**Assignment 2**

**Programming Style**

**Understanding Python’s PEP 8 guidelines**

PEP 8 : Python Enhancement Proposal

PEP 8 (Python Enhancement Proposal 8) provides guidelines for writing clean and readable Python code. Here’s an easy-to-understand summary:

**1. Indentation**

* Use **4 spaces per indentation level**.
* Never use tabs. Stick to spaces for consistency.

**Example:**

def example\_function():

print("Hello")

if True:

print("Indented with 4 spaces")

**2. Line Length**

* Keep each line **under 79 characters**.
* For long lines, break them using parentheses or backslashes.

long\_variable = ("This is an example of a long line broken into multiple lines for clarity.")

**3. Blank Lines**

* Use blank lines to separate sections of code for better readability:
  + **2 blank lines** between top-level functions or class definitions.
  + **1 blank line** between methods in a class.

class MyClass:

def method\_one(self):

pass

def method\_two(self):

pass

**4. Imports**

* Put imports at the top of the file.
* Use separate lines for each import.

import os

import sys

# Incorrect

import os, sys

**5. Naming Conventions**

* **Variables and functions**: Use snake\_case (e.g., my\_variable, calculate\_sum).
* **Classes**: Use PascalCase (e.g., MyClass).
* **Constants**: Use UPPER\_CASE (e.g., PI, MAX\_LIMIT).

**6. Whitespace**

* Avoid unnecessary spaces around parentheses, brackets, or braces.
* Use a single space around operators and after commas.

**Example:**

# Correct

x = (1 + 2) \* 3

# Incorrect

x = ( 1+2 )\*3

**7. Comments**

* Write clear, meaningful comments.
* Use # for single-line comments.
* Use triple quotes (""") for docstrings in functions or classes.

**Example:**

# This is a single-line comment

def example\_function():

"""This is a docstring explaining what the function does."""

pass

**8. Avoid Trailing Whitespace**

* Remove extra spaces at the end of lines.

**9. Use Consistent Quotes**

* Use single (') or double (") quotes consistently for strings.

# Both are correct, but be consistent

my\_string = "Hello"

another\_string = 'World'

**10. Avoid Too Many Nested Levels**

* Refactor if your code is deeply nested.

# Too nested

if condition1:

if condition2:

if condition3:

do\_something()

# Better

if condition1 and condition2 and condition3:

do\_something()

**Indentation, comments, and naming conventions in Python.**

### ****1. Indentation****

In Python, **indentation is mandatory** to define blocks of code. It improves readability and shows the logical structure of the program.

#### ****Rules:****

1. Use **4 spaces** per level of indentation.
2. Never mix tabs and spaces.
3. Ensure all lines in the same block have the same level of indentation.

#### ****Examples:****

# Correct Indentation

def greet(name):

print(f"Hello, {name}") # Indented with 4 spaces

if name:

print("Welcome!") # Nested block indented further

# Incorrect Indentation

def greet(name):

print(f"Hello, {name}") # Inconsistent indentation (less than 4 spaces)

### ****2. Comments****

Comments are notes added to explain the code. They are ignored by Python during execution and help others (or your future self) understand the logic.

#### ****Types of Comments:****

1. **Single-Line Comments**:
   * Use # for short explanations or notes.

python

Copy code

# Assign a value to x

x = 10 # This is also a single-line comment

1. **Multi-Line Comments**:
   * Use triple quotes (""" or ''') for detailed explanations or long comments.

"""

This is a multi-line comment.

It is often used to provide detailed documentation.

"""

def example\_function():

pass

1. **Docstrings**:
   * Special comments at the start of functions, classes, or modules for documentation.

def add\_numbers(a, b):

"""This function adds two numbers and returns the result."""

return a + b

### ****3. Naming Conventions****

Python encourages specific naming styles to make the code readable and consistent.

#### ****Guidelines for Different Elements:****

1. **Variables and Functions**:
   * Use snake\_case: lowercase letters with underscores separating words.

user\_name = "Alice"

def calculate\_sum(a, b):

return a + b

1. **Classes**:
   * Use PascalCase: capitalize the first letter of each word.

class StudentDetails:

pass

1. **Constants**:
   * Use UPPER\_CASE for constants.

PI = 3.14

MAX\_LIMIT = 100

1. **Private Variables**:
   * Prefix with an underscore (\_) to indicate it’s intended for internal use.

\_internal\_variable = 42

1. **Magic/Dunder Methods**:
   * These are special methods surrounded by double underscores (\_\_).

class MyClass:

def \_\_init\_\_(self):

pass