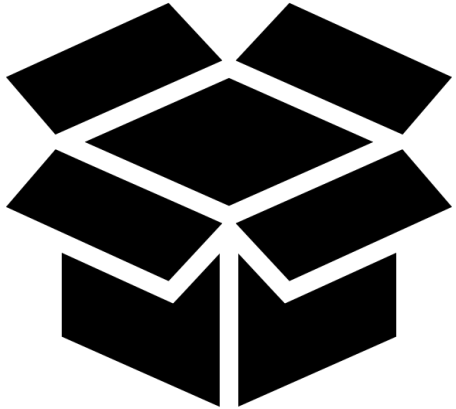
Декораторы в Python

Виталий Полшков

Функции в Python: crash course

Упаковки и распаковки



Декораторы в Python

```
def func(*args, **kwargs):
    print(type(args), args)
    print(type(kwargs), kwargs)

func(1, 2, 3, a=1, b=2, c=3)
```

```
>>> <class 'tuple'> (1, 2, 3)
<class 'dict'> {'a': 1, 'b': 2, 'c': 3}
```

Декораторы в Python

```
def somefunc(a, b, ..., *other_args, kw_a=1, kw_b=2, ..., **other_kwargs)
```

Декораторы в Python

```
first, second, *other = [1, 2, 3, 4, 5]
print(first, second, other)

>>> 1 2 [3, 4, 5]


first, second, *other = (1, 2, 3, 4, 5)
print(first, second, other)


>>> 1 2 [3, 4, 5]
```

Декораторы в Python

```
def sum_of_three(a, b, c):
    return a + b + c


three = [1, 2, 3]
print(sum_of_three(*three))


>>> 6
```

Атрибуты функции

Декораторы в Python

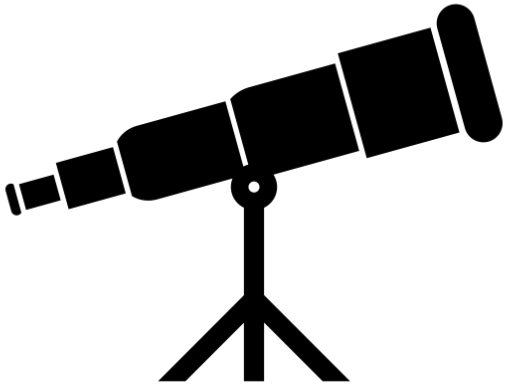
```
def identity(x):
    '''Identity function'''
    return x

f = identity
for attribute in (f.__name__, f.__doc__, f.__module__, f.__class__)
    print(attribute)


>>> identity
Identity function
__main__
<class 'function'>
```

Анонимные функции

Декораторы в Python

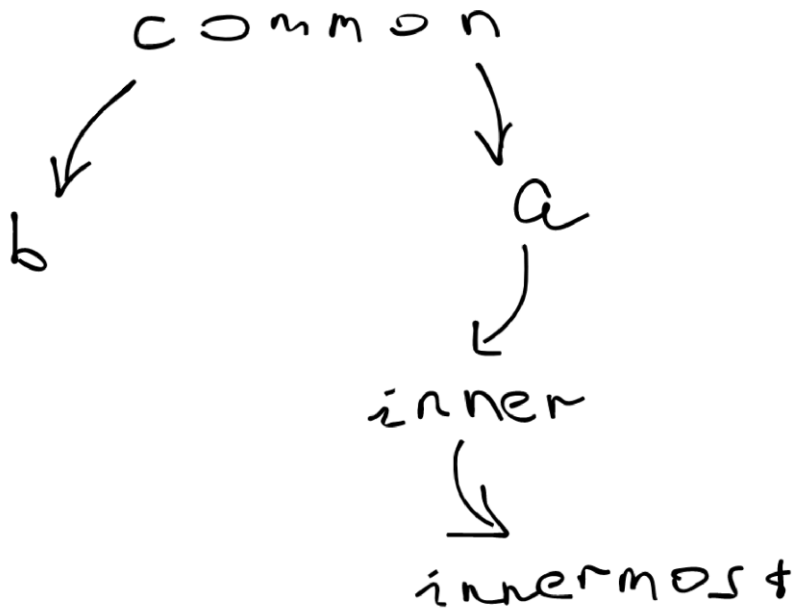
```
def add_f(x, y):
    return x + y

add_l = lambda x, y: x + y

print(add_f(17, 25), add_l(29, 13), (lambda x, y: x + y)(11, 31))


>>> 42 42 42
```

Области видимости



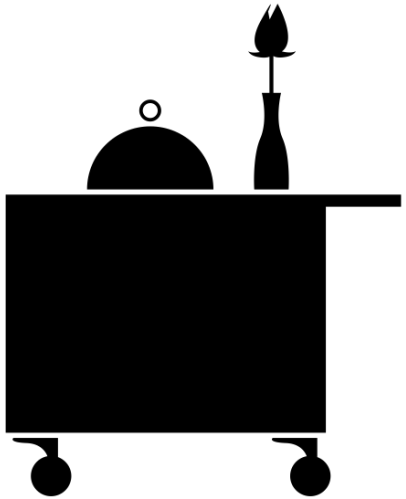
Декораторы в Python

```
def a(a_arg):
    a_var = 20

    def inner(inner_arg):
        inner_var = 42

        def innermost(innermost_arg):
            innermost_var = 101



def b(b_arg):
    b_var = 31415
```



Декораторы в Python

Функции первого класса



Декораторы в Python

```
compute_square = {'circle': lambda r: pi * r**2,
                  'rectangle': lambda a, b: a * b}
```

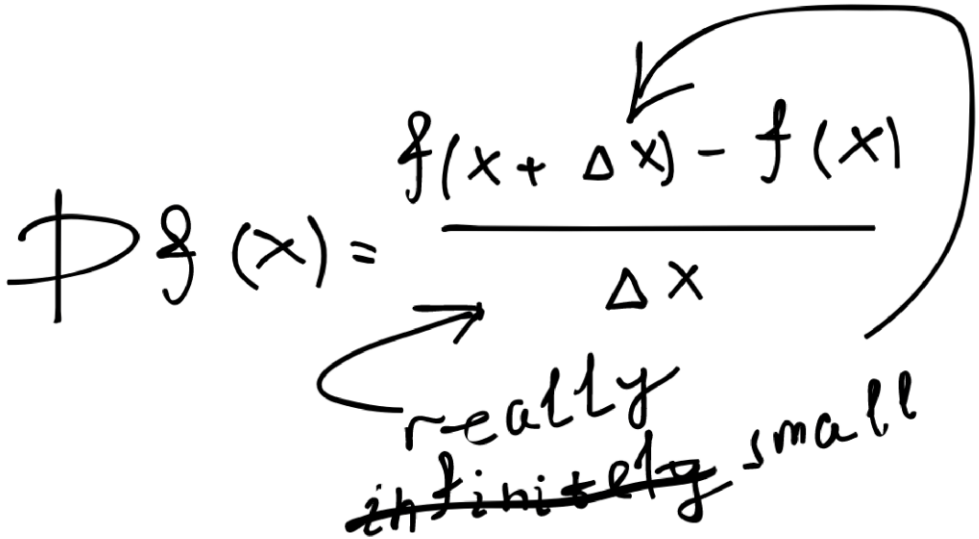
Декораторы в Python

```
xs = [3, 14, 15, 92, 65]
max_index = max((0, 1, 2, 3, 4), key=lambda i: xs[i])


print(max_index)

>>> 3
```

Декораторы в Python

Функции высшего порядка

Декораторы в Python



Декораторы в Python

```
def D(f):
    def derivative(x):
        return (f(x + 0.00001) - f(x)) / 0.00001
    return derivative

def f(x):
    return x**2

g = D(f)
print(g(5))

>>> 10.000009999799886
```

Замыкания

Декораторы в Python

```
def make_greeter(phrase):

    prefix = phrase + ', '

    def greeter(name):
        return prefix + name

    return greeter

casual_greeter = make_greeter('Hello')
informal_greeter = make_greeter('Sup')
mysterious_greeter = make_greeter('So we meet again')
```

Декораторы в Python

```
print('\n'.join((casual_greeter('World'),
                 informal_greeter('Dvach'),
                 mysterious_greeter('my heartache'))))

>>> Hello, World
Sup, Dvach
So we meet again, my heartache
```

Что такое декоратор

Мотивирующий пример

Декораторы в Python

```
from time import time
tmp = time()
func1('<...>')
print('func1 execution time:', time() - tmp)

from time import time
tmp = time()
func2('<...>')
print('func2 execution time:', time() - tmp)
```

Декораторы в Python

Ctrl-c, ctrl-v

Декораторы в Python

```
def execution_time(func, *args, **kwargs):
    from time import time
    tmp = time()
    func(*args, **kwargs)
    return time() - tmp
```

Декораторы в Python

```
def print_execution_time(func, *args, **kwargs):
    from time import time
    tmp = time()
    result = func(*args, **kwargs)
    print(fun.__name__, 'executed; time:', time() - tmp)
    return result

smth1 = print_execution_time(func1, '<...>')
smth2 = print_execution_time(func2, '<...>')
```

Декораторы в Python

Давайте помечтаем

Декораторы в Python

```
def func1('<...>'): # заменим имя func1 на dreamfunc1
    from time import time
    tmp = time()
    '<...>'
    print('func1 executed; time:', time() - tmp)
    return '<...>'


def func2('<...>'): # заменим имя func2 на dreamfunc2
    from time import time
    tmp = time()
    '<...>'
    print('func2 executed; time:', time() - tmp)
    return '<...>'
```

Декораторы в Python

Ctrl-c, ctrl-v

Декораторы в Python

```
def make_dreamfunc(func):

    def dreamfunc(*args, **kwargs):
        tmp = time()
        result = func(*args, **kwargs)
        print(func.__name__, 'executed, time:', time() - tmp)
        return result

    return dreamfunc


dreamfunc1 = make_dreamfunc(func1)
dreamfunc2 = make_dreamfunc(func2)
```

Декораторы в Python

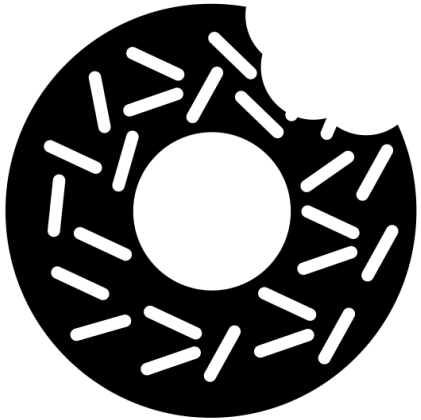
```
def timed(func):

    def timed_func(*args, **kwargs):
        tmp = time()
        result = func(*args, **kwargs)
        print(func.__name__, 'executed, time:', time() - tmp)
        return result

    return timed_func


t_func1 = timed(func1)
t_func2 = timed(func2)
```

Декораторы в Python

```
t_func = timed(func) vs func = timed(func)
```

Декораторы в Python

Пуды сахара



Декораторы в Python

```
@decorator
def func('<...>'):
    '<...>'


# то же, что и


def func('<...>'):
    '<...>'

func = decorator(func)
```

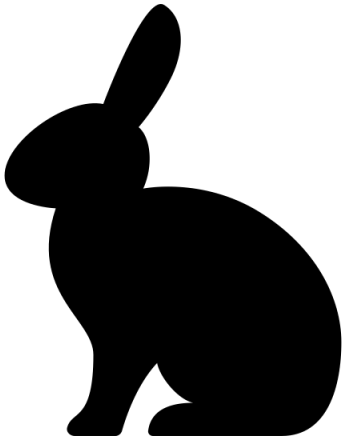
Декораторы в Python

А зачем?

Декораторы в Python

```
@sends_emails_to_bob
@logged
@timed
def fibonacci_number(n):
    '<...>' # здесь считается result
    return result

# vs

def fibonacci_number(n):
    from time import time
    tmp = time()
    logs = open('logs.txt', a)
    '<...>'  # здесь считается result
    execution_time = time() - tmp
    logs.write(func.__name__, execution_time)
    print('fibonacci_number executed; time:', execution_time_)
    send_email_to_bob('Hey! Check out how fast it is!, ', execution_time)
    return result
```

Подопытный кролик



Декораторы в Python

```
@introduce
def identity(x):
    return x

print(identity(42))

>>> identity
42
```

Декораторы в Python

```
def introduce('<...>'):

    return '<...>'
```

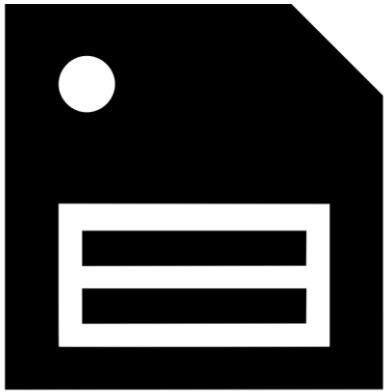
Декораторы в Python

```
def introduce(func):

    def wrapper('<...>'):
        '<...>'
        return '<...>'

    return wrapper
```

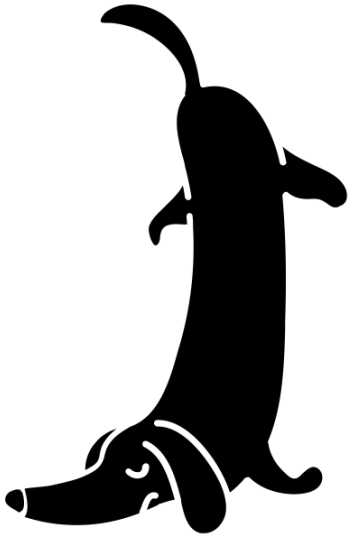
Декораторы в Python

```
def introduce(func):

    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    return wrapper
```

Мемоизация



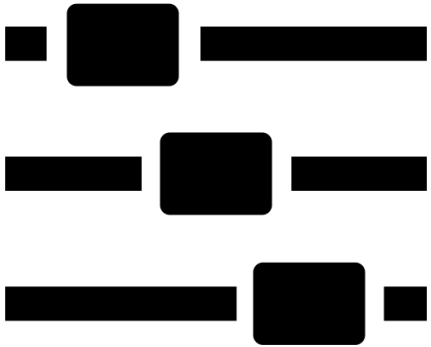
Как перевернуть игру



Декораторы в Python

```
@flip
def div(x, y, show=False):
    res = x / y
    if show:
        print(res)
    return res

div(2, 4, show=True)

>>> 2.0
```

Как добавить опций

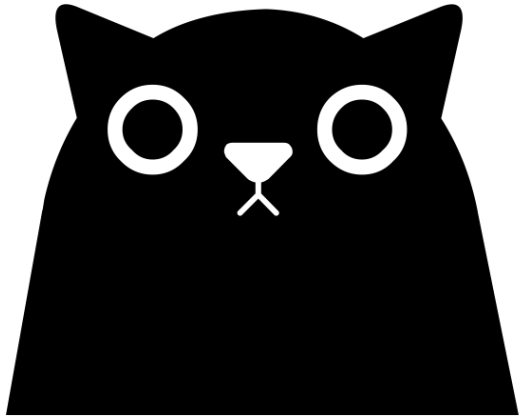


Декораторы в Python

```
@optional_introduce
def identity(x):
    return x


print(identity(20))

>>> 20


print(identity(42, introduce=True))

>>> identity
42
```

Как правильно оборачивать

Проблемы



Декораторы в Python

```
@introduce
def identity(x):
    return x


identity(20)

>>> identity


print(identity.__name__)

>>> wrapper
```

Декораторы в Python

```
def decorator(func):

    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    wrapper.__name__ = func.__name__
    wrapper.__doc__ = func.__doc__
    '<...>'

    return wrapper
```

Декораторы в Python

```
def is_wrapper(wrapper, func):
    wrapper.__name__ = func.__name__
    wrapper.__doc__ = func.__doc__
    '<...>'

def decorator(func):

    def wrapper(*args, **kwargs):
        '<...>'
        return fun(*args, **kwargs)

    is_wrapper(wrapper, func)
    return wrapper
```

Декораторы в Python

Давайте помечтаем

Декораторы в Python

```
def decorator(func):

    @wrapper_decorator
    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

```
def wrapper_deocrator(wrapper):
    wrapper.__name__ = func.__name__
    wrapper.__doc__ = func.__doc__
    '<...>'
    return wrapper
```

Декораторы в Python

```
def make_wrapper_decorator(func):

    def wrapper_decorator(wrapper):
        wrapper.__name__ = func.__name__
        wrapper.__doc__ = func.__doc__
        '<...>'
        return wrapper

    return wrapper_decorator
```

Декораторы в Python

```
def decorator(func):
    wrapper_decorator = make_wrapper_decorator(func)

    @wrapper_decorator
    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

Финт ушами

Декораторы в Python

```
def decorator(func):

    @make_wrapper_decorator(func)
    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

```
def wrapper_decorator(func):

    def actual_decorator(wrapper):
        wrapper.__name__ = func.__name__
        wrapper.__doc__ = func.__doc__
        '<...>'
        return wrapper

    return actual_decorator
```

Декораторы в Python

```
def decorator(func):

    @wrapper_decorator(func)
    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

Изобрели велосипед

Декораторы в Python

```
from functools import wraps
```

Декораторы в Python

```
def decorator(func):

    @wraps(func)
    def wrapper(*args, **kwargs):
        '<...>'
        return func(*args, **kwargs)

    return wrapper
```

Декораторы с параметрами

Хотим ещё так

Декораторы в Python

```
@introduce(4)
def identity(x):
    return x

print(identity(42))

>>> identity
identity
identity
identity

42
```

Декораторы в Python

```
# исходная версия
def introduce(func):

    @wraps(func)
     def wrapper(*args, **kwargs):
        print(func.__name__)
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

```
# используем “внешний” параметр n
def introduce(func):

    @wraps(func)
     def wrapper(*args, **kwargs):
        print((func.__name__ + '\n') * n)
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

```
# предоставляем параметр n
def make_introduce(n):

    def introduce(func):

        @wraps(func)
         def wrapper(*args, **kwargs):
            print((func.__name__ + '\n') * n)
            return func(*args, **kwargs)

        return wrapper

    return introduce
```

Декораторы в Python

```
# переименовываем всё для пущей красоты
def introduce(n):

    def actual_decorator(func):

        @wraps(func)
         def wrapper(*args, **kwargs):
            print((func.__name__ + '\n') * n)
            return func(*args, **kwargs)

        return wrapper

    return actual_decorator
```

Декораторы в Python

Давайте помечтаем

Декораторы в Python

```
def introduce(func, n, newline=True):

    @wraps(func)
    def wrapper(*args, **kwargs):
        print(('\n' if newline else ' ').join([func.__name__] * n))
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

```
@introduce(3, newline=False)
def identity(x):
    return x


print(identity(42))

>>> identity identity identity
42
```

Декораторы для декораторов

Декораторы в Python

```
@parameterized
def introduce(func, n, newline=True):

    @wraps(func)
    def wrapper(*args, **kwargs):
        print(('\n' if newline else ' ').join([func.__name__] * n ))
        return func(*args, **kwargs)

    return wrapper
```

Декораторы в Python

```
def parameterized(init_deco):

    def param_deco(<...>):
        <...>

    return param_deco
```

Декораторы в Python

```
def parameterized(init_deco):

    def param_deco(*decargs, **deckwargs):

        def res_deco(<...>):
            <...>

        return res_decorator

    return param_deco
```

Декораторы в Python

```
def parameterized(init_deco):

    def param_deco(*decargs, **deckwargs):

        def res_deco(func):
            return init_deco(func, *decargs, **deckwargs)

        return res_deco

    return param_deco
```

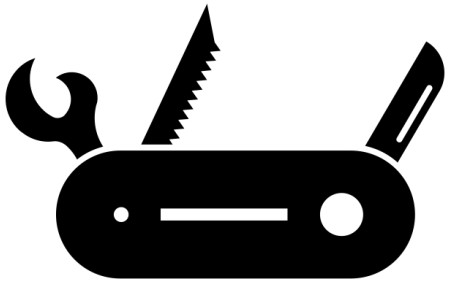
Подведение итогов

Декоратор как паттерн

Декораторы в Python

```
from functools import wraps #  <- всегда используем wraps!

def decorator(initial_function):

    @wraps(initial_function)
    def new_function(*args, **kwargs):
        # <...> всякие полезности
        return initial_function(*args, **kwargs)

    return new_function
```

Декоратор как особенность Python



Декораторы в Python

- Изменение интерфейсов (в т.ч. при помощи “магии”)
- Изменение объектов: атрибуты, методы (добавлять, менять значения)
- Регистрация, аннотация объявляемых объектов (классов, функций)
- Применение конструкторов классов, в т.ч. дескрипторов
- Предпроверка условий, предобработка входных данных
- И другие применения