# REPORT NO. 6.1: CODING STANDARD AND DOCUMENTATION TOOL

## COURSE CODE: CSE 404
## COURSE TITLE: SOFTWARE ENGINEERING AND ISD LABRATORY


Submitted by

Suraiya Mahmuda (364)


Submitted to

Dr. Md. Musfique Anwar, Professor

Dr. Md. Humayun Kabir, Professor



Computer Science and Engineering

Jahangirnagar University

Dhaka, Bangladesh


October 2, 2024

# Contents

# 1. OBJECTIVE

The objective of this work is to familiarize with the application of coding standards and the use of documentation tools. It aims to enhance understanding of maintaining consistency in code structure, improving readability, and generating clear documentation for future reference. Through this process, participants will develop skills in writing clean, well-organized code that adheres to established guidelines, while also learning to utilize tools that automate the creation of documentation for better code maintenance and collaboration.

# 2. SOFTWARE USED

VS Code, Git, Sphinx

# 3. Source Code

Here is the ‘CalculateSubstraction‘ class to find the subtraction of two numbers:

```
class CalculateSubstraction:
    """
    A class to represent basic mathematical operations.

    Attributes
    ----------
```

```
    num1 : float or int
        The first number
    num2 : float or int
        The second number

    Methods
    -------
    subtract():
        Subtracts num2 from num1 and returns the result.
    """

    def __init__(self, num1, num2):
        """
        Constructs all the necessary attributes for the Math
           object.

        Parameters
        ----------
        num1 : float or int
            The first number
        num2 : float or int
            The second number
        """
        self.num1 = num1
        self.num2 = num2

    def subtract(self):
        """
        Subtracts the second number (num2) from the first
           number (num1).

        Returns
        -------
        float or int
            The result of the subtraction (num1 - num2).
        """
        return self.num1 - self.num2

# Example usage:
math_obj = CalculateSubstraction(10, 5)
```

```
result = math_obj.subtract()
print(f"Subtraction result: {result}")
```

Listing 3.1: CalculateSubstraction class to find subtraction of two numbers

# 4. CODING STANDARD THAT HAS USED

# CODING STANDARDS

**Coding standards** are a set of guidelines or best practices that developers follow to write clean, consistent, and maintainable code. These rules cover naming conventions, file structure, formatting, and more. The necessity of coding standards lies in improving readability, reducing errors, and enabling collaboration among developers. By following a consistent approach, teams can avoid confusion, ensure compatibility, and make it easier to maintain and scale projects over time. Coding standards also help in automating quality checks and adhering to industry norms.

## 4.1 Naming Conventions

### 4.1.1 Variables

Use *snake_case* for variable names. Variable names should be meaningful and descriptive.

```
user_name = "John"
total_students = 25
```

Listing 4.1: Variable Example

### 4.1.2 Constants

Use `ALL_CAPS` with underscores separating words. Define constants at the top of the module.

```
MAX_RETRIES = 3
DEFAULT_TIMEOUT = 60
```

Listing 4.2: Constant Example

### 4.1.3 Functions and Methods

Use *snake_case* for function and method names. Function names should describe the action or behavior of the function.

```
def calculate_total_price(item_list):
    return sum(item.price for item in item_list)
```

Listing 4.3: Function Example

### 4.1.4 Classes

Use *PascalCase* for class names. Class names should be nouns and describe the object or entity they represent.

```
class UserProfile:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Listing 4.4: Class Example

### 4.1.5 Modules and Packages

Use *snake_case* for module (file) names and package (directory) names. Avoid using capital letters or special characters.

```
Example:

my_project/
  models.py
  views.py
  forms.py
```

### 4.1.6 Exception Names

The class naming convention applies here. However, you should use the suffix “Error” on your exception names.

```
class CustomError(Exception):
    pass

class FileNotFoundError(Exception):
    pass
```

Listing 4.5: Exception Example

## 4.2 Layout Convention

### 4.2.1 Indentation

Use 4 spaces per indentation level. Never use tabs.

```
def example_function():
    if True:
        print("Indented␣with␣4␣spaces")
```

Listing 4.6: Indentation Example

### 4.2.2 Line Length

Limit all lines to a length of 120 characters.

```
def example_function():
    if True:
        print("Indented␣with␣4␣spaces")
```

Listing 4.7: Line Length Example

### 4.2.3 Blank Space

Use two blank lines before top-level function and class definitions. Use one blank line to separate methods inside a class.

```
def first_function():
    pass

def second_function():
    pass

class MyClass:
    def method_one(self):
        pass

    def method_two(self):
        pass
```

Listing 4.8: Blank Space Example

## 4.3 Comments

### 4.3.1 Inline Comments

Use inline comments sparingly and make sure they explain why the code does something, not what it does. Place inline comments on the same line, separated by two spaces.

```
x = x + 1  # Increment x by 1
```

Listing 4.9: Inline Comment Example

### 4.3.2 Block Comments

Use block comments to explain a section of code. Start each line with a # and a single space, aligning with the indentation level of the code.

```
# This block calculates the average
# from a list of scores.
total = sum(scores)
average = total / len(scores)
```

Listing 4.10: Block Comment Example

### 4.3.3 Docstrings

Use triple quotes (""") for docstrings in functions, classes, and modules. Docstrings should explain the purpose of the function, method, or class.

```
def greet(name):
    """
    This function takes a name as input and returns a greeting
       string.
    """
    return f"Hello, {name}!"
```

Listing 4.11: Docstring Example

## 4.4 Imports

### 4.4.1 Multiple Imports

Multiple imports should usually be on separate lines.

```
import numpy
import pandas
import matplotlib
```

Listing 4.12: Multiple Imports Example

### 4.4.2 Always on the Top

Imports are always put at the top of the file i.e., after any module comments and docstrings, but before module globals and constants.

```
# import the numpy module
import numpy
```

Listing 4.13: Top Imports Example

### 4.4.3 Import Modules in an Order

A good practice is to import modules in the following order:

- Standard library modules – e.g. sys, os, getopt, re.
- Third-party library modules – e.g. ZODB, PIL.Image, etc.
- Locally developed modules.

# 5. DOCUMENTATION

Sphinx is a powerful tool used for generating documentation for Python projects. It allows for the automatic extraction of docstrings from the source code and formats them into beautiful, structured HTML, PDF, or other formats.

In this project, we utilized Sphinx to create documentation for the `CalculateSubtraction` class. This tool simplifies the process of documenting the class methods and their parameters, and it generates a clean and user-friendly interface.

The following image shows the generated documentation for the `CalculateSubtraction` class:

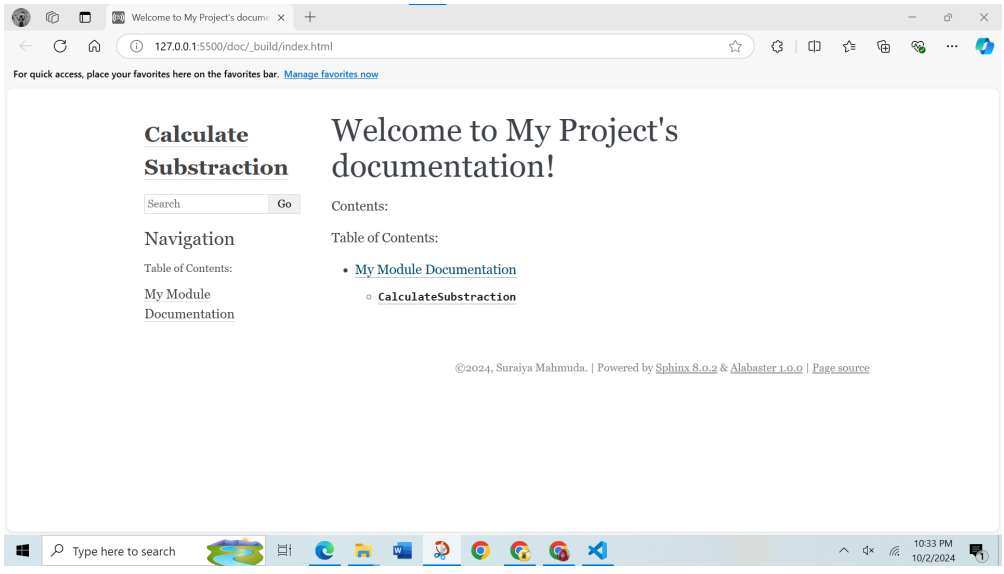


Figure 5.1: CalculateSubstraction Documentation generated using Sphinx

# 6. CONCLUSIONS

In conclusion, this project successfully demonstrated the importance of adhering to coding standards and the value of well-structured documentation. By following best practices, such as consistent naming conventions, appropriate indentation, and clear commenting, the codebase became more readable, maintainable, and scalable. These practices not only reduce the likelihood of errors but also enhance collaboration among team members.