```
#AMIT BABU KHATRI
#2358569
```

Question No 1

```
import numpy as np


# Function to get a valid number input from user
def get_valid_number_input(prompt): #defines a function called get_valid_number_input with parameter prompt
    while True: #check the conditon
        try:
            value = float(input(prompt)) #convert the input string to a floating-point number
            return value
        except ValueError:
            print("Invalid input. Please enter a valid number.") #print if try condition doesnot valid and ask user to enter valid number.


# Create empty 3x3 matrix A
A = np.zeros((3, 3))

# Populate matrix A with user input
for i in range(3):
    for j in range(3):
      #generates a formatted string that serves as the prompt to be displayed to the user
        A[i][j] = get_valid_number_input(f"Enter value for A[{i}][{j}]: ")

# Print matrix A
print("Matrix A:")
print(A)

# Calculate determinant of matrix A
determinant = np.linalg.det(A) #np.linalg.det(A) computes the determinant of matrix A using the det function.
print("Determinant of matrix A:") #print determiant of matrix A
print(determinant)

# Check if matrix A is invertible
if determinant == 0: #check if determinat is equal to zero or not
    print("Matrix A is not invertible")
else:
    # Calculate inverse of matrix A
    inverse_A = np.linalg.inv(A) #np.linalg.inv(A) expression calculates the inverse of matrix A using the inv function.
    print("Inverse of matrix A:") #print inverse of matrix A.
    print(inverse_A)

# Calculate A*A and A*A*A
A_square = np.dot(A, A) #np.dot(A, A) performs the matrix multiplication operation between matrix A and itself.
A_cube = np.dot(A_square, A) #np.dot(A_square, A) performs the matrix multiplication operation between A_square and A.

# Print A*A and A*A*A
print("A*A:")
print(A_square)
print("A*A*A:")
print(A_cube)

# Count number of values in matrix A that are greater than or equal to 10
count = np.count_nonzero(A >= 10)
print(f"Number of values in matrix A that are greater than or equal to 10: {count}")
```

```
Enter value for A[0][0]: 3
Enter value for A[0][1]: 4
Enter value for A[0][2]: 2
Enter value for A[1][0]: 1
Enter value for A[1][1]: 3
Enter value for A[1][2]: 7
Enter value for A[2][0]: 9
Enter value for A[2][1]: 11
Enter value for A[2][2]: 4
Matrix A:
[[ 3.  4.  2.]
 [ 1.  3.  7.]
 [ 9. 11.  4.]]
Determinant of matrix A:
9.000000000000009
```

```
Inverse of matrix A:
[[-7.22222222  0.66666667  2.44444444]
 [ 6.55555556 -0.66666667 -2.11111111]
 [-1.77777778  0.33333333  0.55555556]]
A*A:
[[ 31.  46.  42.]
 [ 69.  90.  51.]
 [ 74. 113. 111.]]
A*A*A:
[[ 517.  724.  552.]
 [ 756. 1107.  972.]
 [1334. 1856. 1383.]]
Number of values in matrix A that are greater than or equal to 10: 1
```

Question No 2

```
#AMIT BABU KHATRI
#2358569
import numpy as np


def get_trace(matrix): #defines a function called get_trace with parameter matrix.
    # Convert the list of lists to a numpy array
    mx = np.array(matrix)
    # Get the trace (sum of diagonal elements)
    trace = mx.trace()
    return trace #return trace


# Test the function with different sized matrices
matrix_3x3 = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
matrix_4x4 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
matrix_6x6 = [[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12], [13, 14, 15, 16, 17, 18], [19, 20, 21, 22, 23, 24],
              [25, 26, 27, 28, 29, 30], [31, 32, 33, 34, 35, 36]]

#get_trace() function calculates the trace of the matrix (the sum of the elements on the main diagonal) and returns the result.
trace_3x3 = get_trace(matrix_3x3) #calls the get_trace() function, passing matrix_3x3 as an argument.
trace_4x4 = get_trace(matrix_4x4) #calls the get_trace() function, passing matrix_4x4 as an argument.
trace_6x6 = get_trace(matrix_6x6) ##calls the get_trace() function, passing matrix_6x6 as an argument.

#prints the value of the trace of a matrix
print(f"The trace of the 3x3 matrix is {trace_3x3}")
print(f"The trace of the 4x4 matrix is {trace_4x4}")
print(f"The trace of the 6x6 matrix is {trace_6x6}")
```

```
The trace of the 3x3 matrix is 15
The trace of the 4x4 matrix is 34
The trace of the 6x6 matrix is 111
```

Question No 3

```
#AMIT BABU KHATRI
#2358569
#defines a function named a that takes two required parameters m and n, and an optional parameter count with a default value of 0.
def a(m, n, count=0):
    count += 1 # increments the count variable by 1.
    if m == 0: #checks if m is equal to 0

    # If true, it returns a tuple (n + 1, count). This represents the base case of the Ackermann function.
        return n + 1, count

    elif m > 0 and n == 0:# checks if both m and n are greater than 0.
        return a(m - 1, 1, count + 1) #call is with arguments (m, n - 1, count + 1), and the result is stored in the variables res and count
    elif m > 0 and n > 0:
        res, count = a(m, n - 1, count + 1)
        return a(m - 1, res, count + 1)


first = int(input('Enter first value for ackermann function: ')) #prompt the user to input  integer values for first value
second = int(input('Enter second value for ackermann function: ')) # prompt the user to input  integer for second value
result, num_calls = a(first, second) #calls the function a with the provided input values and assigns the returned values to result and num_c

#print the result of the Ackermann function (result) and the number of recursive calls made (num_calls).
print("Result:", result)
```

```
print("Number of recursive calls:", num_calls)
```

```
Enter first value for ackermann function: 3
Enter second value for ackermann function: 4
Result: 125
Number of recursive calls: 20613
```

✓ 7s completed at 9:04 PM