Zen of PYTHON

by Tim Peters

PEP 20

Arsalan Ali
@arslanchaos

CODANICS

1- Beautiful is better than ugly

Violation example:

```
1  def this_is_my_custom_Multiplication_function(n1, n2):
2      multiply_output = n1 * n2
3      return multiply_output
4
5  this_is_my_custom_Multiplication_function(3,4)
```

✓ 0.1s

```
12
```

Fulfilment example:

```
1  def multiplication(number_1, number_2):
2      return number_1 * number_2
3
4  multiplication(3,4)
```

✓ 0.1s

```
12
```

2- Explicit is better than implicit

Violation example:

```python
def sum_of_string_digits(string):
    return sum([int(char) for char in string.split() if char.isdigit()])

sum_of_string_digits("10 20 30")
```

✓ 0.1s

```
60
```

Fulfilment example:

```python
def integers_from_string(string):
    integers = []
    for char in string.split():
        if char.isdigit():
            integers.append(int(char))
    return integers

def strings_digits_sum(string):
    integers = integers_from_string(string)
    return sum(integers)

strings_digits_sum("10 20 30")
```

✓ 0.1s

```
60
```

3- Simple is better than complex

Violation example:

```
1  def reverse_string(string):
2      if len(string) == 1:
3          return string
4      else:
5          return reverse_string(string[1:]) + string[0]
6
7
8  string = "sunila"
9
10 reverse_string(string)
```

✓ 0.7s

```
'alinus'
```

Fulfilment example:

```
1  reverse_string = lambda x: x[::-1]
2
3  string = "sunila"
4
5  reverse_string(string)
```

✓ 0.1s

```
'alinus'
```

4- Complex is better than complicated

Violation example:

```python
# Complicated code (even experts would struggle to understand it at first)
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

dot_product([3,4],[5,6])
```

✓ 0.1s

```
39
```

Fulfilment example:

```python
# Complex code (but experts can understand it easily)
def dot_product(list_a, list_b):
    return sum(map(lambda a, b: a*b, list_a, list_b))

dot_product([3,4],[5,6])
```

✓ 0.1s

```
39
```

5- Flat is better than nested

Violation example:

```python
# Nested
a, b, c = 10, 5, "120000000000000"
if a > b:
    if len(c) > a:
        print("1")
else:
    print("2")
```

✓ 0.2s

```
1
```

Fulfilment example:

```python
a, b, c = 10, 5, "120000000000000"
if b < a < len(c):
    print(1)
else:
    print(2)
```

✓ 0.7s

```
1
```

6- Sparse is better than dense

Violation example:

```python
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(f"Sum of positive numbers is {sum_of_values}")

print_sum_of_positive_int_numbers_in_mixed_list([1,-2,"9",3,-4,2,-10, "2"])
```

✓ 0.1s

```
Sum of positive numbers is 6
```

Fulfilment example:

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

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

def sum_of_array_values(array):
    numbers = integers_from_list(array)
    positive = greater_than_zero_integers_from_list(numbers)
    return f"Sum of positive numbers is {sum(positive)}"

sum_of_array_values([1,-2,"9",3,-4,2,-10, "2"])
```

✓ 0.1s

```
'Sum of positive numbers is 6'
```

7- Readability counts

Violation example:

```
VarC = "C"
elemedf_fd_dict = { "A": 1,
 "B": 2
 }
elemedf_fd_dict[VarC] = \
    3
elemedf_fd_dict
```

✓ 0.9s

```
{'A': 1, 'B': 2, 'C': 3}
```

Fulfilment example:

```
char, value = "C", 3
character_dictionary = {"A": 1, "B": 2, "C": 3}
character_dictionary[char] = value

character_dictionary
```

✓ 0.1s

```
{'A': 1, 'B': 2, 'C': 3}
```

8- Special cases aren’t special enough to break the rules

Violation example:

```python
def my_minFunction(*nums):
    [print(f"the smallest is {num}") for num in nums if num == min(nums)]

my_minFunction(2,3,4)
```

✓ 0.1s

```
the smallest is 2
```

Fulfilment example:

```python
def my_minFunction(*nums):
    return f"the smallest is {min(nums)}"

my_minFunction(2,3,4)
```

✓ 0.1s

```
'the smallest is 2'
```

9- Although practicality beats purity

This may seem like a contradiction with the previous statement.
But there are times when you don't have enough time to follow all rules in a project and thus you can break a few rules

10- Errors should never pass silently

Violation example:

```python
def division(a, b):
    try:
        return a / b
    except:
        pass

division(5,0)
```

✓ 0.1s

Fulfilment example:

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

division(5,0)
```

⊗ 0.1s

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
c:\Users\Arsla\OneDrive\Desktop\Python Chilla 2.0\Topics_Posts to Study_Create\Seven
      4     except ZeroDivisionError as error:
      5         raise error
----> 7 division(5,0)
```

11- Unless explicitly silenced

As for some cases where we don't want to get bothered over trivial errors. We silence those errors since practicality beats purity!

Violation example:

```
1  import warnings
2  warnings.filterwarnings('ignore')
```

✓ 0.2s

12- In the face of ambiguity, refuse the temptation to guess

Violation example:

```
def custom_db_mlt(arg1, arg2):
    def sq_args(arg):
        return arg**2
    arg1_update = sq_args(arg1)
    arg2_update = sq_args(arg2)
    return arg1_update*arg2_update

custom_db_mlt(2,2)
```

✓ 0.1s

```
16
```

Fulfilment example:

```
def squaring(num):
    return num**2

def squared_multiplication(num_1, num_2):
    squared_num_1 = squaring(num_1)
    squared_num_2 = squaring(num_2)
    return squared_num_1*squared_num_2

squared_multiplication(2,2)
```

✓ 0.1s

```
16
```

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

Violation example:

```python
# Creating a list
my_list = [1, 2, 3, 4, 5]
my_list_2 = [i for i in range(1, 6)]
my_list_3 = list(range(1, 6))
my_list_4 = [i for i in other_list]
```

Fulfilment example:

```python
# This function can create list out of any mixed or same elements
def create_list(*args):
    return list(args)

create_list(1, "cat", 2, False, "mango", 0)
```

✓ 0.1s

```
[1, 'cat', 2, False, 'mango', 0]
```

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

Its a reference to the creator of Python "Guido van Rossum" since he's a Dutch.
Tim states that there are many ways for a solution but only the Dutch would know the optimal one.
We always go from complex to simpler solutions.

15- Now is better than never

If you know you can optimize the code, you've an idea you want to implement, or there's some tweaking you would like to try then now is the time to do it. Its like don't get stuck in a loop of procrastination.

16- Although never is often better than "*right*" now

"You aren't gonna need it" (YAGNI) principle does apply if your project has a time limit.
You can skip some optimizations while delivering the project at a deadline.

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

If you've completed a project and you're unable to explain it to others in simpler terms.
then it means you either have lack of knowledge or the solution is way too complex.

18- If the implementation is easy to explain, it may be a good idea

If you've completed a project and you're able to explain it to others in simpler terms.
then it means you have knowledge about it even if its a bad code. At least you're on the right track.

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

A namespace is an abstraction used in Python to organize names assigned to objects in a program.
Python using namespaces is an awesome concepts that makes it easier for developers.

```
1  variable = 200
2  print(id(variable))
3  print(id(200))
```

✓ 0.1s

```
2527034415568
2527034415568
```

```
1 import this
```

✓ 0.3s

```
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!
```