IT TOOLS PRACTICE   PRACTICAL 7   NAME: KAJAL TIWARI
AND PLATFORMS   CLASS: FYCS
                                                     ROLL NO: 39

❖ **Implementing coding practices in Python using PEP8**.

- **PEP8:**
  i. The PEP is an abbreviation form of **Python Enterprise Proposal**.

  ii. PEP 8 is a document that provides various guidelines to write the readable in Python.

  iii. PEP 8 describes how the developer can write beautiful code.

  iv. It was officially written in 2001 by Guido van Rossum, Barry Warsaw, and Nick Coghlan.

  v. The main aim of PEP is to enhance the readability and consistency of code.

- **PEP 8 — guidelines for Python code:**

  The very first rule PEP 8 introduces is that rules/guidelines in PEP 8 should only apply when they make our code easier to read. This means we should apply PEP 8 to enhance the readability of the code and to make it less complex. For example, if we're working on a project that was already written with a different coding style (that is, it is already easy to read), we should use that project style for new code. If the PEP 8 rules somehow make the code harder to read or make it complex while writing the code, we should ignore those rules. As Guido Van Rossum, the creator of Python, has noted:

1. Naming Convention

When we write the code, we need to assign name to many things such as variables, functions, classes, packages, and a lot more things. Selecting a proper name will save time and energy. When we look back to the file after sometime, we can easily recall what a certain variable, function, or class represents. Developers should avoid choosing inappropriate names.

The naming convention in Python is slightly messy, but there are certain conventions that we can follow easily. Let's see the following naming convention.

Example -

- Single lowercase letter-

a = 10

- Single upper case letter-

A = 10

- Lowercase-

var = 10

- Lower_case_with_underscores
- number_of_apple = 5
- UPPERCASE-

VAR = 6

- UPPER_CASE_WITH_UNDERSCORES
- NUM_OF_CARS =20
- CapitalizedWords (or CamelCase)
- NumberOfBooks = 100

2. Name Style

Below is the table that specifies some of the common naming styles in Python. Consider the following table.

TYPE	FUNCTION	EXAMPLE
Function	We should use the lowercase words or separates words by the underscore.	myfunction, my_function
Variable	We should use a lowercase letter, words, or separate words to enhance the readability.	a, var, variable_name
Class	The first letter of class name should be capitalized; use camel case. Do not separate words with the underscore.	MyClass, Form, Model
Method	We should use a lowercase letter, words, or separate words to enhance readability.	class_method, method
Constant	We should use a short, uppercase letter, words, or separate words to enhance the readability.	MYCONSTANT, CONSTANT, MY_CONSTANT
Module	We should use a lowercase letter, words, or separate words to enhance the readability.	Module_name.py, module.py

IT TOOLS PRACTICE
AND PLATFORMS

PRACTICAL 7

NAME: KAJAL TIWARI
CLASS: FYCS
ROLL NO: 39

Package	We should use a lowercase letter, words, or separate words to enhance the readability. Do not separate words with the underscore.	package, mypackage,

Above are some common naming conventions that are useful to beautify the Python code. For additional improvement, we should choose the name carefully.

3. <u>Indentation</u>

Unlike other programming languages, the indentation is used to define the code block in Python. The indentations are the important part of the Python programming language and it determines the level of lines of code. Generally, we use the 4 space for indentation. Let's understand the following example.

Example -

```
x = 5

if x ==  5:

    print('x is larger than 5')
```

In the above example, the indented print statement will get executed if the condition of *if statement* is true. This indentation defines the code block and tells us what statements execute when a function is called or condition trigger.

4. <u>Tabs vs. Space</u>

We can also use the *tabs* to provide the consecutive spaces to indicate the indentation, but whitespaces are the most preferable. Python 2 allows the mixing of tabs and spaces but we will get an error in Python 3.

5. <u>Indentation following Line Break</u>

It is essential to use indentation when using line continuations to keep the line to fewer than 79 characters. It provides the flexibility to determining between two lines of code and a single line of code that extends two lines. Let's understand the following example.

Example -

```
# Correct Way:
# Aligned with opening delimiter.
obj = func_name(argument_one, argument_two,
                argument_three, argument_four
```

IT TOOLS PRACTICE   PRACTICAL 7   NAME: KAJAL TIWARI
AND PLATFORMS   CLASS: FYCS
                                                     ROLL NO: 39

We can use the following structure.

```python
# first line doesn't has any argument
# We add 4 spaces from the second line to discriminate arguments from the rest.
def function_name(
    argument_one, argument_two, argument_three,
    argument_four):
  print(argument_two)


# 4 space indentation to add a level.
foo = long_function_name(
  var_one, var_two,
  var_three, var_four)
```

6. <u>Use docstring</u>

Python provides the two types of document strings or docstring - single line and multiple lines.
We use the triple quotes to define a single line or multiline quotes. Basically, these are used to
describe the function or particular program. Let's understand the following example.

Example -

```python
def add(a, b):
  """This is simple add method"""


  """This is
  a
  simple add program to add
  the two numbers. """
```

7. <u>Should a Line Break Before or After a Binary Operator?</u>

The lines break before or after a binary operation is a traditional approach. But it affects
the readability extensively because the operators are scattered across the different screens,
and each operator is kept away from its operand and onto the previous line. Let's understand
the following example.

Example –

```python
# Wrong:

# operators sit far away from their operands
```

```
marks = (engilsh_marks +
    math_marks +
    (science_marks - biology_marks) +
    Physics_marks
```

As we can see in the above example, it seems quite messy to read. We can solve such types of problems by using the following structure.

Example -

```
# Correct:
# easy to match operators with operands
Total_marks = (English_marks
        + math_marks
        + (science_marks - biology_marks)
        + physics_marks
```

Python allows us to break line before or after a binary operator, as long as the convention is consistent locally.

8. **Importing module**

We should import the modules in the separates line as follows.

```
import pygame
import os
import sys
```

> Wrong

```
import sys, os
```

We can also use the following approach.

```
from subprocess import Popen, PIPE
```

The import statement should be written at the top of the file or just after any module comment. Absolute imports are the recommended because they are more readable and tend to be better behaved.

```
import mypkg.sibling
from mypkg import sibling
```

IT TOOLS PRACTICE
AND PLATFORMS

PRACTICAL 7

NAME: KAJAL TIWARI
CLASS: FYCS
 ROLL NO: 39

```
from mypkg.sibling import example
```

9. Blank Lines

Blank lines can be improved the readability of Python code. If many lines of code bunched together the code will become harder to read. We can remove this by using the many blank vertical lines, and the reader might need to scroll more than necessary. Follow the below instructions to add vertical whitespace.

- **Top-level function and classes with two lines -** Put the extra vertical space around them so that it can be understandable.

```python
class FirstClass:
    pass


class SecondClass:
    pass


def main_function():
    return None
```

- **Single blank line inside classes -** The functions that we define in the class is related to one another. Let's see the following example -

```python
class FirstClass:
    def method_one(self):
        return None


    def second_two(self):
        return None
```

- **Use blank lines inside the function -** Sometimes, we need to write a complicated function has consists of several steps before the return statement. So we can add the blank line between each step. Let's understand the following example.

```python
def cal_variance(n_list):
    list_sum = 0
    for n in n_list:
```

```
    list_sum = list_sum + n
  mean = list_sum / len(n_list)


  square_sum = 0
  for n in n_list:
     square_sum = square_sum + n**2
  mean_squares = square_sum / len(n_list)


  return mean_squares - mean**2
```

The above way can remove the whitespaces to improve the readability of code.

- **Put the Closing Braces**

We can break lines inside parentheses, brackets using the Line continuations. PEP 8 allows us to use closing braces in implies line continuations. Let's understand the following example.

- **Line up the closing brace with the first non-whitespace.**

```
list_numbers = [
5, 4, 1,
4, 6, 3,
7, 8, 9
 ]
```

- **Line up the closing braces with the first character of line.**

```
list_numbers = [
   5, 4, 1,
   4, 6, 3,
   7, 8, 9
]
```

10. **Comments**

Comments are the integral part of the any programming language. These are the best way to explain the code. When we documented our code with the proper comments anyone can able to understand the code. But we should remember the following points.

- o Start with the capital latter, and write complete sentence.

- o Update the comment in case of a change in code.

- o Limit the line length of comments and docstrings to 72 characters.

- **Block Comment**

Block comments are the good choice for the small section of code. Such comments are useful when we write several line codes to perform a single action such as iterating a loop. They help us to understand the purpose of the code.

PEP 8 provides the following rules to write comment block.

- o Indent block comment should be at the same level.

- o Start each line with the # followed by a single space.

- o Separate line using the single #.

Let's see the following code.

```
for i in range(0, 5):
    # Loop will iterate over i five times and print out the value of i
    # new line character
    print(i, '\n')
```

We can use more than paragraph for the technical code. Let's understand the following example.

- **Inline Comments**

Inline comments are used to explain the single statement in a piece of code. We can quickly get the idea of why we wrote that particular line of code. PEP 8 specifies the following rules for the inline comments.

- o Start comments with the # and single space.

- o Use inline comments carefully.

- o We should separate the inline comments on the same line as the statement they refer.

Following is the example of inline comments.

```
a = 10    # The a is variable that holds integer value.
```

Sometimes, we can use the naming convention to replace the inline comment.

```
x = 'Peter Decosta' #This is a student name
```

We can use the following naming convention.

Student_name = 'Peter Decosta'

11. Avoid Unnecessary Adding Whitespaces

In some cases, use of whitespaces can make the code much harder to read. Too much whitespaces can make code overly sparse and difficult to understand. We should avoid adding whitespaces at the end of a line. This is known as trailing whitespaces.

Let's see the following example.

Example - 1

```
# Recommended
list1 = [1, 2, 3]

# Not recommended
List1 = [ 1, 2, 3, ]
```

Example - 2

```
x = 5
y = 6

# Recommended
print(x, y)

# Not recommended
print(x , y)
```