print_decor
```

```
def div(a, b):  
    return a / b
```

аналогично

```
div = print_decor(div)
```

# Аргументы для декоратора

```
def some_decor(old_function, arg_1, arg_2):  
    ...  
  
@some_decor  
def foo(*args, **kwargs):  
    return ...
```



# Аргументы для декоратора

```
def decor_with_args(*args, **kwargs):  
    ...  
    def decor(old_function):  
  
        def new_function(*args, **kwargs):  
            ...  
            result = old_function(*args, **kwargs)  
            ...  
            return result  
  
        return new_function  
  
    return decor
```

**decor\_with\_args** - это фабрика декораторов

```
return decor  
some_decor = decor_with_args(...)
```

```
@some_decor  
def foo(*args, **kwargs):  
    ...
```

# Пример

```
import time
import requests

def try_decor(n_tries, timeout):
    def _try_decor(old_function):
        def new_function(*args, **kwargs):

            error = None
            for i in range(n_tries):
                try:
                    return old_function(*args, **kwargs)
                except Exception as er:
                    time.sleep(timeout)
                    error = er

            raise error
        return new_function
    return _try_decor

@try_decor(n_tries=10, timeout=3)
def get_swapi_person(person_id):
    return requests.get(f'https://swapi.dev/api/people/{person_id}').json()
```



## Дополнительные материалы

- <https://python-scripts.com/def-args-kwargs>
- <https://tproger.ru/translations/demystifying-decorators-in-python/>
- <https://pythonist.ru/classmethod-vs-staticmethod-vs-prostye-metody/>
- <https://egorovegor.ru/python-property/>
- <https://pypi.org/project/cachetools/>



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаём в чате вашей группы!
- Задачи можно сдавать по частям.
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**.

 нетология

Задавайте вопросы и напишите отзыв о лекции!

**КИРИЛЛ ТАБЕЛЬСКИЙ**