

# Занятие 30

Unittest, pylint

# Задание 29-1

Дан список, который состоит из одинаковых чисел за исключением одного.  
Найдите это число.

## Задача 29-2

Дана квадратная матрица, напишите функцию, которая возвращает матрицу, полученную вращением по или против часовой стрелки.

```
matrix = [[1, 2, 3],  
          [4, 5, 6],  
          [7, 8, 9]]
```

```
rotate(matrix, «по часовой») # ----> [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

# Задача 29-3

Напишите функцию, которая проверяет, являются ли два слова изоморфными.

Два слова изоморфны, если буквам одного слова можно сопоставить (map) буквам другого слова.

True:

CBAABC DEFFED

XXX YYY

RAMBUNCTIOUSLY THERMODYNAMICS

False:

AB CC

XXY XYY

ABAB CD

# О примерной структуре презентации по работе

1. Титульный слайд – Название работы, автор, курс (зима 2023)
2. Содержание презентации или план
3. Цели
4. Задачи, спецификация, бизнес-проблема
5. Особенности, специфика
6. Функционал, реализованный и планируемый к реализации
7. Теория, подходы к решению
8. Что реализовано, что предстоит в планах реализовать
9. Структуры, схемы, алгоритмы и прочее
10. Исползованные технические средства, модули, и т.д.
11. Демонстрация
12. Любые другие пункты, которые вы хотите включить. Может быть какие-то главные формулы или гениальные строки кода и т.д.

## Комментарии.

1. Презентация не должна быть полноценным докладом с полным текстом
2. Графические представления приветствуются
3. Это скорее опорный конспект, но если вы не уверены в себе, как докладчик, то можно написать ключевые фразы
4. Обязательно проговорите ее перед зеркалом вслух. Выяснится много интересных деталей, засекайте время – не больше 10 минут.
5. Если есть сокращения, то лучше их расшифровать, если это не общепринятые.
6. Это ваша презентация, сколько вам надо слайдов столько и делайте. Если надо больше, значит больше, если надо меньше, значит меньше

BUILDING A PYTHON APP IN

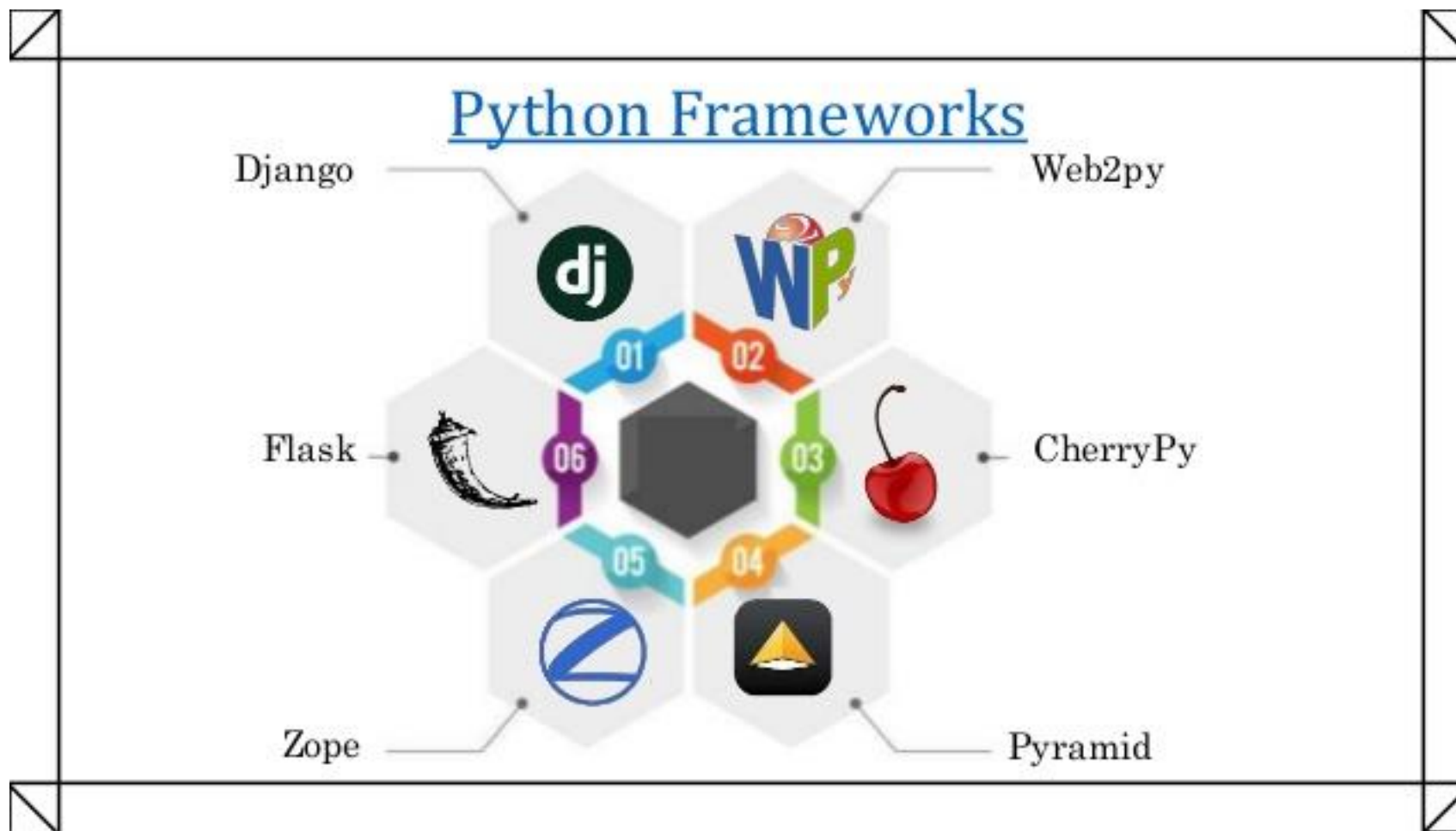
# FLASK



# Web service

- Web service – сервис (набор методов), предоставляемый приложением и доступный по сети
- Стандартизированный способ взаимодействия разнородных приложений
- Представление услуг для любого приложения

# Выбор микрофреймворка





# Установка Flask, Flask-Alchemy

Pycharm: File / Settings / Project ... / Interpretator / + / имя модуля

или

`pip install имя модуля`

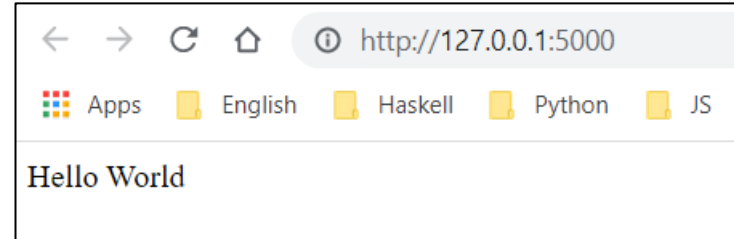
# Первая программа

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')  
def index():  
    return 'Hello World'
```

```
if __name__ == "__main__":  
    app.run(debug = True)
```



# Маршруты (Routing)

- from markupsafe import escape
- `@app.route('/user/<username>')`
- `def show_user_profile(username):`
- `# show the user profile for that user`
- `return f'User {escape(username)}'`
- `@app.route('/post/<int:post_id>')`
- `def show_post(post_id):`
- `# show the post with the given id, the id is an integer`
- `return f'Post {post_id}'`
- `@app.route('/path/<path:subpath>')`
- `def show_subpath(subpath):`
- `# show the subpath after /path/`
- `return f'Subpath {escape(subpath)}'`

# Вторая программа

```
from flask import Flask, render_template
```

```
menu = ["Первый", "Второй", "Третий"]
```

```
app = Flask(__name__)
```

```
@app.route('/index')
```

```
@app.route('/')
```

```
def index():
```

```
    return render_template('index.html', title = 'Про Flask', menu = menu)
```

```
@app.route('/about')
```

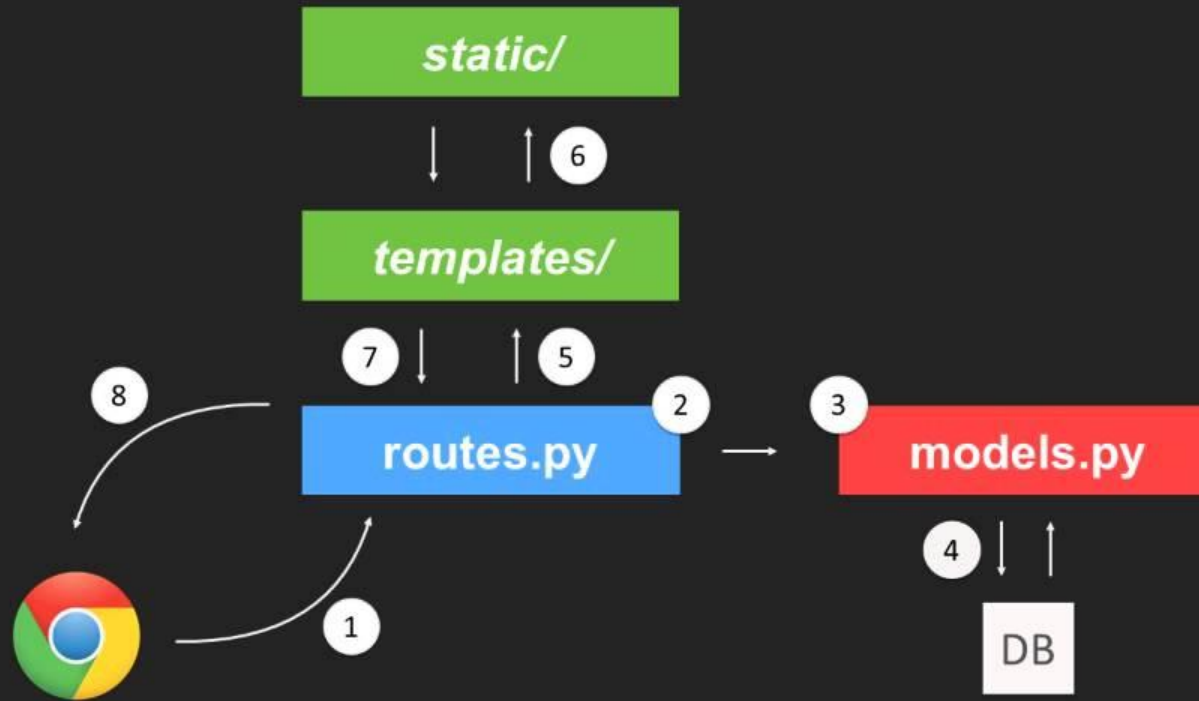
```
def about():
```

```
    return render_template('about.html', title = 'О сайте')
```

```
if __name__ == "__main__":
```

```
    app.run(debug=True)
```

## The Request-Response Cycle



# Шаблон – index.html, about.html (в \templates)

```
<!DOCTYPE html>
<head>
  <title> {{ title }} </title>
</head>
<body>
<H1> The very main page </H1>
<H2> Главная страница </H2>
<H3> {{title}} </H3>
<u1>
{% for m in menu %}
<li> {{m}} </li>
{% endfor %}
</u1>
</body>
</html>
```

```
<!DOCTYPE html>
<head>
  <title> О сайте (about)
</title>
</head>
<body>
<H1> О сайте (about)
</H1>
</body>
</html>
```

# Лучший способ воспользоваться шаблонизатором

- # создадим файл в папке /templates/index.html
- <html>
- <head>
- <title>{{title}} - microblog</title>
- </head>
- <body>
- <h1>Hello, {{user.nickname}}!</h1>
- </body>
- </html>

# HTML5

HTML5 – стандарт языка гипертекстовой разметки. Служит для структурирования и представления материалов в сети WWW

# пример простой страницы

```
<!DOCTYPE html>      <!-- тип документа -->

<html>                <!-- начало документа -->

<head>                <!-- начало заголовка -->

  <meta charset="utf-8">

  <title>Главная страница</title>

</head>

<body>                <!-- тело документа -->

  Привет

</body>

</html>               <!-- окончание документа -->
```



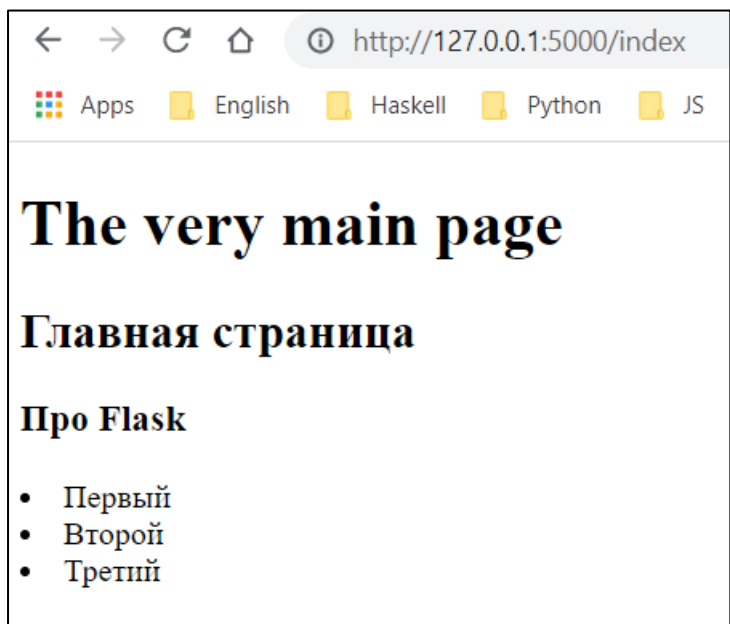
# CSS3

- CSS3 – каскадные таблицы стилей предназначены для задания элементам оформления: размеры блоков, фон, цвета, рамки, отступы, шрифты, эффекты и т.д.
- `<!DOCTYPE HTML>`
- `<html>`
- **`<head>`**
- `<meta charset="utf-8">`
- `<title>Глобальные стили</title>`
- **`<style>`** `<!-- способ 1 , в заголовке html -->`
- `H1 {`
- `font-size: 120%;`
- `font-family: Verdana, Arial, Helvetica, sans-serif;`
- `color: #333366;`
- `}`
- **`</style>`**
- **`</head>`**
- `<body>`
- `<h1>Hello, world!</h1>`
- `</body>`
- `</html>`

# Добавим функцию `render_template`

- В функцию `render_template` передаем шаблон и данные. Функция сама сопоставляет данные с метками в шаблоне и в итоге отдает сгенерированную страницу.
- `from flask import render_template`
- `from app import app`
- `@app.route('/')`
- `@app.route('/index')`
- `def index():`
  - `user = { 'nickname': 'Miguel' } # выдуманный пользователь`
  - `return render_template("index.html",`
  - `title = 'Home',`
  - `user = user)`

# Вторая программа



Измените что-нибудь в обоих шаблонах, затем нажмите «Обновить»

# Где что и как

- Создаем шаблоны страницы в html и храним их в папке **templates**
- Медия ресурсы css, img, audio, video и т.д храним в папке **static**
- Отдаем красивую страницу через шаблонизатор Jinja методом **render\_template**

# Задание

Добавьте страницу `help` с заголовком, с текстом.

Добавьте в программу вызов этой страницы.

Проверьте работоспособность программы, переключите с `index` на `help` и т.д.

# Flask, SQLAlchemy, Postgresql

```
from flask_sqlalchemy import SQLAlchemy
```

```
...
```

```
engine = create_engine("postgresql+psycopg2://postgres:1111@localhost/sqlalchemy_tuts")
```

```
...
```

```
q = session.query(Customer)
```

```
...
```

```
app = Flask(__name__)
```

```
@app.route('/index')
```

```
@app.route('/')  
def index():
```

```
    return render_template('index.html', title = 'Про Flask', menu = menu)
```

```
@app.route('/about')
```

```
def about():
```

```
    return render_template('about.html', title = 'О сайте')
```

```
@app.route('/customer')
```

```
def customer():
```

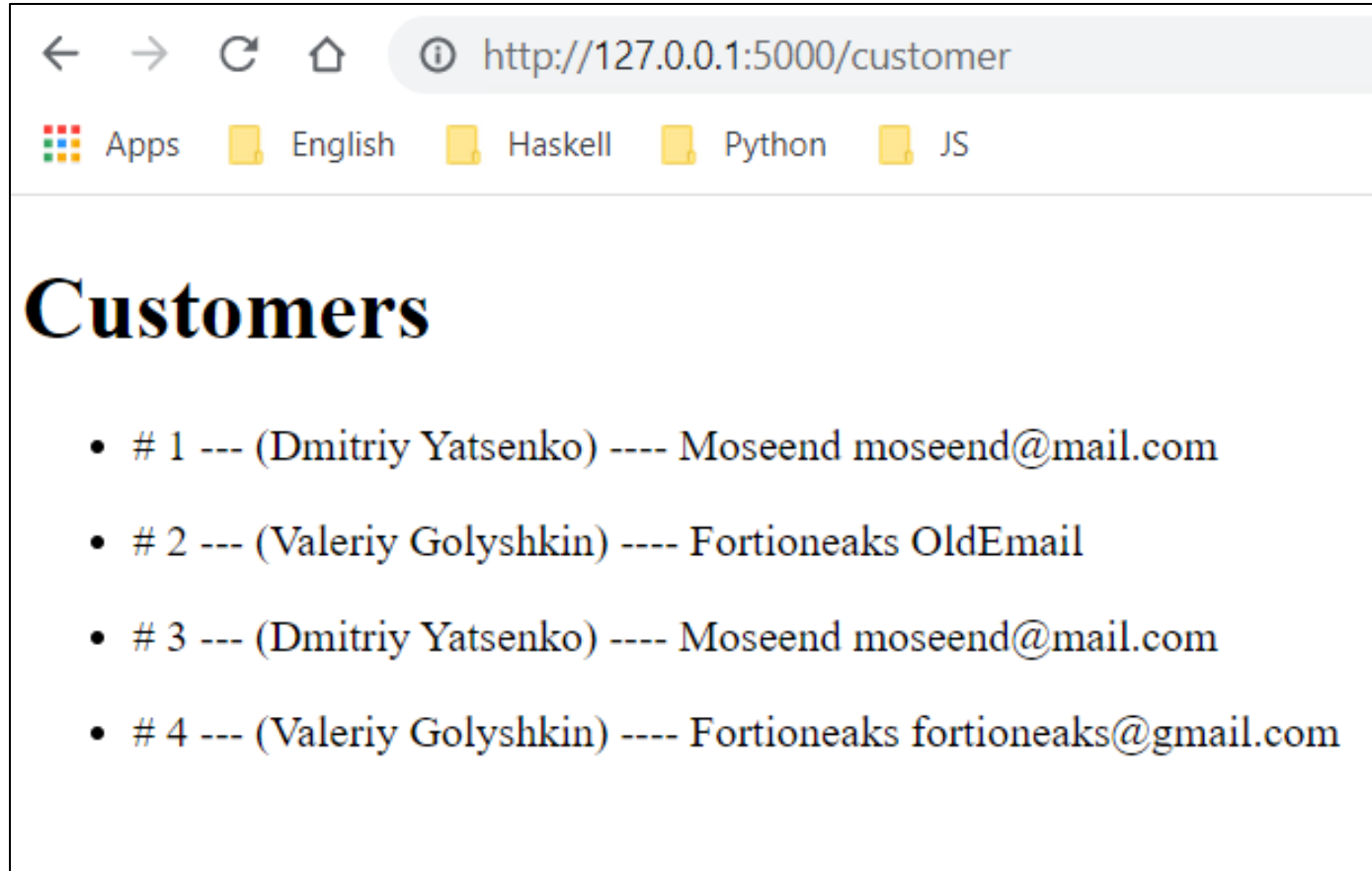
```
    return render_template('customer.html', title = 'Customers', list = q)
```

# Customer.html

```
<!DOCTYPE html>
<head>
  <title>Customers</title>
</head>
<body>
<H1> Customers </H1>
<u1>
{% for m in list %}
  <ul>
    <li> # {{m.id}} --- ({{m.first_name}} {{m.last_name}}) ---- {{m.username}} {{m.email}} </li>

  </ul>
{% endfor %}
</u1>
</body>
</html>
```

# Страница Customers



Давайте разобьем строку на две строки



# Что происходит и в какой последовательности

HTTP запрос (request) приходит на сервер (web server) разбирается под капотом URL, а затем вызывается конкретный роутер который этот запрос обрабатывает.

Далее идет процесс извлечения данных из БД с помощью модели, это может быть (SQLAlchemy) или простые SQL запросы через библиотеку psycopg2.

Как только мы получили данные, идет вызов шаблонизатора, файлы которого находятся в папке templates.

Файлы как правило имеют расширение .html

Данные файлы включают в себя код разметки, стили, ссылки на медиа ресурсы. Все что относится к медиа ресурсам хранится в папке static.

После формирования страницы (шаблон + данные из БД) идет ответ сервера (request) из роутера браузеру.

# Задание

Создайте сайт о себе любимом.

- Биографические данные
- Профессиональный опыт
- Хобби
- Прочее

# Тестирование программного кода

# Виды тестирования

1. Модульные тесты
2. Интеграционные тесты
3. Функциональные тесты
4. Сквозные тесты
5. Приемочное тестирование
6. Тестирование производительности
7. Smoke-тестирование

Ручное vs автоматическое

# Прочие виды тестирования

Анализ покрытия кода

Анализ нагрузки и производительности

Проверка на стандарты PEP-8

(Pylint, PyChecker, PyFlakes, pep8, coala)

Scrum – технология разработки, в которой продукт постоянно в рабочем состоянии – разработка и тестирование не очень большими циклами (пару недель).

# Test Driven Development



# Методология TDD

TDD — Test Driven Development. TDD — это методология разработки ПО, которая основывается на повторении коротких циклов разработки:

- изначально пишется тест, покрывающий желаемое изменение
- затем пишется программный код, который реализует желаемое поведение системы и позволит пройти написанный тест
- затем проводится рефакторинг написанного кода с постоянной проверкой прохождения тестов.

# Общие понятия

- **Модульные тесты** — это сегменты кода, которые проверяют работу других частей кода в приложении, например изолированных функций, классов и т. д. Если приложение успешно проходит все модульные тесты, то вы по меньшей мере уверены, что все низкоуровневые функции работают правильно.



# Задание

Задача.

Функция, которая проверяет, можно ли из трех чисел  $a$ ,  $b$ ,  $c$  составить треугольник. Если да, то выдать `True`, если нет, то `False`.

Напишите тесты для разработчика, выполнение которых обеспечивает правильную работу программы.

Сначала наборы чисел, которые выдают результат `True`.

Потом разные другие наборы.

# Плюсы и минусы тестирования

## Плюсы:

- тесты проверяют корректность кода;
- тесты позволяют безопасно изменять код даже в больших проектах.

## Минусы:

- написание тестов требует времени;
- очень часто получается, что в проекте становится больше тестов чем самого кода;
- работающие тесты не гарантируют корректность выполнения кода.

# Unit testing tools

Инструмент	Источник	Описание	Автор
unittest	Python standard lib	Первый unit test фреймворк, включенный в стандартную библиотеку.	<a href="#">Steve Purcell</a>
doctest	Python standard lib	Удобны для использования в терминале. Могут интегрироваться с системой erudoc.	<a href="#">Tim Peters</a>
pytest	На основе pylib	Не имеют API, автоматическая сборка тестов, простые asserts, поддержка управления через hooks, кастомизированные трейсбэки.	<a href="#">Holger Krekel</a>
nose	--	Надстройка над unittest. Интерфейс напоминает py.test, но более дружелюбный. Имеет много плагинов расширений.	<a href="#">Jason Pellerin</a>
testyfy	--	Модульная тестовая платформа с расширениями, батареи сплит – тестов для распараллеливания, поддерживает стандарты PEP8 и специальный менеджер с большим количеством параметров логгирования.	<a href="#">Yelp team</a>
subunit	--	Запуск тестов в отдельных процессах, Отслеживание результатов в единой интегрированной среде	<a href="#">Robert Collins</a>
Sancho	MEMS Exchange tls	Самостоятельно запускает тесты и сохраняет результаты тестов. Используется для систем, которые не должны немедленно реагировать на ошибки.	<a href="#">MEMS and Nanotechnology Exchange</a>

- <https://wiki.python.org/moin/PythonTestingToolsTaxonomy>

Введем программу тестирования строковых функций

```
import unittest
```

```
class TestStringMethods(unittest.TestCase):
```

```
    def test_upper(self):
```

```
        self.assertEqual('foo'.upper(), 'FOO')
```

```
    def test_isupper(self):
```

```
        self.assertTrue('FOO'.isupper())
```

```
        self.assertFalse('Foo'.isupper())
```

```
    def test_split(self):
```

```
        s = 'hello world'
```

```
        self.assertEqual(s.split(), ['hello', 'world'])
```

```
        # Проверим, что s.split не работает, если разделитель - не строка
```

```
        with self.assertRaises(TypeError):
```

```
            s.split(2)
```

```
if __name__ == '__main__':
```

```
    unittest.main()
```

# Запустим

```
Ran 3 tests in 0.008s
```

```
OK
```

# Изменим условия на неправильные и запустим

```
Ran 3 tests in 0.028s
```

```
FAILED (failures=1)
```

```
F0o != F00
```

```
Expected :F00
```

```
Actual   :F0o
```

```
<Click to see difference>
```

# Некоторые из проверок

`assertEqual(a, b)` — `a == b`

`assertNotEqual(a, b)` — `a != b`

`assertTrue(x)` — `bool(x)` is `True`

`assertFalse(x)` — `bool(x)` is `False`

`assertIs(a, b)` — `a` is `b`

`assertIsNot(a, b)` — `a` is not `b`

`assertIsNone(x)` — `x` is `None`

`assertIsNotNone(x)` — `x` is not `None`

`assertIn(a, b)` — `a` in `b`

`assertNotIn(a, b)` — `a` not in `b`

`assertIsInstance(a, b)` — `isinstance(a, b)`

`assertNotIsInstance(a, b)` — `not isinstance(a, b)`

# Задача

Предположим, что мы написали следующий код, вычисляющий площадь куба:

```
from typing import Union
```

```
def cube_area(side) :
```

```
    return 6*side**2
```

# Стали тестировать. Считает правильно!

```
print(cube_area(3))      → 54
```

```
print(cube_area(3.0))    → 54.0
```

# Но пользователь запустил программу с «кривыми» данными:

```
def cube_area(side):
```

```
    """Функция вычисляет площадь поверхности куба"""
```

```
    return 6*side**2
```

```
side_list = [10, 0, -3 , True , 'five', [1]]
```

```
mess = "Площадь поверхности куба для стороны {side} равна: {result}"
```

```
for side in side_list:
```

```
    result = cube_area(side)
```

```
    print(mess.format(side=side, result=result))
```

```
# На каком данном сломается программа?
```



# Пишем тест

```
import unittest

def cube_area(side):
    """Функция вычисляет площадь поверхности куба"""
    return 6*side**2

# создаем класс для тестирования нашей функции

class TestCubeArea(unittest.TestCase):

    # метод начинаем с префикса test_.

    def test_cube_area(self):

        self.assertEqual(cube_area(3), 54)

        self.assertEqual(cube_area(0), 0)

if __name__ == '__main__':
    unittest.main() # Запуск программы тестирования
```

# Можно запускать из разных файлов

# Флаг -m обозначает что нужно запустить файл как модуль

# Запуск всех тестов в модуле test\_cube\_area.py

\$ `python -m unittest test_cube_area.py`

# Запуск теста из класса TestCubeArea

\$ `python -m unittest test_cube_area.TestCubeArea`

# Запуск конкретного теста из класса TestCubeArea

\$ `python -m unittest test_cube_area.TestCubeArea.test_cube_area`

# Запуск всех модулей теста в текущей папки

\$ `python -m unittest`

Проверим на передачу 0 в функцию.

```
import unittest  
from cube import cube_area
```

```
class TestCubeArea(unittest.TestCase):
```

```
    def test_cube_area(self):  
        self.assertEqual(cube_area(3), 54)
```

```
    def test_value(self):  
        self.assertRaises(ValueError, cube_area, 0)
```

```
#assertRaises(exception, callable, *args, **kwargs)
```

# Получим результат:

- =====
- FAIL: test\_value (test\_cube\_area.TestCubeArea)
- -----
- Traceback (most recent call last):
- File "/home/vitaliy/www/course/students/test\_cube\_area.py", line 11, in test\_value
- self.assertRaises(ValueError, cube\_area, 0)
- AssertionError: ValueError not raised by cube\_area

# Встраиваем обработку 0 в код

```
def cube_area(side)::
```

```
    """Функция вычисляет площадь поверхности куба"""
```

```
    if side == 0: raise ValueError("Передано нулевое значение")
```

```
    return 6*side**2
```

```
# Повторный запуск теста проходит
```

Напоследок добавим явную проверку типов входящих данных

```
import unittest
```

```
from cube import cube_area
```

```
class TestCubeArea(unittest.TestCase):
```

```
    def test_cube_area(self):
```

```
        self.assertEqual(cube_area(3), 54)
```

```
    def test_value(self):
```

```
        self.assertRaises(ValueError, cube_area, 0)
```

```
    def test_types(self):
```

```
        self.assertRaises(TypeError, cube_area, True)
```

```
        self.assertRaises(TypeError, cube_area, [0])
```

```
        self.assertRaises(TypeError, cube_area, {})
```

```
        self.assertRaises(TypeError, cube_area, (1, 2))
```

```
# Какие изменения надо сделать в процедуре вычисления площади куба?
```

# Проверки данных в процедуре

```
def cube_area(side):  
    """Функция вычисляет площадь поверхности куба"""  
    if side <= 0:  
        raise ValueError  
    if type(side) == bool:  
        raise TypeError  
    if not isinstance(side, (int, float)):  
        raise TypeError  
    return 6 * side ** 2
```

# Вывод.

- Подход к тестированию заставил нас проверить функцию на все допустимые вхождения и усовершенствовать реализацию кода.
- Если в дальнейшем мы будем менять что-либо, то запуская тест мы будем контролировать неизменность работы данной функции.



# Задание

Составьте полный набор тестов для решения задачи 29-1.  
Напишите эти тесты и проверьте их.

# Пакет pylint

Установим пакет

```
pip install pylint
```

PyCharm:

File / Settings / **Plugins** / pylint / install

Давайте посмотрим на код. Что в нем не так?

```
import sys
from datetime import datetime
class CarClass:
def __init__(self, color, make, model, year, type):
    self.color = color
    self.make = make
    self.model = model
    self.year = year
    if "Linux" == sys.platform:
        print("You're using Linux!")
    self.weight = self.get_weight(type)
def get_weight(self, type):
    if type == 1:
        return 2000
    return None
```

# Проверка с помощью pylint

- Проверим с помощью pylint
- `pylint crummy_code.py`

# Результат

crummy\_code.py:15:0: **C0304**: Final newline missing (missing-final-newline)

crummy\_code.py:1:0: **C0114**: Missing module docstring (missing-module-docstring)

crummy\_code.py:3:0: **C0115**: Missing class docstring (missing-class-docstring)

crummy\_code.py:10:24: **E0602**: Undefined variable 'platform' (undefined-variable)

crummy\_code.py:12:22: **E1121**: Too many positional arguments for method call (too-many-function-args)

crummy\_code.py:14:4: **C0116**: Missing function or method docstring (missing-function-docstring)

crummy\_code.py:14:4: **C0103**: Method name "getWeight" doesn't conform to snake\_case naming style (invalid-name)

crummy\_code.py:14:4: **E0213**: Method should have "self" as first argument (no-self-argument)

crummy\_code.py:3:0: **R0903**: Too few public methods (1/2) (too-few-public-methods)

crummy\_code.py:1:0: **W0611**: Unused import sys (unused-import)

# Условные обозначения

- С – конвенция (convention)
- R – рефакторинг (refactor)
- W – предупреждение (warning)
- E – ошибка (error)

```
"""Модуль для демонстрации """
import sys
from datetime import datetime

class CarClass:
    """Класс для сущности автомобиль """
    def __init__(self, color, make, model, year, type):
        self.color = color
        self.make = make
        self.model = model
        self.year = year
        if "Linux" == sys.platform:
            print("You're using Linux!")
        self.weight = self.get_weight(type)

    def get_weight(self, type):
        """Функция возвращает вес по типу авто """
        if type == 1:
            return 2000
        return None
```

# Итог

- Линтер помогает делать код отвечающим стандартам языка и соглашениям PEP8.



# Задание

Проверьте любую свою программу на Linter. Что пишет?

# PEP-8 (Python Enhancement Proposal)

- PEP 8, иногда обозначаемый PEP8 или PEP-8, представляет собой документ, содержащий рекомендации по написанию кода на Python.
- Он был составлен в 2001 году Гвидо ван Россумом, Барри Варшавой и Ником Когланом.
- Основная цель PEP 8 – улучшить читабельность и логичность кода на Python.

# <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html>

- Внешний вид кода
  - Отступы
  - Табуляция или пробелы?
  - Максимальная длина строки
  - Пустые строки
  - Кодировка исходного файла
  - Импорты
- Пробелы в выражениях и инструкциях
  - Избегайте использования пробелов в следующих ситуациях:
  - Другие рекомендации
- Комментарии
  - Блоки комментариев
  - "Встрочные" комментарии
  - Строки документации
- Контроль версий
- Соглашения по именованию
  - Главный принцип
  - Описание: Стили имен
  - Предписания: соглашения по именованию
    - Имена, которых следует избегать
    - Имена модулей и пакетов
    - Имена классов
    - Имена исключений
    - Имена глобальных переменных
    - Имена функций
    - Аргументы функций и методов
    - Имена методов и переменных экземпляров классов
    - Константы
    - Проектирование наследования
- Общие рекомендации

# Задание

Какие тесты надо написать, чтобы проверить работу задачи 29-3?

Напишите как можно больше разных и разнообразных тестов, покрывающих как можно больше разных и не очень вариантов.

# Задача 30-1

1. Прорешать все задачи codewars. Можно выбирать по уровню, начиная с 8, можно решать по темам. Обязательно просматривать все чужие решения. Если они непонятны, то обязательно их пытаться повторить. Это самое полезное.
2. Выбрать направление в Python, которое вы считаете своим основным. Web, тестирование, анализ данных, андроид, любое.
3. Посмотреть требования к позициям в hh.ru по этому направлению. Честно оценить свой уровень и составить план совершенствования.
4. Сделать pet-project. Вылизать его и поместить в github. Пусть будет.
5. Пытаться сотрудничать с какой-нибудь командой или опытным разработчиком.
6. Решить задачу / сделать проект для своей предметной области.