Разбор домашнего задания

Роман Булгаков
Спикер курса

Skillbox

Декоратор context manager

Роман Булгаков
Спикер курса

Skillbox

Декораторы с аргументами

Роман Булгаков
Спикер курса

Skillbox

Декораторы с аргументами

```python
def timer_with_precision(precision: int):
    def timer_decorator(func: Callable) -> Callable:
        @functools.wraps(func)
        def wrapped(*args, **kwargs):
            started_at = time.time()
            result = func(*args, **kwargs)
            ended_at = time.time()
            run_time = round(ended_at - started_at, precision)
            print('Функция работала {} секунд(ы)'.format(run_time))
            return result
        return wrapped
    return timer_decorator
```

«Классическая» версия

```python
def timer_with_precision(_func=None, *, precision: int = 10):
    def timer_decorator(func: Callable) -> Callable:
        @functools.wraps(func)
        def wrapped(*args, **kwargs):
            started_at = time.time()
            result = func(*args, **kwargs)
            ended_at = time.time()
            run_time = round(ended_at - started_at, precision)
            print('Функция работала {} секунд(ы)'.format(run_time))
            return result
        return wrapped

    if _func is None:
        return timer_decorator
    else:
        return timer_decorator(_func)
```

Модернизированная версия

Декораторы для классов

Роман Булгаков
Спикер курса

Skillbox

Задача «Время создания»

Условие задачи:

- классы
- объекты (инстансы) классов

Выходные данные:

- дата и время в момент создания каждого объекта

Итоги урока

- def createtime(cls):
- @createtime
  class Functions:
  работает именно с my_funcs

  . . . . .

  my_funcs = Functions(1000)
- @for_all_methods(timer) # timer — другой декоратор

  @for_all_methods()
- dir(cls)
- getattr(cls, i_method_name)
- setattr(cls, i_method_name, decorated_method)

Декоратор как класс

Роман Булгаков
Спикер курса

Skillbox

Реализация декораторов

```python
class CountCalls:
    def __init__(self, func: Callable) -> None:
        functools.update_wrapper(self, func)
        self.func = func
        self.num_calls = 0

    def __call__(self, *args, **kwargs) -> Any:
        self.num_calls += 1
        print("Вызов номер {num} функции {func}".format(
            num=self.num_calls, func=self.func.__name__)
        )
        return self.func(*args, **kwargs)
```

Класс

```python
def count_calls(func: Callable) -> Callable:

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        wrapper.num_calls += 1
        result = func(*args, **kwargs)
        print("Вызов номер {num} функции {func}".format(
            num=wrapper.num_calls, func=func.__name__)
        )
        return result
    wrapper.num_calls = 0
    return wrapper
```

Функция

Декоратор-класс используется **крайне редко**.
Почти всегда используется реализация функцией.

Итоги модуля

- @contextmanager
- def timer_with_precision(precision):

  def timer_decorator(...)
- @createtime

  class Functions:

  . . . . .

  my_funcs = Functions(1000)
- @for_all_methods(timer)

  dir, getattr, setattr
- @CountCalls