SOEN 6441 (Advanced Programming Practices)

Concordia University
Department of Computer Science and Software Engineering

Project: CHEERS
Deliverable 1

Supervisor:
Prof. PANKAJ KAMTHAN

Students:

Team D
Nomesh Palakaluri (40229979)
Darsh Patel (40232273)
Dhairya Patel (40231545)
Krishna Patel (40232651)
Shreya Patel (40230660)
Urvilkumar Patel (40230630)

March 20, 2023

Contents

Introduction

1.1 Objective

For CHEERS, we have to calculate segment length L between two overlapping coasters for which computation of the angle with vertex at the center of the left circle α is necessary. This project has a number of different issues with their unique complications. For this report, we will discuss the most significant discoveries and how they relate to the CHEERS project will also be highlighted in the report.

1.2 Background

Through this project we will be able to find optimal length to place two coasters exactly on top of each other. Therefore, the project's utilization of mathematical formulas and ideas is essential. The project specifically uses the following equations for the length of the segment X1X2 and the angle .

$$l = 2R(1 - cos(\frac{\alpha}{2}))$$

where R is radius of coaster and alpha is calculated by:

$$\alpha - sin(\alpha) = \frac{\pi}{2}$$

.

1.3 Assumptions

We have made few assumptions in order to accomplish the project which are as follows:

1. We have used **11** terms to determine the value of sine and cosine in Taylor series.
    - A good equilibrium between accuracy and computational efficiency for values of x between $[-2\pi, 2\pi]$ is offered by using 11 term in Taylor series expansions for the sine and cosine functions.
2. We are taking 1 as the initial guess for newton's method.
    - The choice of the initial guess is important because it can affect the convergence of the method. One common choice for the initial guess in the Newton's method is to take it as 1. This choice is often used because it is a simple and convenient value that can work well for many functions. Additionally, 1 is a positive number, which is often helpful since many functions have positive roots.
    - However, the choice of the initial guess may also depend on the specific function being evaluated. For some functions, a different initial guess may be more appropriate for faster convergence. Therefore, it is often a good practice to experiment with different initial guesses to find the one that works best for a particular function.

3. We use Newton's technique to get the root of the equation that gives the value of α. We made certain assumptions to ensure it works properly.The Newton's method, also referred to as the Newton-Raphson method, is a sequential numerical technique for locating a function's root. They are as follows:

   - An initial guess for the root that is near to the real root is necessary according to the method.
   - **Continuity**: On the interval containing the root we're searching for, the function must be continuous. This indicates that the function has no breaks, leaps, or asymptotes.
   - **Differentiability**: On the range containing the root, the function must be differentiable. This indicates that the function's derivative is present and continuous within this range.
   - **Non-zero Derivative**: The derivative of the function at the initial guess must be non-zero. Otherwise, the method will not converge and may even diverge.
   - **Single Root**: The method can only find one root at a time. If the function has multiple roots, the method needs to be applied separately for each root.These assumptions are important to keep in mind when applying the Newton's method to find the root of a function. Violating any of these assumptions may result in inaccurate or incorrect results.

Problem 1

2.1 Outline of solution

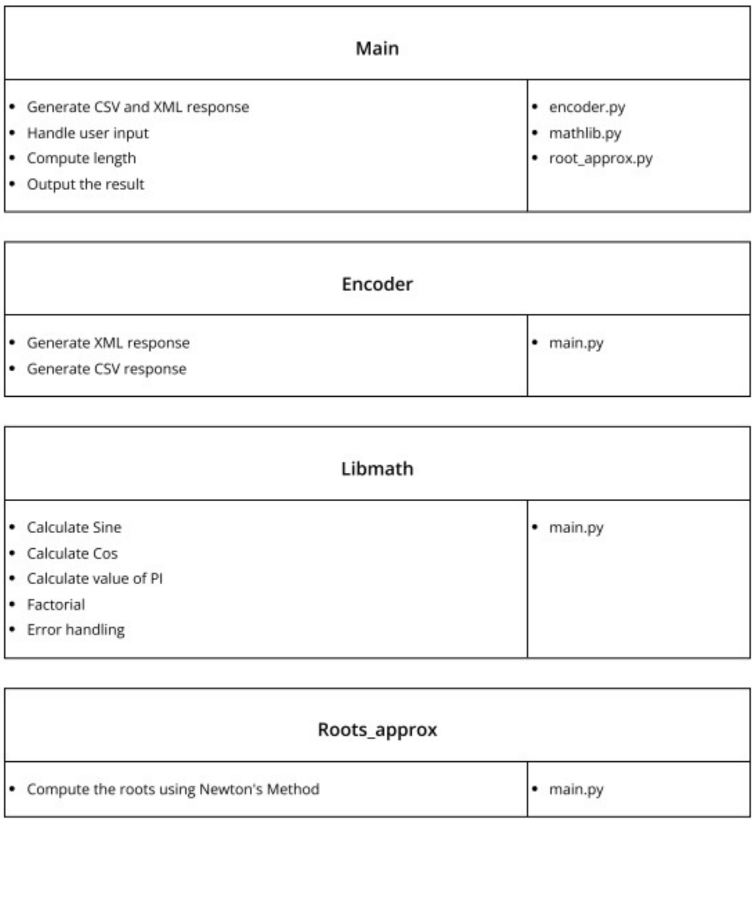
The problem asks to find the length of the segment X1X2 in terms of R, where X1 and $X2$ are two points on a circle of radius R such that the angle α subtended by the chord X1X2 at the center of the circle satisfies the equation

$$\alpha - sin(\alpha) = \frac{\pi}{2}$$

.

To solve the problem, we first need to find the value of α that satisfies the equation. Unfortunately, there is no algebraic expression for α in terms of elementary functions, so we have to use numerical methods to approximate its value. One common method is the Newton-Raphson method, which involves starting with an initial guess for α and then iteratively improving the guess until it converges into a solution.

2.2 CRC Model



Main	
• Generate CSV and XML response • Handle user input • Compute length • Output the result	• encoder.py • mathlib.py • root_approx.py

Encoder	
• Generate XML response • Generate CSV response	• main.py

Libmath	
• Calculate Sine • Calculate Cos • Calculate value of PI • Factorial • Error handling	• main.py

Roots_approx	
• Compute the roots using Newton's Method	• main.py

Problem 2

3.1 sine [1][2]

There are several methods for computing the sine of an angle which are as follows:

- Taylor Series
- CORDIC
- Chebyshev Approximation
- Lookup Tables
- Binary Splitting

The choice of method depends on accuracy, speed, and memory usage requirements. The Taylor series[9] method is a powerful and accurate approach for computing the sine function, especially for small values of the input angle. While the method becomes computationally expensive for larger values of the angle, it remains a viable choice for most applications due to its high accuracy and ease of implementation. Compared to other methods such as CORDIC and lookup table, the Taylor series approach provides a good balance between accuracy and speed. CORDIC is efficient for hardware implementations but less accurate than the Taylor series, while the lookup table is fast but requires significant memory resources to store precomputed values. Ultimately, the choice of method depends on the specific requirements of the application, but the Taylor series method is a reliable and accurate choice for computing sine.

Algorithm 1 Taylor series expansion for $\sin(x)$

```
Require: The value of $x$ and the number of terms $n$ to use in the series
Ensure: The approximate value of $\sin(x)$
1: $result \leftarrow 0$
2: for $i \leftarrow 0$ to $n-1$ do
3:     $result \leftarrow result + \frac{(-1)^i \times x^{2i+1}}{(2i+1)!}$
4: end for
5: return $result$
```

3.2 cosine [3][4][5]

There are several methods to compute the cosine function, which are as follows:

- Taylor Series
- CORDIC
- Lookup Tables
- Chebyshev Approximation

The Taylor series method is considered as one of the most efficient methods for computing cosine, as it balances accuracy and computational efficiency. It uses a series expansion of the cosine function around a specific point, typically zero, to approximate the value of cosine at any point using a finite number of terms in the series. This method is highly useful for small angles where the series converges quickly and provides highly accurate results. However, for larger angles, the number of terms required in the series increases, making it less efficient than other methods such as CORDIC or lookup tables. Nonetheless, it is easy to implement and requires only basic arithmetic operations, making it a popular choice for software-based implementations of cosine.

Algorithm 2 Taylor series expansion for $\cos(x)$

```
Require: The value of $x$ and the number of terms $n$ to use in the series
Ensure: The approximate value of $\cos(x)$
1: $result \leftarrow 0$
2: for $i \leftarrow 0$ to $n-1$ do
3:     $result \leftarrow result + \frac{(-1)^i \times x^{2i}}{(2i)!}$
4: end for
5: return $result$
```

3.3 Pi [6][7][8]

There are several methods to compute the value of pi, which are as follows:

- The Leibniz formula
- The Monte Carlo method
- The Bailey-Borwein-Plouffe (BBP) formula
- The Machin formula
- The Chudnovsky algorithm

The Leibniz formula[10] and other series-based methods, such as the Machin formula, rely on mathematical series and expansions to approximate pi. The Leibniz formula is advantageous due to its simplicity and ease of implementation using basic arithmetic operations. This method also converges quickly and provides accurate approximations with a relatively small number of terms in the series. However, when high precision or many digits is required, more efficient methods like the Chudnovsky algorithm may be preferable. Ultimately, the choice of method depends on factors such as the application's specific requirements, available computational resources, and desired level of precision.

Algorithm 3 Leibniz formula for π

```
Require: The number of terms $n_{terms}$ to use in the series
Ensure: The approximate value of $\pi$
1: $val \leftarrow 0$
2: $sign \leftarrow 1$
3: for $i \leftarrow 0$ to $n_{terms} - 1$ do
4:     $val \leftarrow val + \frac{sign}{2i+1}$
5:     $sign \leftarrow -sign$
6: end for
7: return $val \times 4$
```

3.4 Alpha

Newton's method[11] and the bisection method are both numerical techniques used to find the roots of equations. The bisection method is a reliable and straightforward technique that involves repeatedly dividing an interval in half until a root is found. It guarantees convergence to a root, but convergence can be slow, and the function being evaluated must change sign over the interval. Newton's method is a more powerful approach that uses the derivative of the function to iteratively refine an initial guess for the root. It generally converges faster than the bisection method but may fail to converge under certain circumstances. The optimal method for a specific problem depends on the functional properties and the problem requirements.

Algorithm 4 Newton's Method

```
Require: A function $f(x)$, its derivative $f'(x)$, an initial guess $x_0$, and a tolerance $\epsilon$
Ensure: An approximation to the root of $f(x)$
1: $x \leftarrow x_0$
2: while $|f(x)| > \epsilon$ do
3:     $x \leftarrow x - \frac{f(x)}{f'(x)}$
4: end while
5: return $x$
```

3.5 L

Calculate the value of **l**, which is the length of an arc of a circle with a given radius and subtended by an angle a (computed using the "compute_alpha" function).

The formula for **l** is

$$l = 2R(1 - cos(\frac{\alpha}{2})),$$

where R is the radius and α is the angle in radians. The code prompts the user to input the value of radius and then computes **l** using the previously computed value of a and the user-provided radius.

With python math library function: First defines a function func which represents the equation

$$a - sin(a) = \frac{\pi}{2},$$

where a is the unknown angle. The f_derivative function computes the derivative of func with respect to a. Then, the compute_alpha function uses the Newton-Raphson method to find the solution to the equation

$$a - sin(a) = \frac{\pi}{2}$$

given an initial guess.

After computing a, the code prompts the user to enter the radius of the circle. It then uses the formula

$$l = 2R(1 - cos(\frac{\alpha}{2})),$$

to compute the length **l** of the circular arc segment and prints the result.

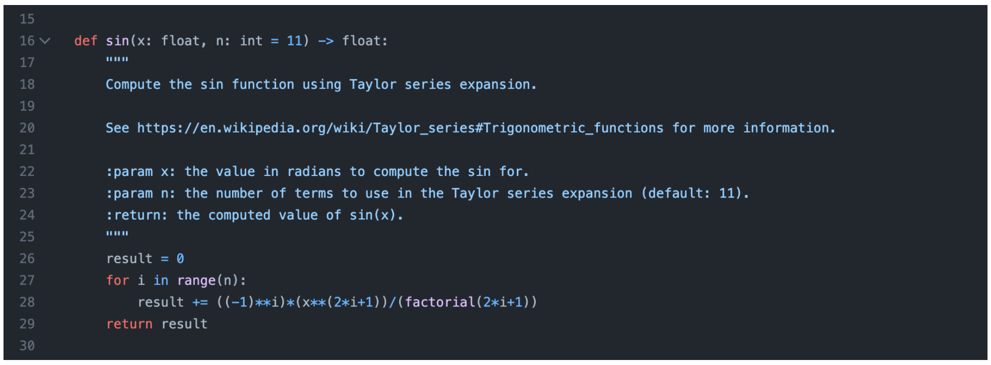
Problem 3

4.1 Incarnation 1

Steps taken towards making Program general, robust, usable, modifiable, readable, reusable, testable and understandable in source code:.

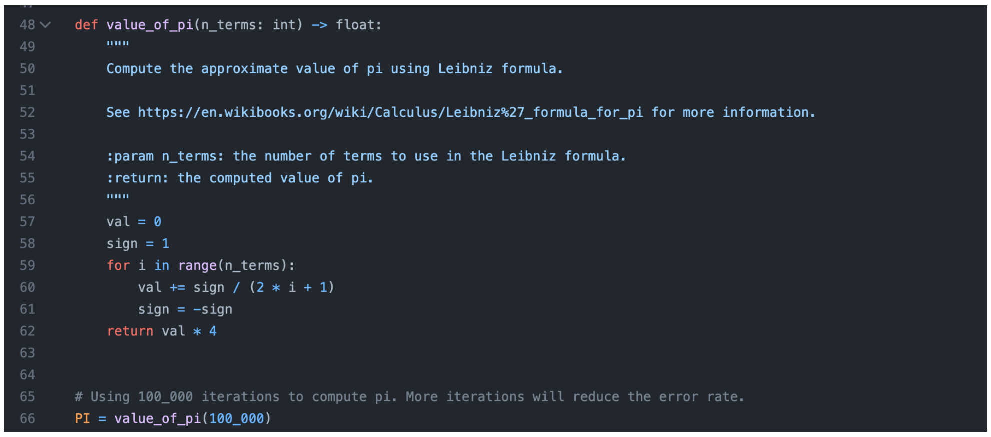
- **General**: Our program includes numerous mathematical functions that are widely used such as Sine, Cosine, PI and factorial. Wide variety of scenarios and applications can make use of these functions.
- **Robust**: As the program consists of error handling mechanisms to ensure the inputs from users are valid. For example, the newton_method function raises a ValueError if the initial guess is not a numeric value or if the desired accuracy is not a positive numeric value. Additionally, the newton_method function checks for divide by zero errors before performing each iteration of Newton's method.
- **Usable**: A very detailed explanation is included for each function such as parameters, return values and mathematical operations. Also, it uses PEP 8 guiding style, which makes the code more readable and easy to understand.
- **Modifiability**: By making the number of terms in the Taylor series for the sine and cosine functions a variable that can be supplied as an argument to the functions, the code enables modifiability. Similar to this, pi's Leibniz formula iteration count can also be supplied as an input. This makes it simple to adjust the results' precision based on the desired number of terms or iterations.
- **Readability**: The code is highly legible due to the use of descriptive function names and comments detailing the purpose and rationale of each function. Also, the code complies with PEP 8's style guidelines for Python, making it simple to read and comprehend for other programmers.
- **Reusability**: Just importing the file or copying the functions into the new program will allow you to reuse the sine, cosine, and pi calculation functions in future applications. As the code is organized into modules and each function completes a certain purpose, it is simple to reuse particular functions as required.
- **Testability**: As each function completes a particular purpose and can be checked independently, the code is very testable. To ensure that the output is accurate, the functions can be checked using various inputs. The code also has comments that describe the desired result and the reasoning behind how it was calculated.
- **Understandability**: Due to the use of descriptive function names, comments, and adherence to the PEP 8 style guide, the code is quite comprehensible. The reasoning used to calculate the outcomes of each function is clearly explained by the way the code is written. The mathematical formulas used in the functions are also referenced in the code, making it simpler to comprehend the underlying mathematical ideas.

Manually computed sin, cos and pi values



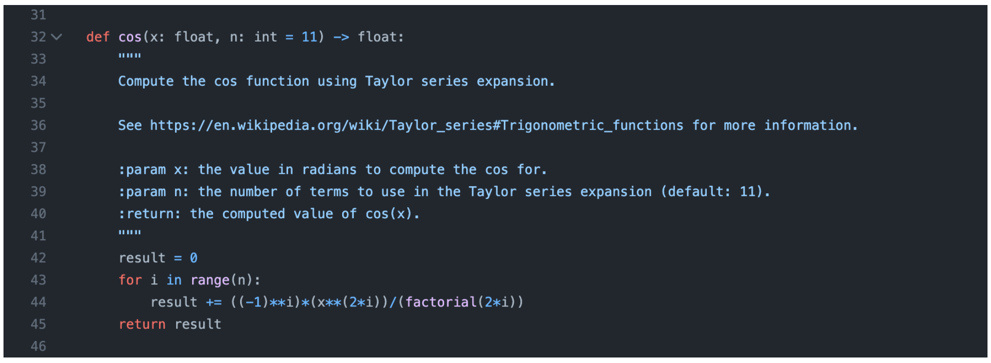
```python

def sin(x: float, n: int = 11) -> float:
    """
    Compute the sin function using Taylor series expansion.

    See https://en.wikipedia.org/wiki/Taylor_series#Trigonometric_functions for more information.

    :param x: the value in radians to compute the sin for.
    :param n: the number of terms to use in the Taylor series expansion (default: 11).
    :return: the computed value of sin(x).
    """
    result = 0
    for i in range(n):
        result += ((-1)**i)*(x**(2*i+1))/(factorial(2*i+1))
    return result

```

Figure 4.1: Computed the value of **sine** using Taylor series



```python
def value_of_pi(n_terms: int) -> float:
    """
    Compute the approximate value of pi using Leibniz formula.

    See https://en.wikibooks.org/wiki/Calculus/Leibniz%27_formula_for_pi for more information.

    :param n_terms: the number of terms to use in the Leibniz formula.
    :return: the computed value of pi.
    """
    val = 0
    sign = 1
    for i in range(n_terms):
        val += sign / (2 * i + 1)
        sign = -sign
    return val * 4


# Using 100_000 iterations to compute pi. More iterations will reduce the error rate.
PI = value_of_pi(100_000)
```

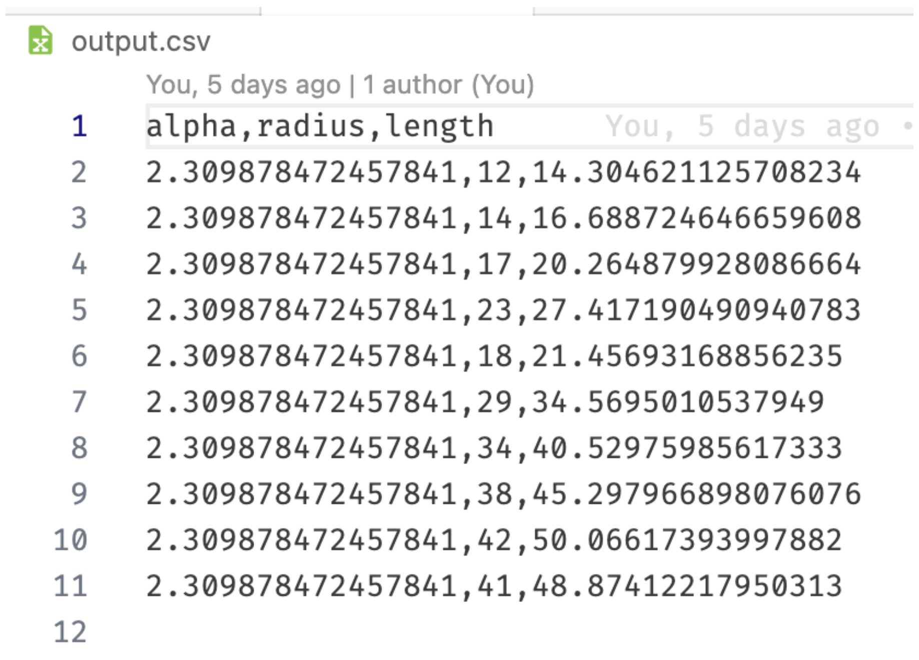
Figure 4.2: Computed the value of **pi** using Taylor series



```python

def cos(x: float, n: int = 11) -> float:
    """
    Compute the cos function using Taylor series expansion.

    See https://en.wikipedia.org/wiki/Taylor_series#Trigonometric_functions for more information.

    :param x: the value in radians to compute the cos for.
    :param n: the number of terms to use in the Taylor series expansion (default: 11).
    :return: the computed value of cos(x).
    """
    result = 0
    for i in range(n):
        result += ((-1)**i)*(x**(2*i))/(factorial(2*i))
    return result

```

Figure 4.3: Computed the value of **cos** using Taylor series

Sample output for different values of R


output.csv

You, 5 days ago | 1 author (You)

```
alphas,radius,length          You, 5 days ago •
2.309878472457841,12,14.304621125708234
2.309878472457841,14,16.688724646659608
2.309878472457841,17,20.264879928086664
2.309878472457841,23,27.417190490940783
2.309878472457841,18,21.45693168856235
2.309878472457841,29,34.5695010537949
2.309878472457841,34,40.52975985617333
2.309878472457841,38,45.297966898076076
2.309878472457841,42,50.06617393997882
2.309878472457841,41,48.87412217950313
```

Figure 4.4: Sample output for different values of **R**

4.2 Incarnation 2

We have created 2 python files to execute our code. One works with all the inbuilt libraries and the other one with the custom-made libraries.

The libraries we have used are: $Math, Sys, CSV, IO, TypingandXml.dom.minidom$

- **Math Library**: In order to verify our computed mathematical values with the original one we are using the math library.
- **Sys**: It is used for taking user inputs from keyboard.
- **CSV**: This function is used to extract the length of segment values for various R values and represent it in a CSV file.
- **IO**: string representing the CSV data.
- **Typing**: From this module we import Tuple ,list so that for every R values we update the list.
- **Xml.dom.minidom**: This module is used for generating XML file for various values of R.

Steps taken towards making S to be readable, modifiable, testable, and understandable after using native libraries:

Readability: We have used clear and descriptive variable names, so that anyone reading the code can quickly understand their purpose.

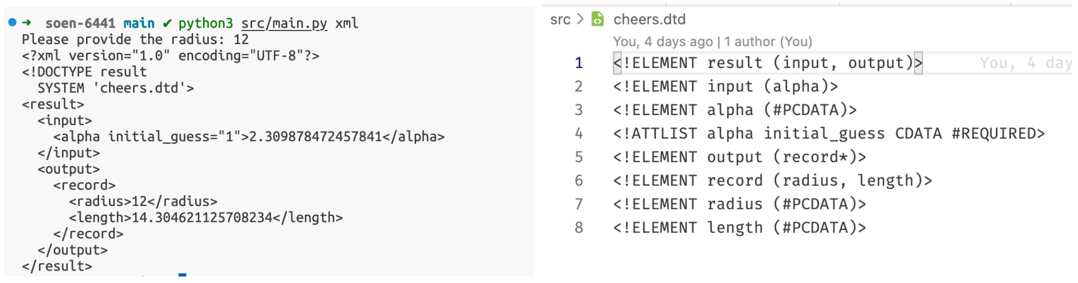
Examples: *compute_alpha* – is a method to compute the value of alpha. *newton_method* - Returns The calculated root value of ‘func‘ using Newton’s method.

Modifiability: We have broke down our code into smaller parts i.e. dividing the entire code into smaller methods and re using it which allows us to understand and modify the code easily.

Understandable: We have added comments for every piece of code we wrote so that if anyone wants make changes to the code they can understand the functionality from comments. Example : The usage of comments in one of our method illustrated below. We have tested various test cases by providing different r values and examined the results obtained.

Steps can be taken to ensure that the relevant P is designed to be comprehensive, robust, and applicable:

Generalization: A method to facilitate adaptation of code based on specific requirements, so less per module A modular system with arguments is to be used. Thus makes it easy to update or design the code according to the user specifications.
Robustness: To enhance our code robustness, we handled every possible exceptions and errors using python exception handling mechanism. This ensures that the code is sufficiently robust and generates informative error messages to users when unsupported arguments are passed.
Usability : To increase the usability of the code, we have made it more reusable due to its modular design, which makes it easy to reuse individual modules. Smaller modules make it easier to integrate code with other programs, thus improving its implementation.



```
● ➜  soen-6441 main ✔ python3 src/main.py xml
  Please provide the radius: 12
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE result
    SYSTEM 'cheers.dtd'>
  <result>
    <input>
      <alpha initial_guess="1">2.309878472457841</alpha>
    </input>
    <output>
      <record>
        <radius>12</radius>
        <length>14.304621125708234</length>
      </record>
    </output>
  </result>
```

src > cheers.dtd
You, 4 days ago | 1 author (You)

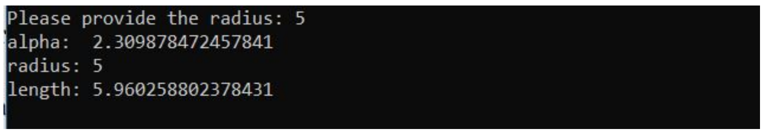
```
<!ELEMENT result (input, output)>        You, 4 day
<!ELEMENT input (alpha)>
<!ELEMENT alpha (#PCDATA)>
<!ATTLIST alpha initial_guess CDATA #REQUIRED>
<!ELEMENT output (record*)>
<!ELEMENT record (radius, length)>
<!ELEMENT radius (#PCDATA)>
<!ELEMENT length (#PCDATA)>
```

Figure 4.5: Sample XML output

Steps taken to Make our S IDE independent :

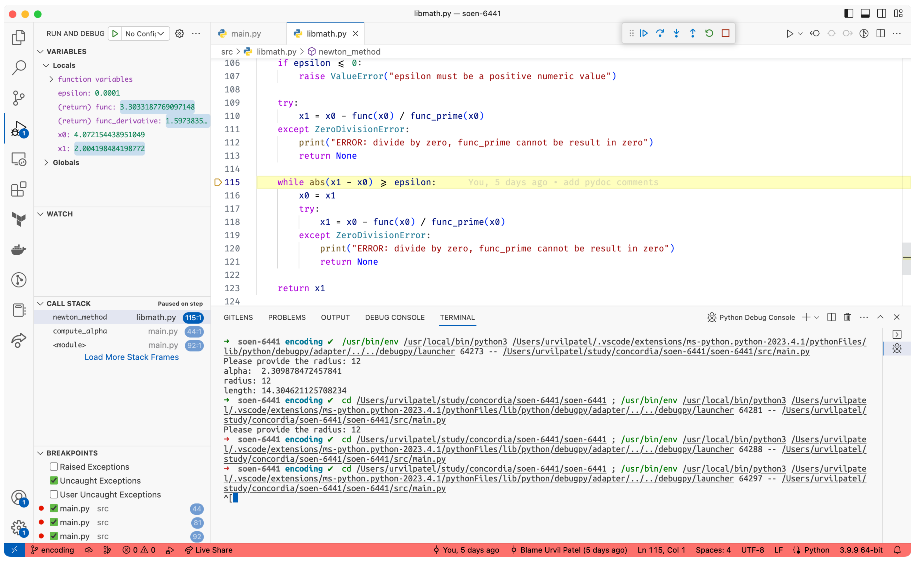
- We have used requirements.txt file , where we listed all the dependencies we have used for our project.
- We have avoided the usage of IDE-specific libraries , so that the libraries we use are pre installed along with python installation, which makes our code IDE independent.
- To manage our code and dependencies, we have used Git version control.
- This enables us to move our code between different environments easily.

- Additionally, we have used virtual environments to isolate our code and its dependencies from the system Python installation. This helps to prevent conflicts between different versions of libraries and ensures that our code runs consistently across different environments.



```
Please provide the radius: 5
alpha:  2.309878472457841
radius: 5
length: 5.960258802378431
```

Figure 4.6: Output in Command line



VARIABLES
Locals
function variables
epsilon: 0.0001
(return) func: 3.3033187769097148
(return) func_derivative: 1.5973835…
x0: 4.072154438951049
x1: 2.004198484198772
Globals

WATCH

CALL STACK — Paused on step
newton_method libmath.py 115:1
compute_alpha main.py 44:1
<module> main.py 92:1
Load More Stack Frames

BREAKPOINTS
☐ Raised Exceptions
☑ Uncaught Exceptions
☐ User Uncaught Exceptions
● ☑ main.py src 44
● ☑ main.py src 81
● ☑ main.py src 92

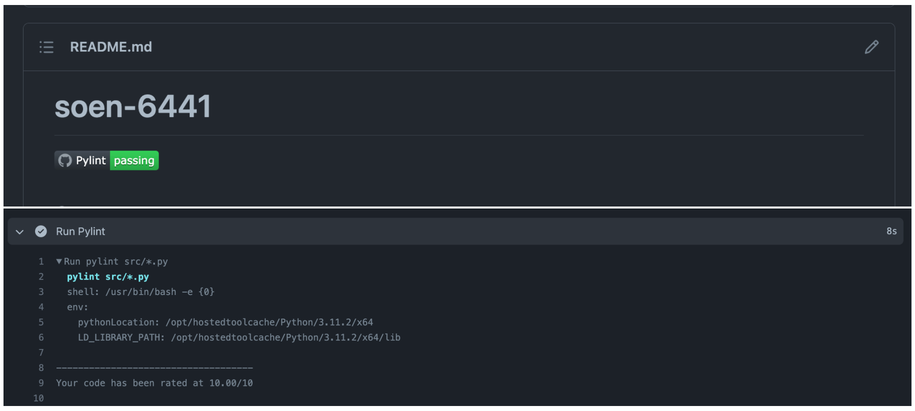
```python
    if epsilon <= 0:
        raise ValueError("epsilon must be a positive numeric value")

    try:
        x1 = x0 - func(x0) / func_prime(x0)
    except ZeroDivisionError:
        print("ERROR: divide by zero, func_prime cannot be result in zero")
        return None

    while abs(x1 - x0) >= epsilon:        You, 5 days ago • add pydoc comments
        x0 = x1
        try:
            x1 = x0 - func(x0) / func_prime(x0)
        except ZeroDivisionError:
            print("ERROR: divide by zero, func_prime cannot be result in zero")
            return None

    return x1
```

```
→ soen-6441 encoding ✓ /usr/bin/env /usr/local/bin/python3 /Users/urvilpatel/.vscode/extensions/ms-python.python-2023.4.1/pythonFiles/
lib/python/debugpy/adapter/../../debugpy/launcher 64273 -- /Users/urvilpatel/study/concordia/soen-6441/soen-6441/src/main.py
Please provide the radius: 12
alpha:  2.309878472457841
radius: 12
length: 14.304621125708234
→ soen-6441 encoding ✓ cd /Users/urvilpatel/study/concordia/soen-6441/soen-6441 ; /usr/bin/env /usr/local/bin/python3 /Users/urvilpate
l/.vscode/extensions/ms-python.python-2023.4.1/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 64281 -- /Users/urvilpatel/
study/concordia/soen-6441/soen-6441/src/main.py
Please provide the radius: 12
→ soen-6441 encoding ✓ cd /Users/urvilpatel/study/concordia/soen-6441/soen-6441 ; /usr/bin/env /usr/local/bin/python3 /Users/urvilpate
l/.vscode/extensions/ms-python.python-2023.4.1/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 64288 -- /Users/urvilpatel/
study/concordia/soen-6441/soen-6441/src/main.py
→ soen-6441 encoding ✓ cd /Users/urvilpatel/study/concordia/soen-6441/soen-6441 ; /usr/bin/env /usr/local/bin/python3 /Users/urvilpate
l/.vscode/extensions/ms-python.python-2023.4.1/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 64297 -- /Users/urvilpatel/
study/concordia/soen-6441/soen-6441/src/main.py
^[
```

encoding ⊗ 0 ⚠ 0 Live Share You, 5 days ago Blame Urvil Patel (5 days ago) Ln 115, Col 1 Spaces: 4 UTF-8 LF Python 3.9.9 64-bit

Figure 4.7: Usage of Debugger

- We have used Pylint for enforcing PEP 8 coding style and to check coding consistency and to improve coding quality.



README.md

soen-6441

Pylint passing

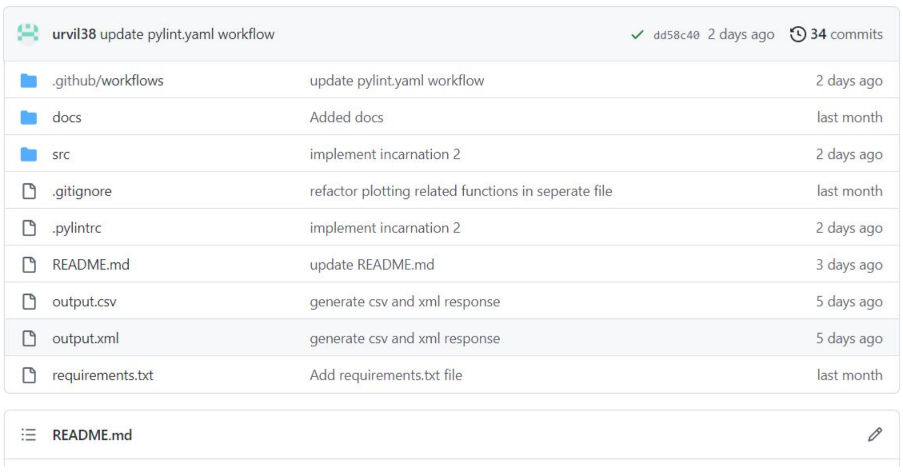
Run Pylint 8s

```
▼Run pylint src/*.py
  pylint src/*.py
  shell: /usr/bin/bash -e {0}
  env:
    pythonLocation: /opt/hostedtoolcache/Python/3.11.2/x64
    LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.11.2/x64/lib

------------------------------------
Your code has been rated at 10.00/10

```

Figure 4.8: Uses of Pylint and PEP8

- In our S we have used a method named compute _PI(no_of_iterations)- which computes the value of PI without using inbuilt libraries.
- Our S has been developed with proper exception and error handling mechanisms using various Error types available in python. We are using Python

debugger module PDB and Pydoc for documentation of the code.



urvil38 update pylint.yaml workflow ✓ dd58c40 2 days ago 34 commits

.github/workflows	update pylint.yaml workflow	2 days ago
docs	Added docs	last month
src	implement incarnation 2	2 days ago
.gitignore	refactor plotting related functions in seperate file	last month
.pylintrc	implement incarnation 2	2 days ago
README.md	update README.md	3 days ago
output.csv	generate csv and xml response	5 days ago
output.xml	generate csv and xml response	5 days ago
requirements.txt	Add requirements.txt file	last month

README.md

Figure 4.9: GitHub repository

All our Source code is listed in the link: https://github.com/urvil38/soen-6441

Description for Processing Source code: https://github.com/urvil38/soen-6441/blob/main/README.md

References

[1] Gene F. Franklin and David J. Powell. Computing the sine function using the taylor series. *Proceedings of the IEEE*, 62(6):1112–1113, 1974.

[2] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications.* Pearson, Upper Saddle River, NJ, 4th edition, 2007.

[3] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, New York, NY, 3rd edition, 2007.

[4] Milton Abramowitz and Irene Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* National Bureau of Standards, Washington, DC, 1964.

[5] S. A. M. Marcum. Evaluation of the sine and cosine integrals. *Journal of Research of the National Bureau of Standards*, 73C(1):11–18, 1969.

[6] Rosetta Code. Methods of computing pi. https://rosettacode.org/wiki/Methods_of_computing_pi.

[7] Eric W. Weisstein. Pi formulas. https://mathworld.wolfram.com/PiFormulas.html.

[8] Wikipedia. Computing pi. https://en.wikipedia.org/wiki/Computing_%CF%80.

[9] Wikipedia. Taylor series. https://en.wikipedia.org/wiki/Taylor_series.

[10] Proof Wiki. Leibniz's formula for pi. https://proofwiki.org/wiki/Leibniz's_Formula_for_Pi.

[11] Wikipedia. Newton's method. https://en.wikipedia.org/wiki/Newton%27s_method.