Разбор домашнего задания

Роман Булгаков
Спикер курса

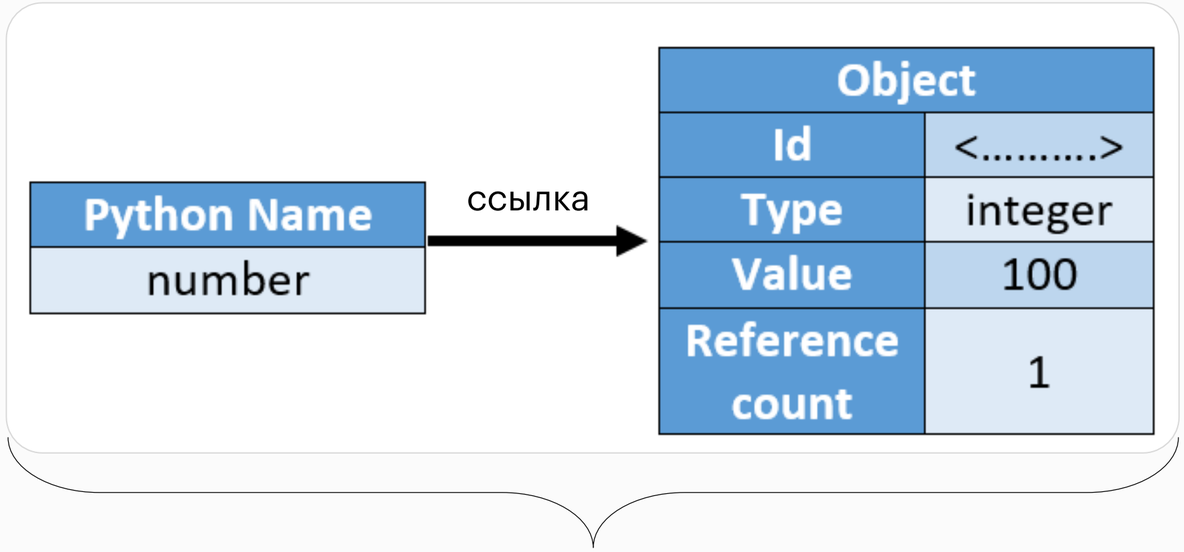
Skillbox

Пространство имён и области видимости

Роман Булгаков
Спикер курса

Skillbox

Пространство имён



Пространство имён (namespace)

Реализовано в виде словаря:
{text: 'example', number: 100, my_list: [1, 2, 3]}

```
def test(num):
    return num + 10


number = 100
text = 'example'
my_list = [1, 2, 3]
```

Виды пространств имён

- **Локальное пространство имён (local namespace).** Это пространство имён содержит локальные имена внутри функции. Создаётся при вызове функции и продолжается до тех пор, пока функция не вернётся

```
def test(num):
    return num + 10
```

- **Глобальное пространство имён (global namespace).** Включает имена из основного кода и из различных импортированных модулей. Существует до завершения скрипта

```
number = 100
text = 'example'
my_list = [1, 2, 3]
```

- **Встроенное пространство имён (build-in namespace).** Содержит встроенные функции и имена исключений

```
list, tuple, sum, ...
```

Области видимости

- Локальная область (**local scope**). Является самой внутренней областью, которая содержит список локальных имён, доступных в текущей функции
- Область всех закрывающих функций (**enclosing scope**). Поиск имени начинается с ближайшей охватывающей области и перемещается наружу
- Глобальная область (**global scope**). Область уровня модуля, содержащая все глобальные имена из текущего модуля. Следующий уровень поиска имени
- «Встроенная» область (**build-in scope**). Содержит список всех встроенных имён. Поиск имени в этой области выполняется самым последним

```
def f3():
    def f4():
        number = 10
        print('Внутри f3/f4 num =', number)
    number = 30
    f4()


number = 100
```

Lambda-функции

Роман Булгаков
Спикер курса

Skillbox

Использование lambda-функций

“

Всегда используйте оператор def вместо оператора присваивания, который связывает лямбду непосредственно с идентификатором.

PEP8

```
def f(x): return 2 * x
```

Хорошо

```
f = lambda x: 2 * x
```

Плохо

```
users = ['user1', 'user2', 'user30', 'user3', 'user22', 'user100']
print(sorted(users))
sorted_users = sorted(users, key=lambda elem: int(elem[4:]))
print(sorted_users)
```

Pythonic way

Плохие примеры использования lambda

- Вызов исключения в lambda-выражении

```
>>> def some_exception(ex): raise ex
...
>>> (lambda: some_exception(Exception('Какая-то ошибка')))()
```

- Cryptonic style

```
>>> (lambda _: list(map(lambda _: _ // 3, _)))([1,2,3,4,5,6,7,8,9,10])
[0, 0, 1, 1, 1, 2, 2, 2, 3, 3]
```

- Lambda как метод класса

```
class Person:
    def __init__(self, name: str, age: int) -> None:...

    name = property(lambda self: getattr(self, '_name'),
                    lambda self, value: setattr(self, '_name', value))
```

Общие проблемы всех примеров:

- каждый из них не следует руководству по стилю Python (PEP 8)
- код выглядит громоздким и трудно читаемым

Функции map и filter

Роман Булгаков
Спикер курса

Skillbox

Функция map против list comprehensions

```
animals = ['cat', 'dog', 'cow']
```

Нужно: ['Cat', 'Dog', 'Cow']

Решение с map:

```
new_animals = list(map(lambda elem: elem.capitalize(), animals))
```

Решение с list comprehension:

```
new_animals = [elem.capitalize() for elem in animals]
```

- map удобно использовать для ленивых вычислений
- lambda замедляет map, но он всё равно **обычно** быстрее list comprehensions (если выражение достаточно простое)

Функциональное программирование

Функции **lambda**, **map**, **filter**, **zip**, а также **reduce** являются элементами функционального программирования.

```python
def timer(func: Callable) -> Callable:
    """ Декоратор. Выводит время работы функции или метода """
    @functools.wraps(func)
    def wrapped(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print('Время работы функции:', end - start)
        return result
    return wrapped
```

Это ФП

```python
@property
def age(self):
    return self.__age

@age.setter
def age(self, age):
    self.__age = age
```

И это ФП

```python
def fibonacci(number):
    cur_val = 0
    next_val = 1
    for _ in range(number):
        yield cur_val
        cur_val, next_val = next_val, cur_val + next_val
```

И это тоже ФП

Специальная переменная __name__

Роман Булгаков
Спикер курса

Skillbox

Итоги модуля

- Пространства имён (namespaces) и области видимости (scopes) (local, global, build-in)
- key=lambda elem: int(elem[4:])
- day, month, year = map(int, date_as_string.split('-'))
- filter(lambda x: x % 2 == 0, result)
- if __name__ == "__main__":
  # основной код