

დავავლებაში მოცემულია 10 რთული დონის ალგორითმული ამოცანა.
თითოეულ ამოცანას გააჩნია თავისი პირობა და 5 test case.

თითოეული ამოცანისთვის უნდა შექმნათ ცალკე პითონის ან js-ის (ამ ორი ენიდან, რომლითაც გინდათ დაწერეთ ამოცანის კოდი) ფაილი და სამივე test case-ზე მიიღოთ იგივე შედეგი.
შესაძლოა ამოცანამ მოცემულ 5 test case-ზე იგივე შედეგი დააბრუნოს, მაგრამ სხვა მაგალითებზე სწორად არ მუშაობდეს. ასეთ დროს ის შესრულებულად არ ჩაითვლება.

!!! ყველა ამოცანას უნდა ქონდეს კომენტარები, სადაც ახსნილი იქნება კოდის რა ნაწილი რა დავალებას ასრულებს.

!!! ამოცანების შესრულებისას დაუშვებელია browser-ის გამოყენება.

1. Check If Two Strings Are Anagrams

Task:

Write a function that determines if two strings are anagrams of each other.

Test Cases:

1. Input: ("listen", "silent") → Output: True
 2. Input: ("triangle", "integral") → Output: True
 3. Input: ("apple", "pale") → Output: False
 4. Input: ("a", "a") → Output: True
 5. Input: ("rat", "car") → Output: False
-

2. Count the Number of Unique Words in a Text

Task:

Write a function that counts the number of unique words in a string, ignoring case sensitivity and punctuation.

Test Cases:

1. Input: "The quick brown fox jumps over the lazy dog" → Output: 8
2. Input: "Hello hello world!" → Output: 2
3. Input: "" → Output: 0
4. Input: "Python is fun. Python is cool." → Output: 4
5. Input: "One word" → Output: 2

3. Reverse the Order of Words in a Sentence

Task:

Write a function that takes a sentence and reverses the order of its words.

Test Cases:

1. Input: "Hello World" → Output: "World Hello"
 2. Input: "Python is great" → Output: "great is Python"
 3. Input: "a b c" → Output: "c b a"
 4. Input: "" → Output: ""
 5. Input: " Spaces " → Output: "Spaces"
-

4. Generate Pascal's Triangle Up to a Given Row

Task:

Write a function to generate Pascal's Triangle up to the specified number of rows.

Test Cases:

1. Input: 1 → Output: `[[1]]`
 2. Input: 2 → Output: `[[1], [1, 1]]`
 3. Input: 3 → Output: `[[1], [1, 1], [1, 2, 1]]`
 4. Input: 5 → Output: `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]`
 5. Input: 0 → Output: `[]`
-

5. Encrypt and Decrypt Strings Using Caesar Cipher

Task:

Write a function to encrypt strings using a Caesar cipher with a given shift value.

Test Cases:

1. Input: ("abc", 2) → Output: "cde"
2. Input: ("xyz", 3) → Output: "abc"
3. Input: ("Hello, World!", 5) → Output: "Mjqqt, Btwqi!"
4. Input: ("Python", 0) → Output: "Python"
5. Input: ("abc", -1) → Output: "zab"

6. Validate a Sudoku Solution

Task Explanation:

Write a function that validates whether a given Sudoku board is solved correctly. The board is represented as a 2D array of size **9x9**. A valid solution must meet these conditions:

1. Each row contains numbers 1-9 without repetition.
2. Each column contains numbers 1-9 without repetition.
3. Each 3x3 subgrid contains numbers 1-9 without repetition.

Test Cases:

1. Valid

Input:

```
[
  [5, 3, 4, 6, 7, 8, 9, 1, 2],
  [6, 7, 2, 1, 9, 5, 3, 4, 8],
  [1, 9, 8, 3, 4, 2, 5, 6, 7],
  [8, 5, 9, 7, 6, 1, 4, 2, 3],
  [4, 2, 6, 8, 5, 3, 7, 9, 1],
  [7, 1, 3, 9, 2, 4, 8, 5, 6],
  [9, 6, 1, 5, 3, 7, 2, 8, 4],
  [2, 8, 7, 4, 1, 9, 6, 3, 5],
  [3, 4, 5, 2, 8, 6, 1, 7, 9]
]
```

Expected Output: **True**

2. Invalid Board with Duplicate in a Row

Input:

```
[
  [5, 3, 4, 6, 7, 8, 9, 1, 2],
  [6, 7, 2, 1, 9, 5, 3, 4, 8],
  [1, 9, 8, 3, 4, 2, 5, 6, 7],
```

```
[8, 5, 9, 7, 6, 1, 4, 2, 3],
[4, 2, 6, 8, 5, 3, 7, 9, 1],
[7, 1, 3, 9, 2, 4, 8, 5, 6],
[9, 6, 1, 5, 3, 7, 2, 8, 4],
[2, 8, 7, 4, 1, 9, 6, 3, 5],
[3, 4, 5, 2, 8, 6, 1, 7, 7] # Invalid: Two 7s in the last row
]
```

Expected Output: `False`

3. Invalid Board with Duplicate in a Column

Input:

```
[
    [5, 3, 4, 6, 7, 8, 9, 1, 2],
    [6, 7, 2, 1, 9, 5, 3, 4, 8],
    [1, 9, 8, 3, 4, 2, 5, 6, 7],
    [8, 5, 9, 7, 6, 1, 4, 2, 3],
    [4, 2, 6, 8, 5, 3, 7, 9, 1],
    [7, 1, 3, 9, 2, 4, 8, 5, 6],
    [9, 6, 1, 5, 3, 7, 2, 8, 4],
    [2, 8, 7, 4, 1, 9, 6, 3, 5],
    [3, 4, 5, 2, 8, 6, 9, 7, 9] # Invalid: Two 9s in the last
column
]
```

Expected Output: `False`

4. Invalid Board with Duplicate in a 3x3 Subgrid

Input:

```
[
    [5, 3, 4, 6, 7, 8, 9, 1, 2],
    [6, 7, 2, 1, 9, 5, 3, 4, 8],
    [1, 9, 8, 3, 4, 2, 5, 6, 7],
    [8, 5, 9, 7, 6, 1, 4, 2, 3],
    [4, 2, 6, 8, 5, 3, 7, 9, 1],
    [7, 1, 3, 9, 2, 4, 8, 5, 6],
    [9, 6, 1, 5, 3, 7, 2, 8, 4],
    [2, 8, 7, 4, 1, 9, 6, 3, 5],
    [3, 4, 5, 2, 8, 6, 1, 1, 9] # Invalid: Two 1s in the
bottom-right subgrid
]
```

Expected Output: `False`

5. Partially Filled Board

Input:

```
[
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]
```

Expected Output: `False` (Not completely filled, so cannot validate)

7. Find All Prime Numbers in a Given Range

Task:

Write a function that takes two integers, `start` and `end`, and returns a list of all prime numbers in the range `[start, end]`. A prime number is a number greater than 1 that has no divisors other than 1 and itself.

Test Cases:

- Input: `start = 10, end = 20`
Output: `[11, 13, 17, 19]`
 - Input: `start = 1, end = 10`
Output: `[2, 3, 5, 7]`
 - Input: `start = 20, end = 30`
Output: `[23, 29]`
 - Input: `start = 24, end = 25`
Output: `[]`
 - Input: `start = 1, end = 1`
Output: `[]`
-

8. Find the Kth Smallest Element in an Array

Task:

Write a function that finds the `k`-th smallest element in an unsorted array.

Test Cases:

- Input: `arr = [3, 2, 1, 5, 6, 4], k = 2`
Output: `2`
Explanation: The 2nd smallest element in the array is `2`.
- Input: `arr = [3, 2, 1, 5, 6, 4], k = 4`
Output: `4`
Explanation: The 4th smallest element in the array is `4`.

- **Input:** `arr = [7, 10, 4, 3, 20, 15], k = 3`
Output: 7
Explanation: The 3rd smallest element in the array is 7.
 - **Input:** `arr = [1, 2, 3, 4, 5], k = 1`
Output: 1
Explanation: The 1st smallest element is 1.
 - **Input:** `arr = [1, 2, 3, 4, 5], k = 5`
Output: 5
Explanation: The 5th smallest element is 5.
-

8. Xbonacci Sequence

Task:

Write a function that generates the Xbonacci sequence. The Xbonacci sequence is a generalization of the Fibonacci sequence, where each number is the sum of the previous `X` numbers in the sequence.

For example, if `X = 3` and the initial sequence is `[1, 1, 1]`, the sequence will proceed as follows:

- The 4th number is the sum of the previous 3: $1 + 1 + 1 = 3$.
- The 5th number is the sum of the previous 3: $1 + 1 + 3 = 5$.
- The 6th number is the sum of the previous 3: $1 + 3 + 5 = 9$.

Your function should take two arguments:

1. `X`: The number of previous terms to sum.
2. `n`: The number of terms to generate.

Test Cases:

- **Input:** `X = 3, n = 10`
Output: `[1, 1, 1, 3, 5, 9, 17, 31, 57, 105]`
Explanation: The first 10 numbers of the Xbonacci sequence starting with `[1, 1, 1]` and `X = 3`.
- **Input:** `X = 2, n = 10`
Output: `[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]`
Explanation: This is the Fibonacci sequence, where each number is the sum of the previous two numbers.
- **Input:** `X = 4, n = 6`
Output: `[1, 1, 1, 1, 4, 8]`

Explanation: Starting with $[1, 1, 1, 1]$, the next terms are calculated by summing the last 4 terms.

- **Input:** $X = 5, n = 8$

Output: $[1, 1, 1, 1, 1, 5, 9, 17]$

Explanation: The sequence starts with five 1s, and each subsequent number is the sum of the previous 5 numbers.

- **Input:** $X = 3, n = 1$

Output: $[1]$

Explanation: Only one term in the sequence is required, so the output is $[1]$.

9. Primorial of a Number

Task:

Write a function that calculates the **primorial** of a number. The **primorial** of a number n is the product of all prime numbers less than or equal to n . For example, the primorial of 5 is the product of the primes less than or equal to 5: $2 * 3 * 5 = 30$.

Your function should take an integer n and return the primorial of that number.

Test Cases:

- **Input:** $n = 5$

Output: 30

Explanation: The prime numbers less than or equal to 5 are 2, 3, and 5. Their product is $2 * 3 * 5 = 30$.

- **Input:** $n = 10$

Output: 210

Explanation: The prime numbers less than or equal to 10 are 2, 3, 5, and 7. Their product is $2 * 3 * 5 * 7 = 210$.

- **Input:** $n = 1$

Output: 1

Explanation: There are no primes less than or equal to 1, so the primorial is 1 by definition.

- **Input:** $n = 7$

Output: 210

Explanation: The prime numbers less than or equal to 7 are 2, 3, 5, and 7. Their product is $2 * 3 * 5 * 7 = 210$.

- **Input:** $n = 11$

Output: 2310

Explanation: The prime numbers less than or equal to 11 are 2, 3, 5, 7, and 11. Their product is $2 * 3 * 5 * 7 * 11 = 2310$.

10. Sum Fractions Using GCD and LCM

Task:

Write a function that sums two fractions and returns the result in its simplest form. The fractions will be given as two tuples, where each tuple consists of two integers: the numerator and the denominator. To simplify the result, you need to compute the Least Common Multiple (LCM) and Greatest Common Divisor (GCD) of the denominators.

Then, simplify the result by dividing both the numerator and denominator by their GCD.

Test Cases:

- **Input:** `frac1 = (1, 2), frac2 = (1, 3)`
Output: `(5, 6)`
Explanation: The LCM of 2 and 3 is 6. The sum is $(1 * 3) / 6 + (1 * 2) / 6 = 3/6 + 2/6 = 5/6$.
- **Input:** `frac1 = (1, 4), frac2 = (1, 4)`
Output: `(1, 2)`
Explanation: The LCM of 4 and 4 is 4. The sum is $(1/4 + 1/4) = 2/4 = 1/2$.
- **Input:** `frac1 = (2, 5), frac2 = (1, 5)`
Output: `(3, 5)`
Explanation: The LCM of 5 and 5 is 5. The sum is $(2/5 + 1/5) = 3/5$.
- **Input:** `frac1 = (3, 4), frac2 = (5, 6)`
Output: `(19, 12)`
Explanation: The LCM of 4 and 6 is 12. The sum is $(3 * 3) / 12 + (5 * 2) / 12 = 9/12 + 10/12 = 19/12$.
- **Input:** `frac1 = (5, 12), frac2 = (7, 15)`
Output: `(139, 60)`
Explanation: The LCM of 12 and 15 is 60. The sum is $(5 * 5) / 60 + (7 * 4) / 60 = 25/60 + 28/60 = 53/60$.

To calculate the **Greatest Common Divisor (GCD)** of two numbers in Python, you can use the **Euclidean algorithm**, which is an efficient way to find the GCD. Here's an explanation of how to build the GCD function and how the algorithm works:

Understanding the Euclidean Algorithm

The Euclidean algorithm finds the GCD of two integers `a` and `b` by repeatedly applying the following steps:

1. **Divide `a` by `b`** and calculate the remainder (`a % b`).
2. Replace `a` with `b` and `b` with the remainder from the previous step.

3. Repeat the process until the remainder becomes 0. The GCD will be the last non-zero value of **b**.

Steps to Build GCD Function in Python

1. **Initial Check:** If **b** becomes 0, then **a** is the GCD.
2. **Iterate:** Continue the division and remainder operation until the remainder becomes 0.

Understanding and Building LCM (Least Common Multiple) in Python


The **Least Common Multiple (LCM)** of two integers is the smallest positive integer that is divisible by both numbers. The relationship between the LCM and GCD of two numbers can be used to efficiently compute the LCM.


$$\text{LCM}(a, b) = |a * b| / \text{GCD}(a, b)$$


შეფასების სისტემა

თითოეული ამოცანა ფასდება მაქსიმუმ 3 ქულით.

ბარათები ქულების რაოდენობის მიხედვით:

ქულა >= 27: 

ქულა >= 24: 

ქულა >= 21: 

ქულა < 21: 