|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | 20 May 2022 | |
| NUMERICAL COMPUTING (CS325) | | | | |
| **Group Members:**   * Mohammad Basil Ali Khan (20K-0477) * Ali Jodat (20K-0155) * Abdul Ahad Shaikh (20K-0319) * Mohammad Umer (20K-0225) | | | | |
| Course Instructor:  Sir Jamil Usmani | |  |  |  |

**NUMERICAL COMPUTING (CS325)**

**PROJECT**

**LAB – 1**

**Group Members:**

* Mohammad Basil Ali Khan (20K-0477)
* Ali Jodat (20K-0155)
* Abdul Ahad Shaikh (20K-0319)
* Mohammad Umer (20K-0225)

**Project Title:**

**LAB 1: Solution of Non Linear Equation in one Variable f(x) = 0**

**Aim:**

To understand the fundamental concepts of scientific programming using python.

**Description:**

We selected three methods of Lab1.

1. Bisection Method
2. Regular Falsi Method
3. Secant Method

First we have studied the algorithm of then we have written the programming of that method.

**IDE and Programming Language:**

We have chosen python programming language and IDE we are using is Visual Studio Code.

**Library Used:**

We have imported 3 libraries:

1. sympy library for to get equation solution on particular intervals and can initialize symbols.
2. tabulate library to generated table on each iteration.
3. array library to save each iteration values to use in next iteration.

**Implementation and Code Snippets:**

* **Bisection Method:**

**Formula:**

**Algorithm:**

Step 1: Find two points, say a and b such that a < b and f(a)\* f(b) < 0

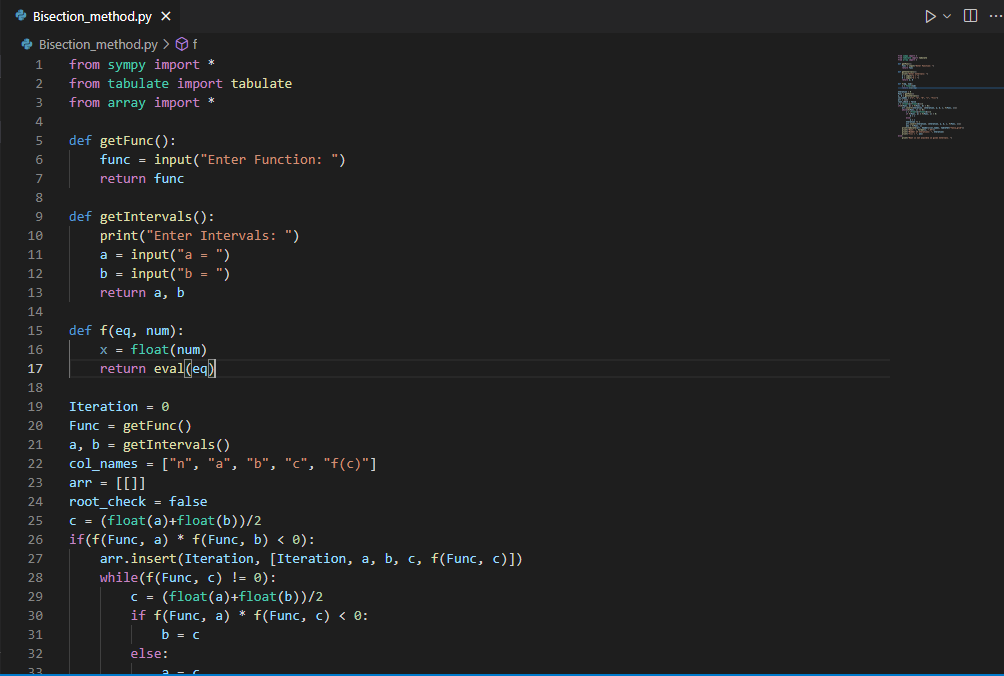
Step 2: Find the midpoint of a and b, say “c”

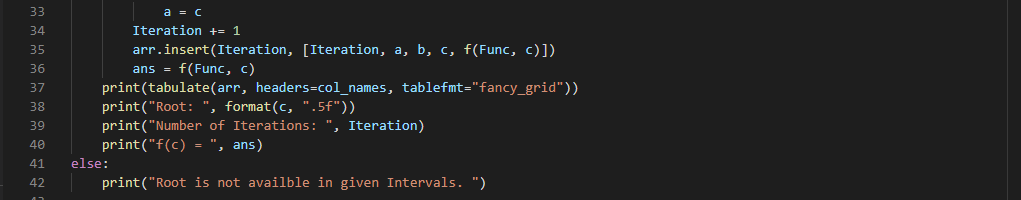
Step 3: c is the root of the given function if f(c) = 0; else follow the next step

Step 4: Divide the interval [a, b] – If f(c)\*f(a) <0, there exist a root between t and a  
 else if f(c) \*f (b) < 0, there exist a root between t and b

Step 5: Repeat above three steps until f(c) = 0.

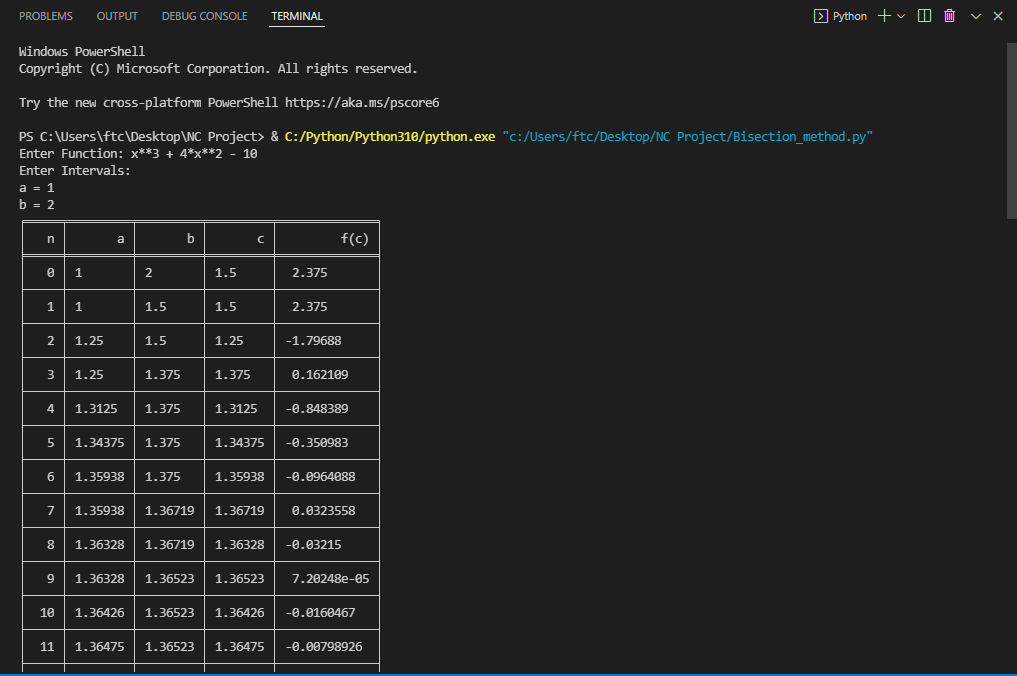
**Code Snippets:**

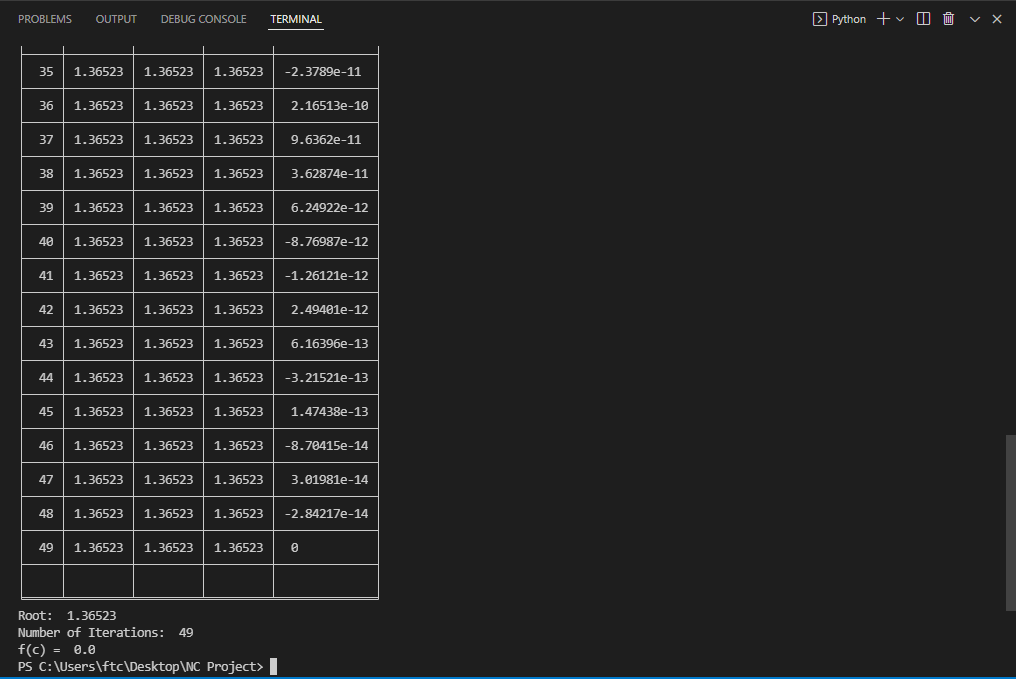




**Output:**

Input:





* **Secant Method:**

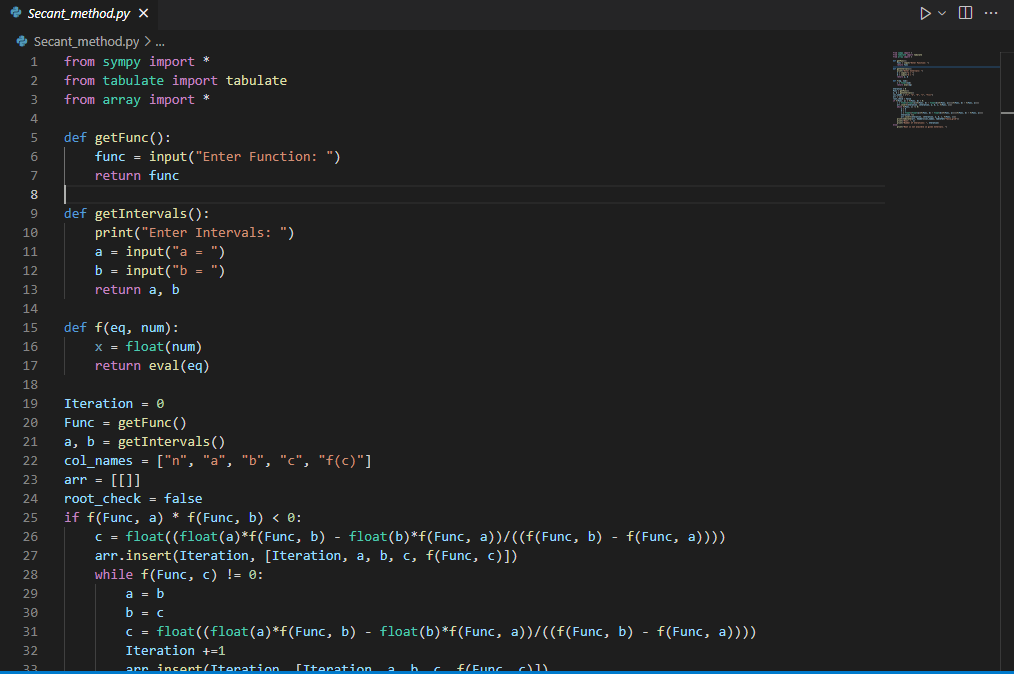
**Formula:**

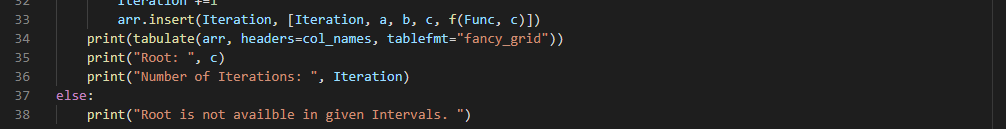
**Algorithm:**

Given an equation f(c) = 0   
Let the initial guesses be a and b  
Do

while (f(c) not equals 0)

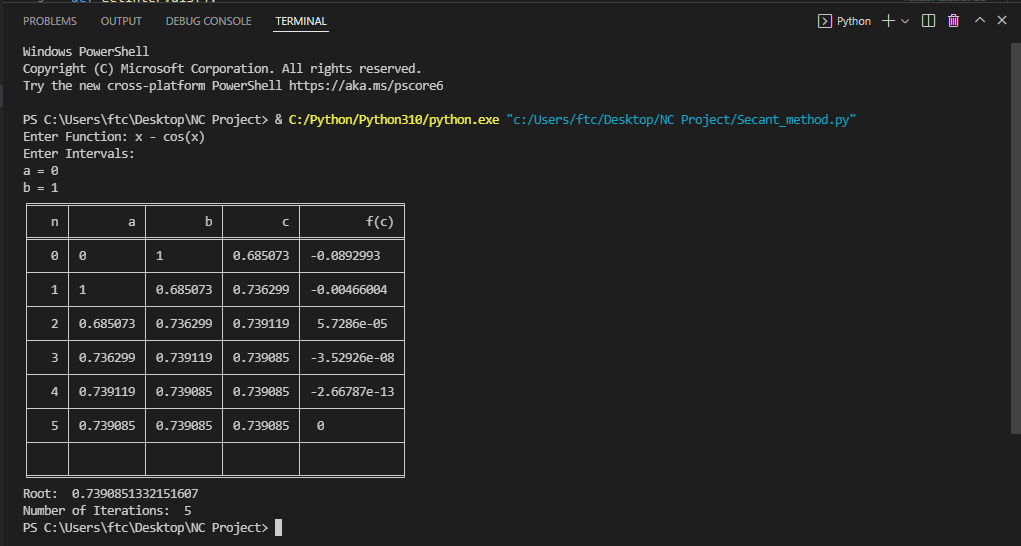
**Code Snippets:**





**Output:**

Input :



* **Regular Falsi Method:**

**Formula:**

**Algorithm:**

Step 1: Find two points, say a and b such that a < b and f(a)\* f(b) < 0

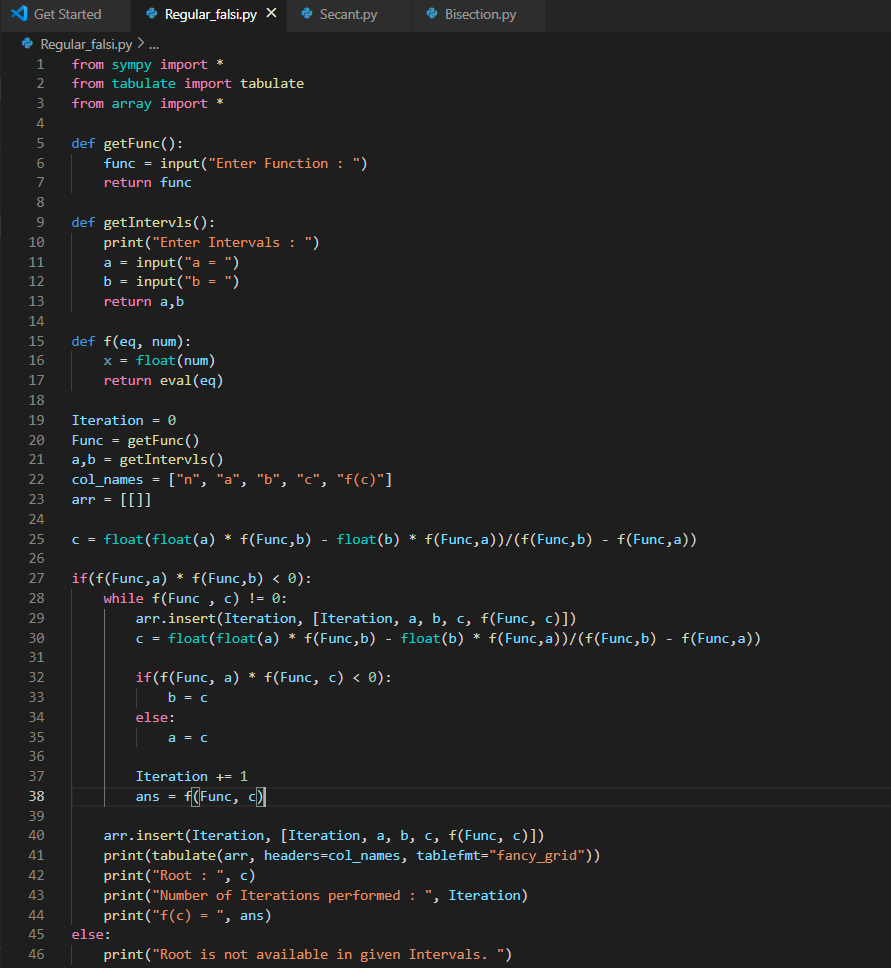
Step 2:

Step 3: c is the root of the given function if f(c) = 0; else follow the next step

Step 4: Divide the interval [a, b] – If f(c)\*f(a) <0, there exist a root between t and a  
 else if f(c) \*f (b) < 0, there exist a root between t and b

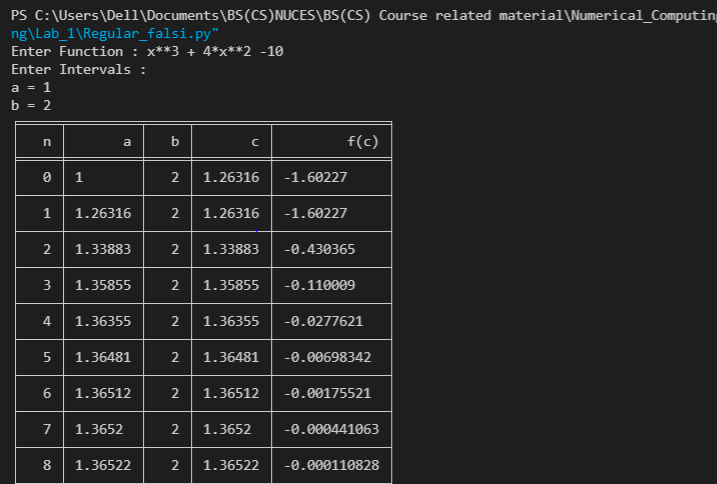
Step 5: Repeat above three steps until f(c) = 0.

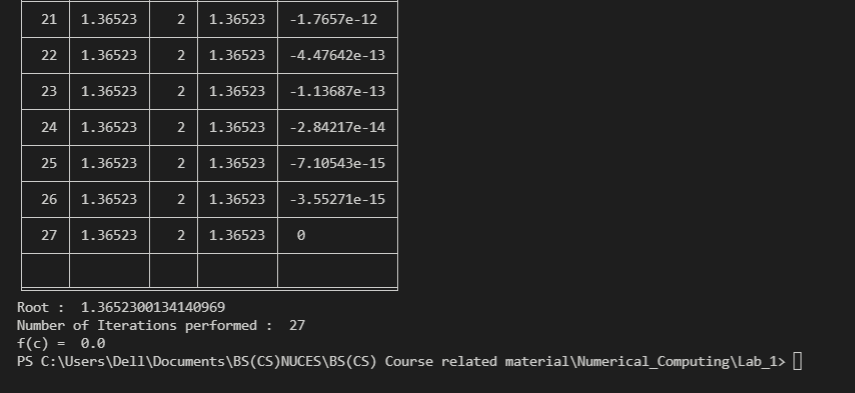
**Code Snippets:**



**Input :**

**Output:**





**NUMERICAL COMPUTING (CS325)**

**PROJECT**

**LAB – 2**

**Group Members:**

* Mohammad Basil Ali Khan (20K-0477)
* Ali Jodat (20K-0155)
* Abdul Ahad Shaikh (20K-0319)
* Mohammad Umer (20K-0225)

**Project Title:**

**LAB 2: Interpolation and Polynomial Approximation**

**Aim:**

To understand the fundamental concepts of scientific programming using python.

**Description:**

We selected three methods of Lab2.

1. Lag grange Interpolation
2. Newton Divided Difference
3. Newton Forward and Backward

First we have studied the algorithm of then we have written the programming of that method.

**IDE and Programming Language:**

We have chosen python programming language and IDE we are using is Visual Studio Code.

**Library Used:**

* Used panda library to make data frame

**Implementation and Code Snippets:**

* **Lag grange Interpolation:**

**Formula:**

Where can be written as;

**Algorithm:**

Step 1: Read number of data N.

Step 2: Read data Xi and Yi from I = 0 to I = N.

Step 3: Read value of independent variables say x whose corresponding value of dependent say y is to be determined.

Step 4: Initialize: y = 0

Step 5: For i = 0 to N

Set p = 1

For j =0 to N

If i ≠ j then

Calculate product = product \* (x - Xj)/(Xi - Xj)

End If

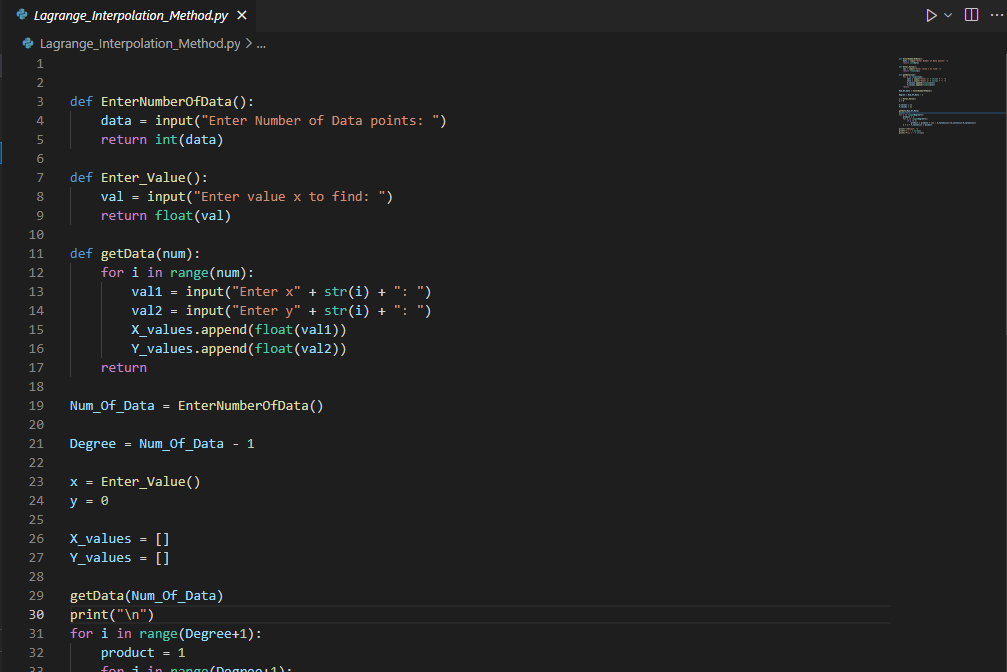
Next j

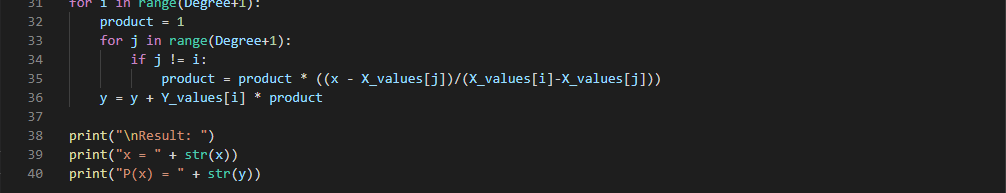
Calculate y = y + product \* Yi

Next i

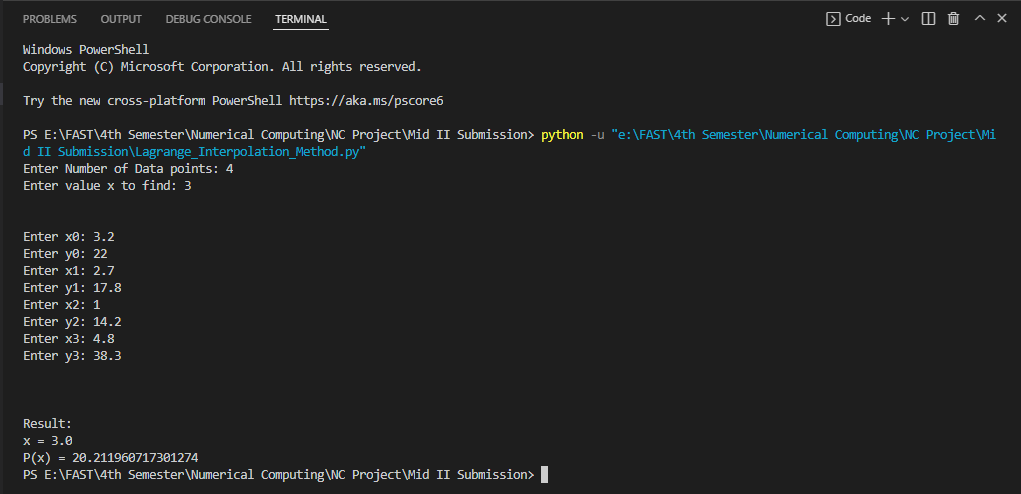
Step 6: Display value of y as interpolated value.

**Code Snippets:**





**Output:**



* **Newton Forward and Backward:**

**Formula:**

* **Forward**
* **Backward**

**Algorithm:**

* **Forward**

Step 1: Read number of data (n)

Step 2: Read data points for x and y:

For i = 0 to n-1

Read Xi and Yi,0

Next i

Step 3: Read calculation point where derivative is required (xp)

Step 4: Generate forward difference table

For i = 1 to n-1

For j = 0 to n-1-i

Yj,i = Yj+1,i-1 - Yj,i-1

Next j

Next i

Step 5: Calculate finite difference: h = X1 - X0

Step 6: Set sum = 0 and sign = 1

Step 7: Calculate sum of different terms in formula to find derivatives using Newton's forward difference formula:

For i = 1 to n-1-index

term = (Yindex, i)i / i

sum = sum + sign \* term

sign = -sign

Next i

Step 8: Divide sum by finite difference (h) to get result first\_derivative = sum/h

Step 9: Display value of first\_derivative

* **Backward**

Step 1: Read number of data (n)

Step 2: Read data points for x and y:

For i = 0 to n-1

Read Xi and Yi,0

Next i

Step 3: Read calculation point where derivative is required (xp)

Step 4: Generate backward difference table

For i = 1 to n-1

For j = n-1 to i (Step -1)

Yj,i = Yj,i-1 - Yj-1,i-1

Next j

Next i

Step 5: Calculate finite difference: h = X1 - X0

Step 6: Set sum = 0

Step 7: Calculate sum of different terms in formula to find derivatives using Newton's backward difference formula:

For i = 1 to index

term = (Yindex, i)i / i

sum = sum + term

Next i

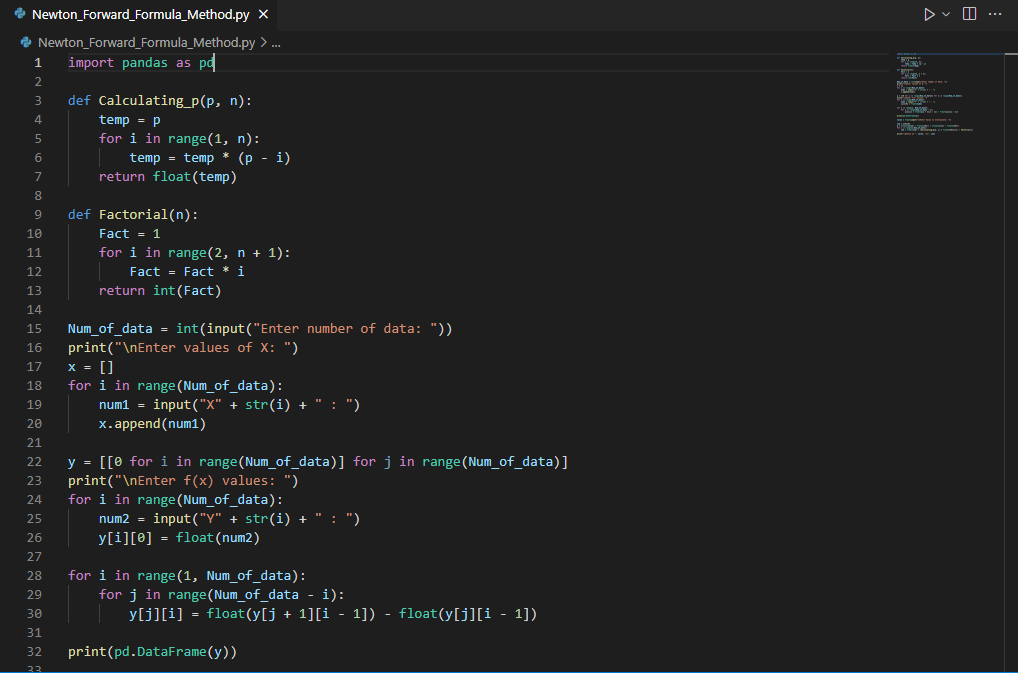
Step 8: Divide sum by finite difference (h) to get result

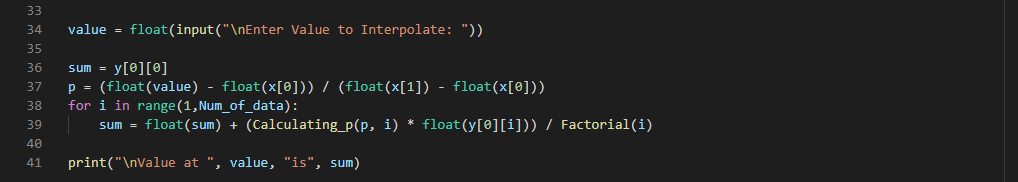
first\_derivative = sum/h

Step 9: Display value of first\_derivative

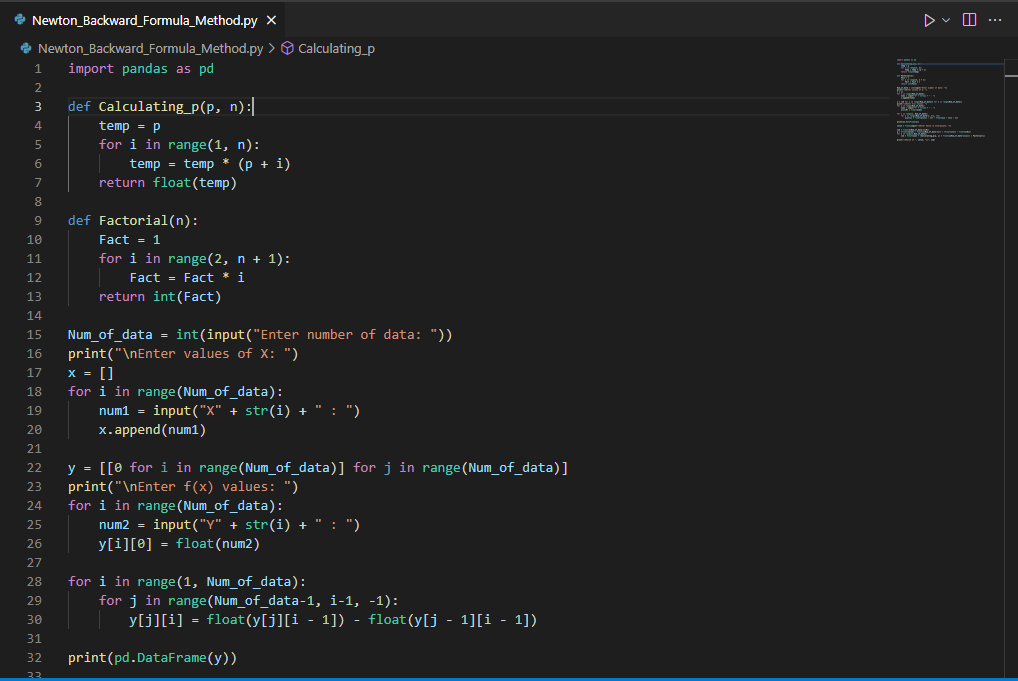
**Code Snippets:**

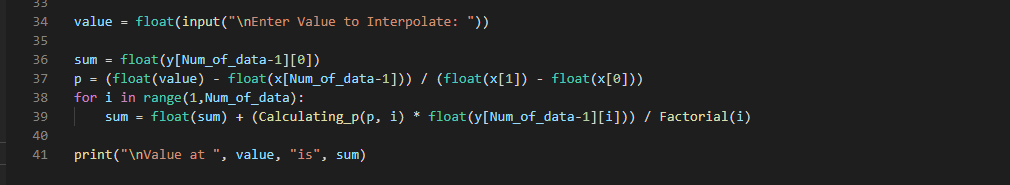
* **Forward**





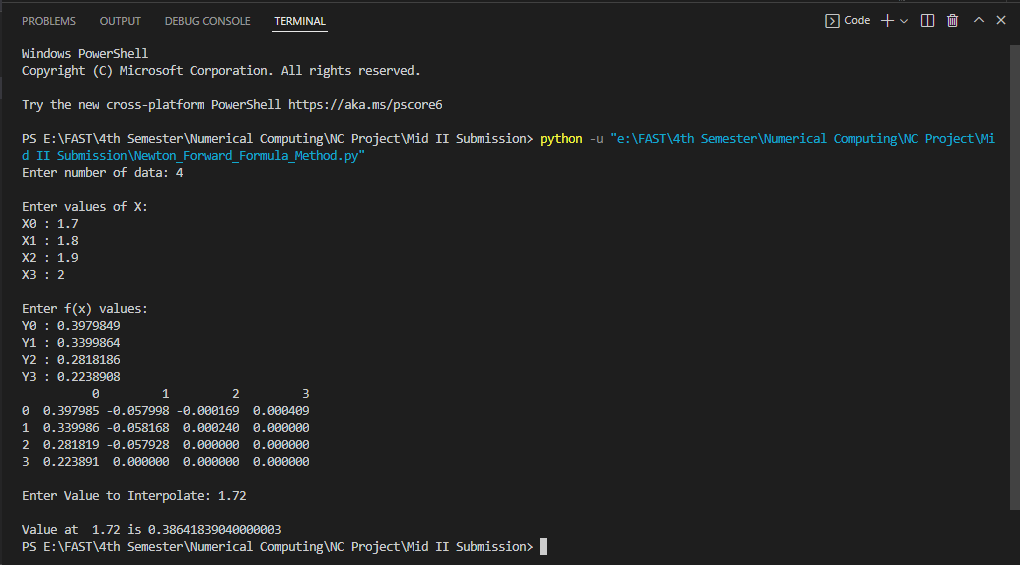
* **Backward**



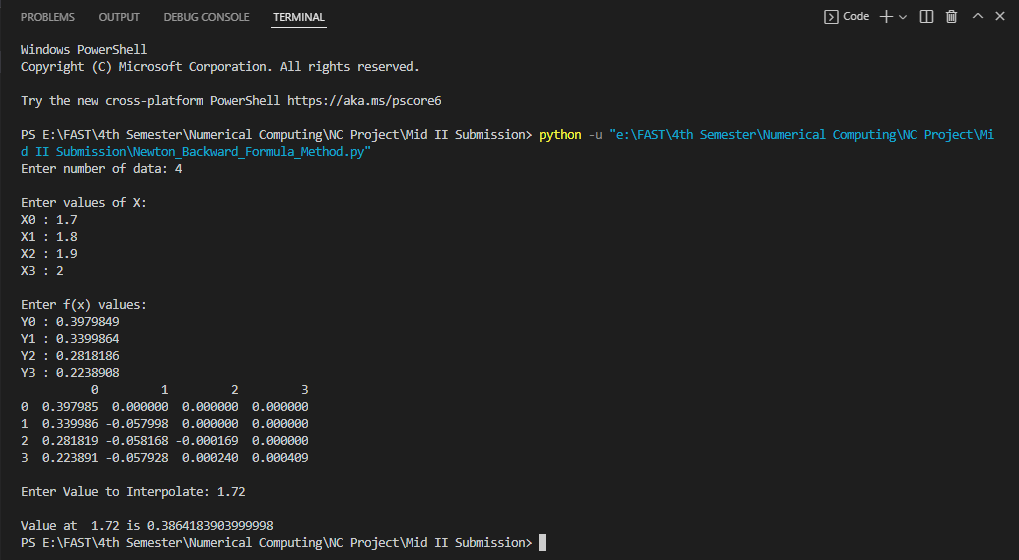


**Output:**

* **Forward**



* **Backward**



* **Newton Divided Difference:**

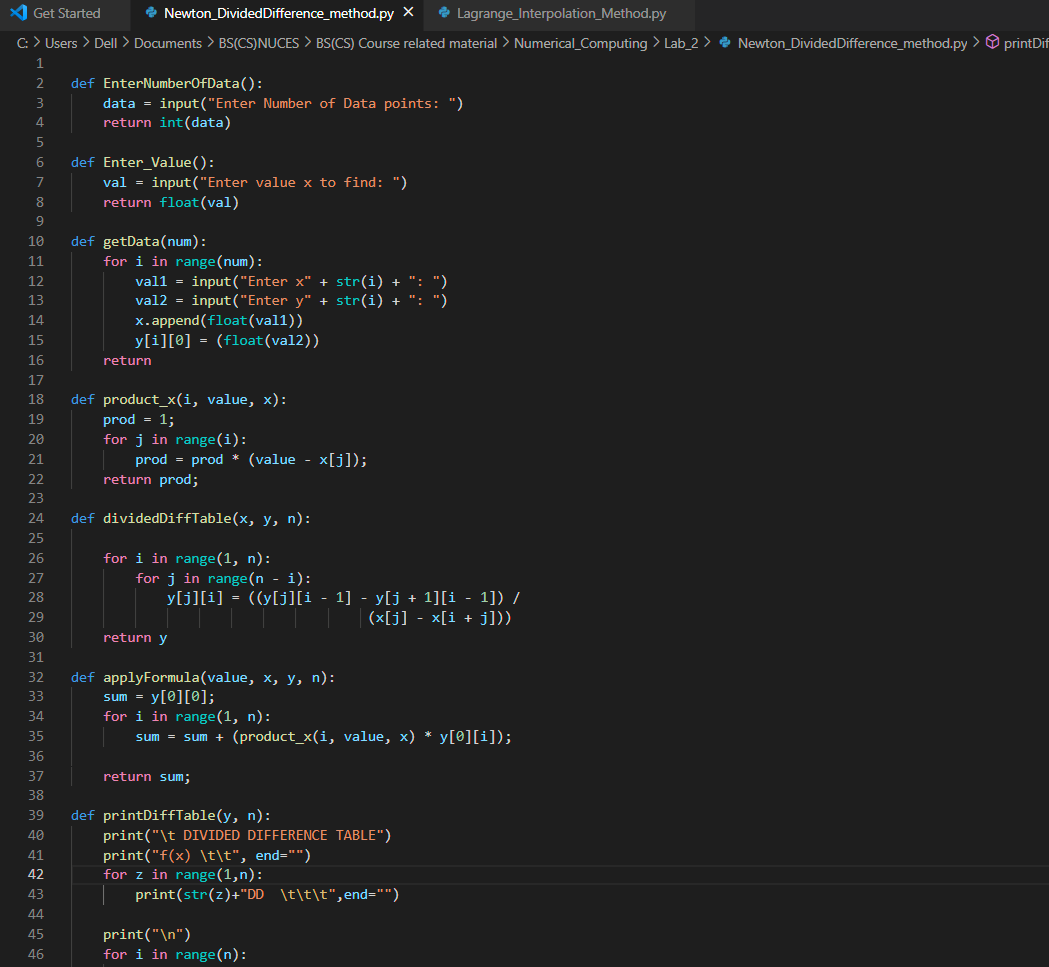
**Formula:**

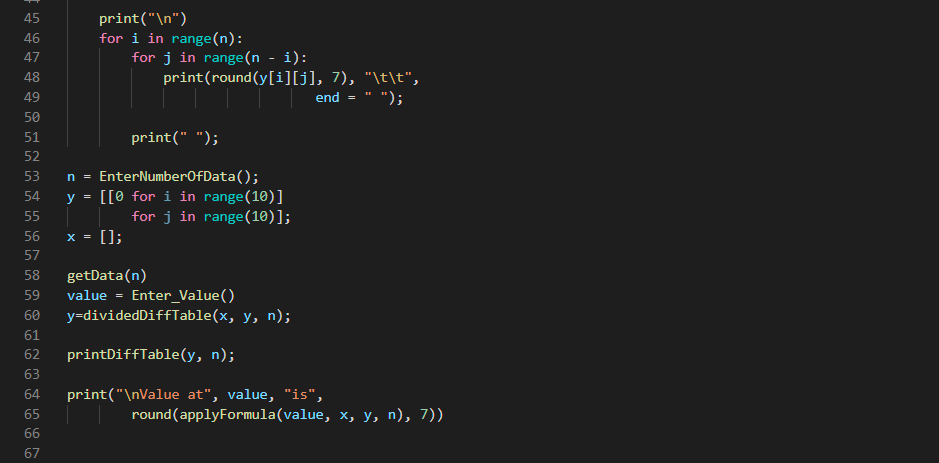
**f(x) = f [x0] + (x - x0) f [x0, x1] + (x - x0) (x - x1) f [x0, x1, x2]**   
 **+ . . . + (x - x0) (x - x1) . . . (x - xk-1) f [x0, x1, . . ., xk]**

Where

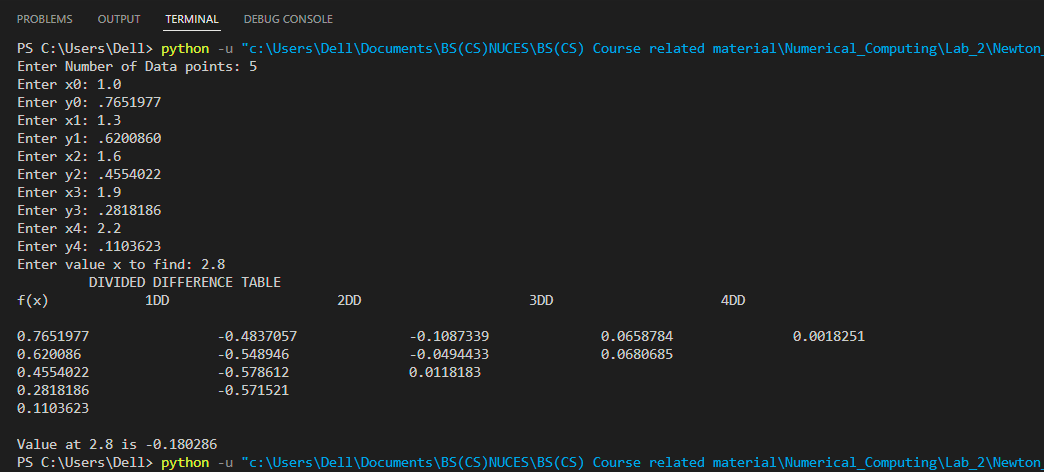
**f [x0, x1, . . ., xk]**  **= (f [x1, x2, . . ., xk] - f [x0, x1, . . ., xk-1]) / (xk - x0)**

**Code Snippets:**





**Output:**

****

**NUMERICAL COMPUTING (CS325)**

**PROJECT**

**LAB – 3**

**Group Members:**

* Mohammad Basil Ali Khan (20K-0477)
* Ali Jodat (20K-0155)
* Abdul Ahad Shaikh (20K-0319)
* Mohammad Umer (20K-0225)

**Project Title:**

**LAB 3: Numerical Integration**

**Aim:**

To understand the fundamental concepts of scientific programming using python.

**Description:**

We selected three methods of Lab1.

1. Newton Cotes CLOSED quadrature formula.
2. Newton Cotes OPEN quadrature method.
3. Composite Midpoint rule

First we have studied the algorithm of then we have written the programming of that method.

**IDE and Programming Language:**

We have chosen python programming language and IDE we are using is Visual Studio Code.

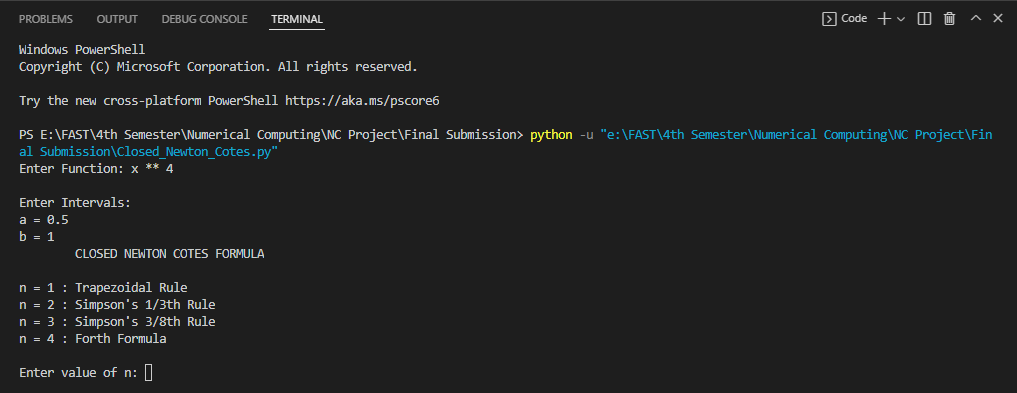
**Library Used:**

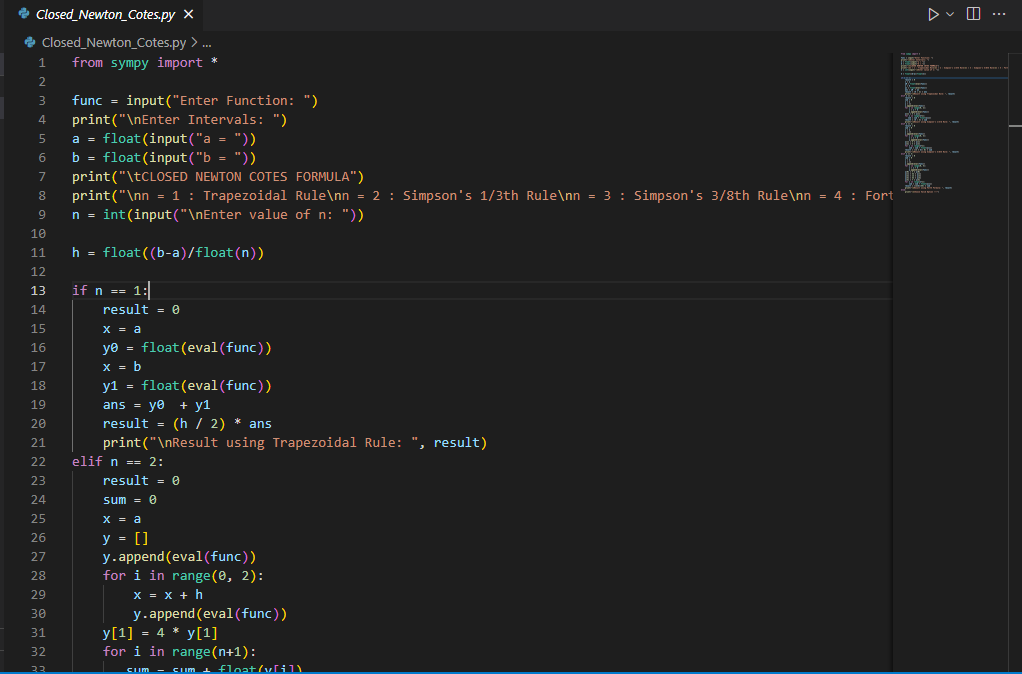
We have imported 3 libraries:

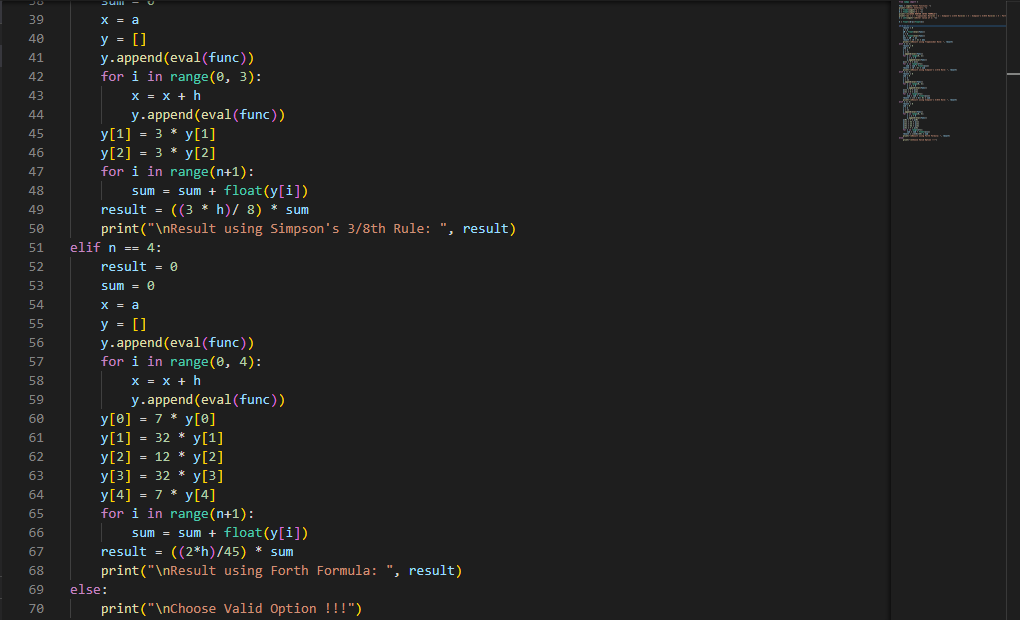
* sympy library for to get equation solution on particular intervals and can initialize symbols.

**Implementation and Code Snippets:**

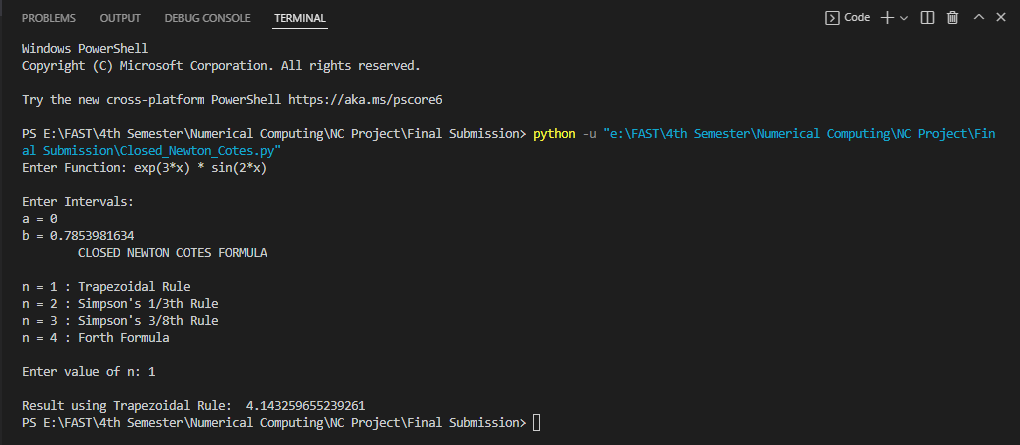
**CLOSED NEWTON COTES:**



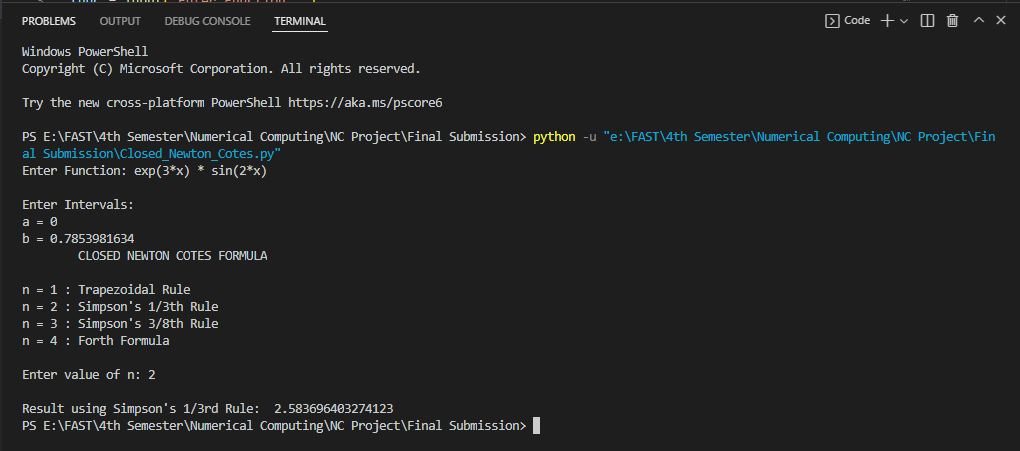




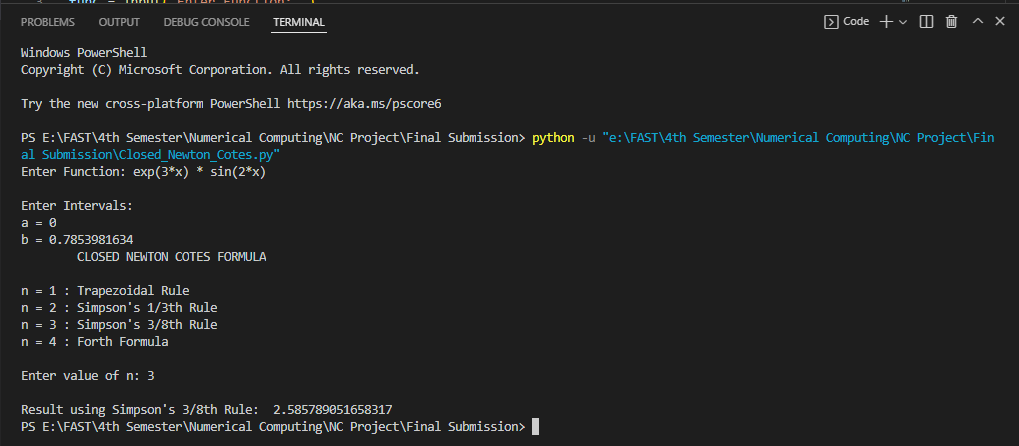
* **Trapezoidal Rule:**



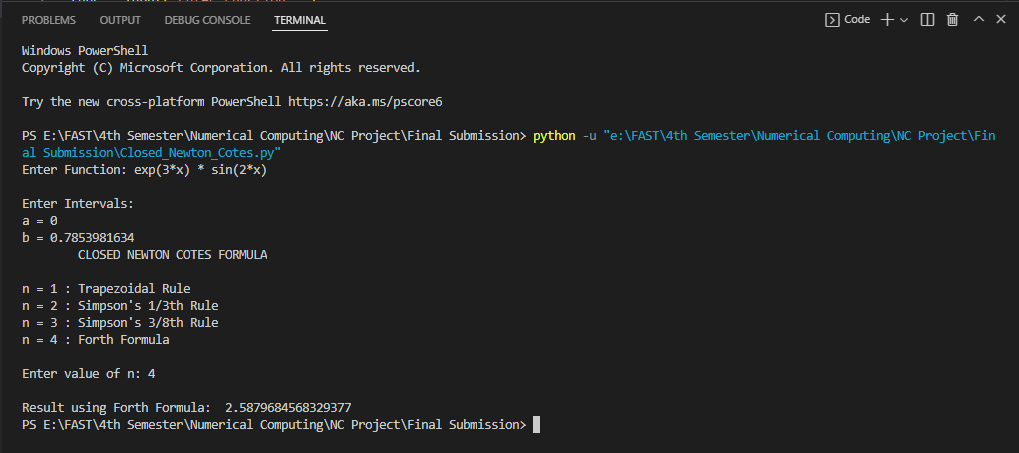
* **Simpsons 1/3rd Rule:**



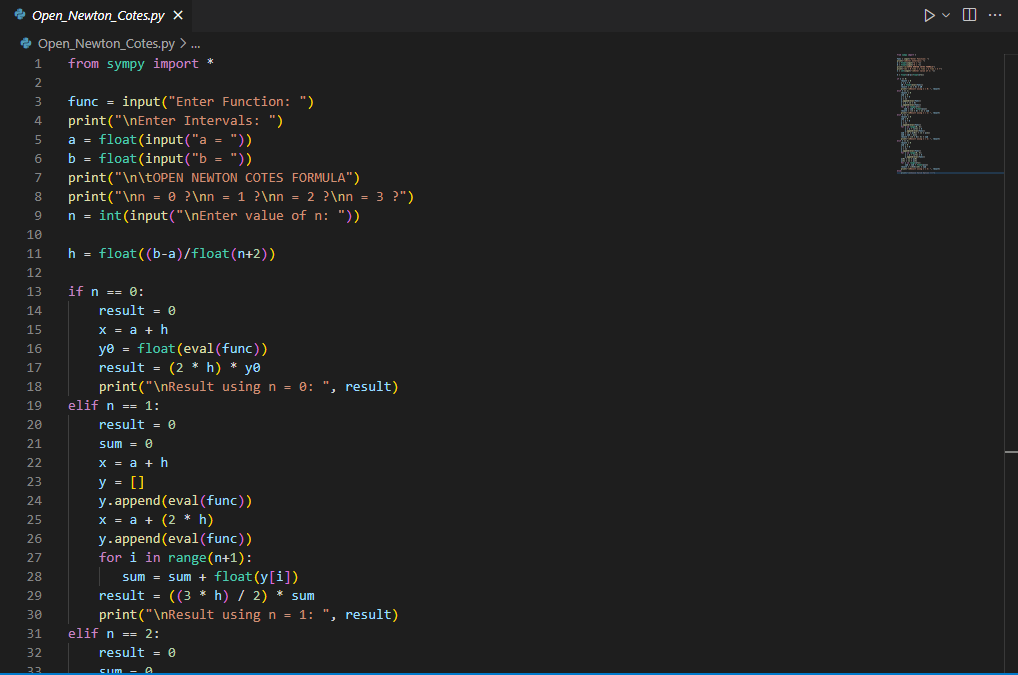
* **Simpsons 3/8th Rule:**

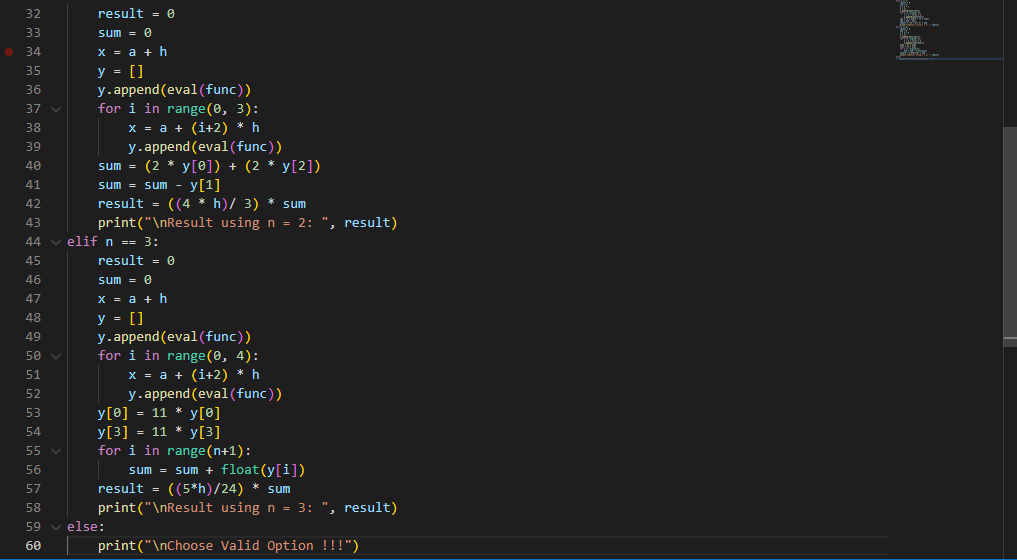


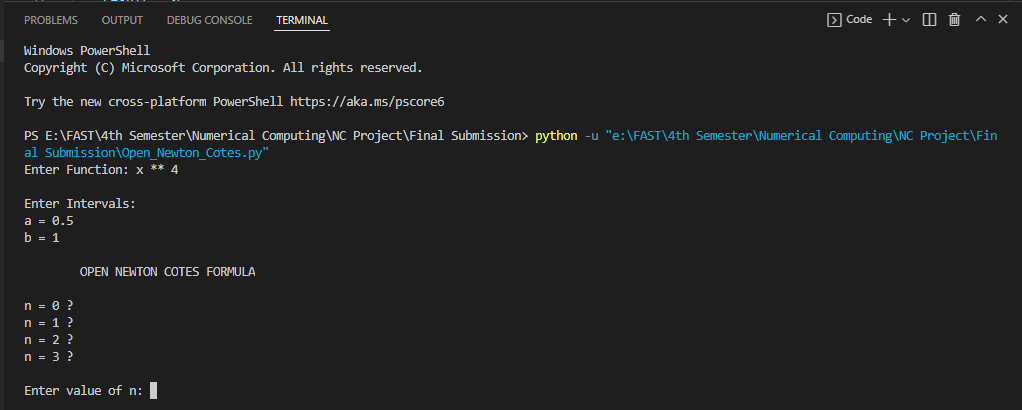
* **N = 4:**



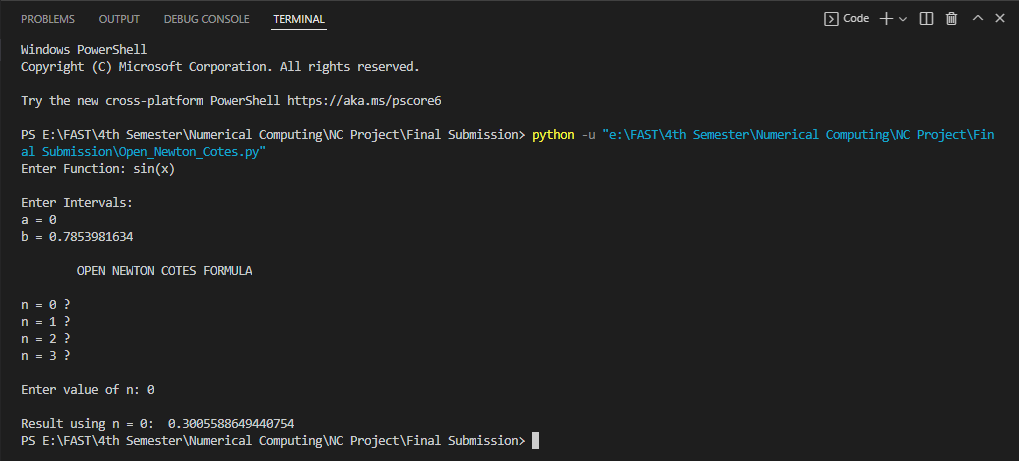
**OPEN NEWTON COTES**



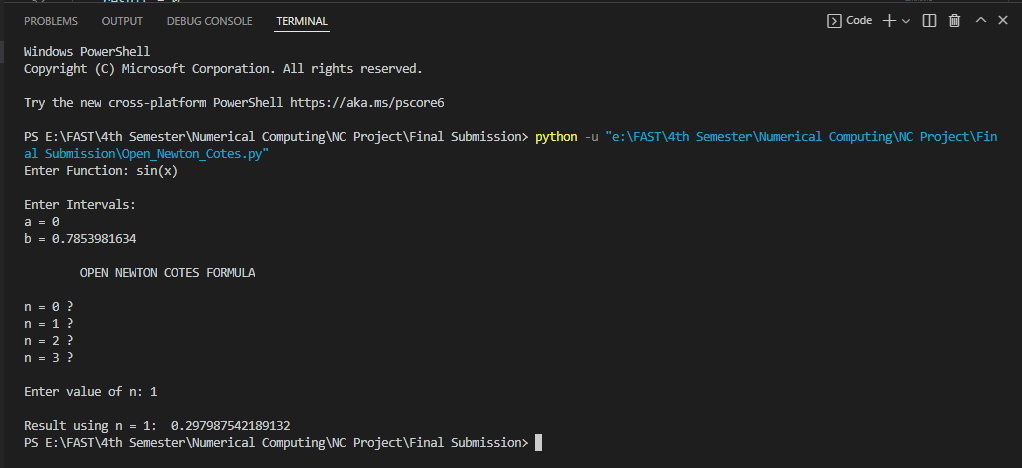




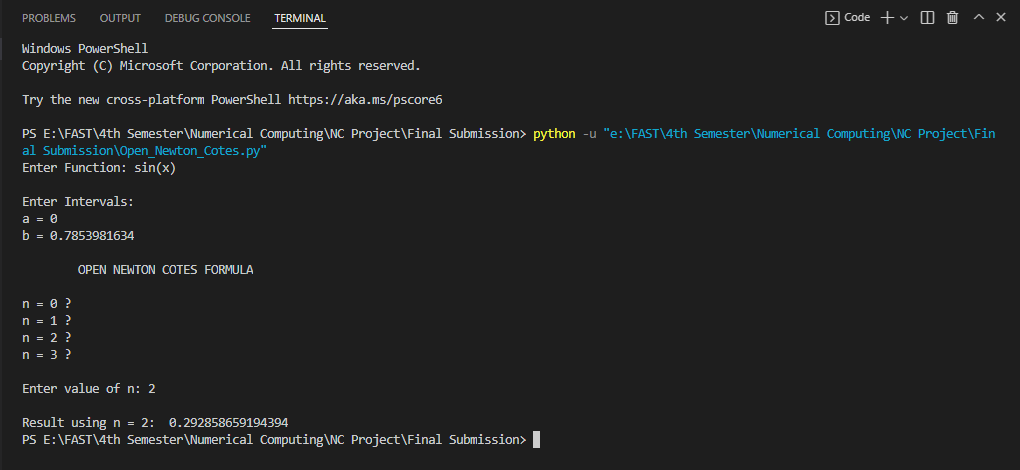
* **N = 0:**



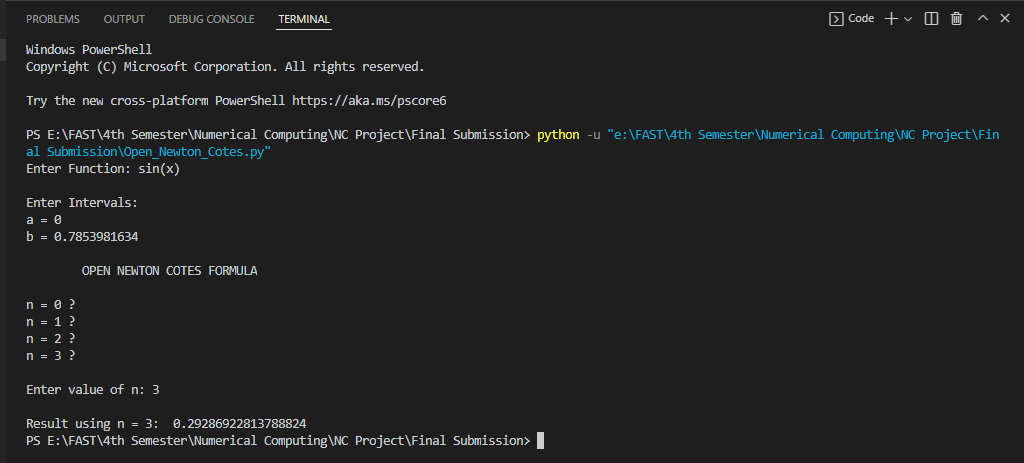
* **N = 1:**



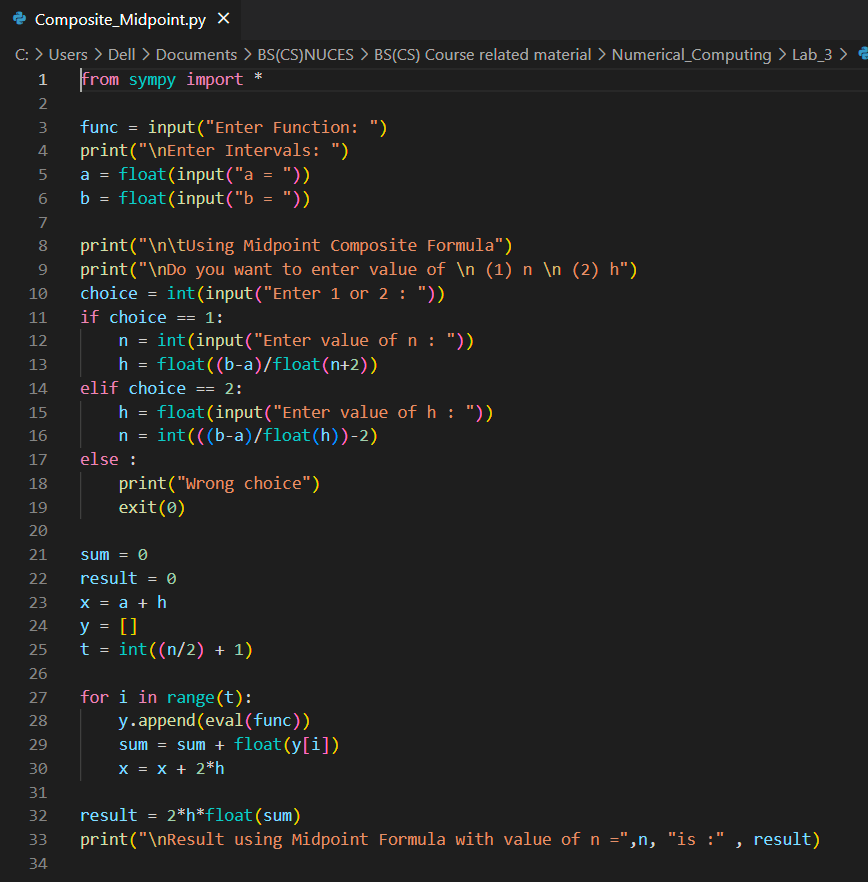
* **N = 2:**



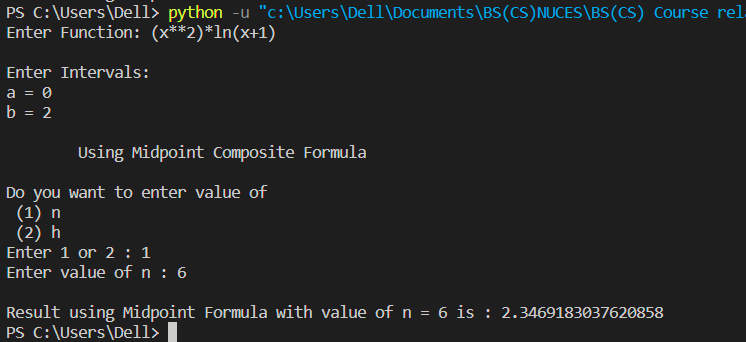
* **N = 3:**



**Composite Midpoint Rule**

****

* **When ‘n’ is given :**

****

* **When ‘h’ is given :**

