Python Good Practice (PEP8 Guide)

Naming Conventions

Variables: `my_well_named_variable`

Constants: `MY_WELL_NAMED_CONSTANT`

Functions: `my_well_named_function`

Modules and Packages: `mywellnamedmodule` **or** `my_well_named_module`

Classes: `MyWellNamedClass`

_Modules and Packages may use underscores if it improves readability - but ideally should be exclusively lowercase.

The Look

No Unnecessary Whitespace:

`add(x, y)` **not** `add ( x , y )`

Tabs Not Spaces!

This is actually the opposite of convention, which says indentation should always be be exactly 4 spaces. But we find that is harder for learners to grasp!

Line Length should be no more than 80 characters!

This is to make your code readable on small screens, in windowed view, on vertical screens, when formatted in browsers etc...

Python Good Practice (PEP8 Guide)

No Commas

```
import math
import random
```

not

```
import math, random
```

```
name = "dan"
age = 30
```

not

```
name, age = "dan", 30
```

Python can do this but it is generally seen as bad practice! There are specific circumstances where it is accepted.

No Magic Numbers

```
SENSIBLE_NAME = 30
if my_variable > SENSIBLE_NAME:
```

not

```
if my_variable > 30:
```

0 and 1 are allowed as magic numbers!

No Errors

Possible errors should always be handled!

```
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

Python Good Practice (PEP8 Guide)

Commenting

Good use of commenting is essential!

This should be within reason - not every line of code requires a comment, the code itself should be self evident. However, complicated iterations/selections may require clarification.

```
#This iterates through list and does something
```

Use DocStrings!

At the top of every module, function, and class, describe the purpose of that module, function or class for the user! This should not explain the code, just the general idea, what parameters must be provided, and what it returns!

```
"""
This function does this thing.
Args: n (int): It does it to this integer n
Returns: new_n (int): It returns the thing
"""
```

Small Scope

Funtions and Modules should have a small restricted scope!

It is better to have lots of small funtions that each perform simple tasks and can be reused. This also goes for the modules in your Python package.
This helps you adhere to the Python Community Standards:
DRY *(Don't Repeat Yourself)*, *Simple Is Better Than Complex*, and *Readability Counts*.

Python Good Practice (PEP8 Guide)

Python File Structure

```
"""
module docstring
"""

import standardlibrary
import thirdpartylibrary
import projectmodules


MODULE_LEVEL_CONSTANT = 0


define all_functions(parameters):

    """
    function docstring
    """

    FUNCTION_LEVEL_CONSTANT = 0

    function_level_variable = 0


define allClasses:

    """
    class docstring
    """


main()
```

This is only the logical and necessary order of elements in a Python file.
Not all elements have to be incorporated in every file - and as mentioned small reusable modules are better!
***Module Level Variables** (Global) are possible but generally bad practice.*
***Classes** are A-Level content.*

***main()** should always call the main function that begins the program.*

In other words there should be no code floating around at the bottom of the python file - the code that starts the program should be defined in main!

Python Good Practice (PEP8 Guide)

Python Project Structure

```
my_package/
├── my_package/              # Main package directory
│   ├── module1.py           # Module 1
│   ├── module2.py           # Module 2
│   └── subpackage/          # Subpackage directory
│       ├── submodule1.py
│       └── submodule2.py
├── tests/                   # Tests directory
│   ├── test_module1.py
│   └── test_module2.py
├── resources/               # Resources directory
│   ├── input_data.txt
│   └── output_data.csv
├── README.txt               # Documentation
└── requirements.txt         # List of dependencies
```

This is only a basic overview of the file structure you may encounter in Python projects - whether they are libraries you have downloaded or online representations such as on GitHub.

There are many more files and directories that may be included in a professional Python package.

***Note** that creating a Package of this complexity, or recalling this structure, is not a requirement of the GCSE CS course.*

This is simply a useful overview and background knowledge.

PEP8 Style Guide: peps.python.org/pep-0008/