Python Базовий

PEP8 стандарти оформлення коду

Introduction

Вікторія Бойчук
Python Developer, тренер CBS

Вікторія Бойчук

Тема уроку

PEP8 стандарти оформлення коду

План уроку

1. Що таке PEP8
2. Що таке дзен Python. Огляд правил та практичні приклади
3. Висновок з курсу

Що таке PEP8

PEP - Python Enhancement Proposals або пропозиції щодо покращення Python.

Існує безліч PEP, їх можна знайти тут - https://www.python.org/dev/peps/

Конкретний **PEP8** - це набір правил поліпшення візуальної складової коду. Його можна вивчити тут - https://www.python.org/dev/peps/pep-0008/

Що таке 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/)

Beautiful is better than ugly

Дослівно - "гарне краще потворного". Дане правило є збірним, тобто включає безліч інших понять, які викладені в PEP 8.

Головна думка - "краще писати красивий і зрозумілий код за прийнятими стандартами, ніж незрозумілий".

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 []
```

Explicit is better than implicit

Дослівно - “явне краще за неявне”.

Головна думка даного правила така: "якщо хочете запрограмувати якийсь алгоритм, зробіть це кількома простими та зрозумілими діями, ніж однією купою коду з купою незрозумілих імен змінних та функцій".

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
```

Simple is better than complex

Дослівно - “просте краще за складне”.

Головна думка цього правила така: "якщо є можливість спростити ваш код, користуйтеся нею".

Це правило може охопити багато інших пов'язаних зі спрощенням коду.

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
```

Complex is better than complicated

Дослівно - "складне краще, ніж заплутане".

Головна думка цього правила така: "якщо завдання з яким ви працюєте саме по собі складна, то не варто ще й заплутувати її вирішення".

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))
```

Flat is better than nested

Дослівно - "плоське краще вкладеного".

Головна думка цього правила така: "використовуйте розгалуження розумно, не робіть їх по 10 рівнів у глибину, пишіть відразу правильні умови".

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)
```

Sparse is better than dense

Дослівно - “розріджене краще за щільне”.

Головна думка цього правила така: "якщо ваш код виконує безліч різних завдань, то розділіть їх на окремі методи з мінімальною відповідальністю, не засовуйте весь код в одну функцію".

Тут добре підходить SRP – single responsibility principle (принцип єдиної відповідальності).

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)
```

Readability counts

Дослівно - "читаність (коду) має значення".

Головна думка даного правила така: "пишіть код, що легко читається і це вам повернеться приємним бонусом у вигляді ясного розуміння коду після того як ви не будете ним займатися тривалий час".

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
```

Special cases aren't special enough to break the rules

Дослівно - “особливі випадки не такі особливі, щоб порушувати правила”.

```
def calculate_data(data):
    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}
    return result


```

Головна думка цього правила така: "якщо вам здається, що саме ваш випадок є винятком із правил, то подумайте двічі і зробіть за правилами".

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

```

Although practicality beats purity

Дослівно - "проте практичність важливіша за правильність".

Головна думка цього правила така: "але якщо ви вибираєте між дотриманням правил і практичністю, то виберіть практичність".

```
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
```

Іноді така реалізація все ж таки краща за правилами (у даному випадку comprehensions спрацюють швидше звичайних циклів).

Але це тільки в тому випадку, якщо вам дуже потрібна швидкість і ви готові пожертвувати читанням коду.

Errors should never pass silently

Дослівно - "помилки ніколи не повинні заглушуватися".

Головна думка цього правила така: "якщо у вашому коді помилка, ламайте програму і виводьте цю помилку в консоль, не замовчуйте її".

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
```

Unless explicitly silenced

Дослівно - "тільки якщо вони не заглушені явно".

Головна думка цього правила така: "проте ви можете не ламати програму, але помилку все ж таки доведеться оголосити".

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)
```

In the face of ambiguity refuse the temptation to guess

Дослівно - "Перед обличчям двозначності відмовтеся від спокуси вгадати".

Головна думка цього правила така: "якщо ви не впевнені як працює код, то розбийте його на прості компоненти і складіть пазл замість 10 спроб вгадування".

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
```

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

Дослівно - "Має бути один і тільки один очевидний спосіб зробити це".

Головна думка цього правила така: “Python пропонує вам багато гнучкості у рішеннях, але завжди спершу фокусуйтеся на проблемі, яку вирішуєте. Для неї найкраще рішення лише одне”.

```
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
```

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

Дослівно - "Проте цей спосіб може бути не очевидний з першого разу тільки якщо ви не голландець".

Головна думка цього правила така: "практика, практика і ще раз практика, тоді ви зможете бачити найкращі рішення відразу".

PS: голландець - посилання на творця мови Python Гвідо ван Россума, він голландець.

Now is better than never

Дослівно - “зараз краще ніж ніколи”.

Головна думка цього правила така: "якщо ви знаєте як покращити щось, то покращуйте!".

Although never is often better than *right* now

Дослівно - "проте ніколи часто краще ніж прямо зараз".

Головна думка цього правила така: "якщо ви хочете щось оптимізувати о 3 годині ночі, то краще поспати і зі свіжою головою це переглянути".

Принцип YAGNI - You Aren’t Gonna Need It

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

Дослівно - "якщо реалізацію складно пояснити, це погана ідея".

Головна думка даного правила така: "вчитеся робити прості та ефективні рішення і вчіться їх пояснювати так само просто".

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

Дослівно - "якщо реалізацію легко пояснити, можливо, це гарна ідея".

Головна думка цього правила така: "навіть якщо вам вдалося щось пояснити, то це не запорука успіху, 2+2=5 це просто, але неправильно".

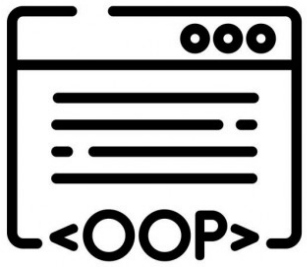
Namespaces are one honking great idea -- let's do more of those !

Дослівно - "простори імен це чудова ідея, давайте зробимо ще більше їх!".

Головна думка цього правила така: "у Python все пов'язано одним великим глобальним простором імен, зрозумійте його і вам стане простіше зрозуміти мову".

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

Висновок з курсу



OOP

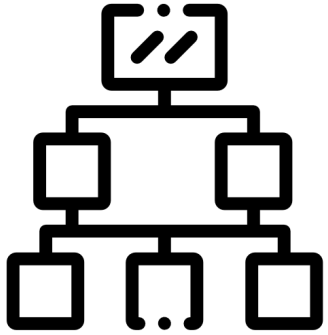
- Наслідування
- Інкапсуляція
- Поліморфізм
- Абстракція

Recursion

- Приклади використання рекурсії
- Дерева пошуку

Files

- Читання та запис
- JSON, XML, CSV



Modules

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

PEP8

- PEP
- PEP8
- Python Zen

Інформаційний відеосервіс для розробників програмного забезпечення

TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA

Перевірка знань

TestProvider.com

TestProvider – це online сервіс перевірки знань з інформаційних технологій. За його допомогою Ви можете оцінити Ваш рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і для загальної оцінки знань IT спеціаліста.

Перевірте як Ви засвоїли даний матеріал на TestProvider.com

Успішне проходження фінального тестування дозволить Вам отримати відповідний Сертифікат.

Дякую за увагу! До нових зустрічей!

Вікторія Бойчук
Python Developer, тренер CBS