Name:- Gautam Chandrakant Mandaliya

Roll No:- 27 Class:- FYCS

Subject:- IT TOOLS

Practical 7:- Implementing Coding Practices in PYTHON using PEP8

What is PEP8?

Indeed coding and applying logic is the foundation of any programming language but there's also another factor that every coder must keep in mind while coding and that is the coding style. Keeping this in mind, Python maintains a strict way of order and format of scripting. Following this sometimes mandatory and is a great help on the user's end, to understand. Making it easy for others to read code is always a good idea, and adopting a nice coding style helps tremendously for that.

For Python, PEP 8 has emerged as the style guide that most projects adhere to; it promotes a very readable and eye-pleasing coding style. Every Python developer should read it at some point; here are the most important points extracted:-

1. ***Use 4-space indentation and no tabs:-*** The 4 space rule is not always mandatory and can be overruled for continuation line.

Eg:-

a.

```
# Aligned with opening delimiter.
grow = function_name(variable_one, variable_two,
                     variable_three, variable_four)
```

b.

```
# First line contains no argument. Second line onwards more indentation included to distinguish this from the rest.
def function_name(
    variable_one, variable_two, variable_three,
    variable_four):
  print(variable_one)
```

2. ***Use docstrings:-*** There are both single and multi-line docstrings that can be used in Python. However, the single line comment fits in one line, triple quotes are used in both cases. These are used to define a particular program or define a particular function.

Eg:-

```
def exam():
    """This is single line docstring"""


    """This is
    a
    multiline comment"""
```

3. ***Wrap lines so that they don't exceed 79 characters:-*** The Python standard library is conservative and requires limiting lines to 79 characters. The lines can be wrapped using parenthesis, brackets, and braces. They should be used in preference to backslashes.

Eg:-

```
with open('/path/from/where/you/want/to/read/file') as file_one, \
     open('/path/where/you/want/the/file/to/be/written', 'w') as file_two:
    file_two.write(file_one.read())
```

4. ***Use of regular and updated comments are valuable to both the coders and users:-*** There are also various types and conditions that if followed can be of great help from programs and users point of view. Comments should form complete sentences. If a comment is a full sentence, its first word should be capitalized, unless it is an identifier that begins with a lower case letter. In short comments, the period at the end can be omitted. In block comments, there are more than one paragraphs and each sentence must end with a period. Block comments and inline comments can be written followed by a single '#'.

Eg:-

```
sum = sum + 1              # Increment
```

5. ***Use of trailing commas:-*** This is not mandatory except while making a tuple.

Eg:-

```
tup = ("gautam",)
```

6. ***Use spaces around operators and after commas, but not directly inside bracketing constructs:-***

Eg:-

a = f(1, 2) + g(3, 4)

7. ***Naming Conventions:-*** There are few naming conventions that should be followed in order to make the program less complex and more readable. At the same time, the naming conventions in Python is a bit of mess, but here are few conventions that can be followed easily.There is an overriding principle that follows that the names that are visible to the user as public parts of API should follow conventions that reflect usage rather than implementation.

Here are few other naming conventions:-

b (single lowercase letter)

B (single upper case letter)

lowercase

lower_case_with_underscores

UPPERCASE

UPPER_CASE_WITH_UNDERSCORES

CapitalizedWords (or CamelCase). This is also sometimes known as StudlyCaps.

Note: While using abbreviations in CapWords, capitalize all the letters

of the abbreviation. Thus HTTPServerError is better than HttpServerError.

mixedCase (differs from CapitalizedWords by initial lowercase character!)

Capitalized_Words_With_Underscores

In addition to these few leading or trailing underscores are also considered.

a.single_leading_underscore:- weak “internal use” indicator.

Eg:-

from M import * does not import objects whose name starts with an underscore.

b. ***single_trailing_underscore_:-*** used to avoid conflicts with Python keyword.

Eg:-

Tkinter.Toplevel(master, class_='ClassName')

c. ***__double_leading_underscore:-*** when naming a class attribute, invokes name mangling.

Eg:-

(inside class FooBar, __boo becomes _FooBar__boo;).

d***__double_leading_and_trailing_underscore__:-*** "magic" objects or attributes that live in user-controlled namespaces.

Eg:-

__init__, __import__ or __file__. Only use them as documented.

8. ***Characters that should not be used for identifiers:-*** 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names as these are similar to the numerals one and zero.

9***. Don't use non-ASCII characters in identifiers:-*** If there is only the slightest chance people speaking a different language will read or maintain the code.

10. ***Name your classes and functions consistently:-*** The convention is to use CamelCase for classes and lower_case_with_underscores for functions and methods. Always use self as the name for the first method argument.

11***. While naming of function of methods always use self for the first argument to instance methods and cls for the first argument to class methods.If a functions argument name matches with reserved words then it can be written with a trailing comma.***

Eg:-

class_

Program 1:-

Code:-

pep8ii.py - D:/Semester-2/OOP/Python/pep8ii.py (3.9.1)

File Edit Format Run Options Window Help

```
# Python program to find the factorial of a number provided by the user.

# change the value for a different result
num = int(input("Enter a number: "))
factorial = 1

# check if the number is negative, positive or zero
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,num + 1):
        factorial = factorial*i

print("The factorial of",num,"is",factorial)
```

Output:-

IDLE Shell 3.9.1

File Edit Shell Debug Options Window Help

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/Semester-2/OOP/Python/pep8ii.py ==========
Enter a number: 6
The factorial of 6 is 720
>>>
```

Program 2:-

Code:-

pep8.py - D:/Semester-2/OOP/Python/pep8.py (3.9.1)

File Edit Format Run Options Window Help

```
# Writing this PYTHON CODE to get the user details using PEP8.

class Student:
    """Student Class"""

    def __init__(self, name = None, rollno = None, age = None, marks = None):
        self.name = name
        self.rollno = rollno
        self.age = age
        self.marks = marks

    def setAge(self, age):
        self.age = age

    def setMarks(self, marks):
        self.marks = marks

    def Display(self):
        print("Student Name is: ", self.name)
        print("Student Roll Number is: ", self.rollno)
        print("Student Age is: ", self.age)
        print("Student Marks is: ", self.marks)


std = Student("Gautam Chandrakant Mandaliya", 27)
std.setAge(18)
std.setMarks(82)
std.Display()
```

Output:-

IDLE Shell 3.9.1

File Edit Shell Debug Options Window Help

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/Semester-2/OOP/Python/pep8.py ==========
Student Name is:  Gautam Chandrakant Mandaliya
Student Roll Number is:  27
Student Age is:  18
Student Marks is:  82
>>>
```

Program 3:-

Code:-

pep8iii.py - D:/Semester-2/OOP/Python/pep8iii.py (3.9.1)

File Edit Format Run Options Window Help

```
# Python program to perform Addition Subtraction Multiplication and Division of two numbers

num1 = int(input("Enter First Number: "))
num2 = int(input("Enter Second Number: "))

print("Enter which operation would you like to perform?")
op = input("Enter any of these char for specific operation +,-,*,/: ")

result = 0
if op == '+':
    result = num1 + num2
elif op == '-':
    result = num1 - num2
elif op == '*':
    result = num1 * num2
elif op == '/':
    result = num1 / num2
else:
    print("Input character is not recognized!")

print(num1, op , num2, ":", result)
```

Output:-

IDLE Shell 3.9.1

File Edit Shell Debug Options Window Help

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/Semester-2/OOP/Python/pep8iii.py ==========
Enter First Number: 5
Enter Second Number: 6
Enter which operation would you like to perform?
Enter any of these char for specific operation +,-,*,/: +
5 + 6 : 11
>>>
========== RESTART: D:/Semester-2/OOP/Python/pep8iii.py ==========
Enter First Number: 7
Enter Second Number: 4
Enter which operation would you like to perform?
Enter any of these char for specific operation +,-,*,/: -
7 - 4 : 3
>>>
========== RESTART: D:/Semester-2/OOP/Python/pep8iii.py ==========
Enter First Number: 9
Enter Second Number: 5
Enter which operation would you like to perform?
Enter any of these char for specific operation +,-,*,/: *
9 * 5 : 45
>>>
========== RESTART: D:/Semester-2/OOP/Python/pep8iii.py ==========
Enter First Number: 24
Enter Second Number: 3
Enter which operation would you like to perform?
Enter any of these char for specific operation +,-,*,/: /
24 / 3 : 8.0
>>>
```

----------X----------