**1) Imports** **:** The Python import statement imports code from one module into another program. A module is a file that contains functions and values that we can reference from our program.

The import statement syntax is: **import modulename**.

**Ex** :

import time

current\_time=time.ctime()

print(current\_time)

import statements appear at the top of a Python file, beneath any comments that may exist. This is because importing modules or packages at the top of a file makes the structure of the code clearer.

We can override the default module name using the ‘as’ keyword. This is useful if the name of a module conflicts with a variable that we declared in the code. It is also useful if we want to reference another module that shares the same name.

**Ex :**

import time as t

time = t.ctime()

print(time)

**2) Indentation :** Indentation in Python refers to the spaces and tabs that are used at the beginning of a statement. The statements with the same indentation belong to the same group called a suite.

**Ex :**

if a==1:

print(a)

if b==2:

print(b)

print(‘end’)

In the above code, the first and last line of the statement is related to the same suite because there is no indentation in front of them. So after executing the first "if statement", the Python interpreter will go into the next statement. If the condition is not true, it will execute the last line of the statement.

The third statement: "if b==2" will be executed if the first statement: "if a==1" is true.

In the next statement, "print (b)" is in a separate suite, and it will execute only if the statement "if b==2" is true.

**3) Length of the line :**

Generally, it's good to aim for a line length of 79 characters in the Python code.

**The advantages are :**

* It is possible to open files side by side to compare;
* can view the whole expression without scrolling horizontally which adds to better readability and understanding of the code.

Comments should have 72 characters of line length.

**4) Naming Conventions :**

Use grammatically correct variable names, the class name should start with an uppercase and must follow camelCase convention If more than two words are to be used. In the same way, a function name should be joined with an underscore, and it must be lowercase. In method arguments, always use self as the first argument to declare an instance variable. In the same way, use ‘cls’ for the first argument for the class method. If the function name clashes with a reserved argument, use an underscore instead of a wrong spelling. Constants are declared in all capital letters.

**Ex :**

# class name follows camelcase convention

class StudentDetails:

def \_\_init\_\_(self, first\_name, last\_name):

self.first\_name = first\_name

self.last\_name = last\_name

# Method name, variable names in lowercase joined with an underscore

def grade(self, marks\_obtained):

# constants in capital

GRACE = 2

marks\_obtained = GRACE + marks\_obtained

if marks\_obtained > 90:

self.student\_grade = 'A'

elif marks\_obtained > 70:

student\_grade = 'B'

else:

student\_grade = 'C'

**5) Blank Lines :**

In Python scripts, top-level functions and classes are separated by two blank lines.

Method definitions inside classes should be separated by one blank line.

**6) Whitespaces in Expressions and Statements :**

One should avoid extra white spaces, there must be a single white space around both sides of an operator.

**7) Comments :**

Comments are used for in-code documentation in Python. They add to the understanding of the code. There are lots of tools that you can use to generate documentation, such as comments and docstrings, for our own module. Comments should be more verbose so that when someone reads the code, the person would get the proper understanding of the code and how it is being used with other pieces of the code.

Comments start with the # symbol. Anything written after the hashtag does not get executed by the interpreter.

**Ex :**

# This is a python single line comment

print(“This is a Python comment”)

**8) Documentation of a Method :**

Documenting every method with proper specification of parameters, return type, and data types. Try to avoid multiple returns from a function, a single generic return is preferred.

**Ex :**

# documenting a function

def get\_grades(marks):

if marks > 90:

grade = 'A'

elif marks > 70:

grade = 'B'

else:

grade = 'C'

return grade

9) **Use DRY (Don’t Repeat Yourself) :**

Always use the DRY principle to reuse the code. The best way to do it is to use functions and classes. The common functions can be put into a separate utils.py file and can be used several times instead of creating similar functions again and again.

Suppose if you need to read three files, instead of writing code for file read thrice, you can read it as a function and save your time.

**Ex :**

# function to read the file read

def file\_read(filename):

with open(filename, 'r') as f:

return f.read()

qualities = file\_read('quality.txt')

description = file\_read('description.txt')

summary = file\_read('summary.txt')

**10) What to use? Tuples, Lists of Dictionaries**

Use tuples when data is non-changeable, dictionaries when we need to map things, and lists if the data can change later on.

**Ex :**

# tuples are used when data is constant

colors\_of\_rainbow = ('V', 'I', 'B', 'G', 'Y', 'O', 'R')

# lists where data can be mutated

movies\_to\_watch = ['Inception', 'Iron Man', 'Wonder Woman']

# Using lists in mapping is wrong

# O(n) time would be taken to extract marks

marks\_obtained = [['History', 30], ['English', 35], ['Physics', 45]]

# dicts when mapping is needed

# dicts take O(1) time to get key value

marks\_obtained = {'History': 30, 'English': 35, 'Physics': 45}