Функции, встроенные функции и тестирование

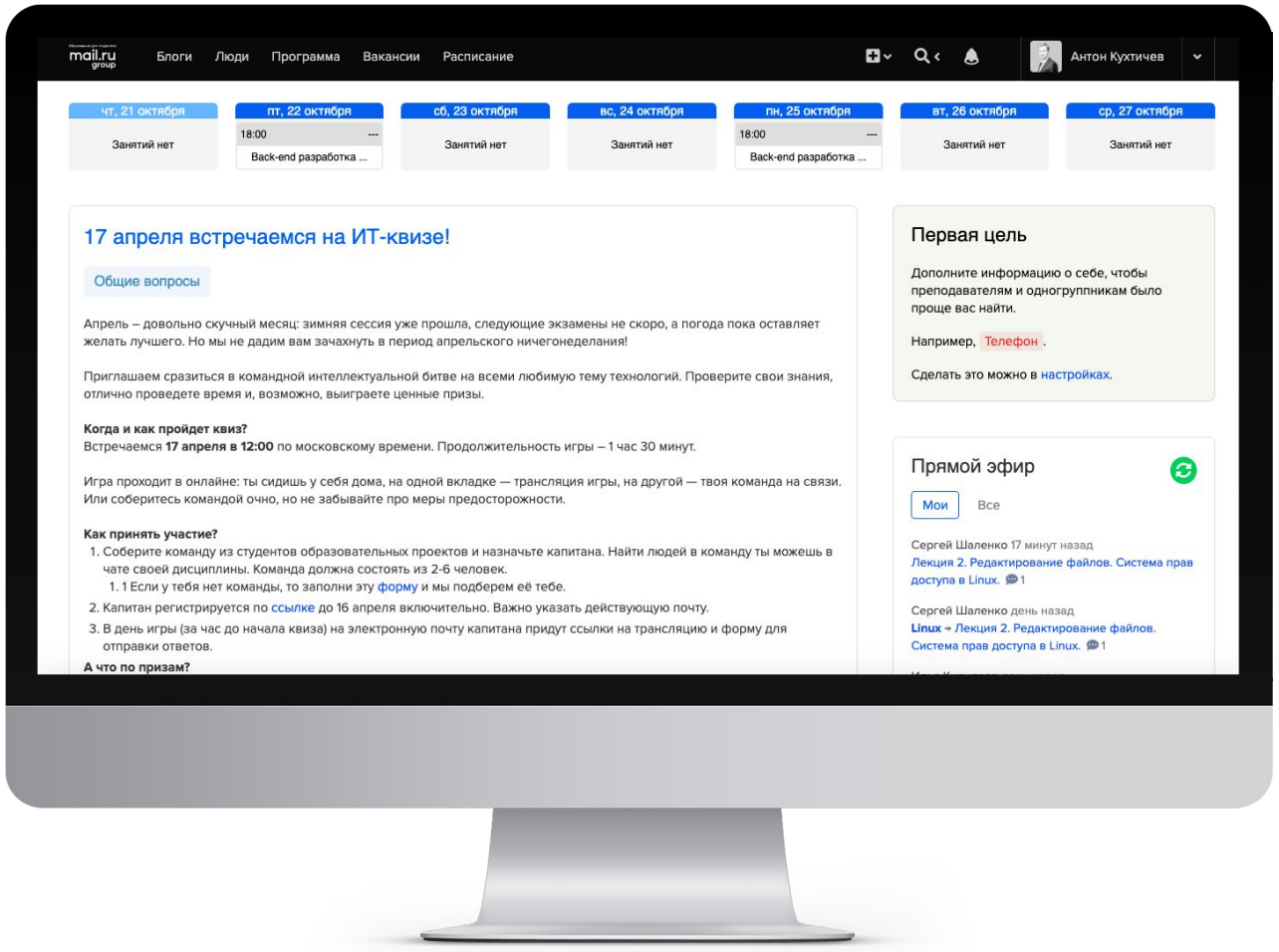
Антон Кухтичев

Содержание занятия

- Квиз
- Функции
- Аргументы функции
- Декораторы
- λ-функция
- Встроенные функции
- Тестирование

Напоминание отметиться на портале

и оставить отзыв после лекции



Квиз #1

По первой лекции
https://forms.gle/GqS6RKRiFLKNtjBy9

Функции и их аргументы

Функции

```
def square(x):
    return x * x
>>> square(4)
16
```

Правила наименования:

- имя функции состоит из букв, чисел, знака подчёркивания (_);
- название функции не должно начинаться с цифры;
- лидирующий знак подчёркивания - соглашение, что функция приватная.

Аргументы функции

```
>>> def func(a, b, c=2): # c - необязательный аргумент
...     return a + b + c
```

- может принимать произвольное количество аргументов;
- а может и не принимать их вовсе;
- может принимать и произвольное число именованных аргументов;
- у аргументов может быть значение по умолчанию;

Декоратор

```
def square(func):
    def wrapper(*args, **kwargs):
        # какая-то логика
    return wrapper
```

Декоратор - это функция, принимающая единственный аргумент - другую функцию и выполняющая дополнительную логику.

λ-функция

```
f = lambda x: x * x
>>> f(2)
4
```

- Работают быстрее классических функций;
- Полезны в случае, когда нужна одноразовая функция;
- Потенциально повышают читаемость кода, а могут понизить.

Встроенные функции

map

```
map(function, iterable, [iterable 2, iterable 3, ...])
```

```
def func(el1, el2):
    return '%s|%s' % (el1, el2)

list(map(func, [1, 2], [3, 4, 5]))  # ['1|3', '2|4']
```

Применяет указанную функцию к каждому элементу указанной последовательности/последовательностей.

reduce

```
from functools import reduce

reduce(function, iterable[, initializer])

items = [1,2,3,4,5]

sum_all = reduce(lambda x,y: x + y, items)
```

Применяет указанную функцию к элементам последовательности, сводя её к единственному значению.

filter

```
def is_even(x):
    return x % 2 == 0:
>>> print(list(filter(is_even, [1, 3, 2, 5, 20, 21])))
[2, 20]
```

Функция filter предлагает элегантный вариант фильтрации элементов последовательности. Принимает в качестве аргументов функцию и последовательность, которую необходимо отфильтровать.

zip

```
>>> a = [1,2,3]
>>> b = "xyz"
>>> c = (None, True)
>>> print(list(zip(a, b, c)))
[(1, 'x', None), (2, 'y', True)]
```

Функция zip объединяет в кортежи элементы из последовательностей переданных в качестве аргументов.

compile

```
compile(source, filename, mode, flag, dont_inherit, optimize)

# выполнение в exec
>>> x = compile('x = 1\nz = x + 5\nprint(z)', 'test', 'exec')
>>> exec(x)
# 6

# выполнение в eval
>>> y = compile("print('4 + 5 =', 4+5)", 'test', 'eval')
>>> eval(y)
# 4 + 5 = 9
```

exec

```
exec(obj[, globals[, locals]]) -> None
```

eval

```
eval(expression[, globals[, locals]])
```

- в eval() запрещены операции присваивания;
- SyntaxError также вызывается в случаях, когда eval()не удается распарсить выражение из-за ошибки в записи;
- Аргумент globals опционален. Он содержит словарь, обеспечивающий доступ eval() к глобальному пространству имен;
- В locals-словарь содержит переменные, которые eval() использует в качестве локальных имен при оценке выражения.

Тестирование

Цели тестирования

«Тестирование показывает присутствие ошибок, а не их отсутствие.»

— Дейкстра

Тестированием можно доказать неправильность программы, но нельзя доказать её правильность.

Цели тестирования

- Проверка правильности реализации;
- Проверка обработки внештатных ситуаций и граничных условий;
- Минимизация последствий.

Виды тестирования (1)

- **Unit-тестирование**

  Проверяют функциональное поведение для отдельных компонентов, часто классов и функций.

- **Интеграционное тестирование**

  Проверка совместной работы групп компонентов. Данные тесты отвечают за совместную работу между компонентами, не обращяя внимания на внутренние процессы в компонентах.

Виды тестирования (2)

- **Функциональное тестирование (несколько функций)**;
- **Тестирование производительности (performance testing)**
  - Нагрузочное тестирование (load testing)
  - Стресс тестирование (stress testing)
- **Регрессионное тестирование (regression testing)**

  Тесты которые воспроизводят исторические ошибки (баги). Каждый тест вначале запускается для проверки того, что баг был исправлен, а затем перезапускается для того, чтобы убедиться, что он не был внесен снова с появлением новых изменений в коде.

Test driven development

TDD – test driven development – техника разработки ПО, основывается на повторении коротких циклов разработки: пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода.

TDD. Алгоритм

- Пишем тест, в котором проверяем, что функция someCode() возвращает нужные значения в разных ситуациях;
- Проверяем, что тесты упали (кода еще нет);
- Пишем код функции очень просто — так чтобы тесты прошли;
- Проверяем, что тесты прошли;
- На этом шаге можем задуматься о качестве кода. Можем спокойно рефакторить и изменять код как угодно;
- Повторяем все вышеуказанные шаги еще раз.

Покрытие тестов

Степень покрытия тестами (test coverage)

`coverage` - библиотека для проверки покрытия тестами.

```
pip install coverage
# общий случай
coverage run tests.py
coverage report -m
coverage html
# django
coverage run --source='.' manage.py test myapp
coverage report
```

Инструменты для тестирования в Python

Инструменты для тестирования в Python

- doctest
- pytest
- hypothesis
- unittest

doctest

```
def multiply(a, b):
    """
    >>> multiply(4, 3)
    12
    >>> multiply('a', 3)
    'aaa'
    """
    return a * b
```

Запуск

```
python -m doctest <file>
```

pytest

```
class TestClass(object):
    def test_one(self):
        x = 'this'
        assert 'h' in x
    def test_two(self):
        x = 'hello'
        assert hasattr(x, 'check')
```

Запуск

```
pytest <file>
```

hypothesis

```
from hypothesis import given

from hypothesis.strategies import text


@given(text())

def test_decode_inverts_encode(s):

    assert decode(encode(s)) == s
```

https://hypothesis.readthedocs.io/en/latest/quickstart.html

unittest

unittest (1)

- `def setUp(self)` — Установки запускаются перед каждым тестом
- `def tearDown(self)` — Очистка после каждого метода
- `def test_<название теста>(self)` — Код теста;

unittest (2)

```
import unittest

class FirstTestClass(unittest.TestCase):
    def test_upper(self):
        self.assertEqual('text'.upper(), 'TEXT')

if __name__ == '__main__':
    unittest.main()
```

Запуск

```
python -m unittest <file>
```

unittest. Расширенный набор проверок assert

- `assertEqual(a, b)`
- `assertNotEqual(a, b)`
- `assertTrue(x)`
- `assertFalse(x)`
- `assertIs(a, b)`
- `assertIsNot(a, b)`
- `assertIsNone(x)`
- `assertIn(a, b)`
- `assertIsInstance(a, b)`
- `assertLessEqual(a, b)`
- `assertListEqual(a, b)`
- `assertDictEqual(a, b)`
- `assertRaises(exc, fun, *args, **kwargs)`
- `assertJSONEqual(a, b)`

mock (1)

Mock — это просто объект, который создает пустой тест для определенной части программы.

- Высокая скорость
- Избежание нежелательных побочных эффектов во время тестирования

```
from unittest.mock import patch

class TestUserSubscription(TestCase):
    @patch('users.views.get_subscription_status', return_value=True)
    def test_subscription(self, get_subscription_status_mock):
        ...
```

mock (2)

Атрибуты объекта Mock с информацией о вызовах

- `called` — вызывался ли объект вообще
- `call_count` — количество вызовов
- `call_args` — аргументы последнего вызова
- `call_args_list` — список всех аргументов
- `method_calls` — аргументы обращений к вложенным методам и атрибутам
- `mock_calls` — то же самое, но в целом и для самого объекта, и для вложенных

```
# Пример
self.assertEqual(get_subscription_status_mock.call_count, 1)
```

Factory boy

factory_boy

Библиотека `factory_boy` служит для генерации тестовых объектов (в т.ч. связанных) по заданным параметрам.

```
# Объявляем фабрику
class RandomUserFactory(factory.Factory):
    class Meta:
        model = models.User
    first_name = factory.Faker('first_name', locale='ru_RU')
    last_name = factory.Faker('last_name')
    email = factory.Sequence(lambda n: f'person{n}@example.com')
```

factory_boy, провайдеры

- `paragraph(nb_sentences=3, variable_nb_sentences=True, ext_word_list=None)`
- `sentence(nb_words=6, variable_nb_words=True, ext_word_list=None)`
- `text(max_nb_chars=200, ext_word_list=None)`
- `word(ext_word_list=None)`
- `first_name(), last_name(), name()`

Подробнее тут

factory_boy

```
# Создаёт объект User, которые не сохранён.

user = RandomUserFactory.build()

# Возвращает сохранённый объект User.

user = RandomUserFactory.create()

# Returns a stub object (just a bunch of attributes)

obj = RandomUserFactory.stub()

users = RandomUserFactory.build_batch(10, first_name="Tuco")
```

Запуск тестов. Общий случай

```
# Найти и выполнить все тесты
python -m unittest discover

# Тесты нескольких модулей
python -m unittest test_module1 test_module2

# Тестирование одного кейса - набора тестов
python -m unittest tests.SomeTestCase

# Тестирование одного метода
python -m unittest tests.SomeTestCase.test_some_method
```

Selenium

Selenium (1)

Selenium WebDriver – это программная библиотека для управления браузерами. WebDriver представляет собой драйверы для различных браузеров и клиентские библиотеки на разных языках программирования, предназначенные для управления этими драйверами.

```
# Установка
pip install selenium
```

Selenium (2)

- Требует конкретного драйвера для конкретного браузера (Chrome, Firefox и т.д.);
- Автоматическое управление браузером;
- Поддержка Ajax;
- Автоматические скриншоты;

Виды тестирования. Selenium

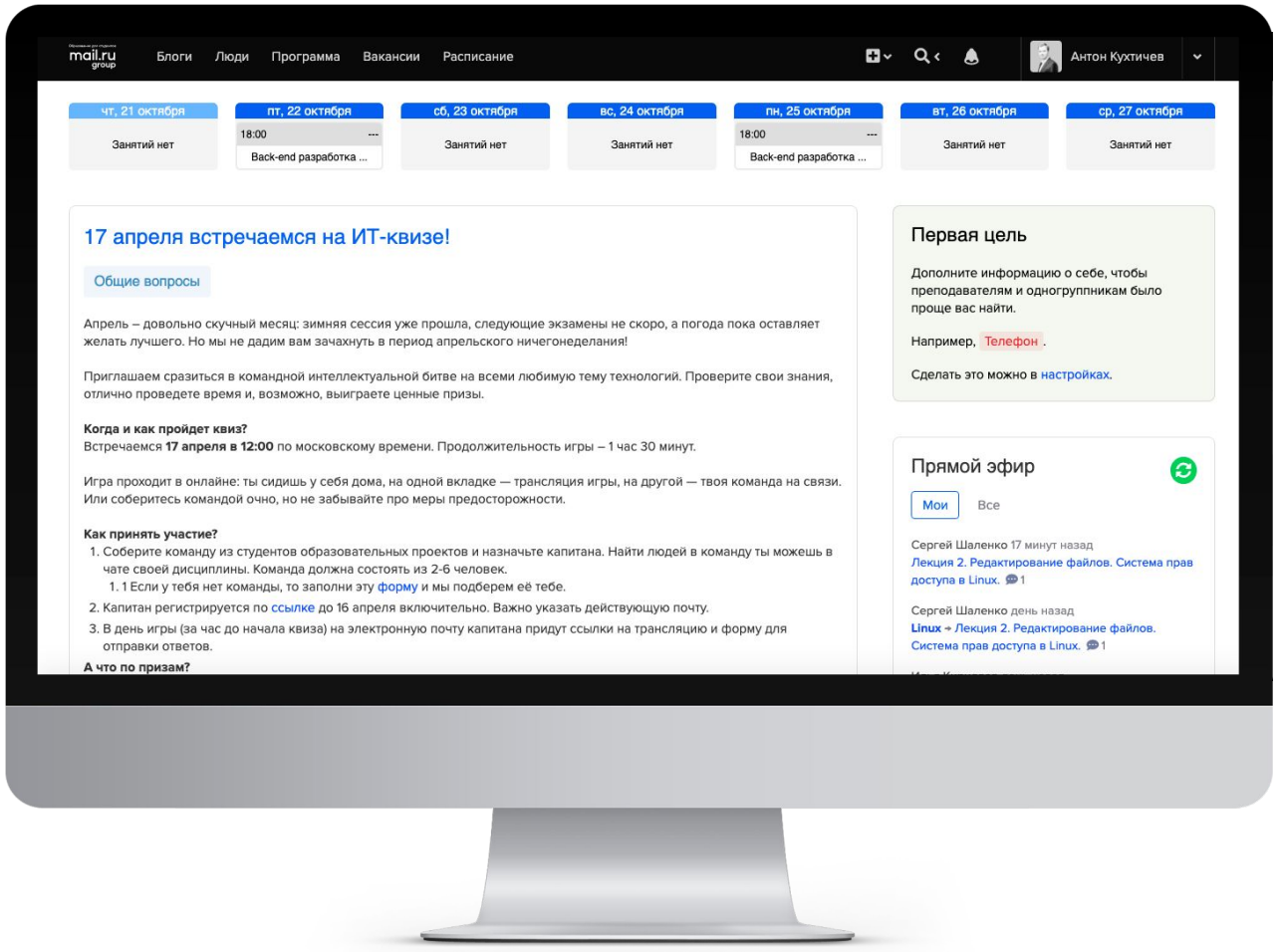
```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

class PythonOrgSearch(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()

    def test_search_in_python_org(self):
        driver = self.driver
        driver.get("http://www.python.org")
        self.assertIn("Python", driver.title)
        ...
        elem = driver.find_element_by_name("q")
        elem.send_keys("pycon")
        elem.send_keys(Keys.RETURN)
        assert "No results found." not in
driver.page_source

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    unittest.main()
```

Домашнее задание

- Написать функцию, которая в качестве аргументов принимает:
  - строку html;
  - 3 функции-обработчика тегов: открытие тега, текст между тегами, закрытие тега
- Использовать mock-объект при тестировании;
- Использовать factory boy;
- Узнать степень покрытия тестами с помощью библиотеки coverage.

Напоминание оставить отзыв

Это правда важно



Спасибо за внимание!

Вопросы?