Experiment – 1

Aim: Write a program to find the factorial of a number.

pseudocode :
1. Function factorial(n):
   a. Initialize result = 1
   b. For i from 1 to n:
      i. result *= i
   c. Return result
2. Input number
3. Call factorial function and print the result

source code :
```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

num = int(input("Enter a number: "))
print(f"Factorial of {num} is {factorial(num)}")
```

output:

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 1_factorial.py
Enter a number: 4
24
```

Experiment – 2

Aim: Generate Fibonacci series of N terms.

pseudocode :
1. Function fibonacci(n):
   a. Initialize fib_series as empty list
   b. Initialize a = 0, b = 1
   c. For _ from 0 to n-1:
      i. Append a to fib_series
      ii. Update a, b to b, a + b
   d. Return fib_series
2. Input number of terms
3. Call fibonacci function and print the series

source code :
```
def fibonacci(n):
    fib_series = []
    a, b = 0, 1
    for _ in range(n):
        fib_series.append(a)
        a, b = b, a + b
    return fib_series

terms = int(input("Enter the number of terms: "))
print(f"Fibonacci series: {fibonacci(terms)}")
```

output:

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 2_fibonacci.py
Enter the number of terms: 5
Fibonacci series: [0, 1, 1, 2, 3]
```

Experiment – 3

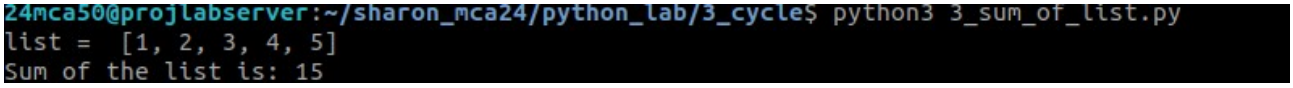
Aim: Write a program to find the sum of all items in a list. [Using for loop]

pseudocode :
1. Function sum_of_list(lst):
   a. Initialize total = 0
   b. For each item in lst:
      i. total += item
   c. Return total
2. Define a list of numbers
3. Call sum_of_list function and print the sum

source code :
```
def sum_of_list(lst):
    total = 0
    for item in lst:
        total += item
    return total

numbers = [1, 2, 3, 4, 5]  # Example list
print(f"Sum of the list is: {sum_of_list(numbers)}")
```

output:



Experiment – 4

Aim: Generate a list of four-digit numbers in a given range with all their digits even and the number is a perfect square.

pseudocode :

1. Function even_perfect_squares(start, end):
   a. Initialize results as empty list
   b. For num from start to end:
      i. If num is between 1000 and 9999 and num is even:
         A. Find the square root of num
         B. If square root squared is num:
            a. Convert num to string
            b. If all digits are even:
               i. Append num to results
   c. Return results
2. Input start and end range
3. Call even_perfect_squares function and print the results

source code :

```
def even_perfect_squares(start, end):
    results = []
    for num in range(start, end + 1):
        if 1000 <= num <= 9999 and num % 2 == 0:
            root = int(num**0.5)
            if root * root == num:  # Check if it's a perfect square
                digits = str(num)
                if all(int(digit) % 2 == 0 for digit in digits):
                    results.append(num)
    return results

start_range = int(input("Enter start range: "))
end_range = int(input("Enter end range: "))
print(f"Four-digit even perfect squares: {even_perfect_squares(start_range, end_range)}")
```

ouput:

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 4_even_perfect_squares.py
Enter start range: 4000
Enter end range: 9000
Four-digit even perfect squares: [4624, 6084, 6400, 8464]
```

Experiment – 5

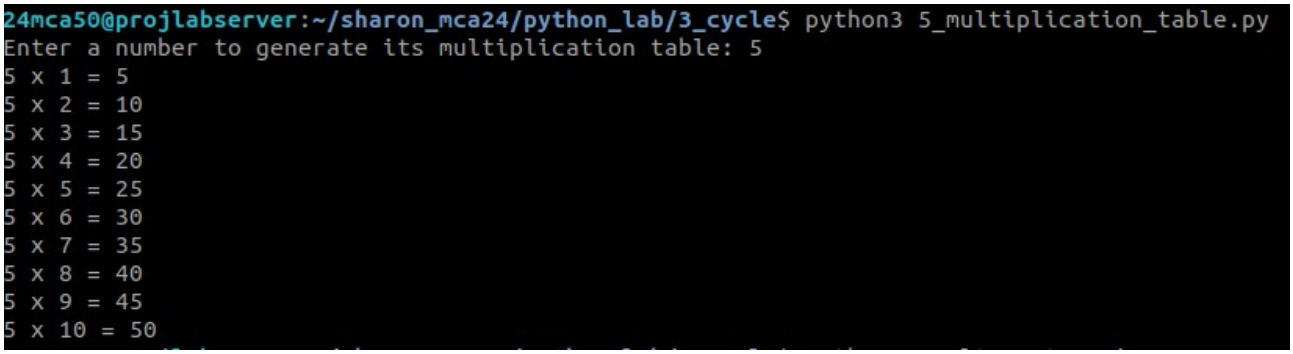
Aim: Write a program using a for loop to print the multiplication table of n, where n is entered by the user.

pseudocode :
1. Function multiplication_table(n):
   a. For i from 1 to 10:
      i. Print n x i = n * i
2. Input n
3. Call multiplication_table function

source code :
```
def multiplication_table(n):
    for i in range(1, 11):
        print(f"{n} x {i} = {n * i}")

num = int(input("Enter a number to generate its multiplication table: "))
multiplication_table(num)
```

output:



Experiment – 6

Aim: Write a program to display alternate prime numbers till N (obtain N from the user).

pseudocode :

1. Function is_prime(num):
   a. If num < 2:
      i. Return False
   b. For i from 2 to sqrt(num):
      i. If num % i == 0:
         A. Return False
   c. Return True
2. Function alternate_primes(n):
   a. Initialize count = 0
   b. For num from 2 to n:
      i. If is_prime(num):
         A. Increment count
         B. If count is odd:
            a. Print num
3. Input upper limit
4. Call alternate_primes function

source code :

```
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def alternate_primes(n):
    count = 0
    for num in range(2, n + 1):
        if is_prime(num):
            count += 1
            if count % 2 == 1:  # Print alternate primes
                print(num)

limit = int(input("Enter the upper limit: "))
print("Alternate prime numbers up to N:")
alternate_primes(limit)
```

output:

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 6_alternate_primes.py
Enter the upper limit: 30
Alternate prime numbers up to N:
2
5
11
17
23
```

Experiment – 7

Aim: Write a program to compute and display the sum of all integers that are divisible by 6 but not by 4, and that lie below a user-given upper limit.

pseudocode :

1. Function sum_divisible_by_6_not_4(limit):
   a. Initialize total = 0
   b. For i from 0 to limit-1:
      i. If i % 6 == 0 and i % 4 != 0:
         A. total += i
   c. Return total
2. Input upper limit
3. Call sum_divisible_by_6_not_4 function and print the sum

source code :

```
def sum_divisible_by_6_not_4(limit):
    total = 0
    for i in range(limit):
        if i % 6 == 0 and i % 4 != 0:
            total += i
    return total

upper_limit = int(input("Enter the upper limit: "))
print(f"Sum of integers divisible by 6 but not by 4 below {upper_limit}:
{sum_divisible_by_6_not_4(upper_limit)}")
```

output:

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 7_numbers_divisible_by_6_but_not_4_in_list.py
Enter the upper limit: 20
Sum of integers divisible by 6 but not by 4 below 20: 24
```

Experiment – 8

Aim: Calculate the sum of the digits of each number, within a specified range, and print the sum, only if it is prime.

pseudocode :

1. Function sum_of_digits(n):
   a. Initialize sum = 0
   b. For each digit in n:
      i. sum += digit
   c. Return sum
2. Function is_prime(num):
   a. Check if num < 2
   b. For i from 2 to sqrt(num):
      i. If num % i == 0:
         A. Return False
   c. Return True
3. Function sum_digits_in_range(upper_limit):
   a. For num from 1 to upper_limit:
      i. Calculate digit_sum using sum_of_digits
      ii. If digit_sum is prime:
         A. Print digit_sum
4. Input upper limit and Call sum_digits_in_range function

source code :

```
def sum_of_digits(n):
    return sum(int(digit) for digit in str(n))

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def sum_digits_in_range(upper_limit):
    for num in range(1, upper_limit + 1):
        digit_sum = sum_of_digits(num)
        if is_prime(digit_sum):
            print(f"Sum of digits of {num} is {digit_sum}, which is prime.")

limit = int(input("Enter an upper limit: "))
sum_digits_in_range(limit)
```

output :

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 8_digit_sum_only_for_primes_in_range.py
Enter an upper limit: 10
Sum of digits of 2 is 2, which is prime.
Sum of digits of 3 is 3, which is prime.
Sum of digits of 5 is 5, which is prime.
Sum of digits of 7 is 7, which is prime.
```

Experiment – 9

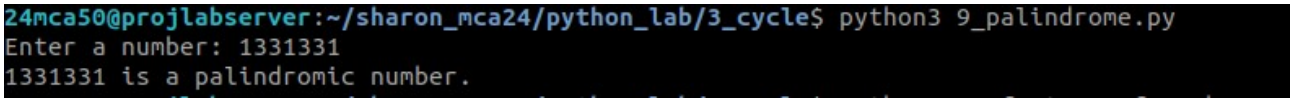
Aim: A number is input through the keyboard. Write a program to determine if it’s palindromic.

pseudocode :
1. Function is_palindrome(num):
   a. Convert num to string
   b. If string is equal to reverse of string:
      i. Return True
   c. Return False
2. Input number
3. Call is_palindrome function and display result

source code :
```
def is_palindrome(num):
    return str(num) == str(num)[::-1]

number = int(input("Enter a number: "))
if is_palindrome(number):
    print(f"{number} is a palindromic number.")
else:
    print(f"{number} is not a palindromic number.")
```

output:



Experiment – 10

Aim: Write a program to generate all factors of a number. [use while loop]

pseudocode :
1. Function factors(num):
   a. Initialize result as empty list
   b. Initialize i = 1
   c. While i <= num:
      i. If num % i == 0:
         A. Append i to result
      ii. Increment i
   d. Return result
2. Input number
3. Call factors function and print the result

source code :
```
def factors(num):
    result = []
    i = 1
    while i <= num:
        if num % i == 0:
            result.append(i)
        i += 1
    return result

number = int(input("Enter a number to find its factors: "))
print(f"Factors of {number}: {factors(number)}")
```

output :

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 10_factors_of_number.py
Enter a number to find its factors: 4
Factors of 4: [1, 2, 4]
```

Experiment – 11

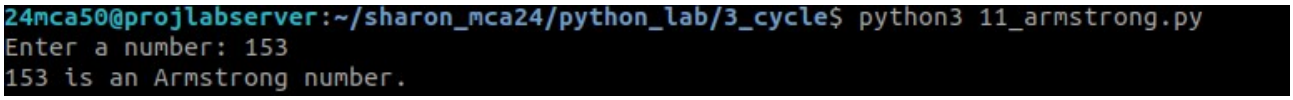
Aim: Check if a number is an Armstrong number (using a while loop).

pseudocode :
1. Function is_armstrong(num):
   a. Initialize sum_of_cubes = 0
   b. Store original_num = num
   c. While num > 0:
      i. Extract last digit
      ii. Update sum_of_cubes += digit^3
      iii. Remove last digit
   d. If sum_of_cubes equals original_num:
      i. Return True
   e. Return False
2. Input number
3. Call is_armstrong function and display result

source code :
```
def is_armstrong(num):
    sum_of_cubes = 0
    original_num = num
    while num > 0:
        digit = num % 10
        sum_of_cubes += digit ** 3
        num //= 10
    return sum_of_cubes == original_num

number = int(input("Enter a number: "))
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

output :



Experiment – 12

Aim: Display a pyramid based on user input.

pseudocode :
1. Function display_pyramid(n):
   a. For i from 1 to n:
      i. For j from 1 to i:
         A. Print i * j
      ii. Move to the next line
2. Input number of steps
3. Call display_pyramid function

source code :
```
def display_pyramid(n):
    for i in range(1, n + 1):
        for j in range(1, i + 1):
            print(i * j, end=' ')
        print()

steps = int(input("Enter the number of steps for the pyramid: "))
display_pyramid(steps)
```

output :

```
24mca50@projlabserver:~/sharon_mca24/python_lab/3_cycle$ python3 12_pyramid.py
Enter the number of steps for the pyramid: 5
1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
```

Experiment – 13

Aim: Construct a pattern using nested loops.

pseudocode :
1. Function print_pattern():
   a. For i from 1 to 5:
      i. Print '*' repeated i times
   b. For i from 4 down to 1:
      i. Print '*' repeated i times
2. Call print_pattern function

source code :
```
def print_pattern():
    for i in range(1, 6):
        print('* ' * i)
    for i in range(4, 0, -1):
        print('* ' * i)

print_pattern()
```

output :