LEVEL - 2

1. Let's say you are managing a software development project, and you need to estimate the effort required for upcoming tasks. You notice that as tasks become more complex, the time taken increases in a manner similar to the Fibonacci sequence. This is because each new task builds upon the complexity of previous tasks.

Let:

T_n be the estimated time for the n-th task.

The sequence T follows the Fibonacci series.

Thus:

T(0)=T0

T(1)=T1

.

.

T(n)=T(n−1)+T(n−2) for n≥2

where T0 and T1 are the estimated times for the first and second tasks, respectively.

Example

Suppose the time for the initial tasks is estimated as follows:

- T0=1 hour (for the initial setup)
- T1=2 hours (for setting up the development environment)

Then the time estimation for subsequent tasks would be:

T(2)=T(1)+T(0)=2+1=3 hours

T(3)=T(2)+T(1)=3+2=5 hours

T(4)=T(3)+T(2)=5+3=8 hours

T(5)=T(4)+T(3)=8+5=13 hours

Constraint:

- 0 < n <= 20
- T0 = 1
- T2 = 2

Test Case 1:

Input: n = -1

Expected Output: Invalid Input

Test Case 2:

Input: n = 0

Expected Output: 1

Test Case 3:

Input: n = 1

Expected Output: 2

Test Case 4:

Input: n = 2

Expected Output: 3

Test Case 5:

Input: n = 3

Expected Output: 5

Test Case 6:

Input: n = 4

Expected Output: 8

Test Case 7:

Input: n = 5

Expected Output: 13

Test Case 8:

Input: n = 10

Expected Output: 144

Test Case 9:

Input: n = 25

Expected Output: Invalid

Test Case 10:

Input: n = 20

Expected Output: 17711

2. Develop a project for a data analysis firm that specializes in processing binary data obtained from various sources such as sensors, digital devices, and network logs. As part of the data processing pipeline, develop a utility to convert binary numbers into decimal format for further analysis and visualization.
This utility will be crucial for converting binary-encoded information, such as sensor readings or network packets, into human-readable decimal values, facilitating data interpretation and decision-making. Write a program that efficiently converts a binary number into its decimal equivalent.

Horner's method is often used to evaluate polynomials efficiently and can be applied to binary-to-decimal conversion.

Given a binary number $b = b_{n-1}b_{n-2}\ldots b_1b_0$, the decimal value D can be computed using:

$$D = ((\ldots((b_{n-1} \cdot 2 + b_{n-2}) \cdot 2 + b_{n-3}) \cdot 2 + \ldots + b_1) \cdot 2 + b_0$$

Let's denote the following variable:
b → Binary number provided as input

Constraints

$b \in \{0, 1\}^*$

Test Case 1:

Input:
0
Output:
The decimal equivalent of 0 is: 0

Test Case 2:

Input:
1
Output:
The decimal equivalent of 1 is: 1

Test Case 3:

Input:
10101
Output:
The decimal equivalent of 10101 is: 21

Test Case 4:

Input:
-101001
Output:
Invalid Input

Test Case 5:

Input:
1234
Output:
Invalid Input

Test Case 6:

Input:
A01B1
Output:
Invalid Input

Test Case 7:

Input:
1010$&1
Output:
Invalid Input

3. Picture yourself as a data scientist embarking on a project to unravel insights from a diverse dataset filled with numerical values. Your task involves crafting a program that computes the sum of cubes for all even numbers up to a specified limit.
This endeavor is essential for understanding patterns and trends within the dataset, particularly how even-numbered data points contribute to overall trends and statistical measures.
By meticulously calculating the sum of cubes, your tool illuminates hidden relationships and amplifies the clarity of decision-making processes for stakeholders relying on precise numerical analysis.
Calculating sum of the cubes of the first n even numbers using the formula :

$$2n^2(n+1)^2.$$

Constraints:

- The limit entered by the user must be a positive integer.
- 1<= n <= 2^31 - 1
- The program should use a while loop for iterative computation
- The program should not use any standard libraries or pre-defined functions to compute the sum of cubes of even numbers.

Test Case 1:

Input: limit = 10
Expected Output: The sum of cubes of even numbers up to 10 is 1800.
Explanation: Even numbers up to 10 are 2, 4, 6, 8, 10. Their cubes are 8, 64, 216, 512, 1000 respectively. Sum of these cubes gives the result as 1800.

Test Case 2:

Input: limit = 0
Expected Output: The sum of cubes of even numbers up to 0 is 0.

Test Case 3:

Input: limit = 100
Expected Output: The sum of cubes of even numbers up to 100 is 13005000.

Test Case 4:

Input: limit = 2
Expected Output: The sum of cubes of even numbers up to 2 is 8.

Test Case 5:

Input: limit = 1
Expected Output: The sum of cubes of even numbers up to 1 is 0.

Test Case 6:

Input: limit = -10

Expected Output: Invalid.

Test Case 7:

Input: limit = 7559

Expected Output: The sum of cubes of even numbers up to 10000 is 1534557472.

Test Case 8:

Input: limit = 10000

Expected Output: The sum of cubes of even numbers up to 10000 is -1153066880 (Garbage Value)

4. Imagine you're developing a software tool for a financial analysis firm that handles vast amounts of numerical data daily. As part of your task, you need to count the number of digits in each input integer. This crucial information helps analysts understand the scale and complexity of financial transactions and investments.
Your program will be used to process the accounts balance, transaction volumes, and customer data across various financial instruments. By accurately counting digits in integers, you ensure the integrity and precision of subsequent analyses and reports.

To determine the number of digits in a given positive integer n, you can use logarithms. The mathematical formula to count the number of digits in n is:

Number of digits = ⌊log10(n)⌋+1

Constraints:

- The input integer can be positive or negative.
- -2^63 <= n <= 2^63 - 1
- Absolute_value(n) is considered for the processing.
- The program should use a do-while loop to count the digits.

Test Case 1:

Input: number = 123
Expected Output: The number of digits in 123 is 3.
Explanation: The integer 123 has 3 digits.

Test Case 2:

Input: number = 0
Expected Output: The number of digits in 0 is 1.

Test Case 3:

Input: number = -4567
Expected Output: The number of digits in -4567 is 4.

Test Case 4:

Input: number = 9
Expected Output: The number of digits in 9 is 1.

Test Case 5:

Input: number = 9876543210
Expected Output: The number of digits in 9876543210 is 10.

Test Case 6:

Input: number = -123456789
Expected Output: The number of digits in -123456789 is 9.

Test Case 7:

Input: number = 1000000000

Expected Output: The number of digits in 1000000000 is 10.

Test Case 8:

Input: number = -1

Expected Output: The number of digits in -1 is 1.

Test Case 9:

Input: number = 00000000

Expected Output: The number of digits in 0 is 1.

Test Case 10:

Input: number = -999999999999999999

Expected Output: 19

5. Imagine you are working as a software engineer for a telecommunications company. Your team is responsible for developing a new system for error detection and correction in data transmission. In digital communications, data is often transmitted in binary form, and one common technique used for error detection is to generate the one's complement of binary data. This technique helps in detecting bit errors during data transmission. The program will use a loop to iterate through each bit of the binary number and compute its one's complement. [The one's complement of a binary number is obtained by flipping all the bits (i.e., converting all '0's to '1's and all '1's to '0's)].

Let's denote the following variable:

data → input signal value from telecom devices

Constraints:

- 1<= data <= 2^63 - 1

Test Case 1:

Input:
01000011
Output:
Ones complement: 10111100

Test Case 2:

Input:
-01000011
Output:
Invalid Input

Test Case 3:

Input:
1010
Output:
Ones complement: 0101

Test Case 4:

Input:
0000
Output:
Ones complement: 1111

Test Case 5:

Input:
1111
Output:

Ones complement: 0000

Test Case 6:

Input:

1

Output:

Ones complement: 0

Test Case 7: *(Empty)*

Input:

Output:

Ones complement:

Test Case 8:

Input:

101010

Output:

010101

6. A Software project development involves analyzing and processing large volumes of data. To estimate the effort required for different stages of data processing, you notice that the complexity of each stage increases in a pattern similar to the prime number sequence.

Let's denote the following variable:

n → number of stages involved in a software project

P(n) → estimated time required for the n-th stage, where the time follows the prime number sequence as given below,

- P(0) → 2 (first prime number)
- P(1) → 3 (second prime number)
- P(n) is the (n+1)th prime number for n ≥ 0.

Example:

If the number of stages involved in the project is 5, then the time for the stages is estimated as follows:

P(0) = 2 hours (for basic data pre-processing)

P(1) = 3 hours (for initial data validation)

Then the time estimation for subsequent stages would be:

- P(2) = 5 hours
- P(3) = 7 hours
- P(4) = 11 hours

Constraints:

0 < n <= 10^2

P(0) = 2

P(1) = 3

Test Case 1:

Input: Enter the stage involved in project (up to 100): 1

Expected Output:

Estimated Times for Data Processing Stages:

P(0) = 2 hours

Test Case 2:

Input: Enter the number of prime numbers to generate (up to 20): -1

ExpectedOutput:

Invalid input. Please enter a number between 1 and 20.

Test Case 3:
Input: Enter the number of prime numbers to generate (up to 20): 3
Expected Output:

Estimated Times for Data Processing Stages:
P(0) = 2 hours
P(1) = 3 hours
P(2) = 5 hours

Test Case 4:
Input: Enter the number of prime numbers to generate (up to 20): 20
Expected Output:

Estimated Times for Data Processing Stages:
P(0) = 2 hours
P(1) = 3 hours
P(2) = 5 hours
P(3) = 7 hours
P(4) = 11 hours
P(5) = 13 hours
P(6) = 17 hours
P(7) = 19 hours
P(8) = 23 hours
P(9) = 29 hours

Test Case 5:
Input: Enter the number of prime numbers to generate (up to 20): 0
ExpectedOutput:

Invalid input. Please enter a number between 1 and 20.

7. In a distributed computing environment, multiple nodes communicate over a network. The time it takes for a message to travel from one node to another is known as network latency. When multiple nodes are involved in a communication process, the total latency can be approximated using the harmonic series.

For instance, consider a distributed system where a central server communicates with n clients. If the communication time between the server and each client is inversely proportional to the client's position in a queue, the total communication time can be approximated by the sum of the first n terms of the harmonic series.

The harmonic series sum H_n for n terms is given by: $H_n = \sum_{i=1}^{n} \frac{1}{i}$

Constraints:

- 0 <= n <= 10^6, n → number of clients
- n must be a non-negative integer.
- n must not be a decimal or fractional number.

Test Case 1:

Input: n=1
Expected Output:
Sum: 1.00000

Test Case 2:

Input: n=5
Expected Output:
Sum: 2.28333 (approximately)

Test Case 3:

Input: n=10
Expected Output:
Sum: 2.92897 (approximately)

Test Case 4:

Input: n=1000
Expected Output:
Sum: 7.48547 (approximately)

Test Case 5:

Input: n=1500

Expected Output:

Sum: 7.89077 (approximately)

Test Case 6:

Input: n=0

Expected Output:

Error or prompt to enter a valid number greater than zero.

Test Case 7:

Input: n=−5

Expected Output:

Error or prompt to enter a valid number greater than zero.

Test Case 8:

Input: n=2.5

Expected Output:

Error or prompt to enter a valid integer.

Test Case 9:

Input: n=10^7 *(beyond practical computational limit)*

Expected Output:

Error or warning about the input size being too large.

8. Develop a program to print a square pattern of asterisks (*) where the interior of the square is hollow and filled with spaces. The size of the square (number of rows and columns) is provided by the user as input. Write a program that efficiently generates and prints this pattern.

Let n be the size of the square.

Constraints:

- 2 <= size <= 50
- n should be a positive integer.

Test Case 1:

Input: 3

Output:

```
***
* *
***
```

Test Case 2:

Input: 5

Output:

```
*****
*   *
*   *
*   *
*****
```

Test Case 3:

Input: 6

Output:

```
******
*    *
*    *
*    *
*    *
******
```

Test Case 4:

Input: 1

Output: Invalid Input

Explanation : It does not meet the minimum size requirement to form a hollow square

Test Case 5:

Input: -2

Output: Invalid input

Test Case 6:

Input: 6.7

Output: Invalid input

Test Case 7:

Input: 0

Output: Invalid input

9. Design a number manipulation tool that swaps the first and last digits of a given integer. This tool will be used by a data entry team to quickly modify numbers in a dataset based on this specific requirement.
The program should prompt the user to enter an integer. It will calculate the number of digits in the number using a while loop. After determining the number of digits, the program will extract the first and last digits of the number. Using a for loop, it will swap these digits and print the modified number.
Input: An integer N
Output: An integer M such that the first and last digits of N are swapped

- d1 is the first digit of N
- dn is the last digit of N
- The output M will be the integer formed by swapping d1 and dn alone ignoring the sign of the integer N

Constraints:

- 1<= N<= 2^31 - 1
- Ensure that leading zeros are ignored in the output.

Test Case 1:
Input: 5
Output: 5
Test Case 2:
Input: 10
Output: 1
Test Case 3:
Input: 12345
Output: 52341
Test Case 4:
Input: 987654321
Output: 187654329
Test Case 5:
Input: -42020
Output: -2024
Test Case 6:
Input: 1020
Output: 21

10. In financial forecasting, particularly when analyzing investment risks or predicting stock market movements, binomial models are often used. These models help in evaluating the potential outcomes of various financial scenarios over time. Pascal's Triangle can be used to compute the probabilities of different outcomes in a binomial tree, which is crucial for options pricing and other financial predictions.

Let's denote the following variable:
n → Number of rows in the pascal's triangle
k → Number of column in the pascal's triangle

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Constraints:

- n >= 0
- n, must be a positive integer.

Test Case 1:

Input: n=1
Expected Output:
1

Test Case 2:

Input: n = 3
Expected Output:
1
1 1
1 2 1

Test Case 3:

Input: n = 5
Expected Output:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Test Case 4:

Input: n = 10

Expected Output:

```
    1
      1 1
     1 2 1
    1 3 3 1
   1 4 6 4 1
  1 5 10 10 5 1
 1 6 15 20 15 6 1
 1 7 21 35 35 21 7 1
 1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

Test Case 5:

Input: n = 0

Expected Output: Invalid input. Number of rows must be at least 1.

11. In applications where users input numerical data, such as financial transactions or data logging systems, ensuring the integrity of the input data is crucial. An Armstrong number program could be used as a form of error-checking mechanism. For instance, if a user is expected to input a number and it needs to be verified before further processing, checking if it's an Armstrong number could be one way to validate the input.

Let's denote the following variable:
N →Positive integer provided by the user

Constraints:

- 0<=N<=2^31-1
- Return 1, if N is a armstrong number, and 0 otherwise

Test Case 1: Minimum value (N = 0)
Input: N = 0
Expected Output: 1
Test Case 2: Maximum value (N = 2^31 - 1)
Input: N = 2147483647
Expected Output: 0
Test Case 3: Single-digit Armstrong number (N = 9)
Input: N = 9
Expected Output: 1
Test Case 4: Three-digit Armstrong number (N = 153)
Input: N = 153
Expected Output: 1
Test Case 5: Three-digit non-Armstrong number (N = 154)
Input: N = 154
Expected Output: 0
Test Case 6: Large Armstrong number (N = 548834)
Input: N = 548834
Expected Output: 1
Test Case 7: Large non-Armstrong number (N = 548835)
Input: N = 548835
Expected Output: 0
Test Case 8: Edge case with all nines (N = 99999)

Input: N = 99999

Expected Output: 0

Test Case 9: Edge case with all zeros (N = 1000)

Input: N = 1000

Expected Output: 0

Test Case 10: Negative number

Input: N = -885

Expected Output: Invalid output

12. Develop a utility for a banking application to validate transaction IDs. A transaction ID is considered valid if it forms a palindrome. Depending on whether the transaction ID is a palindrome, the utility will perform a different calculation on the digits of the ID. If the ID is a palindrome, the utility will multiply all its digits together. If it is not a palindrome, the utility will sum all its digits. The utility will be able to handle only positive integers.

Let's denote the following variable:
T —> Transaction ID

Constraints:

- 0 <= T <= 2^31 - 1
- If T is a palindrome, the utility will multiply all its digits together.
- If T is not a palindrome, the utility will sum all its digits.

Test Case 1: Minimum value (T = 0)
Input: T = 0
Expected Output: T is a palindrome, product of digits = 0

Test Case 2: Maximum value (T = 2^31 - 1)
Input: T = 2147483647
Expected Output: T is not a palindrome, sum of digits = 46

Test Case 3: Single-digit palindrome (T = 5)
Input: T = 5
Expected Output: T is a palindrome, product of digits = 5

Test Case 4: Three-digit palindrome (T = 121)
Input: T = 121
Expected Output: T is a palindrome, product of digits = 2

Test Case 5: Three-digit non-palindrome (T = 345)
Input: T = 345
Expected Output: T is not a palindrome, sum of digits = 12

Test Case 6: Large palindrome (T = 123454321)
Input: T = 123454321
Expected Output: T is a palindrome, product of digits = 2880

Test Case 7: Large non-palindrome (T = 987654321)
Input: T = 987654321
Expected Output: T is not a palindrome, sum of digits = 45

Test Case 8: Negative number

Input: T = -120

Expected Output: Invalid input

Test Case 9: Edge case with all nines (T = 99999)

Input: T = 99999

Expected Output: T is a palindrome, product of digits = 59049

Test Case 10: Negative number

Input: T = -12345

Expected Output: Invalid input

13. Develop a program to generate Floyd's Triangle based on the number of rows specified by the user. This program will be used in a mathematics class to visually demonstrate sequential number patterns and triangular number structures.

Let's denote the following variable:
n → number of rows in Floyd's Triangle

The number in Floyd's triangle should alternate between 1 and 0, where the first row always starts with the number 1.

Constraints:

- 0<= n<= 2^31 - 1
- If n<=0 → no output
- For a given row i (where 1<=i<=n)

Test Case 1: Minimum valid input

Input: n = 1
Expected Output:
1

Test Case 2: Small number of rows

Input: n = 3
Expected Output:
1
0 1
0 1 0

Test Case 3: Zero rows (no output expected)

Input: n = 0
Expected Output: (no output)

Test Case 4: Negative number (no output expected)

Input: n = -5
Expected Output: (no output)

Test Case 5: Large number of rows

Input: n = 10
Expected Output:
1

0 1
0 1 0
1 0 1 0
1 0 1 0 1
0 1 0 1 0 1
0 1 0 1 0 1 0
1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1

Test Case 6: Alternating pattern for large n

Input: n = 5

Expected Output:

1
0 1
0 1 0
1 0 1 0
1 0 1 0 1

14. Imagine a warehouse inventory that stores the products supplied in Walmart. The inventory management system tracks the number of units sold for each product. Develop a system that identifies the products that are selling exceptionally well, based on a unique criteria.

Criteria:
High_Selling_Index - The number of units sold equals to the sum of the factorials of their digits.
Low_Selling_Index - The number of units sold is not equal to the sum of the factorials of their digits.

Let's denote the following variable:

N → number of units sold in a product
Sum → calculate the sum of factorials of the digit
Flag → Set the flag variable to true, if N=Sum (High_Selling_Index)
Set the flag variable to false, if N≠Sum (Low_Selling_Index)

Constraints:

- 0 <= N <= 2^31 - 1

Test Case 1:
Input: 1
Expected Output: Number is a strong

Test Case 2:
Input: 145
Expected Output: Number is a strong

Test Case 3:
Input: 40585
Expected Output: Number is a strong
Test Case 4:
Input: 123
Expected Output: Number is not a strong
Test Case 5:
Input: 7
Expected Output: Number is not a strong

Test Case 6:
Input: 0
Expected Output: Number is a strong
Test Case 7:
Input: 1457
Expected Output: Number is not a strong
Test Case 8:
Input: 40586
Expected Output: Number is not a strong
Test Case 9:
Input: 9474
Expected Output: Number is not a strong
Test Case 10:
Input: 146
Expected Output: Number is not a strong

15. Develop a program to determine whether a given number is a Kaprekar number or not. This function will be used in a mathematical analysis tool to identify specific properties of numbers, specifically focusing on the properties defined by Kaprekar numbers. For the given number n, find the square. Split the square at different positions and see if the sum of two parts in any split becomes equal to n. Then the given number is said to be a Kaprekar number.

Let's denote the following variable:
n → The number to be checked for the Kaprekar property.

Example:
Input : n = 45
Output : Yes
Explanation : 45*45 = 2025 and 20 + 25 is 45

Input : n = 13
Output : No
Explanation : 13*13= 169. Neither 16 + 9 nor 1 + 69 is equal to 13

Input : n = 10
Output : No
Explanation: 10*10 = 100. It is not a Kaprekar number even if sum of 100 + 0 is 100. This is because of the condition that none of the parts should have value 0.

Constraints
n>=1

Test Case 1:
Enter a number:45
Expected Output: yes

Test Case 2:
Enter a number :15
Expected Output: no

Test Case 3:
Enter a number: 55
Expected Output: yes

Test Case 4:

Enter a number: 1

Expected Output: yes

Test Case 5:

Enter a number: 297

Expected Output: yes

Test Case 6:

Enter a number: −5

Expected Output: no output

Test Case 7:

Enter a number: 0

Expected Output: no output

Test Case 8:

Enter a number: 703

Expected Output: yes

Test Case 9:

Enter a number: 10

Expected Output: no

Test Case 10:

Enter a number: 999

Expected Output: yes

16. Develop a program to calculate and display the sum of a series. Consider the first term as 9, whereas the next term is generated by concatenating the digit 9 to the previous term. The series will be generated as per the number of terms (N). This program is intended to be used in a financial analysis context where you need to compute cumulative amounts based on a given sequence of increasing values.

Let's denote the following variable:

N → Number of terms in the series.

For N=1, the series is: 9
For N=2, the series is: 9, 99
For N=3, the series is: 9, 99, 999
And so on…

Constraints:

- 1<= N <= 10
- If N<=0, print "Invalid input"

Test Case 1: Minimum Boundary

Input: N=1
Expected Output: Sum = 9

Test Case 2: Small Value

Input: N=2
Expected Output: Sum = 108

Test Case 3: Medium Value

Input: N=3
Expected Output: Sum =1107

Test Case 4: Larger Value

Input: N=7
Expected Output: Sum = 11111103

Test Case 5: Larger Value

Input: N=8
Expected Output: Sum = 111111102

Test Case 6: Larger Value

Input: N=10
Expected Output: Sum =11111111100

Test Case 7:

Input: N=3

Expected Output: 1107

Test Case 8: Invalid Input

Input: N=0

Expected Output: Invalid input

Test Case 9: Invalid Input

Input: N = −1

Expected Output: Invalid input

Test Case 10:

Input: N=11

Expected Output: 111111111099