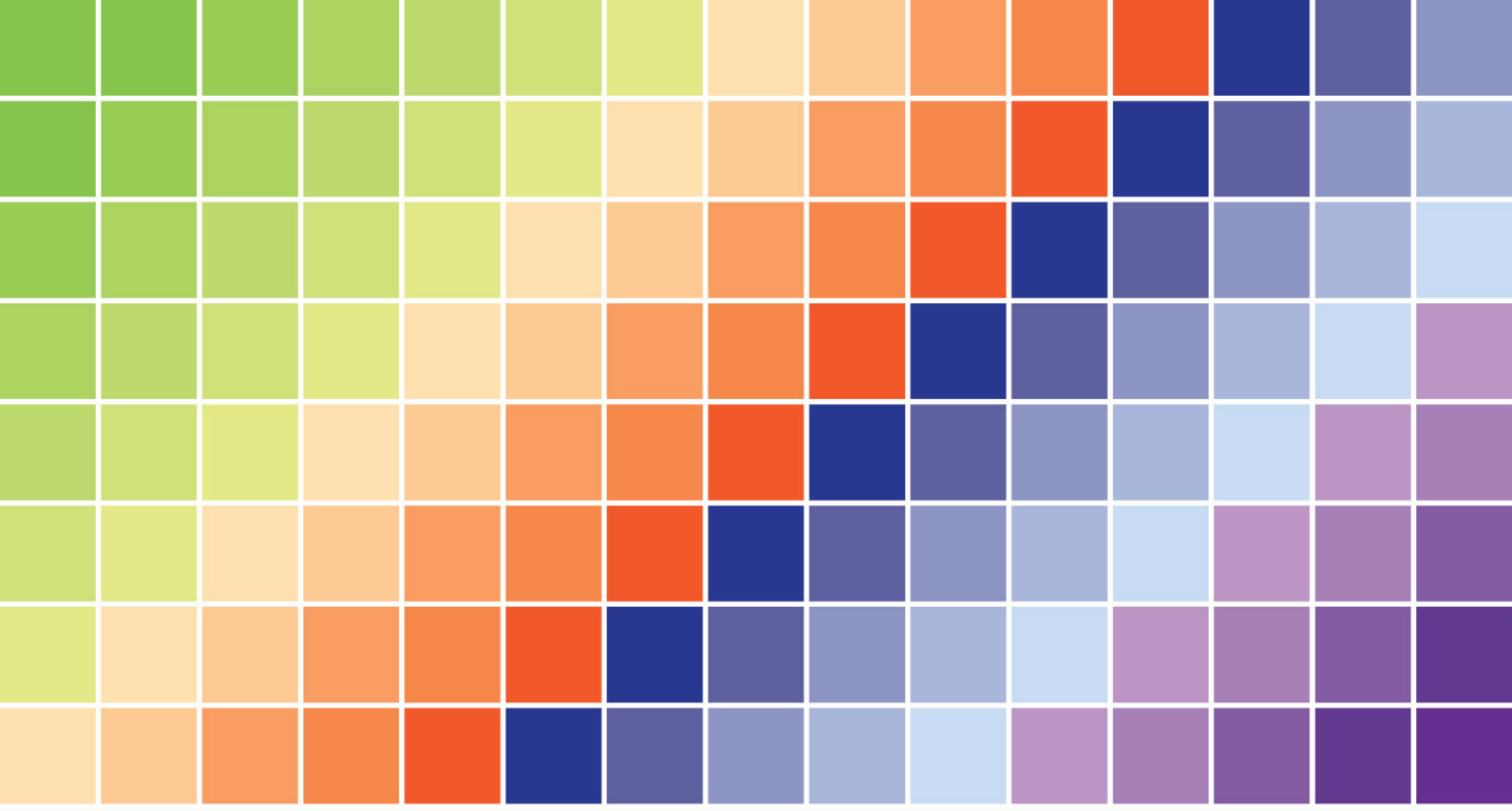

School of Computer Science

СТАНДАРТЫ ОФОРМЛЕНИЯ КОДА

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы

СТАНДАРТЫ ОФОРМЛЕНИЯ КОДА
ПРИЧИНЫ ПОЯВЛЕНИЯ

- 80% времени для жизненного цикла кода затрачивается на его сопровождение
- Практически отсутствует программный код, который сопровождается только его автором
- Стандарты улучшают читаемость кода, позволяя другим инженерам быстрее и лучше понимать его
- Code Review – обычная процедура в современных командах разработки
- Если исходный код поставляется в составе продукта (open source или коммерческого), мы должны быть уверены, что он хорошо оформлен и понятен

СТАНДАРТ ОФОРМЛЕНИЯ КОДА В PYTHON
PEP 8 И GOOGLE STYLE GUIDE

- PEP 8 – Python Enhancement Proposal #8
  - Оригинальное описание: www.python.org/dev/peps/pep-0008
  - Русскоязычные материалы:

    Перевод: defpython.ru/pep8

    Трактовка:
    - geekbrains.ru/posts/pep8
    - netology-university.bitbucket.io/codestyle/python
    - www.calculate-linux.ru/main/ru/python_style_guide
- Google Style Guide:
  - google.github.io/styleguide/pyguide.html
  - habr.com/post/179271 и habr.com/post/180509

СТАНДАРТЫ ОФОРМЛЕНИЯ КОДА
ОСОБЕННОСТИ

- Одно из положений дзен-философии Python гласит: «*Должен существовать один – и, желательно, только один – очевидный способ сделать это*»
- В рекомендациях документа PEP 8 делается попытка изложить такой стиль написания кода
- Подход Google описан подробно в книге Бретта Слаткина «Секреты Python»

ПРОБЕЛЬНЫЕ СИМВОЛЫ
ПРОБЕЛЫ В PEP 8

- Используйте пробелы, а не символы табуляции для создания отступов
- Используйте по 4 пробела для каждого уровня синтаксически значимых отступов
- Помещайте по одному и только одному пробелу до и после оператора присваивания и других двуместных операторов
- Не окружайте пробелами:
  - Вызовы функций
  - Индексы элементов списков
  - Операторы присваивания значения именованным аргументам функций

ДЛИНЫ И ПЕРЕВОДЫ СТРОК
УПРАВЛЕНИЕ СТРОКАМИ В PEP 8

- Длина строк не должна превышать 79 символов
- Дополнительные строки, являющиеся продолжением длинных выражений, должны выделяться четырьмя дополнительными пробелами сверх обычного их количества для отступов данного уровня
- Между определениями функций и классов в файлах следует вставлять две пустые строки
- Между определениями методов в классах следует вставлять одну пустую строку

ИМЕНОВАНИЕ ИДЕНТИФИКАТОРОВ
ИМЕНА В PEP 8

- PEP 8 предлагает свой стиль имен для различных элементов языка Python
- Это нужно, чтобы в процессе чтения кода легко определять какому типу соответствует то или иное имя
- Базовый принцип именования идентификаторов объектов описан в регуляции PEP 3131:
  - *все идентификаторы обязаны содержать только ASCII символы, и означать английские слова везде, где это возможно*

ИМЕНА ПЕРЕМЕННЫХ. ТРЕБОВАНИЯ
СИНТАКСИС ИМЕН PYTHON

- Используются для идентификации ссылок на объекты в Python
- Состоят из букв, цифр и знака подчеркивания
- Не могут начинаться с цифры
- Регистр в Python ИМЕЕТ значение:
  - Переменные accountBalance и AccountBalance – РАЗНЫЕ

ПЛОХИЕ ИМЕНА
ИМЕНА В PEP 8

- Имена, которых следует избегать:
  - Односимвольные имена, исключая счетчики
  - Никогда не используйте символы l (маленькая латинская буква «эль»), O,o (латинская буква «о») или I (заглавная латинская буква «ай»)
  - Не используйте дефисы в именах модулей и пакетов, используйте знак подчеркивания
  - Двойные подчеркивания в начале и конце имен зарезервированы для языка, не используйте их

РЕКОМЕНДОВАННЫЕ ИМЕНА
ИМЕНА В PEP 8

- Имена функций, их параметров, переменных и атрибутов должны следовать формату snake_case
- Имена внутренних (protected) атрибутов экземпляра должны соответствовать формату _leading_underscore
- Имена закрытых (private) атрибутов экземпляра должны соответствовать формату __leading_underscore

РЕКОМЕНДОВАННЫЕ ИМЕНА
ИМЕНА В PEP 8

- Имена классов и исключений должны следовать формату CapitalizedWord
- Константы на уровне модуля и внутри классов должны записываться в формате ALL_CAPS
- В определениях методов экземпляров классов в качестве имени первого параметра следует всегда использовать self

СВОДКА ПО ИМЕНАМ
РЕКОМЕНДАЦИИ ГВИДО ВАН РОССУМА

Тип объекта	Внешний формат имени	Внутренний формат имени
Модуль	lower_with_under	_lower_with_under
Класс	CapsWords	_CapsWords
Исключение	CapsWords	
Функция	lower_with_under()	_lower_with_under()
Глобальная/Внутриклас-совая константа	CAPS_WITH_UNDER	_CAPS_WITH_UNDER
Глобальная/Внутриклас-совая константа	lower_with_under	_lower_with_under
Переменная экземпляра класса	lower_with_under	_lower_with_under или __lower_with_under (private)
Имя метода класса	lower_with_under()	_lower_with_under() or __lower_with_under() (private)
Параметр функции/метода	lower_with_under	
Локальная переменная	lower_with_under	

ВЫРАЖЕНИЯ С NONE
PEP 8

- Сравнения с None должны обязательно выполняться с использованием операторов is или is not, а не с помощью операторов ==, !=
- Не пишите if x, если имеете в виду if x is not None – если , к примеру, при тестировании такая переменная примет значение не-None, то при приведении к булевскому типу может получиться False
- Используйте оператор is not None вместо not …is None – эти подходы идентичны, но первый лучше читается:
  - YES: if foo is not None: …
  - NO: if not foo is None: …

ПРОВЕРКА ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ
PEP 8

- Используйте встроенные отрицания (if a is not b), а не отрицания утвердительных выражений (if not a is b)
- Не тестируйте пустые значения (такие, как [] или ''), проверяя их длину (if len(somelist) == 0). Вместо этого используйте проверку if not somelist, исходя из того, что пустое значение трактуется как False
- То же самое касается и непустых значений (таких как [1] или 'hi'): в инструкции if somelist результат вычисления непустого значения трактуется как True

ПРОЧИЕ ПРАВИЛА
PEP 8

- Избегайте записи инструкций if, циклов for и while, а также сложных инструкций except в одной строке
- Всегда помещайте инструкции import в самом начале файла
- Импортируемые модули должны располагаться в разделах, указываемых в таком порядке:
  - Модули стандартных библиотек
  - Модули сторонних разработчиков
  - Ваши собственные модули

  В каждом разделе модули следует располагать в алфавитном порядке

СВОЙСТВА ИСКЛЮЧЕНИЙ
GOOGLE STYLE GUIDE

- Exception является средством выхода из нормального потока управления для обработки ошибок или других исключительных ситуаций
- Плюсы использования исключений:
  - Поток управления обычного кода не перемешивается с кодом перехвата ошибок
  - Возможность пропускать N-ое количество блоков кода при выполнении условия, например, возврат из N-вложенных функций за один шаг вместо множественной обработки кодов возврата по всем N функциям
- Минусы исключений:
  - Можно потерять контроль над потоком исполнения, упустив некоторые исключения
  - Например, можно пропустить ситуацию, в которой будет возбуждена ошибка при вызове функции из библиотеки

ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ
GOOGLE STYLE GUIDE

- Глобальные переменные – это переменные, которые определены на уровне модуля
- Не используйте глобальные переменные в пользу переменных класса
- Несколько исключений из этого правила:
  - Стандартные настройки скриптов
  - Константы уровня модуля. Например, PI = 3.14159. Константы должны быть именованы с использованием только заглавных букв и символа подчеркивания по правилам PEP 8
  - Иногда полезно использовать глобальные переменные, чтобы кэшировать значения, необходимые для функции или возвращаемые функцией
  - При необходимости, глобальные переменные должны быть созданы внутри модуля и доступны через общедоступные фукнции уровня модуля

ЛОКАЛЬНЫЕ КЛАССЫ И ФУНКЦИИ
GOOGLE STYLE GUIDE

- Класс может быть определен внутри метода, функции или другого класса
- Функция может быть определена внутри метода или другой функции
- Вложенные функции имеют доступ только на чтение к переменным, определенным в родительской области
- Это делает возможным определение вспомогательных классов и функций, которые будут использованы только внутри очень ограниченного пространства
- Такие сущности рекомендуются к использованию

ГЕНЕРАТОРЫ ВМЕСТО MAP(), FILTER()
GOOGLE STYLE GUIDE

- Генераторы лучше, чем код, в котором используются функции map() и filter()
- Для генераторов не нужны дополнительные лямбда-выражения
- Генераторы списков позволяют легко игнорировать ненужные элементы входного списка, в то время как в случае использования функции map() для этого приходится задействовать еще и функцию filter()
- См. примеры в list_gen_shell.txt

ЛЯМБДА-ФУНКЦИИ
GOOGLE STYLE GUIDE

- Определяют анонимные функции в выражении; альтернатива обычной функции
- Плюсы: удобно
- Минусы:
  - Труднее читать и по сравнению с локальными функциями
  - Труднее отлаживать из-за отсутствия имени функции в стеке вызовов
  - Ограниченные возможности – лямбда может содержать только одно выражение
- Хорошо подходят для инлайновых выражений
- Если код внутри лямбда-функции длиннее чем 60-80 символов, то, возможно, лучше определить данную функцию как обычную

АРГУМЕНТЫ ФУНКЦИИ ПО УМОЛЧАНИЮ
GOOGLE STYLE GUIDE

- Значения для параметров функции в конце списка, например def foo(a, b=0)
- Плюсы: частое использование функции с коротким списком аргументов, имитация перегрузки функций
- Минусы:
  - Значения аргументов вычисляются один раз во время загрузки модуля
  - Для изменяемых объектов в качестве значений по умолчанию это может быть проблемой
- Решение:

Хорошо:

```
def foo(a, b=None):
    if b is None:
        b = []
```

Плохо:

```
def foo(a, b=[]):
```

ИСПОЛЬЗОВАНИЕ КРУГЛЫХ СКОБОК
GOOGLE STYLE GUIDE

- Используйте скобки экономно, например, не нужно использовать скобки с return, если значение помещается на одной строке
- Скобки хорошо использовать для явного обозначения кортежей

Хорошо:

```
if foo:
    bar()

while x:
    x = bar()

if x and y:
    bar()

if x and y:
    bar()
return foo

for (x, y) in dict.items:
```

Плохо:

```
if (x):
    bar()
if not(x):
    bar()
return (foo)
```

СТРОКИ ДОКУМЕНТИРОВАНИЯ
GOOGLE STYLE GUIDE

- Снабжайте строками документирования каждую функцию, класс и модуль
- Функция должна иметь строку документации во всех случаях, кроме описанных ниже:
  - Не видима снаружи модуля
  - Очень короткая
  - Очевидная (легко читаемая)
- Строка документирования должна давать достаточно информации, чтобы оформить вызов функции без чтения ее исходного кода

ТИПОВОЙ СОСТАВ DOCSTRING ДЛЯ ФУНКЦИИ

1. Типы параметров и возвращаемых значений
2. Описание того, что делает функция
3. Условия ее использования
4. Возбуждаемые исключения (если есть)
5. Примеры вызовов в стиле Python Shell

- Смотрите примеры в triangle.py
- Для пояснений хитрого алгоритма комментарии внутри исходного кода более предпочтительны, чем строки документации
- Рекомендованный стандарт для строк документирования PEP 0257

DOCSTRING ДЛЯ КЛАССОВ
GOOGLE STYLE GUIDE

- Классы должны иметь строку документации ниже своего объявления
- Первая строка – описание назначения класса в одно предложение
- Публичные атрибуты класса должны быть документированы следом
- Каждый общедоступный метод должен снабжаться строками документирования
- См. пример в google_class_doc.py

DOCSTRING ДЛЯ МОДУЛЕЙ
GOOGLE STYLE GUIDE

- Каждый модуль должен снабжаться строками документирования верхнего уровня
- Этот строковый литерал должен быть первой инструкцией в файле модуля
- Docstring содержит ознакомительные сведения о модуле и его содержимом:
  - 1-я строка – одно предложение, описывающее назначение модуля
  - Далее – описание работы модуля, рассчитанное на его пользователей
- См. пример в google_unit_doc.py

ПОДГОТОВКА HTML-ДОКУМЕНТАЦИИ TRIANGLE.HTML С ПОМОЩЬЮ PYDOC

```
>>> import pydoc
>>> import os
>>>
>>> os.getcwd()
'C:\\Лекции в IT-школе\\Lesson 25\\Scripts'
>>> os.chdir(r"C:\Лекции в IT-школе\Lesson 25\Scripts")
>>> os.getcwd()
'C:\\Лекции в IT-школе\\Lesson 25\\Scripts'
>>>
>>> import triangle
>>> pydoc.writedoc(triangle)
wrote triangle.html
>>>
```

ИСПОЛЬЗОВАНИЕ ФУНКЦИИ MAIN()
GOOGLE STYLE GUIDE

- Определяйте функцию main() для старта вашей программы
- Запускайте данную функцию только в том случае, если ваш модуль исполняется, а не импортируется
- Проверяйте if __name__ == '__main__' перед исполнением функции main():
  - Это означает, что ваш модуль не будет полностью исполнен при импортировании его в другую программу
  - Используйте этот подход даже если ваш скрипт не создавался для того, чтобы быть импортированным

КОММЕНТАРИИ
РЕКОМЕНДАЦИИ ГВИДО ВАН РОССУМА

- Комментарии, противоречащие коду, хуже, чем отсутствие комментариев. Всегда исправляйте комментарии, если меняете код!
- Комментарии должны быть законченными предложениями
- Если комментарий – фраза или предложение, первое слово должно быть написано с большой буквы
- Используйте английский язык для написания комментариев
- Используйте два пробела после точки в конце предложения

БЛОКИ КОММЕНТАРИЕВ
РЕКОМЕНДАЦИИ ГВИДО ВАН РОССУМА

- Блок комментариев обычно объясняет код (весь или только некоторую часть), идущий после блока, и должен иметь тот же отступ, что и сам код
- Каждая строчка такого блока должна начинаться с символа # и одного пробела после него
- Абзацы внутри блока комментариев разделяются строкой, состоящей из одного символа #
- Лучше всего отделять блоковые комментарии пустыми строками сверху и снизу

ВНУТРИСТРОКОВЫЕ КОММЕНТАРИИ
РЕКОМЕНДАЦИИ ГВИДО ВАН РОССУМА

- Находятся на той же строке, что и выполняемая инструкция
- Они должны начинаться с # и одного пробела
- Внутристроковые комментарии надо отделять как минимум двумя пробелами от предшествующего оператора
- Некоторые комментарии излишни и отвлекают, если их значение и так очевидно:
  `x = x + 1  # Увеличение x на единицу`
- Но иногда могут быть полезными:
  `x = x + 1  # Для компенсации толщины рамки`

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ ?

School of Computer Science