Программирование
на Python

Презентация занятия

# Рефакторинг кода.

21 занятие

инжиниум®

МГТУ им. Н.Э. Баумана

2019

# Тема: Рефакторинг кода.

## 1. КАК ПРОГРАММИСТЫ ОБЩАЮТСЯ?

### 1.1 Что такое код-ревью?

**Код-ревью (рецензирование кода, инспекция кода)** - систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными в начальной фазе разработки.

**Возможные проблемы, которые могут быть найдены:**

- состояние гонки
- утечка памяти
- переполнение буфера

# Тема: Рефакторинг кода.

## 1.2 Для чего нужна инспекция кода команде?

**Цель** - улучшить качество программного продукта и усовершенствовать навыки разработчика.

В результате код-ревью могут появиться следующие артефакты:

1. Описание способа решения задачи (design review)
2. Комментарии к стилю кода (code review)
3. Более правильный вариант (быстрый, легкочитаемый) реализации (design review, code review)
4. Указание на ошибки в коде (забытое условие в switch, и т.д.) (code review)
5. Юнит тесты (design review, code review)

*Все результаты должны быть внесены в СКВ*

# Тема: Рефакторинг кода.

## 1.3 Кто это проводит?

Код-ревью чаще всего проводят более опытные разработчики и/или члены команды разработки, занимающиеся схожей задачей.

***Одной из особенностью является то**, что во время его проведения разработчики знакомятся с кодом членов своей команды, тем самым улучшают свое представление о состоянии проекта в целом.*

## 1.4 Инструменты для инспекции кода

В основном – **СКВ**.

# Тема: Рефакторинг кода.

## 2. ЧИТАБЕЛЬНОСТЬ КОДА
## 2.1 Как писать читабельный код

- Для начала необходимо сформулировать критерии читаемости.

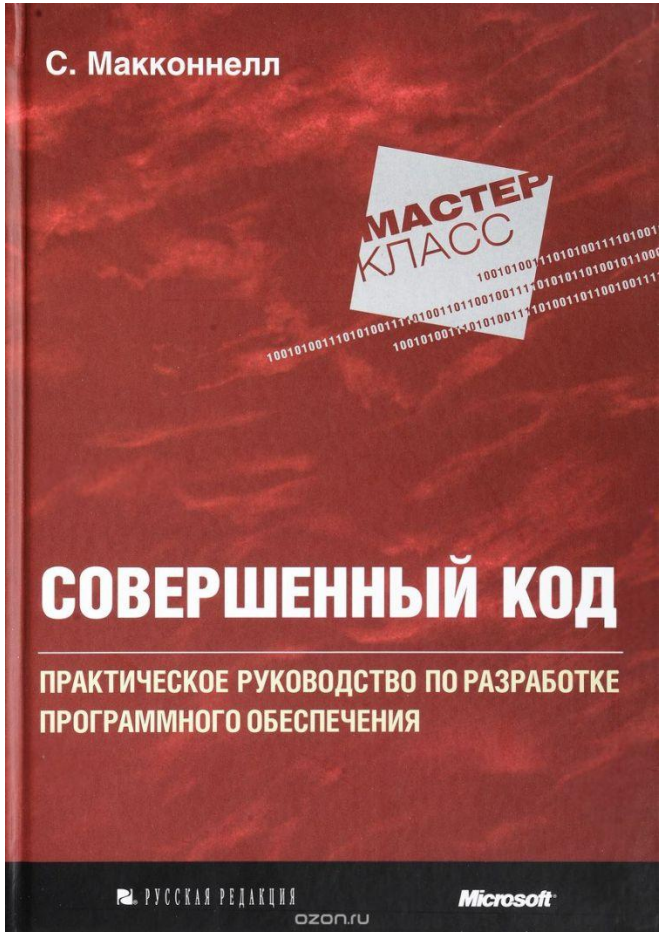
В Python критерии читаемости отражены в **стандарте PEP8.**

## 2.2 Что такое синтаксические нормы и стандарты

**Синтаксические нормы и стандарты** - свод правил, принятых разработчиками по всему миру, нацеленные на написание единого и общепонятного кода программистами.

# Тема: Рефакторинг кода.

## 2.3 Полезная литература



# Тема: Рефакторинг кода.

## 2.4 Стандарт PEP8

**PEP8** - руководство по написанию кода на Python.

PEP8 создан на основе рекомендаций Гуидо ван Россума с добавлениями от Барри (создатель Python).

Ключевая идея Гуидо:
***«Код читается намного больше раз, чем пишется.»***

**Две причины для того, чтобы нарушить данные правила:**

1. Когда применение правила сделает код менее читаемым даже для того, кто привык читать код, который следует правилам.

1. Чтобы писать в едином стиле с кодом, который уже есть в проекте и который нарушает правила (возможно, в силу исторических причин) — впрочем, это возможность переписать чужой код.

# Тема: Рефакторинг кода.

## 3. Основные правила PEP8
## 3.1 Отступы

Рекомендуется использовать 4 пробела на каждый уровень отступа. Python 3 запрещает смешивание табуляции и пробелов в отступах.

**Хорошо**

```
def no_tab_using():
    no_tab = 'Using 4 spaces'
```

**Плохо**

```
def use_tab():
→   one_tab_using = 'Ugly'
```

# Тема: Рефакторинг кода.

## 3.1 Отступы

Если функция слишком длинная
**Максимальная длина строки – 79 символов**

Правильно:

```
# Выровнено по открывающему разделителю
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

Неправильно:

```
# Аргументы на первой линии запрещены, если не используется вертикальное
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

# Тема: Рефакторинг кода.

## 3.1 Отступы

Закрывающие круглые/квадратные/фигурные скобки в многострочных конструкциях могут находиться под первым непробельным символом последней строки списка, например:

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )
```

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

# Тема: Рефакторинг кода.

## 3.2 Импорты

Каждый импорт, как правило, должен быть на отдельной строке.

Правильно:

```
import os
import sys
```

Неправильно:

```
import sys, os
```

В то же время, можно писать так:

```
from subprocess import Popen, PIPE
```

*Импорты всегда помещаются в начале файла, сразу после комментариев к модулю и строк документации, и перед объявлением констант.*

# Тема: Рефакторинг кода.

## 3.3 Соглашения по именованию

*Соглашения по именованию переменных в python немного туманны, поэтому их список никогда не будет полным.*

**Предпочтительнее писать в едином стиле.**

Некоторые стили:

1. lowercase (слово в нижнем регистре)
2. lower_case_with_underscores (слова из маленьких букв с подчеркиваниями)
3. UPPERCASE (заглавные буквы)
4. UPPERCASE_WITH_UNDERSCORES (слова из заглавных букв с подчеркиваниями)
5. CapitalizedWords (слова с заглавными буквами. **CapWords/CamelCase**).

# Тема: Рефакторинг кода.

## 3.3 Соглашения по именованию

**Имена модулей и пакетов**
import sys
import this_my_modul

**Имена функций**
def my_new_function():

**Имена классов**
class MyClass:

**Имена методов и переменных экземпляров классов**
Используйте тот же стиль, что и для имен функций

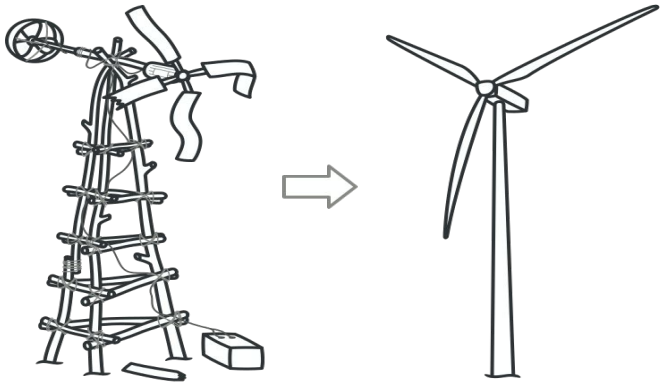
# Тема: Рефакторинг кода.

## 4. РЕФАКТОРИНГ
## 4.1 Что это?

**Рефакторинг -** процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения.

**Цель** — сделать код программы более легким для понимания; без этого рефакторинг нельзя считать успешным.

- В основе рефакторинга лежит последовательность небольших преобразований, сохраняющих поведение.
- Может продолжаться до бесконечности.



# Тема: Рефакторинг кода.

## 4.2 Причины применения

- Необходимо исправить ошибку, причины возникновения которой сразу не ясны
- Сложная логика программы

**Рефакторинг нужно применять постоянно при разработке кода!**

# Тема: Рефакторинг кода.

## 4.3 Признаки плохого кода

- дублирование кода;
- длинный метод;
- большой класс;
- длинный список параметров;
- «жадные» функции — это метод, который чрезмерно обращается к данным другого объекта;
- избыточные временные переменные;
- Не сгруппированные данные.

# Тема: Рефакторинг кода.

## 4.4 Правильный порядок

Внутри единого блока (скрипта) всегда должен быть соблюден единый порядок:

1. **наверху** - блок с import'ами и подключением всех необходимых библиотек и файлов.
2. **следом** - все необходимые классы
3. **затем** - явное объявление всех необходимых функций
4. **в конце** - блок использования кода

# Тема: Рефакторинг кода.

## 5. pylint

Для того, чтобы иметь инструмент автоформатирования и автопроверок на соответствие PEP8 в Python существует 2 утилиты, дополняющие друг друга:

1. pylint
2. autopep

### 5.1 Установка pylint

Установим и активируем виртуальное окружение:

```
virtualenv PRG1
PRG1\Scripts\activate.bat
```

```
pip install pylint
```

# Тема: Рефакторинг кода.

## 5.2 Запуск

**pylint file.py**

Попробуйте найти ошибки в коде не запуская код.

```
import sys, math


class CarClass:
    """"""

    def __init__(self, color, make, model, year):
        """Constructor"""
        self.color = color
        self.make = make
        self.model = model
        self.year = year

        if "Windows" in platform.platform():
            print("You're using Windows!")

        self.weight = self.getWeight(1, 2, 3)

    def getWeight(this):
        """"""
        return "2000 lbs"
```

# Тема: Рефакторинг кода.

## 5.3 Обозначения pylint

### Обозначение букв:

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

Попробуем исправить наши ошибки:

1. Сначала исправим **getWeight** на **get_weight**, так как *camelCase* не используется в названиях методов
2. В **get_weight** поместим первым аргументом **self**
3. Уберем неиспользуемые модули
4. Подключим модуль platform
5. Что можно сделать еще?

# Тема: Рефакторинг кода.

## Задания

### 1. Исправьте код, чтобы проходили тесты pylint

```
import os
import notexistmodule

def Function(num,num_two):
return num

class MyClass:
"""class MyClass ""«

def __init__(self,var):
self.var=var

def out(var):
print(var)

if __name__ == "__main__":
my_class = MyClass("var")
my_class.out("var")
notexistmodule.func(5)
```

# Тема: Рефакторинг кода.

**2. Найти количество четных числе в массиве**
Входные данные: nums = [12,345,2,6,7896]
Выходные данные: 4

**3. Вывести сумму всех разрядов числа**
Входные данные: n = 234
Выходные данные: 9

Объяснение:
2 + 3 + 4 = 9

**Все задания должны проходить тесты pylint**

# Тема: Рефакторинг кода.

## Рефлексия

1. В чем отличие код-ревью от рефакторинга?
2. Какой порядок в коде необходимо соблюдать?
3. Цель код-ревью?
4. Цель рефакторинга?
5. Что еще мы сегодня узнали и чему научились?