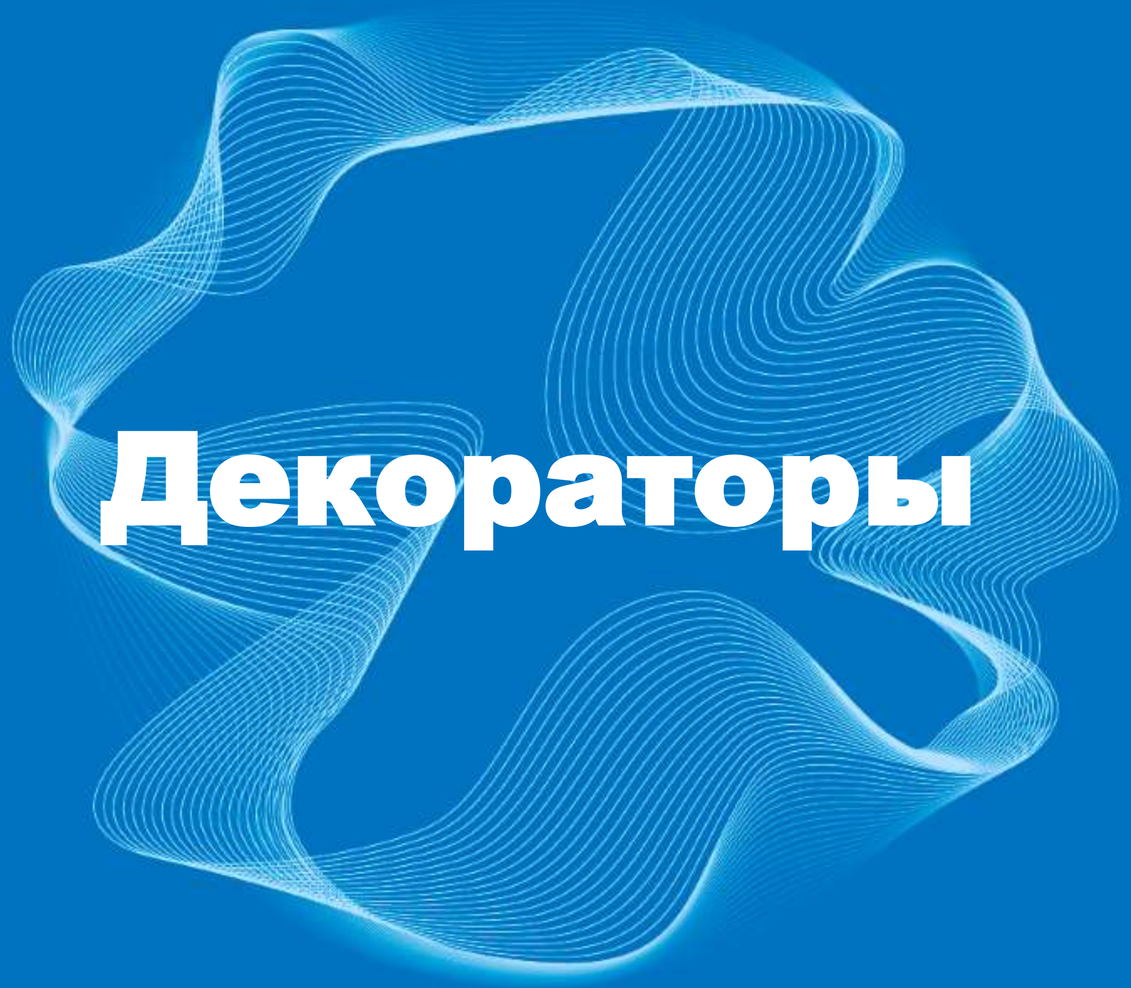
МФТИ



Декораторы

Докладчик: Евграфов Михаил

Замыкания

Глобальные переменные

```
def print_vars(var_local: int) -> None:
    print(f'{var_local = };')
    print(f'{var_global = };')
```

```
>>> var_global = 5
>>> print_vars(10)
var_local = 10;
var_global = 5;
```

Изменение глобальных объектов

```
def print_list_info(
    list_value: list, list_name: str
) -> None:
    print(
        f'{list_name} value: {list_value};',
        f'{list_name} id: {id(list_value)};',
        sep='\n',
        end='\n\n',
    )

def change_list() -> None:
    list_global = list(range(10))
    print_list_info(list_global, 'list_global')
```

Изменение глобальных объектов

```
>>> list_global = [1, 2]
>>> print_list_info(list_global, 'list_global')
>>> change_list()
>>> print_list_info(list_global, 'list_global')
list_global value: [1, 2];
list_global id: 140667361610688;

list_global value: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
list_global id: 140667119931264;

list_global value: [1, 2];
list_global id: 140667361610688;
```

Квалификация переменных

```
def func(num: int) -> None:
    print(f'{num = };')
    print(f'{some_number = };')
    print('')

>>> some_number = 9
>>> func(3)
num = 3;
some_number = 9;
```

Квалификация переменных

```
def func(num: int) -> None:
    some_number = 6
    print(f'{num = };')
    print(f'{some_number = };')
    print('')

>>> some_number = 9
>>> func(3)
num = 3;
some_number = 6;
```

Квалификация переменных

```
def func(num: int) -> None:
    print(f'{num = };')
    print(f'{some_number = };')
    some_number = 6
    print('')

>>> some_number = 9
>>> func(3)
num = 3;
...
UnboundLocalError: ...
```

global

```
def func(num: int) -> None:
    global some_number
    print(f'{num = };')
    print(f'{some_number = };')
    some_number = 6

>>> some_number = 3
>>> func(3)
>>> print(f'{some_number = };')
num = 3;
some_number = 3;
some_number = 6;
```

Квалификация переменных

```
def outer_func(num: int) -> Callable:
    outer_num = num
    def inner_func(num: int) -> None:
        print(f'inner {num = };')
        print(f'{outer_num = };')
    print(f'{outer_num = };')
    return inner_func

>>> inner_func = outer_func(10)
>>> inner_func(3)
outer_num = 10;
inner num = 3;
outer_num = 10;
```

Квалификация переменных

```
def outer_func(num: int) -> Callable:
    outer_num = num
    def inner_func(num: int) -> None:
        print(f'inner {num = };')
        print(f'{outer_num = };')
        outer_num = 5
    print(f'{outer_num = };')
    return inner_func

>>> inner_func = outer_func(10)
>>> inner_func(3)
outer_num = 10;
inner num = 3;
...
UnboundLocalError: ...
```

nonlocal

```
def outer_func(num: int) -> Callable:
    outer_num = num
    def inner_func(num: int) -> None:
        nonlocal outer_num
        print(f'inner {num = };')
        print(f'{outer_num = };')
        outer_num = 5
    inner_func(num)
    print(f'{outer_num = };')
    return inner_func

>>> inner_func = outer_func(10)
inner num = 10;
outer_num = 10;
outer_num = 5;
```

Когда не работает global

```
def count() -> int:
    global counter
    counter += 1
    return counter

>>> counter = 0
>>> count()
1
>>> count()
2
>>> count()
3
```

Когда не работает global

```
def count() -> int:
    global counter
    counter += 1
    return counter

>>> counter = 0
>>> count1 = count
>>> count2 = count
>>> count1()
1
>>> count2()
2
```

Замыкания

```
from typing import Callable


def make_counter() -> Callable[[], int]:
    counter = 0
    def count() -> int:
        nonlocal counter
        counter += 1
        return counter
    return count
```

Замыкания

```
>>> counter1 = make_counter()
>>> counter2 = make_counter()
>>> for i in range(3):
>>>     counter1()
>>> for i in range(5):
>>>     counter2()
>>> print(f'counter1: {counter1()};')
>>> print(f'counter1: {counter2()};')
counter1: 4;
counter1: 6;
```

Где лежат данные

```
>>> print(f'local variables: {counter1.__code__.co_varnames}')
>>> print(f'free variables: {counter1.__code__.co_freevars}')
local variables: ()
free variables: ('counter',)

>>> print(f'counter1 closure: {counter1.__closure__}')
>>> for i, cell in enumerate(counter1.__closure__):
>>>     print(f'counter1 cell_{i} content: {cell.cell_contents};')
counter1 closure: (<cell at 0x7fefac20e370: ...>,)
counter1 cell_0 content: 4;
```

Декораторы

Декораторы

```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print('start function')
        result = func(*args, **kwargs)
        print('finished function')
        return result
    return wrapper

@my_decorator
def do_something():
    print('do_something')

>>> do_something()
start function
do_something
finished function
```

Синтаксический сахар

```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print('start function')
        result = func(*args, **kwargs)
        print('finished function')
        return result
    return wrapper

def do_something():
    print('do_something')

>>> do_something = my_decorator(do_something)
>>> do_something()
start function
do_something
finished function
```

Подмена

```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print('start function')
        result = func(*args, **kwargs)
        print('finished function')
        return result
    return wrapper

@my_decorator
def do_something():
    print('do_something')

>>> do_something.__name__
'wrapper'
```

wraps

```
from functools import wraps

def my_decorator(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('start function')
        result = func(*args, **kwargs)
        print('finished function')
        return result
    return wrapper

@my_decorator
def do_something():
    print('do_something')

>>> do_something.__name__
'do_something
```

Момент выполнения декораторов

```
def decorate(func):
    print('run decorate')
    return func

@decorate
def do_something() -> None:
    print('do_something')

@decorate
def do_another_thing() -> None:
    print('do_another_thing')

run decorate
run decorate
```

Композиция декораторов

```
def outer(func):
    def wrapper(*args, **kwargs):
        print('outer')
        result = func(*args, **kwargs)
        return result
    return wrapper

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

Композиция декораторов

```
@outer
@inner
def do_something() -> None:
    print('do_something')

>>> do_something()
outer
inner
do_something
```

Композиция декораторов

```
def do_something() -> None:
    print('do_something')

>>> do_something = outer(inner(do_something))
>>> do_something()
outer
inner
do_something
```

Параметризованный декоратор

```
from functools import wraps
from typing import Callable

def print_greeting(greeting: str) -> Callable:
    def _print_greeting(func):
        @wraps(func)
        def _wrapper(*args, **kwargs):
            print(f'{greeting}
{func.__name__}')
            result = func(*args, **kwargs)
            return result
        return _wrapper
    return _print_greeting
```

Параметризованный декоратор

```
@print_greeting('Hello world! Now I will run:')
def do_something() -> None:
    print('do something')

>>> do_something()
Hello world! Now I will run: do_something
do something
```

Параметризованный декоратор

```
def do_something() -> None:
    print('do something')

>>> do_something = print_greeting("Run:
")(do_something)
>>> do_something()
Run:  do_something
do something
```

Семинар