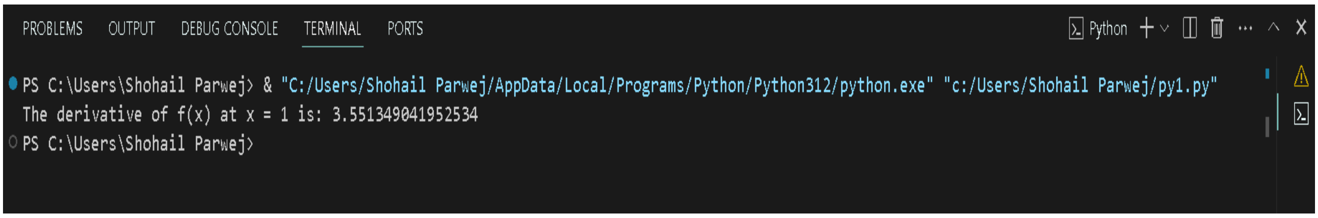
1. Write a Python program to find the derivate of using limit of the difference coefficient method at x=1.

$$f(x) = e^{x^2} + \sin(x) - \tan(x) + \log(x)$$

Code :-

```
#Author : Shohail Parwej

#Regd.No. : 2141016146

import math

def  f(x):

   return math.exp(x**2) + math.sin(x) - math.tan(x) + math.log(x)

def  derivative_at_1(f, h=1e-6):

   # Calculate the difference coefficient

   derivative = (f(1 + h) - f(1)) / h

   return derivative

# Calculate the derivative at x = 1

derivative_at_x_1 = derivative_at_1(f)

print("The derivative of f(x) at x = 1 is:", derivative_at_x_1)
```

Output :



2. Write a Python program to find gradient of Rosenbrock function using limit of the difference coefficient method at the point (1,2). Rosenbrock function is defined below.

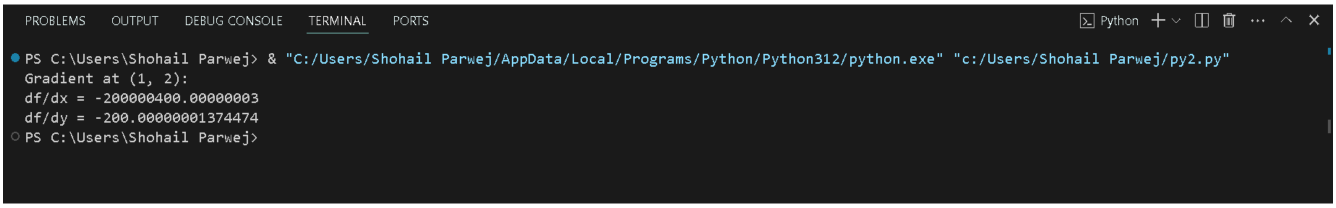
$$f(x, y) = (1 - x)^2 + 100(y - x^2)^{\wedge 2}$$

Code

```
#Author : Shohail Parwej

#Regd.No. : 2141016146


def rosenbrock(x, y):
```

```
    return (1 - x)**2 + 100 * (y - x**2)**2
def partial_x(f, x, y, h=1e-6):
    return (-2 * (1 - x) - 400 * x * (y - x**2) - f(x - h, y) + f(x + h, y)) / (2 * h)
def partial_y(f, x, y, h=1e-6):
    return (f(x, y - h) - f(x, y + h)) / (2 * h)

# Evaluate partial derivatives at point (1, 2)
x = 1
y = 2
gradient_x = partial_x(rosenbrock, x, y)
gradient_y = partial_y(rosenbrock, x, y)
print("Gradient at (1, 2):")
print("df/dx =", gradient_x)
print("df/dy =", gradient_y)
```

Output :



3. Write a Python program to find the point of minima of function using Gradient Descent method taking initial solution x0 = 2.

$$f(x) = x^2 + \sin(x)$$

Code

```
#Author : Shohail Parwej
#Regd.No. : 2141016146
import math
# Define the function f(x)
def f(x):
```

```
    return x**2 + math.sin(x)
# Define the derivative of f(x)
def df(x):
    return 2*x + math.cos(x)


# Gradient Descent function to find the minimum
def gradient_descent(f, df, x0, learning_rate=0.1, tolerance=1e-6, max_iterations=1000):
    x = x0
    iteration = 0
    while True:
        gradient = df(x)
        x_new = x - learning_rate * gradient
        if abs(x_new - x) < tolerance or iteration >= max_iterations:
            break
        x = x_new
        iteration += 1
    return x


x0 = 2
minimum_point = gradient_descent(f, df, x0)
print("Minimum point (x) found by Gradient Descent method:", minimum_point)
print("Minimum value of f(x) at the minimum point:", f(minimum_point))
```

Output

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                   Python
PS C:\Users\Shohail Parwej> & "C:/Users/Shohail Parwej/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Shohail Parwej/py3.py"
Minimum point (x) found by Gradient Descent method: -0.45018008157897416
Minimum value of f(x) at the minimum point: -0.23246557514304614
PS C:\Users\Shohail Parwej>
```

4. Write a Python program to find the point of minima of Rosenbrock function using Gradient Descent method taking initial solution (0,0). Rosenbrock function is defined below.

$$f(x, y) = (1 - x)^2 + (y - x^2)^2$$

Code

```
#Author : Shohail Parwej

#Regd.No. : 2141016146

import math

# Rosenbrock function

def rosenbrock(x, y):

    return (1 - x)**2 + 100 * (y - x**2)**2

# Gradient of Rosenbrock function

def rosenbrock_gradient(x, y):

    df_dx = -2 * (1 - x) - 400 * x * (y - x**2)

    df_dy = 200 * (y - x**2)

    return [df_dx, df_dy]

# Gradient Descent method

def gradient_descent(rosenbrock, gradient, initial_solution, learning_rate=0.001, tolerance=1e-6,
max_iterations=10000):

    solution = initial_solution

    for i in range(max_iterations):

        grad = gradient(solution[0], solution[1])

        norm_grad = math.sqrt(grad[0]**2 + grad[1]**2)

        if norm_grad < tolerance:

            break

        solution[0] -= learning_rate * grad[0]

        solution[1] -= learning_rate * grad[1]

    return solution


initial_solution = [0, 0]

minimum_point = gradient_descent(rosenbrock, rosenbrock_gradient, initial_solution)

print("Minimum point (x, y) found by Gradient Descent method:", minimum_point)

print("Minimum value of f(x, y) at the minimum point:", rosenbrock(*minimum_point))
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                    Python
PS C:\Users\Shohail Parwej> & "C:/Users/Shohail Parwej/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Shohail Parwej/py4.py"
Minimum point (x, y) found by Gradient Descent method: [0.9944009477070963, 0.9888107640653774]
Minimum value of f(x, y) at the minimum point: 3.139992492505519e-05
PS C:\Users\Shohail Parwej>
```

5. Let X be a binomial random variable with parameters n = 100 and p = 0.6. Write a Python program to find the approximate probability that:

1. X lies above 60.

2. X lies between 50 and 70 ; using normal approximation to binomial distribution.

Code :

```
#Author : Shohail Parwej
#Regd.No. : 2141016146


import math
n = 100
p = 0.6
# Mean and standard deviation of the binomial distribution
mu = n * p
sigma = math.sqrt(n * p * (1 - p))
# Standard normal distribution CDF function
def std_normal_cdf(x):
    return 0.5 * (1 + math.erf(x / math.sqrt(2)))


# 1. Probability that X lies above 60
z_score_60 = (60 - mu) / sigma
prob_above_60 = 1 - std_normal_cdf(z_score_60)


# 2. Probability that X lies between 50 and 70
z_score_50 = (50 - mu) / sigma
z_score_70 = (70 - mu) / sigma
```

```
prob_between_50_and_70 = std_normal_cdf(z_score_70) - std_normal_cdf(z_score_50)

print("Approximate probability that X lies above 60:", prob_above_60)

print("Approximate probability that X lies between 50 and 70:", prob_between_50_and_70)
```

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                    Python
PS C:\Users\Shohail Parwej> & "C:/Users/Shohail Parwej/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Shohail Parwej/py5.py"
Approximate probability that X lies above 60: 0.5
Approximate probability that X lies between 50 and 70: 0.9587731666628363
PS C:\Users\Shohail Parwej>
```

6. Define p-value and write a python program to find the two-sided p-value with and without continuity correction when the values of x(observed no. of heads), mean and standard deviation are 110, 100, 5 respectively.

Code

```
#Author : Shohail Parwej

#Regd.No. : 2141016146

import math


def normal_cdf(x, mu, sigma):

    return 0.5 * (1 + math.erf((x - mu) / (sigma * math.sqrt(2))))


def two_sided_p_value(x, mu, sigma, continuity=False):

# Z-score calculation

    z = (x - mu) / sigma

    # Apply continuity correction if specified

    if continuity:

        z = abs(z) - 0.5

    else:

        z = abs(z)
```

```
    # Calculate one-tailed probability using the CDF of the standard normal distribution
    p_one_tail = normal_cdf(z, 0, 1)
    # Two-sided p-value (sum of probabilities in both tails)
    p_value = 2 * (1 - p_one_tail)
    return p_value

x = 110
mu = 100
sigma = 5
# Calculate p-value with and without continuity correction
p_value_without_correction = two_sided_p_value(x, mu, sigma)
p_value_with_correction = two_sided_p_value(x, mu, sigma, continuity=True)
print("Two-sided p-value (without continuity correction):", p_value_without_correction)
print("Two-sided p-value (with continuity correction):", p_value_with_correction)
```

Output :