

MODULE 01

1. Write a program that generates the Collatz sequence for every number from 1 to N, and prints the number that produces the longest sequence and the maximum value reached across all sequences

```
# Collatz Sequence Analyzer
# For every number from 1 to N:
# - Find which starting number produces the longest Collatz sequence
# - Find the maximum value reached across ALL sequences

def collatz_longest_and_global_max(N):
    memo = {1: 1}          # Cache for sequence lengths
    best_start = 1
    best_length = 1
    global_max_value = 1

    for start in range(1, N + 1):
        n = start
        local_max = n
        path = []

        # Generate Collatz sequence until we hit a memoized number
        while n not in memo:
            path.append(n)
            local_max = max(local_max, n)

            if n % 2 == 0:
                n = n // 2
            else:
                n = 3 * n + 1

            local_max = max(local_max, n)

        # Add lengths back through the stored path
        length = memo[n]
        for value in reversed(path):
            length += 1
            memo[value] = length

        # Update longest sequence info
        seq_length = memo[start]
        if seq_length > best_length:
            best_length = seq_length
            best_start = start
```

```

        # Update global max value
        global_max_value = max(global_max_value, local_max)

    return best_start, best_length, global_max_value

# Example usage
if __name__ == "__main__":
    N = int(input("Enter N: "))
    start, length, max_value = collatz_longest_and_global_max(N)

    print("Starting number with longest sequence:", start)
    print("Length of that sequence:", length)
    print("Maximum value reached across all sequences:", max_value)

```

OUTPUT:

Enter N: 100

Starting number with longest sequence: 97

Length of that sequence: 119

Maximum value reached across all sequences: 9232

2. Given a paragraph, create a two-dimensional table (26×26) where cell (i, j) stores the number of times character i is followed by character j in the text. Ignore spaces and convert everything to lowercase

```

# build 26x26 pair-count table for letters (ignore non-letters, lowercase)
def pair_table(text):
    s = ''.join(ch for ch in text.lower() if 'a' <= ch <= 'z')
    table = [[0]*26 for _ in range(26)]
    for a, b in zip(s, s[1:]):
        table[ord(a)-97][ord(b)-97] += 1
    return table

# usage
if __name__ == "__main__":
    txt = input("Enter paragraph: ")
    tbl = pair_table(txt)
    for row in tbl:
        print(' '.join(map(str, row)))

```

OUTPUT:

Enter paragraph: Hey there! How are you? Hope you are doing well...

```
0000000000000000002000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000001000000000000
00010001000100000100000030
00000000000000000000000000000
0000000000000000000000001000
00002000000000200000000000000
00000000000000100000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000010000000000000000
00000000000000000000000000000
00000001000000000000000000000
00000000100000010000201000
00001000000000000000000000000
00000000000000000000000000000
00003000000000000000000000000
00000000000000000000000000000
00000001000000000000000000000
10000001000000000000000000000
00000000000000000000000000000
10001000000000000000000000000
00000000000000000000000000000
000000000000000200001000000
00000000000000000000000000000
```

3. Read marks of n students for m subjects into a matrix.

Print:

Subject averages

Student ranks

Highest & lowest in each subject

Use nested loops only

```
n = int(input("Students: "))
m = int(input("Subjects: "))

marks = []
for i in range(n):
    row = []
    for j in range(m):
        row.append(float(input(f"Mark of student {i+1}, subject {j+1}: ")))
    marks.append(row)

# subject averages
sub_avg = [0]*m
for j in range(m):
    s = 0
    for i in range(n):
        s += marks[i][j]
    sub_avg[j] = s / n

# student totals for ranking
totals = [0]*n
for i in range(n):
    t = 0
    for j in range(m):
        t += marks[i][j]
    totals[i] = t

# ranks (simple descending)
rank = [1]*n
for i in range(n):
    for k in range(n):
        if totals[k] > totals[i]:
            rank[i] += 1

# highest & lowest in each subject
high = [marks[0][j] for j in range(m)]
low = [marks[0][j] for j in range(m)]
for j in range(m):
    for i in range(n):
        if marks[i][j] > high[j]:
            high[j] = marks[i][j]
        if marks[i][j] < low[j]:
            low[j] = marks[i][j]
```

```
print("Subject averages:", sub_avg)
print("Student ranks:", rank)
print("Highest per subject:", high)
print("Lowest per subject:", low)
```

OUTPUT:

Students: 2

Subjects: 4

Mark of student 1, subject 1: 50

Mark of student 1, subject 2: 60

Mark of student 1, subject 3: 35

Mark of student 1, subject 4: 80

Mark of student 2, subject 1: 70

Mark of student 2, subject 2: 40

Mark of student 2, subject 3: 65

Mark of student 2, subject 4: 90

Subject averages: [60.0, 50.0, 50.0, 85.0]

Student ranks: [2, 1]

Highest per subject: [70.0, 60.0, 65.0, 90.0]

Lowest per subject: [50.0, 40.0, 35.0, 80.0]

MODULE 02

1. Write a program to compress a string:

"aaabbccaa" → "a3b2c2a2"

Do not use any library. Use loops + slicing.

```
2. s = input("Enter string: ")
3. res = ""
4. i = 0
5.
6. while i < len(s):
7.     ch = s[i]
8.     c = 1
9.     j = i + 1
10.    while j < len(s) and s[j] == ch:
11.        c += 1
12.        j += 1
13.    res += ch + str(c)
14.    i = j
15.
16. print(res)
17.
```

OUTPUT:

Enter string: aaabbccaa

a3b2c2a2

2. Write a Python program to perform multiplication of two matrices without using NumPy. Use nested loops only

```
r1 = int(input("Rows of A: "))
c1 = int(input("Cols of A: "))
r2 = int(input("Rows of B: "))
c2 = int(input("Cols of B: "))

if c1 != r2:
    print("Matrix multiplication not possible")
    exit()

A = []
B = []
print("Enter A:")
for i in range(r1):
    row = []
```

```

    for j in range(c1):
        row.append(int(input()))
    A.append(row)

print("Enter B:")
for i in range(r2):
    row = []
    for j in range(c2):
        row.append(int(input()))
    B.append(row)

# result matrix
C = [[0]*c2 for _ in range(r1)]

for i in range(r1):
    for j in range(c2):
        s = 0
        for k in range(c1):
            s += A[i][k] * B[k][j]
        C[i][j] = s

print("Result:")
for row in C:
    print(row)

```

OUTPUT:

Rows of A: 2

Cols of A: 2

Rows of B: 2

Cols of B: 2

Enter A:

1

2

3

4

Enter B:

6

7

8

9

Result:

[22, 25]

[50, 57]

3. Generate Pascal Triangle up to N rows using only lists of lists

```
N = int(input("Rows: "))
tri = []

for i in range(N):
    row = [1] * (i + 1)
    for j in range(1, i):
        row[j] = tri[i-1][j-1] + tri[i-1][j]
    tri.append(row)

for r in tri:
    print(r)
```

OUTPUT:

Rows: 5

[1]

[1, 1]

[1, 2, 1]

[1, 3, 3, 1]

[1, 4, 6, 4, 1]

1. Write a Python program using NumPy to analyze weather data stored in a 2D NumPy array, where each row represents a month and each column represents a day's temperature. Your program should compute the following:

- The monthly average temperature for all 12 months.
- The hottest day of the entire year and its temperature.
- The count of days that were above the overall yearly average temperature.
- A boolean mask identifying cold-wave days (temperature < 15°C).

```
• import numpy as np
•
• def analyze(temps):
•     monthly_avg = temps.mean(axis=1)
•     flat_idx = temps.argmax()
•     month, day = np.unravel_index(flat_idx, temps.shape) #
0-based
•     hottest = temps[month, day]
•     overall_avg = temps.mean()
•     count_above = int((temps > overall_avg).sum())
•     cold_mask = temps < 15
•     return monthly_avg, (month + 1, day + 1, hottest),
count_above, cold_mask
•
• if __name__ == "__main__":
•     # Example: 12 months x 30 days (replace with real data)
•     np.random.seed(0)
•     temps = np.random.normal(loc=20, scale=6, size=(12,
30)) # synthetic temps
•
•     monthly_avg, hottest_info, count_above, cold_mask =
analyze(temps)
•
•     print("Monthly averages:", np.round(monthly_avg, 2))
•     print(f"Hottest day: Month {hottest_info[0]}, Day
{hottest_info[1]}, Temp {hottest_info[2]:.2f}°C")
•     print("Days above yearly average:", count_above)
•     print("Cold-wave mask (shape = temps.shape):")
•     print(cold_mask.astype(int)) # print 1 for cold-wave
days, 0 otherwise
•
```

OUTPUT:

Monthly averages: [22.66 18.26 19.2 23.07 20.58 19.6 19.4 19.53 18.18 21.19 18.93 18.12]

Hottest day: Month 5, Day 25, Temp 34.30°C

Days above yearly average: 172

Cold-wave mask (shape = temps.shape):

```
[[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0]
 [0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0]
 [1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0]
 [0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0]]
```

2. You are given a dictionary with the following structure:

```
students = {
    roll1: {"name": ..., "marks": [...], "avg": ...},
    roll2: {"name": ..., "marks": [...], "avg": ...},
    ...
}
```

Write a Python program to compute the following from the dictionary:

The overall topper (student with highest average marks).

The subject topper for each subject (highest mark in each subject).

A list of students scoring below the class average (based on average marks).

```
def analyze(students):
    # overall topper
    topper = max(students.items(), key=lambda x: x[1]["avg"])[0]

    # subject toppers
    m = len(next(iter(students.values()))["marks"])
    subj_top = []
    for j in range(m):
        best = None
```

```

        best_roll = None
        for r, info in students.items():
            if best is None or info["marks"][j] > best:
                best = info["marks"][j]
                best_roll = r
        subj_top.append((j + 1, best_roll, best))

# class average
class_avg = sum(students[r]["avg"] for r in students) / len(students)
below_avg = [r for r in students if students[r]["avg"] < class_avg]

return topper, subj_top, below_avg

# example
students = {
    1: {"name": "A", "marks": [80, 90, 85], "avg": 85},
    2: {"name": "B", "marks": [70, 60, 75], "avg": 68.3},
    3: {"name": "C", "marks": [95, 88, 92], "avg": 91.6}
}

topper, subj_top, below_avg = analyze(students)

print("Overall topper roll:", topper)

print("\nSubject toppers:")
for sub, roll, mark in subj_top:
    print(f"Subject {sub}: Roll {roll} with {mark} marks")

print("\nStudents below class average:", below_avg)

```

OUTPUT:

Overall topper roll: 3

Subject toppers:

Subject 1: Roll 3 with 95 marks

Subject 2: Roll 1 with 90 marks

Subject 3: Roll 3 with 92 marks

Students below class average: [2]

MODULE 4

1. Write a program that has a class store which keeps a record of code and price of each product. Display a menu of all products to the user and prompt him to enter the quantity of each item required. Generate a bill and display the total amount.

```
2. class Store:
3.     def __init__(self):
4.         # code : (name, price)
5.         self.items = {
6.             101: ("Bread", 30),
7.             102: ("Milk", 25),
8.             103: ("Eggs", 6),
9.             104: ("Juice", 40)
10.        }
11.
12.    def display(self):
13.        print("Code Item Price")
14.        for code, (name, price) in self.items.items():
15.            print(f"{code} {name:<8} {price}")
16.
17.    def bill(self):
18.        total = 0
19.        for code, (name, price) in self.items.items():
20.            qty = int(input(f"Enter quantity for {name}: "))
21.            total += qty * price
22.        print("\nTotal Bill Amount:", total)
23.
24. s = Store()
25. s.display()
26. s.bill()
27.
```

OUTPUT:

Code Item Price

101 Bread 30

102 Milk 25

103 Eggs 6

104 Juice 40

Enter quantity for Bread: 3

Enter quantity for Milk: 5

Enter quantity for Eggs: 20

Enter quantity for Juice: 10

Total Bill Amount: 735

2. Write a program that has classes such as Student, Course and Department. Enroll a student in a course of a particular Department.

```
class Department:
    def __init__(self, name):
        self.name = name
        self.courses = []

    def add_course(self, course):
        self.courses.append(course)

class Course:
    def __init__(self, name, dept):
        self.name = name
        self.dept = dept
        self.students = []
        dept.add_course(self)

    def enroll(self, student):
        self.students.append(student)
        student.courses.append(self)

class Student:
    def __init__(self, name, department):
        self.name = name
        self.department = department
        self.courses = []

# departments
cs = Department("Computer Science")
math = Department("Mathematics")

# courses
c1 = Course("Python Programming", cs)
c2 = Course("Linear Algebra", math)

# students with home departments
s1 = Student("Arjun", cs)
s2 = Student("Nisha", math)

# enrollments
c1.enroll(s1)
c2.enroll(s1)
```

```

c1.enroll(s2)

# display
for s in [s1, s2]:
    print("\nStudent:", s.name)
    print("Home Department:", s.department.name)
    print("Enrolled Courses:")
    for c in s.courses:
        print("  ", c.name, "(Dept:", c.dept.name + ")")

```

OUTPUT:

Student: Arjun

Home Department: Computer Science

Enrolled Courses:

Python Programming (Dept: Computer Science)

Linear Algebra (Dept: Mathematics)

Student: Nisha

Home Department: Mathematics

Enrolled Courses:

Python Programming (Dept: Computer Science)

3. Write a program that has an abstract class polygon. Derive two classes Rectangle and Triangle from polygon and write methods to get the details of their dimensions and hence calculate the area.

```

from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def get_data(self):
        pass

    @abstractmethod
    def area(self):
        pass

```

```

class Rectangle(Polygon):
    def get_data(self):
        self.l = float(input("Length: "))
        self.b = float(input("Breadth: "))

    def area(self):
        return self.l * self.b

class Triangle(Polygon):
    def get_data(self):
        self.b = float(input("Base: "))
        self.h = float(input("Height: "))

    def area(self):
        return 0.5 * self.b * self.h

# example
print("1. Rectangle\n2. Triangle")
ch = int(input("Enter choice: "))

if ch == 1:
    obj = Rectangle()
else:
    obj = Triangle()

obj.get_data()
print("Area =", obj.area())

```

OUTPUT:

```

1. Rectangle
2. Triangle
Enter choice: 2
Base: 30
Height: 50
Area = 750.0

```

-END-

MODULE 4

1. Write a program that has a class store which keeps a record of code and price of each product. Display a menu of all products to the user and prompt him to enter the quantity of each item required. Generate a bill and display the total amount.

```
2. class Store:
3.     def __init__(self):
4.         # code : (name, price)
5.         self.items = {
6.             101: ("Bread", 30),
7.             102: ("Milk", 25),
8.             103: ("Eggs", 6),
9.             104: ("Juice", 40)
10.        }
11.
12.    def display(self):
13.        print("Code Item Price")
14.        for code, (name, price) in self.items.items():
15.            print(f"{code} {name:<8} {price}")
16.
17.    def bill(self):
18.        total = 0
19.        for code, (name, price) in self.items.items():
20.            qty = int(input(f"Enter quantity for {name}: "))
21.            total += qty * price
22.        print("\nTotal Bill Amount:", total)
23.
24.s = Store()
25.s.display()
26.s.bill()
27.
```

OUTPUT:

Code	Item	Price
------	------	-------

101	Bread	30
-----	-------	----

102	Milk	25
-----	------	----

103	Eggs	6
-----	------	---

104	Juice	40
-----	-------	----

Enter quantity for Bread: 3

Enter quantity for Milk: 5

Enter quantity for Eggs: 20

Enter quantity for Juice: 10

Total Bill Amount: 735

2. Write a program that has classes such as Student, Course and Department. Enroll a student in a course of a particular Department.

```
class Department:
    def __init__(self, name):
        self.name = name
        self.courses = []

    def add_course(self, course):
        self.courses.append(course)

class Course:
    def __init__(self, name, dept):
        self.name = name
        self.dept = dept
        self.students = []
        dept.add_course(self)

    def enroll(self, student):
        self.students.append(student)
        student.courses.append(self)

class Student:
    def __init__(self, name, department):
        self.name = name
        self.department = department
        self.courses = []

# departments
cs = Department("Computer Science")
math = Department("Mathematics")

# courses
c1 = Course("Python Programming", cs)
c2 = Course("Linear Algebra", math)

# students with home departments
s1 = Student("Arjun", cs)
s2 = Student("Nisha", math)
```

```

# enrollments
c1.enroll(s1)
c2.enroll(s1)
c1.enroll(s2)

# display
for s in [s1, s2]:
    print("\nStudent:", s.name)
    print("Home Department:", s.department.name)
    print("Enrolled Courses:")
    for c in s.courses:
        print("  ", c.name, "(Dept:", c.dept.name + ")")

```

OUTPUT:

Student: Arjun

Home Department: Computer Science

Enrolled Courses:

Python Programming (Dept: Computer Science)

Linear Algebra (Dept: Mathematics)

Student: Nisha

Home Department: Mathematics

Enrolled Courses:

Python Programming (Dept: Computer Science)

3. Write a program that has an abstract class polygon. Derive two classes Rectangle and Triangle from polygon and write methods to get the details of their dimensions and hence calculate the area.

```

from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def get_data(self):
        pass

    @abstractmethod
    def area(self):

```

```

        pass

class Rectangle(Polygon):
    def get_data(self):
        self.l = float(input("Length: "))
        self.b = float(input("Breadth: "))

    def area(self):
        return self.l * self.b

class Triangle(Polygon):
    def get_data(self):
        self.b = float(input("Base: "))
        self.h = float(input("Height: "))

    def area(self):
        return 0.5 * self.b * self.h

# example
print("1. Rectangle\n2. Triangle")
ch = int(input("Enter choice: "))

if ch == 1:
    obj = Rectangle()
else:
    obj = Triangle()

obj.get_data()
print("Area =", obj.area())

```

OUTPUT:

```

1. Rectangle
2. Triangle
Enter choice: 2
Base: 30
Height: 50
Area = 750.0

```

-END-