Practical 7) Implementing coding practices in Python using PEP8

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 for you:

1. Use 4-space indentation and no tabs.
Examples:

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

The 4 space rule is not always mandatory and can be overruled for continuation line.

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.
Example:

```
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.
Example:

```
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 '#'. Example of inline comments:

```
geek = geek + 1                # Increment
```

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

```
tup = ("geek",)
```

5. **Use Python's default *UTF-8* or *ASCII* encodings and not any fancy encodings**, if it is meant for international environment.

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

```
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. Examples:

single_leading_underscore: weak "internal use" indicator. E.g. from M import * does not import objects whose name starts with an underscore.

single_trailing_underscore_: used to avoid conflicts with Python keyword. Example:

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

__double_leading_underscore: when naming a class attribute, invokes name mangling.
(inside class FooBar, __boo becomes _FooBar__boo;).
__double_leading_and_trailing_underscore__: "magic" objects or attributes that live in user-controlled namespaces. E.g. *__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.