

Занятие 30

Unittest, pylint

Что напечатает?

```
lst = ['hello']
```

```
lst.extend('world')
```

```
print(lst[1])
```

```
a = {'B': 5, 'A': 9, 'C': 7}
```

```
result = sorted(a)
```

```
print(result)
```

Задание 29-1

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

Задача 29-2

Напишите функцию, результатом которой является расстояние Хемминга двух строк одинаковой длины, равное количеству несовпадающих букв на одинаковых позициях.

Например:

abc и abc – 0

abc и abd – 1

abc и xyz - 3

Попробуйте написать эту функцию в одну строчку)

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

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

Установка Flask

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

или

`pip install имя модуля`

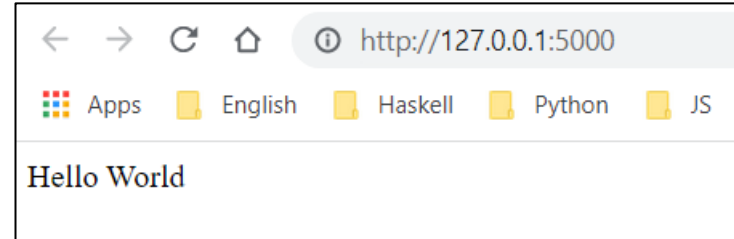
Первая программа

```
from flask import Flask
```

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

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

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



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')
```

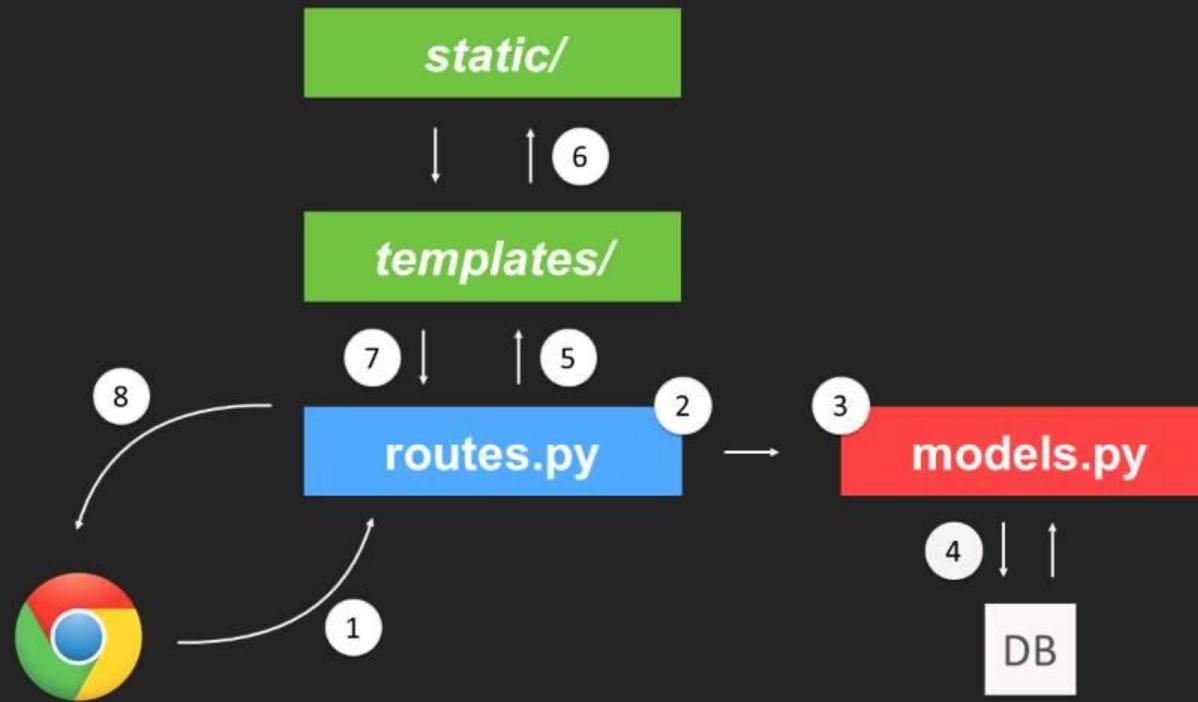
```
def about():
```

```
    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`

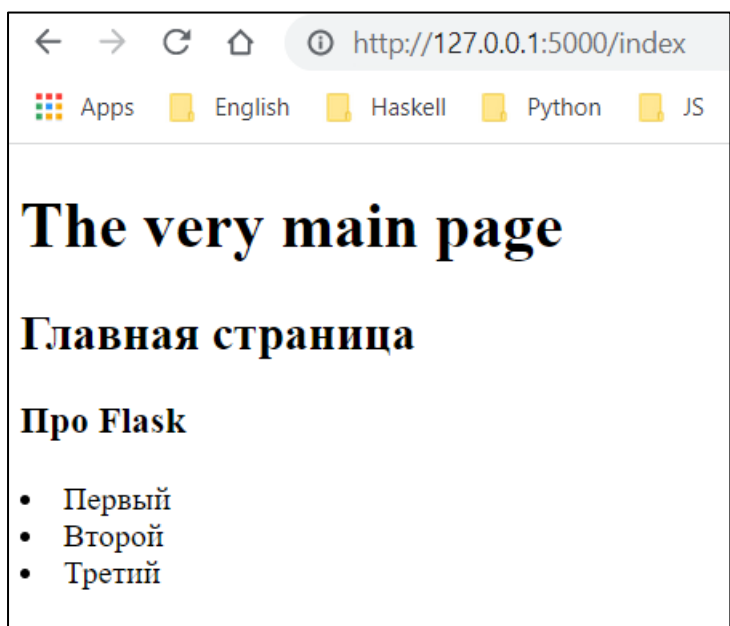
В функцию `render_template` передаем шаблон и данные. Функция сама сопоставляет данные с метками в шаблоне и в итоге отдает сгенерированную страницу.

```
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 `contacts` с заголовком, с текстом.

Добавьте в программу вызов этой страницы.

Проверьте работоспособность программы, переключите с `index` на `contacts` 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)
```

Books

```
class Book(Base):
    __tablename__ = 'book'
    book_id = Column(Integer(), primary_key=True)
    title = Column(String(50), nullable=False)
    author_id = Column(Integer(), nullable=False)
    price = Column(Integer(), nullable=False)
    amount = Column(Integer(), nullable=False)
# q = session.query(Book)
#-----
@app.route('/book')
def book():
    q = session.query(Book)
    return render_template('book.html', title = 'Books', list = q)
```

Book.html

```
<!DOCTYPE html>
```

```
<head>
```

```
  <title>Books</title>
```

```
</head>
```

```
<body>
```

```
<H1> Books </H1>
```

```
<u1>
```

```
{% for m in list %}
```

```
  <ul>
```

```
    <li> {{m.book_id}} {{m.title}} {{m.author_id}} {{m.price}} {{m.amount}} </li>
```

```
  </ul>
```

```
{% endfor %}
```

```
</u1>
```

```
</body>
```

```
</html>
```

Author

```
class Author(Base):
```

```
    __tablename__ = 'author'
```

```
    author_id = Column(Integer(), primary_key = True)
```

```
    name = Column(String(50), nullable = False)
```

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

```
def author():
```

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

```
    return render_template('author.html', title = 'Author', list = q)
```

Author.html

```
<!DOCTYPE html>
<head>
  <title>Books</title>
</head>
<body>
<H1> Authors </H1>
<u1>
{% for m in list %}
  <ul>
    <li> {{m.author_id}} {{m.name}} </li>
  </ul>
{% endfor %}
</u1>
</body>
</html>
```


Books1

```
@app.route('/book1')
```

```
def book1():
```

```
    q = session.query(Book, Author).join(Author, Book.author_id ==  
    Author.author_id).all()
```

```
    return render_template('book1.html', title = 'Book1', list = q)
```

```
#book1.html
```

```
{% for m in list %}
```

```
    <ul>
```

```
        <li> {{m.Book.book_id}} {{m.Author.name}} {{m.Book.title}} {{m.Book.price}}  
        {{m.Book.amount}} </li>
```

```
    </ul>
```

```
{% endfor %}
```

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

Уровни тестирования

1. Компонентное или Модульное тестирование (Component Testing or Unit Testing)
2. Интеграционное тестирование (Integration Testing)
3. Системное тестирование (System Testing)
4. Приемочное тестирование (Acceptance Testing)

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).

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 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 вычисляет площадь прямоугольника

```
def rect_square(x, y):  
    return x * y
```

Давайте предложим тесты, выполнение которых обеспечивает правильную работу программы.

Что надо проверить в аргументах перед их перемножением?

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

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

Случай из практики программиста

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

```
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]]
```

```
for side in side_list:
```

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

```
    print(f "Площадь поверхности куба для стороны {side} равна: {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
```

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
```

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)`

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:

"Дан список, который состоит из одинаковых чисел за исключением одного.

Найдите это число."

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, ctype):
        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(ctype)
    def get_weight(ctype):
        if ctype == 1:
            return 2000
        return None
```


Проверка с помощью pylint

Проверим с помощью pylint

```
pylint crummy_code.py
```

Результат

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.

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

Проверьте любую свою программу на Linter. Что пишет?

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. Если есть проблемы с выбором темы работы, то написать мне.
2. Если проблем нет, то написать мне, какая тема выбрана и прислать вариант презентации на посмотреть.
3. Накануне дня защиты (03 апреля) прислать мне презентацию и код.
4. 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.
5. Выбрать направление в Python, которое вы считаете своим основным. Web, тестирование, анализ данных, андроид, backend, любое другое.
6. Посмотреть требования к позициям в hh.ru по этому направлению. Честно оценить свой уровень и составить план совершенствования. Пройти тесты на hh.
7. Сделать pet-project. Вылизать его и поместить в github, ссылку на него в резюме.
8. Попытаться сотрудничать с какой-нибудь командой или опытным разработчиком.
9. Решить задачу / сделать проект для своей предметной области, использовать в своей работе.
10. Можно присылать мне резюме на "посмотреть" и задавать любые другие вопросы.