Activity Questions

Problem-1

Accept a positive integer `x` as input and print the number of factors of `x`. Divide your test cases into two classes. The first class should have numbers which are perfect squares, while the second class should have numbers which are not perfect squares. Do you observe any pattern?

Answer

```python
x = int(input())
count = 0
for i in range(1, x + 1):
    if x % i == 0:
        count += 1
print(count)

# Test cases
# squares = [1, 4, 6, 9, 25, 36, 49, 64, 81, 100]
# non_squares = [5, 10, 15, 18, 20, 32, 44, 78, 80, 95]
```

- For perfect squares, the number of factors is odd.
- For all other numbers, the number of factors is even.

Problem-2

Consider the following sequence:

$$7, 77, 777, 7777, \cdots$$

Let x be the smallest element in this sequence that is divisible by 2003. How many digits does x have?

Answer

```python
seq = 7
while seq % 2003 != 0:
    seq = seq * 10 + 7
print(len(str(seq)))
```

Problem-3

A bot starts at the origin and can make the following moves:

- UP
- DOWN
- LEFT
- RIGHT

Each move has a magnitude of 1 unit. You are given the sequence of moves made by the bot. The first entry in the sequence is always START while the last entry in the sequence is always STOP. A sample sequence is given below:

```
START
UP
RIGHT
LEFT
LEFT
DOWN
UP
STOP
```

Find the distance of the bot from the origin. By distance we mean the Euclidean distance.

Answer

```python
x, y = 0, 0          # start at the origin
seq = input()
while seq != 'STOP':
    if seq == 'UP':
        y += 1
    if seq == 'DOWN':
        y -= 1
    if seq == 'LEFT':
        x -= 1
    if seq == 'RIGHT':
        x += 1
    seq = input()
# pow is a built in function
# pow(x, y) is equivalent to x ** y
dist = pow(x ** 2 + y ** 2, 0.5)
# rounding off to two decimal places
print(f'{dist:0.2f}')
```

Problem-4

Accept two positive integer `n` and `p` as input. A string is formed by writing all the integers from 1 to `n` side by side on the same line. Find the number that is at the `p`th position in this sequence. For example, if `n = 15`, the sequence will look like this: `123456789101112131415`. The 11^{th} number in this sequence is 0.

Answer

```python
n, p = int(input()), int(input())

S = ''
for i in range(1, n + 1):
    S += str(i)
# Assume that p is less than or equal to len(S)
print(S[p - 1])
```

Problem-5

When an unbiased dice with 12 faces having the numbers from 1 to 12 is rolled, the probability of obtaining a prime number is $5/12$. Set up a computational experiment to verify this.

Answer

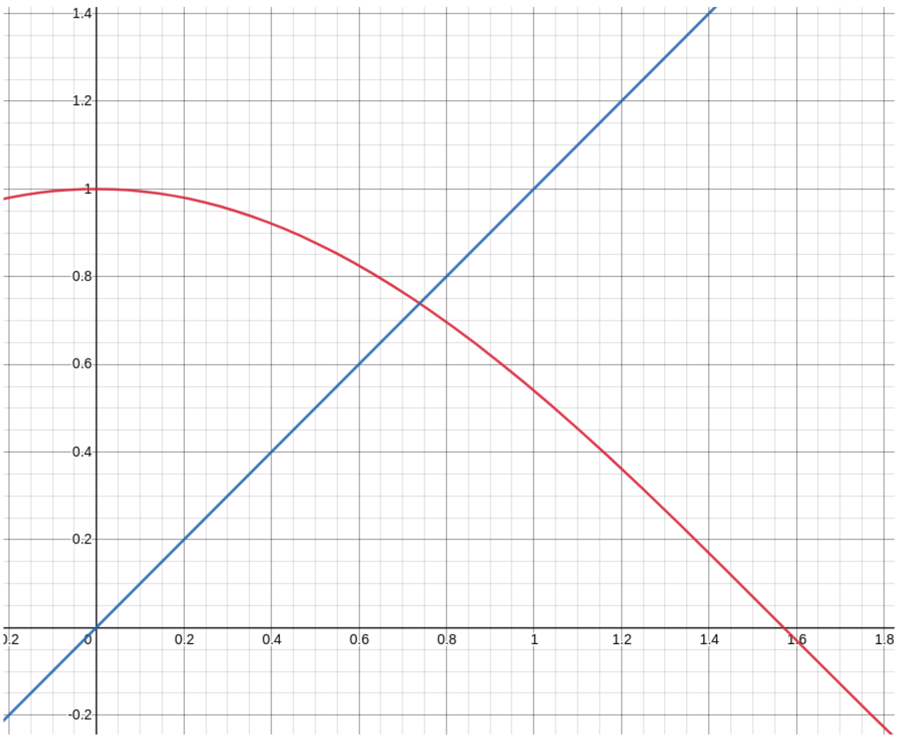
```python
import random

rolls = [ ]
# Roll the dice 100,000 times
for i in range(100_000):
    roll = random.randint(1, 12)
    rolls.append(roll)

# Count the number of times primes occur
count = 0
primes = [2, 3, 5, 7, 11]
for roll in rolls:
    if roll in primes:
        count += 1

# Report the results
print(f'rolls = {len(rolls)}')
print(f'actual = {(5 / 12):0.3f}')
print(f'estimate = {(count / len(rolls)):0.3f}')
```

Problem-6

Consider the following sequence.

$$\begin{aligned} x_0 &= 0 \\ x_n &= \cos(x_{n-1}) \end{aligned}$$

- It is known that this sequence converges to a limit. Find an approximate value of the limit of this sequence. Use a tolerance of 0.00001 between successive terms in this sequence to terminate the iteration. Round off your answer to four decimal places.
- How many iterations are needed to achieve this approximation?
- How is the limit related to the following graph? Can you see the connection?



Answer

```python
from math import cos

x_prev, x_curr, count = 1000, 0, 0
while abs(x_prev - x_curr) > 0.00001:
    x_prev, x_curr = x_curr, cos(x_curr)
    count += 1
print(f'Number of iterations needed = {count}')
print(f'Approximate value = {x_curr:0.4f}')
```

The graph on display has two functions:

- $y = \cos(x)$
- $y = x$

The point of intersection of this graph is the solution to the following equation:

$$x - \cos(x) = 0$$

The solution to this equation is called a fixed point for the function $f(x) = \cos(x)$. The code given above is a way to find an approximate value of the fixed point and is called the fixed point iteration method! The mathematically courageous folks can go through this resource.

Problem-7

Consider the following equation. x_n and y_n are integers:

$$\left(\sqrt{2}-1\right)^n = x_n + \sqrt{2}\cdot y_n$$

For $n = 1$, $x_1 = -1$ and $y_1 = 1$. Find the value of $\frac{-x_{100}}{y_{100}}$. Compare it with $\sqrt{2}$. What do you observe?

Answer

If $(\sqrt{2}-1)^n = x_n + y_n \cdot \sqrt{2}$, then:

$$\begin{aligned}(\sqrt{2}-1)^{n+1} &= (x_n + y_n \cdot \sqrt{2}) \cdot (\sqrt{2}-1)\\ &= (2y_n - x_n) + (x_n - y_n) \cdot \sqrt{2}\\ &= x_{n+1} + y_{n+1} \cdot \sqrt{2}\end{aligned}$$

The equation given above defines what is called a recurrence relation: each new term in the sequence is a function of the preceding terms. In this sequence we have $x_1 = -1, y_1 = 1$. For $n > 0$, the pair of equations given below forms the recurrence relation:

$$\begin{aligned}x_{n+1} &= 2y_n - x_n\\ y_{n+1} &= x_n - y_n\end{aligned}$$

Loops are useful tools when it comes to computing terms in such sequences:

```python
n = int(input())    # sequence length
x_n, y_n = -1, 1    # x_1 and y_1
for i in range(n - 1):
    x_n, y_n = 2 * y_n - x_n, x_n - y_n
```

This in turn provides a way to approximate $\sqrt{2}$ using rational numbers:

$$\sqrt{2} \approx \frac{-x_n}{y_n}$$

As n becomes large, this approximation will become increasingly accurate. For $n = 100$, we have $\frac{-x_{100}}{y_{100}}$:

$$\frac{228725309250740208744750893347264645481}{161733217200188571081311986634082331709}$$

This is pretty close to $\sqrt{2}$.

Problem-8

Accept a multiple of 3 as input and print the following pattern. For $n = 9$, the pattern is as follows:

```
|123|456|789|
```

Answer

```python
n = int(input())

for i in range(1, n + 1, 3):
    print(f'|{i}{i + 1}{i + 2}', end = '')
print('|')
```

Problem-9

Write a program to print the following table. The numbers on each line are separated by the tab character.

```
	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100
```

Answer

This is a multiplication table.

```python
print('\t', end = '')
for i in range(1, 11):
    print(i, end = '\t')
print()
for i in range(1, 11):
    print(i, end = '\t')
    for j in range(1, 11):
        print(i * j, end = '\t')
    print()
```

Problem-10

There are five boxes arranged from left to right. You keep adding a variable number of coins sequentially in each box. Start from box-1 and keep going right. Once you reach the last box, head back to box-1 and then keep adding coins. In any given turn, the number of coins added to a box is always less than 10.

Find the box which has the maximum number of coins. If there are two boxes which have the same maximum number of coins, output the smaller of the two box numbers. The sequence of coins is represented by a string. For example, if the input is `'3972894910'`, this is how coins are added:

Box	Coins
1	3
2	9
3	7
4	2
5	8
1	3 + 9 = 12
2	9 + 4 = 13
3	7 + 9 = 16
4	2 + 1 = 3
5	8 + 0 = 8

In this case, 3 is the output as box-3 has the maximum number of coins in it.

Answer

```python
boxes = [ ]
for i in range(5):
    boxes.append(0)

# Add coins to boxes
coins = input()
# Search the web to learn more about enumerate
for index, coin in enumerate(coins):
    # Use mod operator to cycle back
    box_index = (index + 1) % 5 - 1
    boxes[box_index] += int(coin)

# Find box with maximum coins
max_coins, max_box = boxes[0], 0
for box_index, coins in enumerate(boxes):
    if coins > max_coins:
        max_coins = coins
        max_box = box_index
```

```

print(max_box + 1)
```

Problem-11

Consider a bouncing ball. It is dropped from a height of 10 meters. For every bounce, it looses $1/10^{th}$ of its height. For example, after the first bounce, it will loose 1 meter. This means, it will only go as high as 9 meters. How many bounces does it take for its height to drop below 1 meter.

Answer

```python
h, bounces = 10, 0
print('—' * 20)     # added for aesthetic reasons
print('Bounces\t|\tHeight')
print('—' * 20)
while h > 1:
    # rebound height is (1 - 0.1) * h or 0.9 * h
    h = 0.9 * h
    bounces += 1
    print(f'{bounces}\t\t|\t{h:.2f}')
print('—' * 20)
print(f'It takes {bounces} bounces for the height to drop below 1 meter')
```

Problem-12

Consider the following polynomial:

$$f(x) = x^2 + x + 41$$

$f(0)$ is divisible by 41. Find the smallest positive integer x for which $f(x)$ is divisible by 41.

Answer

```python
x = 1
while (x ** 2 + x + 41) % 41 != 0:
    x += 1
print(x)
# check correctness of answer
# assert statement may be new to you
# assert False will throw an AssertionError
assert (x ** 2 + x + 41) % 41 == 0
```

Problem-13

Simulating Nim

Simulate the following game. There are n marbles in a pile to begin with. Two players take turns to play the game. Each player can remove at most four marbles from the pile, but has to remove at least one marble. The player who removes the last marble is the loser.

One of the players will be the computer. The other player will be a human. For every turn, you must accept an input from the user which will be the number of marbles that the user has to remove from the pile. If the user makes a wrong move — more than four marbles or less than one marble — flash a warning message and keep accepting input until a valid input is received.

For simulating the computer's moves, use the `random` library. This game has a winning strategy for the person who makes the first move. Can you figure it out?

Answer

```python
import random
# Assumptions
# (1) Computer starts first
# (2) n > 4
# (3) User always enters an integer

# no. of marbles
n = int(input('Enter the number of marbles you wish to play with: '))
print('------------------')
print(f'Game | {n} marbles')
print('------------------')
### Start ###
turn = 'computer'
# Game continues as long as there are marbles in the pile
while n > 0:
    # check whose turn it is
    if turn == 'human':
        human = int(input('Your turn: '))
        # second condition is needed
        # you can't remove more marbles than what is there in the pile
        if not(1 <= human <= 4) or (n < human):
            print('Invalid move')
            continue
        # successful move
        n -= human
        # it is now the computer's turn
        turn = 'computer'
        # After every turn, notify the user
        print(f'##### Marbles remaining: {n}')
    elif turn == 'computer':
        computer = random.randint(1, 4)
        if n < computer:
            continue
        n -= computer
        turn = 'human'
        print(f'Computer\'s choice: {computer}')
        print(f'##### Marbles remaining: {n}')

```

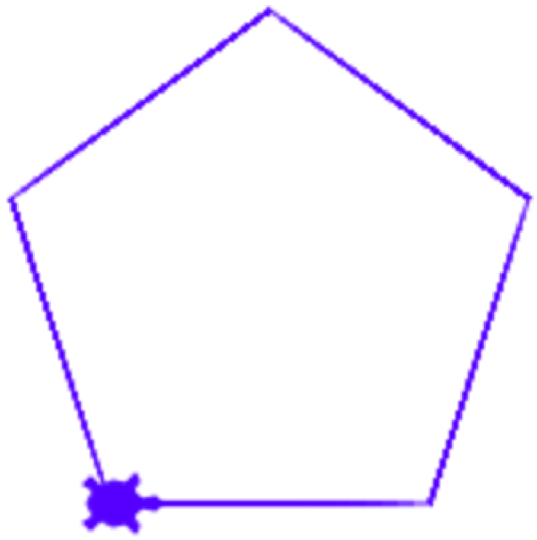
```python
# At this stage, n = 0
# If it is your turn now, then the last marble was removed by the computer
# So you win
if turn == 'human':
    print('You win!')
# If it is the computer's turn now, then the last marble was removed by you
# So the computer wins
if turn == 'computer':
    print('Computer wins!')
```

Problem-14

Accept a value of n and animate a regular polygon of n sides using Turtle. A regular polygon of n sides is also called an n-gon. You need a `Python (with Turtle)` repl to code this. This can be selected from the drop down menu while choosing the language of your repl. You may have to go deep down the menu. Here is some boilerplate code to get you started:

```python
import turtle

### You don't need to worry about these lines
t = turtle.Turtle()
t.shape('turtle')
t.color('blue')

### Draw an equilateral triangle
t.forward(100)
t.left(120)
t.forward(100)
t.left(120)
t.forward(100)
t.left(120)
```

Sample output for $n = 5$:



Answer

```python
import turtle

### You don't need to worry about these lines
t = turtle.Turtle()
t.shape('turtle')
t.color('blue')

# Number of sides
```

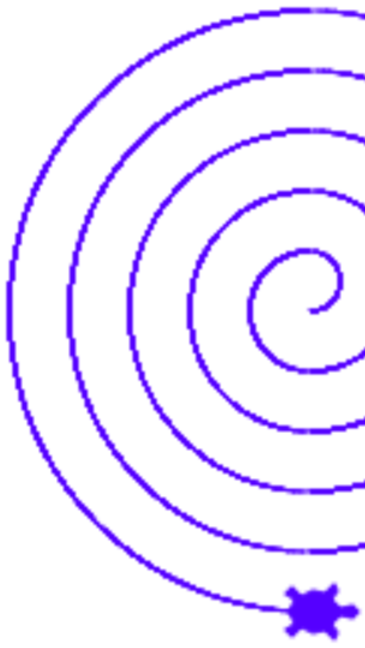
```python
n = int(input())

# Angle by which the turtle has to turn
angle = 360 / n
while n > 0:
    t.forward(100)
    t.left(angle)
    n -= 1
```

Problem-15

Animate a spiral using Turtle. Here is some boilerplate code to get you started:

```python
import turtle

t = turtle.Turtle()
t.shape('turtle')
t.color('blue')

# radius, angle, steps
# ignore the steps parameter for now
t.circle(100, 180, steps = 100)
```

Your output should look as follows:



Answer

```python
import turtle

t = turtle.Turtle()
t.shape('turtle')
t.color('blue')

# radius, angle, steps
# ignore the steps parameter for now

n = 1
while n <= 10:
    t.circle(10 * n, 180, steps = 100)
    n += 1
```

Problem-16

Simulate a bouncing ball. Here is some boilerplate code to get you started:

```python
import turtle
import time

t = turtle.Turtle()
t.shape('circle')
t.color('blue')

t.penup()
t.goto(0, -90)
t.goto(0, 100)
```

The ball should keep bouncing forever.

Answer

```python
import turtle
import time

t = turtle.Turtle()
t.shape('circle')
t.color('blue')

t.penup()
while True:
    t.goto(0, -90)
    t.goto(0, 100)
```

Problem-17

Given a list of integers, find the number of times an integer x appears in the list. The input will have two lines.

- First line will be a comma-separated sequence of integers.
- Second line will be an integer x.

The output should be a non-negative integer.

Test Cases

Input

```
1,2,3,4,5
2
```

Output

```
1
```

Input

```
1,1,2,3,4,1,2,1
1
```

Output

```
4
```

Input

```
1,2,3,4,5
6
```

Output

```
0
```

Input

```
1,1,1,1,1,1,1
1
```

Output

```
7
```

Answer

```python
L = [ ]
for num in input().split(','):
    L.append(int(num))
x = int(input())
count = 0
for y in L:
    if y == x:
        count += 1
print(count)
```

Problem-18

Accept a sequence of space-separated integers as input. Print `ascending` if the sequence is in ascending order. Print `descending` if the sequence is in descending order. Print `unsorted` if the sequence is not in any order. You can assume that all numbers in the sequence are distinct.

Test Cases

Input	Output
`1 2 3 4 5`	`ascending`
`-1 10 0 9 39 900`	`unsorted`
`100 90 -10 -100`	`descending`

Answer

```python
L = [ ]
for num in input().split(' '):
    L.append(int(num))

# Assumption given in question
# All numbers are distinct

ascending, descending = True, True
for i in range(1, len(L)):
    if L[i] > L[i - 1]:
        descending = False
    elif L[i] < L[i - 1]:
        ascending = False

if ascending:
    print('ascending')
elif descending:
    print('descending')
else:
    print('unsorted')
```

Problem-19

Two vectors are parallel to each other if one can be expressed as a scalar multiple of the other. For example, $[1, 2, 3]$ and $[4, 8, 12]$ are parallel because $[4, 8, 12] = 4 \cdot [1, 2, 3]$. The expression $4 \cdot [1, 2, 3]$ stands for the vector obtained by multiplying each element of $[1, 2, 3]$ by 4. In general, two vectors u and v are parallel, if we there is a non-zero scalar c such that:

$$v = c \cdot u$$

Accept two vectors as input from the user and determine if they are parallel to each other.

Input	Output

`1 2 3 4`
`5 10 15 20`
	`parallel`

`1 0 2 3 4 9 1`
`10 0 20 -30 40 90 10`
	`not parallel`

Answer

```python
v1 = [ ]
for num in input().split():
    v1.append(float(num))

v2 = [ ]
for num in input().split():
    v2.append(float(num))

result = 'parallel'
# two vectors of unequal length can't be parallel
if len(v1) != len(v2):
    result = 'not parallel'
else:
    # the scalar
    c = 0
    length = len(v1)
    # Find the scalar
    for i in range(length):
        # condition is important
        # otherwise we will end up
        # dividing by zero
        if v1[i] != 0:
            c = v2[i] / v1[i]
            break
    # check if they are parallel
    for i in range(length):
        if v2[i] != c * v1[i]:
            result = 'not parallel'
            break
print(result)
```

Problem-20

Refer to the lecture on birthday paradox. Birthdays are numbered from 1 to 365 (both endpoints included). Given a list of birthdays of people in a room, find the number of pairs of people who share a birthday. The input will be a single line of comma-separated integers. The output will be a non-negative integer. For example:

Input

```
1,22,22,34,45,100,100,150
```

Output

```
2
```

Explanation

In this case, there are two pairs. The first pair corresponds to the number 22 while the second pair corresponds to the number 100.

Test Cases

Input	Output
`300,200,365,100,50,33,1,365`	`1`
`100,90,200,100,30,50,90,80,90`	`4`
`1,1,1,1,1`	`10`
`1,2,3,4,5`	`0`

Answer

```python
bdays = [ ]
for bday in input().split(','):
    bdays.append(int(bday))

count = 0
for i in range(len(bdays)):
    for j in range(i + 1, len(bdays)):
        if bdays[i] == bdays[j]:
            count += 1
print(count)
```

Problem-21

Write a program to accept a comma-separated sequence of distinct non-negative integers in any order and find the sum of `n` largest numbers in it. `n` is a positive integer and should also be taken as input from the user.

If the user enters only one value, the output will be the same as the input. If only two values are entered separated by commas, the output will be the sum of the two values. For more than 2 values entered, the program asks the user to enter the value of `n`.

Test Cases

Input	Output
`1,2`	`3`

`3,2,4,5,1`
`3`
	`12`

`1`	`1`
`1,0`	`1`

Answer

```python
L = [ ]
for num in input().split(','):
    L.append(int(num))

if len(L) == 1:
    print(L[0])
elif len(L) == 2:
    print(sum(L))
else:
    n = int(input())
    # assume that n is less than or equal to len(L)
    answer = 0
    while n > 0:
        max_num = L[0]
        for i in range(len(L)):
            if L[i] > max_num:
                max_num = L[i]
        L.remove(max_num)
        answer += max_num
        n -= 1
    print(answer)
```

Problem-22

Accept a square matrix as input and find the sum of all the elements in it. The input will have $n + 1$ lines. The first line will be a positive integer n. n Each of the next n lines will be a list of comma-separated integers.

Test cases

Input

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

Output

```
45
```

Input

```
2
1,0
0,-1
```

Output

```
0
```

Input

```
4
1,2,3,4
5,6,7,8
9,10,11,12
13,14,15,16
```

Output

```
136
```

Input

```
1
10
```

Output

```
10
```

Answer

```python
n = int(input())
M = [ ]
for i in range(n):
    M.append([ ])
    for num in input().split(','):
        M[-1].append(int(num))

msum = 0
for i in range(n):
    for j in range(n):
        msum += M[i][j]

print(msum)
```

Problem-23

The trace of a square matrix is the sum of elements on the main diagonal (from the upper left to the lower right). Accept a square matrix as input and compute its trace. The input will have $n + 1$ lines. The first line will be a positive integer n. Each of the next n lines will be a list of comma-separated integers.

Sample Test Case

Input

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

Output

```
15
```

Explanation

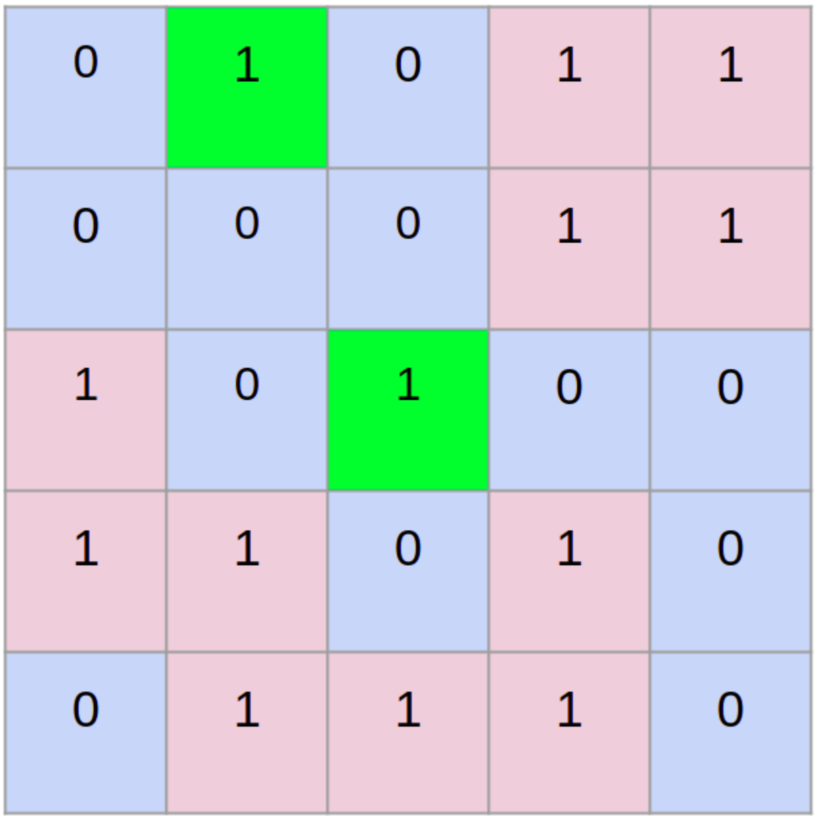
The first line of the input suggests that it is a 3×3 matrix. The trace is `1 + 5 + 9 = 15`

Answer

```python
n = int(input())
M = [ ]
for i in range(n):
    M.append([ ])
    for num in input().split(','):
        M[-1].append(int(num))

trace = 0
for i in range(n):
        trace += M[i][i]

print(trace)
```

Problem-24

Consider an $n \times n$ grid-world. `1` corresponds to land and `0` corresponds to water. An island is a piece of land surrounded by water on all its sides. Given an arbitrary world, find the number of islands in it. An instance of a 5×5 world is given below. It has two islands:



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

Accept the input as follows. The first line will contain a single integer n, which gives the dimension of the world. Each of the next n lines, will be a comma-separated sequence of length n which corresponds to one row in the world. The above grid world will have the following input-output:

Input

```
5
0,1,0,1,1
0,0,0,1,1
1,0,1,0,0
1,1,0,1,0
0,1,1,1,0
```

Output

```
2
```

Answer

```python
n = int(input())
world = [ ]
for i in range(n):
    world.append([ ])
    for num in input().split(','):
        world[-1].append(int(num))

# embed world in a new matrix
# pad it with zeros on all four sides
mod_world = [ ]
for i in range(n + 2):
    mod_world.append([ ])
    for j in range(n + 2):
        if 1 <= i <= n and 1 <= j <= n:
            mod_world[-1].append(world[i - 1][j - 1])
        else:
            mod_world[-1].append(0)

count = 0
for i in range(1, n + 1):
    for j in range(1, n + 1):
        # Potential island
        if mod_world[i][j] == 1:
            # UP, LEFT, RIGHT, DOWN
            if (  mod_world[i - 1][j] ==
                  mod_world[i][j - 1] ==
                  mod_world[i][j + 1] ==
                  mod_world[i + 1][j] == 0
            ):
                count += 1

print(count)
```

Problem-25

The lectures covered the multiplication of two square matrices. Let us now crank up the difficulty level. Accept two matrices A and B of dimensions $m \times n$ and $p \times q$ respectively as input. We wish to compute the product AB. If they are not of compatible dimensions, then print `IMPOSSIBLE`. If they can be multiplied, then print the product `AB`.

The first line of the input will be `m,n`. The next `m` lines will have a sequence of `n` comma-separated numbers on each line. This is followed by `p,q` in a new line. The next `p` lines will have a sequence of `q` comma-separated values on each line. If the dimensions are compatible, then the output will have `m` lines with a sequence of `q` comma-separated values on each line.

Input

```
3,4
1,2,3,4
5,6,7,8
9,10,11,12
4,2
13,14
15,16
17,18
19,20
```

Output

```
170,180
426,452
682,724
```

Answer

```python
dims = input().split(',')
m, n = int(dims[0]), int(dims[1])

A = [ ]
for i in range(m):
    A.append([ ])
    for num in input().split(','):
        A[-1].append(int(num))

dims = input().split(',')
p, q = int(dims[0]), int(dims[1])

B = [ ]
for i in range(p):
    B.append([ ])
    for num in input().split(','):
        B[-1].append(int(num))

if n != p:
    print('IMPOSSIBLE')
else:
    C = [ ]
    for i in range(m):
        C.append([ ])
        for j in range(q):
            C[-1].append(0)
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
            if j != q - 1:
                print(C[i][j], end = ',')
            else:
                print(C[i][j])
```