

# Python Базовый

PEP8 стандарты оформления кода


# Python Базовый

## Introduction



**Бондаренко Кирилл**

Senior Data scientist, CreatorIQ

 [profile.php?id=100011447245832](https://www.facebook.com/profile.php?id=100011447245832)

 [kirill-bond/](https://www.linkedin.com/in/kirill-bond/)

 [@bond.kirill.alexandrovich](https://www.telegram.com/@bond.kirill.alexandrovich)



# Python Базовый

## Тема урока

### PEP8 стандарты оформления кода

# Python Базовый

## План урока

1. Что такое PEP8
2. Что такое дзен Python. Разбор правил и практические примеры
3. Заключение по курсу

# Python Базовый

## Что такое PEP8

**PEP** - Python Enhancement Proposals или предложения по улучшению Python.

Существует множество PEP, их можно найти тут - <https://www.python.org/dev/peps/>

Конкретный **PEP8** - это набор правил для улучшения визуальной составляющей кода. Его можно изучить тут - <https://www.python.org/dev/peps/pep-0008/>



# Python Базовый

## Что такое Python Zen

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python Zen = PEP20 (<https://www.python.org/dev/peps/pep-0020/>)

# Python Базовый

## Beautiful is better than ugly

Дословно - "красивое лучше уродливого". Данное правило является собирательным, то есть включает в себя множество других понятий, которые изложены в PEP 8.

Главная мысль - "лучше писать красивый и понятный код по принятым стандартам, чем сложно читаемый и непонятный".

### Violation example:

```
def LongCamelCaseFuncNameDoSomeStuff(arg1, arg2, arg3, arg4, arg5):  
    if (arg1 | arg2):  
        return 1;  
    elif (arg4 > arg5):  
        return 2;  
    else:  
        return arg3;  
    localVar = map(lambda x,y,z,a,b,c: x+y+z+a+b+c, [1], [2], [3],  
[4], [5], [6]);  
    return localVar
```

### Fulfilment example:

```
def define_time(length, velocity):  
    return length / velocity if velocity else 0  
  
def calculate_time_for_batch(lengths_array, velocities_array):  
    time_array = map(  
        define_time,  
        lengths_array,  
        velocities_array  
    )  
    return time_array if sum(time_array) else []
```

# Python Базовый

## Explicit is better than implicit

Дословно - "явное лучше неявного".

Главная мысль данного правила такая:  
"если хотите запрограммировать какой-то алгоритм, сделайте это несколькими простыми и понятными действиями, чем одной кучей кода с кучей непонятных имен переменных и функций".

Violation example:

```
def main_function_of_program(argument_first, argument_second):
    result_first = argument_first + argument_second
    results_in_general = []
    if result_first > 0:
        results_in_general.append(result_first)
    else:
        results_in_general.append(0)
    another_action = argument_first * argument_second
    if another_action > results_in_general[0]:
        results_in_general.append(another_action)
    else:
        results_in_general.append(0)
    sum_of_results = 0
    for value in results_in_general:
        sum_of_results += value
    return sum_of_results
```

Fulfilment example:

```
def get_sum_of_values_in_two_arrays(array_a, array_b):
    return sum(array_a) + sum(array_b)

def create_integers_array_from_string(string):
    integers_array = []
    for char in string:
        if char.isdigit():
            integers_array.append(int(char))
    return integers_array

def strings_digits_sum(string_a, string_b):
    int_array_a = create_integers_array_from_string(string_a)
    int_array_b = create_integers_array_from_string(string_b)
    sum_of_int_a_b = get_sum_of_values_two_arrays(
        int_array_a, int_array_b
    )
    print(sum_of_int_a_b)

if __name__ == "__main__":
    strings_digits_sum("hello123", "222goodbye") # 12
    strings_digits_sum("abc1", "def22") # 5
    strings_digits_sum("abc", "abc") # 0
```



# Python Базовый

## Simple is better than complex

Дословно - "простое лучше сложного".

Главная мысль данного правила такая:  
"если есть возможность упростить ваш код,  
пользуйтесь ею".

Данное правило может охватить многие  
другие связанные с упрощением кода.

Violation example:

```
def define_how_many_letters_are_digits(string):  
    alphabet = "abcdefghijklmnopqrstuvwxyz"  
    digits = "0123456789"  
    count_of_digits = 0  
    for char in string:  
        char_is_letter = None  
        if char.lower() in alphabet:  
            char_is_letter = True  
        elif char in digits:  
            char_is_letter = False  
        if isinstance(char_is_letter, bool):  
            if not char_is_letter:  
                count_of_digits += 1  
    return count_of_digits
```

Fulfilment example:

Two examples both simple but first is using list comprehension and second one is the classic one.

```
# example 1 via comprehension  
def define_how_many_letters_are_digits_1(string):  
    return len([1 for char in string if char.isdigit()])  
  
# example 2 classic for loop  
def define_how_many_letters_are_digits_2(string):  
    count_of_digits = 0  
    for char in string:  
        if char.isdigit():  
            count_of_digits += 1  
    return count_of_digits
```

# Python Базовый

## Complex is better than complicated

Дословно - "сложное лучше чем запутанное".

Главная мысль данного правила такая: "если задача с которой вы работаете сама по себе сложная, то не стоит еще и запутывать ее решение".

### Complicated (violation example):

```
def dot_product(list_a, list_b):  
    list_of_products = []  
    for a, b in zip(list_a, list_b):  
        list_of_products.append(a * b)  
    sum_of_list = 0  
    for product in list_of_products:  
        sum_of_list += product  
    return sum_of_list
```

### Complex (fulfilment example):

```
def dot_product(list_a, list_b):  
    return sum(map(lambda a, b: a*b, list_a, list_b))
```

# Python Базовый

## Flat is better than nested

Дословно - "плоское лучше вложенного".

Главная мысль данного правила такая: "используйте ветвления разумно, не делайте их по 10 уровней в глубину, пишите сразу правильные условия".

**Violation example (nested):**

```
a = 10
b = 5
if a > b:
    if len(c) > a:
        print(1)
else:
    print(2)
```

**Fulfilment example (flat/chained):**

```
a = 10
b = 5
if b < a < len(c):
    print(1)
else:
    print(2)
```

# Python Базовый

## Sparse is better than dense

Дословно - “разреженное лучше плотного”.

Главная мысль данного правила такая: “если ваш код выполняет множество разных задач, то разделите их на отдельные методы с минимальной ответственностью, не засовывайте весь код в 1 функцию”.

Здесь хорошо подходит SRP - single responsibility principle (принцип единой ответственности).

Violation example (two examples of dense code):

```
# first example is bad
def print_sum_of_positive_int_numbers_in_mixed_list(array):
    numbers = [value for value in array if isinstance(value, int)]
    positive = [value for value in numbers if value > 0]
    sum_of_values = sum(positive)
    print("Sum of positive numbers is %s . Thank you." %
          sum_of_values)

# but this one even worse
def print_sum_of_positive_int_numbers_in_mixed_list(array):
    print("Sum of positive numbers is %s . Thank you." %
          sum([value for value in
               [value for value in array if isinstance(value, int)]
               if value > 0]))
```

Fulfilment example (sparse code for the same problem)

```
def get_integers_from_list(array):
    return [value for value in array if isinstance(value, int)]

def get_greater_than_zero_integers_from_list(array):
    return [value for value in array if value > 0]

def get_sum_of_array_values(array):
    return sum(array)

def print_information(value):
    print("Sum of positive numbers is %s . Thank you." % value)

def print_sum_of_positive_int_numbers_in_mixed_list(array):
    numbers = get_integers_from_list(array)
    positive = get_greater_than_zero_integers_from_list(numbers)
    sum_of_values = get_sum_of_array_values(positive)
    print_information(sum_of_values)
```

# Python Базовый

## Readability counts

Дословно - "читаемость (кода) имеет значение".

Главная мысль данного правила такая: "пишите легко читаемый код и это вам вернется приятным бонусом в виде ясного понимания кода после того как вы не будете им заниматься длительное время".

### Violation example:

```
VarA = 10
elemedf_fd_dict_A = { "a": "hello", "v": 2
,"c": 1}
elemedf_fd_dict_A[VarA] = \
"elemet!"
```

### Fulfilment example:

```
char, index = "C", 3 # using tuples

char_indices_dict = {
    "A": 1,
    "B": 2,
    "C": 3,
    char: index
} # simple meaning for dict is char:char_index without any comments
```

# Python Базовый

## Special cases aren't special enough to break the rules

Дословно - "особые случаи не настолько особые, чтобы нарушать правила".

```
1 def calculate_data(data):
2     result = {k:k**2 for k in [a for a in [value for value in data.get('values') if value<0]if a > 4] if k is not None}
3     return result
4
5
```

Главная мысль данного правила такая: "если вам кажется, что именно ваш случай является исключением из правил, то подумайте дважды и сделайте по правилам".

```
1 def calculate_data(data):
2     values = []
3     for value in data.get('values'):
4         if value < 0:
5             values.append(value)
6     a_list = []
7     for a in values:
8         if a > 4:
9             a_list.append(a)
10    result = {}
11    for k in a_list:
12        if k is not None:
13            result[k] = k**2
14    return result
15
```

# Python Базовый

## Although practicality beats purity

Дословно - "однако практичность важнее правильности".

Главная мысль данного правила такая: "но если вы выбираете между следованием правилам и практичностью, то выберите практичность".

```
1 def calculate_data(data):  
2     result = {k:k**2 for k in [a for a in [value for value in data.get('values') if value<0]if a > 4] if k is not None}  
3     return result  
4  
5
```

Иногда такая реализация все же лучше следования правилам (в данном случае comprehensions работают быстрее обычных циклов).

Но это только в том случае, если вам очень нужна скорость и вы готовы пожертвовать читаемостью кода.

# Python Базовый

## Errors should never pass silently

Дословно - "ошибки никогда не должны заглушаться".

Главная мысль данного правила такая: "если в вашем коде ошибка, ломайте программу и выводите эту ошибку в консоль, не замалчивайте ее".

**Violation :**

```
def division_function(a, b):  
    try:  
        return a / b  
    except:  
        pass # or print("Hey bro, don't divide on 0, please")  
    finally:  
        return 0
```

**Fulfilment example:**

```
def division_function(a, b):  
    try:  
        return a / b  
    except ZeroDivisionError as error:  
        raise error
```



# Python Базовый

## Unless explicitly silenced

Дословно - "только если они не заглушены явно".

Главная мысль данного правила такая: "однако вы можете не ломать программу, но ошибку все же придется огласить".

**Fulfilment example:**

```
def division_function(a, b):
    result = 0
    try:
        result = a / b
    except ZeroDivisionError as error:
        result = error[0]
    finally:
        return result

if __name__ == "__main__":
    value_a, value_b = 5, 0
    division = division_function(value_a, value_b)
    if division == "division by zero":
        pass
        # here we can send error message to developer
    else:
        print("Division result: %s" % division)
```

# Python Базовый

In the face of ambiguity refuse the temptation to guess

Дословно - "Перед лицом двусмысленности откажитесь от соблазна угадать".

Главная мысль данного правила такая: "если вы не уверены как работает код, то разбейте его на простые компоненты и сложите пазл вместо 10 попыток угадываний".

## Violation example:

```
def huge_function(arg1, arg2): # where is a function description ?
    component_1 = function_huge_2(arg1) # you skip to debug it
    component_2 = function_huge_3(arg2) # you skip to debug it
    component_result = sqrt(component_1+component_2)
    return component_result + 1 # where is an explanation ?
```

## Fulfilment example:

```
def my_custom_formula(arg1, arg2):
    """
    Custom function to calculate
    how ... <here is explanation>
    <also raw formula>
    :param arg1: integer, 0 < arg1 < 10
    :param arg2: float, 0 < arg2 < 1
    :return: square root of sum of
    argument 1 in power of 10 and
    second argument tangent.
    """
    component_1 = function_huge_2(arg1) # pow to 10
    component_2 = function_huge_3(arg2) # tangent
    component_result = sqrt(component_1+component_2)
    return component_result + 1 # see method docstring
```

# Python Базовый

There should be one --and preferably only one--  
obvious way to do it

Дословно - "Должен быть один и только один очевидный способ сделать это".

Главная мысль данного правила такая:  
"Python предлагает вам много гибкости в решениях, но всегда сперва фокусируйтесь на проблеме, которую решаете. Для нее самое лучшее решение только одно".

```
1 def create_list(n):  
2     return list(range(n))  
3  
4  
5     n = 10  
6  
7     a = [x for x in range(n)]  
8     b = [int(x) for x in [str(i) for i in range(n)]]  
9     c = [int(j) for j in list("0123456789")]  
10
```

# Python Базовый

Although that way may not be obvious at first unless  
you're Dutch

Дословно - "Однако этот способ может быть не очевиден с первого раза только если вы не голландец".

Главная мысль данного правила такая: "практика, практика и еще раз практика, тогда вы сможете видеть самые лучшие решения сразу".

PS: голландец - отсылка на создателя языка Python Гвидо ван Россума, он голландец.



# Python Базовый

Now is better than never

Дословно - "сейчас лучше чем никогда".

Главная мысль данного правила такая: "если вы знаете как улучшить что-то, то улучшайте!".



# Python Базовый

Although never is often better than \*right\* now

Дословно - "однако никогда часто лучше чем прямо сейчас".

Главная мысль данного правила такая:  
"если вы хотите что-то оптимизировать в 3 часа ночи, то лучше поспать и со свежей головой это пересмотреть".

Принцип YAGNI - You Aren't Gonna Need It



# Python Базовый

If the implementation is hard to explain, it's bad idea

Дословно - "если реализацию сложно объяснить, это плохая идея".

Главная мысль данного правила такая: "учитесь делать простые и эффективные решения и учитесь их объяснять так же просто".



# Python Базовый

If the implementation is easy to explain, it maybe a good idea

Дословно - "если реализацию легко объяснить, возможно, это хорошая идея".

Главная мысль данного правила такая: "даже если вам удалось что-то объяснить, то это не залог успеха,  $2+2=5$  это просто, но неправильно".





# Python Базовый

Namespaces are one honking great idea -- let's do more of those !

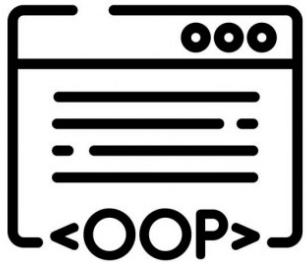
Дословно - "пространства имен это отличная идея, давайте сделаем больше их!".

Главная мысль данного правила такая: "в Python все связано одним большим глобальным пространством имен, поймите его и вам станет проще понять язык".

```
variable = 10
print(id(variable)) # 4464290896
variable = variable + 10 # variable = 20
print(id(variable)) # 4464291216
print(id(20)) # 4464291216
```

# Python Базовый

## Заключение по курсу



OOP

- Наследование
- Инкапсуляция
- Полиморфизм
- Абстракция



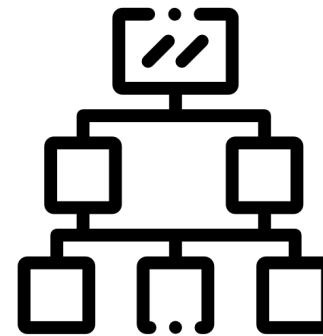
Recursion

- Примеры использования рекурсии
- Деревья поиска



Files

- Чтение и запись
- JSON, XML, CSV



Modules

- random
- math
- collections
- itertools
- re
- datetime



PEP8

- PEP
- PEP8
- Python Zen

# Информационный видеосервис для разработчиков программного обеспечения



# Проверка знаний

TestProvider.com



Проверьте как Вы усвоили данный материал на [TestProvider.com](http://TestProvider.com)

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

# Python Базовый

Спасибо за внимание! До новых встреч!



Бондаренко Кирилл  
Senior Data scientist, CreatorIQ

