Implementing the coding practice in python using PEP8.

Ans :-

It gets difficult to understand a messed up handwriting, similarly an unreadable and unstructured code is not accepted by all. However, you can benefit as a programmer only when you can express better with your code. This is where PEP comes to the rescue.

Python Enhancement Proposal or PEP is a design document which provides information to the Python community and also describes new features and document aspects, such as style and design for Python. Python is a multi-paradigm programming language which is easy to learn and has gained popularity in the fields of Data Science and Web Development over a few years and PEP 8 is called the style code of Python. It was written by Guido van Rossum, Barry Warsaw, and Nick Coghlan in the year 2001. It focuses on enhancing Python's code readability and consistency. Join the certification course on Python Programming and gain skills and knowledge about various features of Python along with tips and tricks.

The Zen of Python

It is a collection of 19 'guiding principles' which was originally written by Tim Peters in the year 1999. It guides the design of the Python Programming Language. Python was developed with some goals in mind. You can see those when you type the

following code and run it:

>>> 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!

The Need for PEP 8

Readability is the key to good code. Writing good code is like an art form which acts as a subjective topic for different developers.

Readability is important in the sense that once you write a code, you need to remember what the code does and why you have written it. You might never write that code again, but you'll have to read that piece of code again and again while working in a project.

PEP 8 adds a logical meaning to your code by making sure your variables are named well, sufficient whitespaces are there or not and also by commenting well. If you're a beginner to the language, PEP 8 would make your coding experience more pleasant.

Following PEP 8 would also make your task easier if you're working as a professional developer. People who are unknown to you and have never seen how you style your code will be able to easily read and understand your code only if you follow and recognize a particular guideline where readability is your de facto.

And as Guido van Rossum said— "Code is read much more than it is often written".

The Code Layout

Your code layout has a huge impact on the readability of your code.

Indentation

The indentation level of line is computed by the leading spaces and tabs at the beginning of a line of logic. It influences the grouping of statements.

The rules of PEP 8 says to use 4 spaces per indentation level and also spaces should be preferred over tabs.

An example of code to show indentation:

```
x = 5

if x < 10:

print('x is less than 10')
```

Tabs or Spaces?

Here the print statement is indented which informs Python to execute the statement only if the if statement is true. Indentation also helps Python to know what code it will execute during function calls and also when using classes. PEP 8 recommends using 4 spaces to show indentation and tabs should only be used to maintain consistency in the code.

Python 3 forbids the mixing of spaces and tabs for indentation. You can either use tabs or spaces and you should maintain consistency while using Python 3. The errors areautomatically displayed:

python hello.py

File "hello.py", line 3

print(i, j)

^

TabError: inconsistent use of tabs and spaces in indentation

However, if you're working in Python 2, you can check the consistency by using a -t flag in your code which will display the warnings of inconsistencies with the use of spaces and tabs.

You can also use the -tt flag which will show the errors instead of warnings and also the location of inconsistencies in your code.

Maximum Line Length and Line Breaking

The Python Library is conservative and 79 characters are the maximum required line limit as suggested by PEP 8. This helps to avoid line wrapping. Since maintaining the limit to 79 characters isn't always possible, so PEP 8 allows wrapping lines using Python's implied line continuation with parentheses, brackets, and braces:

```
def function(argument_1, argument_2,
             argument_3, argument_4):
    return argument_1
```

Or by using backslashes to break lines:

```
with open('/path/to/some/file/you/want/to/read') as example_1, \
     open('/path/to/some/file/being/written', 'w') as example_2:
    file_2.write(file_1.read())
```

When it comes to binary operators, PEP 8 encourages to break lines before the binaryoperators. This accounts for more readable code.

Let us understand this by comparing two examples:

```
# Example 1

# Do

total = ( variable_1 + variable_2 - variable_3 )

# Example 2

# Don't

total = ( variable_1 + variable_2 - variable_3 )
```

In the first example, it is easily understood which variable is added or subtracted, since the operator is just next to the variable to which it is operated. However, in the second example, it is a little difficult to understand which variable is added or subtracted.

Indentation with Line Breaks

Indentation allows a user to differentiate between multiple lines of code and a single line of code that spans multiple lines. It enhances readability too.

The first style of indentation is to adjust the indented block with the delimiter:

```
def function(argument_one, argument_two,
             argument_three, argument_four):
    return argument_one
```

You can also improve readability by adding comments:

```
x = 10

if (x > 5 and
   x < 20):
   # If Both conditions are satisfied

print(x)
```

Or by adding extra indentation:

```
x = 10

if (x > 5 and
   x < 20):
   print(x)
```

Another type of indentation is the hanging indentation by which you can symbolize a

continuation of a line of code visually:

```
foo = long_function_name(
    variable_one, variable_two,
    variable_three, variable_four)
```

You can choose any of the methods of indentation, following line breaks, in situations where the 79 character line limit forces you to add line breaks in your code, which will ultimately improve the readability.

Blank lines

Blank lines are also called vertical whitespaces. It is a logical line consisting of spaces, tabs, formfeeds or comments that are basically ignored.

Using blank lines in top-level-functions and classes:

```
class my_first_class:
    pass
class my_second_class:
    pass
def top_level_function():
   return None
```

Adding two blank lines between the top-level-functions and classes will have a clear separation and will add more sensibility to the code.

Using blank lines in defining methods inside classes:

```
class my_class:
    def method_1(self):
    return None


def method_2(self):
    return None
```

Here, a single vertical space is enough for a readable code. You can also use blank spaces inside multi-step functions. It helps the reader to gather the logic of your function and understand it efficiently. A single blank line will work in such case.

An example to illustrate such:

```
def calculate_average(number_list):
    sum_list = 0
```

```
for number in number_list:
    sum_list = sum_list + number


average = 0
    average = sum_list / len(number_list)


return average
```

Above is a function to calculate the average. There is a blank line between each step and also before the return statement. The use of blank lines can greatly improve the readability of your code and it also allows the reader to understand the separation of the sections of code and the relation between them.