This is the **solutions manual** for the **final contest**.

This problem set has **13 problems**. The items in the problem list below are links to each problem. You can click a problem name to jump to its page in the PDF file.

Solution for Problem A: AM-GM Inequality

We first store a list of floats, then get the sum and product of all items in the list.

Solution:

```
n = int(input())
data = [float(i) for i in input().split()]

sum = 0
for x in data:
      sum = sum + x

product = 1
for x in data:
      product = product * x

print( "%.2f" % (sum/n) )
print( "%.2f" % (product**(1/n)) )
```

Solution for Problem B: Color Codes

We get a string as input.

Then, we go through each character of the string one-by-one and print out the output if we find an A.

This solution can be considered as an application of the Filter pattern.

Solution:

```
S = input()
for i in range(len(S)):
      if S[i] == 'a' or S[i] == 'A':
            print( S[i] + " found at index " + str(i))
```

Solution for Problem C: Field of Numbers

```
# This function checks if a list of numbers is just [1,2,3,4,5,6,7,8,9]
def checkList( L ):
      valid = True
      L.sort()
      for x in range(9):
            if L[x] != x+1:
                  valid = False
      return valid

# This function checks if column K in the grid is correct
def checkCol( grid , K ):
      L = []
      for r in range(9):
            L.append(grid[r][K])
      return checkList(L)

# This function checks if row K in the grid is correct
def checkRow( grid, K ):
      L = []
      for c in range(9):
            L.append(grid[K][c])
      return checkList(L)

# This function checks if the box with upper-left corner R,C is correct
def checkBox( grid, R, C ):
      L = []
      for a in range(3):
            for b in range(3):
                  L.append( grid[R+a][C+b] )
      return checkList(L)
# Get the grid as input
grid = []
for row in range(9):
      numrow = [int(i) for i in input().split()]
      grid.append(numrow)
# Use the All pattern to check the grid.
valid = True
for i in range(9):
      if not checkRow(grid, i):
            valid = False
      if not checkCol(grid, i):
            valid = False
for a in [0,3,6]:
      for b in [0,3,6]:
            if not checkBox(grid, a, b):
                  valid = False

if valid:
      print("Sudoku!")
else:
      print("Sad aku.")
```

Solution for Problem D: Diy Latin

We check the first letter. If it is a vowel, we add "ay" to the string. Otherwise, we get portions of the string by slicing to obtain the answer.

Solution:

```
S = input()

if S[0] == 'a' or S[0] == 'e' or S[0] == 'i' or S[0] == 'o' or S[0] == 'u':
      print(S + "ay")
else:
      print(S[1:]+S[0]+"ay")
```

Solution for Problem E: Hacking to the Gate

First, we get all the input. Then, we simulate each jump.

We can do this by keeping track of our current position (S) and replacing it with our new position K times.

Solution:

```
inputNums = [int(i) for i in input().split()]
N = inputNums[0]
S = inputNums[1]
K = inputNums[2]
D = [int(i) for i in input().split()]

for jumps in range(K):
      S = D[S-1]

print("Location: " + str(S))
```

Solution to Problem F: Cake

We import the math library, then output the answer using the given formula.

Solution:

```
import math
r = float(input())
print( "%.3f" % (2*r*math.pi))
```

Solution for Problem G: Maximino Triples

We can loop for each of the N test cases. For each test case, we take a list of 3 numbers and output the maximum.

Solution:

```
N = int(input())
for t in range(N):
      nums = [int(i) for i in input().split()]
      print("Student "+ str(t+1) + ": " + str(max(nums)))
```

Solution for Problem H: Range Median Query

We can create a function to get the median of a list.

Then, we can use that function repeatedly to answer each query.

Solution:

```
def getMedian( L ):
      L.sort()
      if len(L) % 2 == 1:
            return L[len(L)//2]
      else:
            return (L[len(L)//2] + L[len(L)//2 - 1])/2

N = int(input())
A = [float(i) for i in input().split()]
Q = int(input())
for query in range(Q):
      bounds = [int(i) for i in input().split()]
      x = bounds[0]
      y = bounds[1]
      print( "%.4f" % (getMedian(A[x:y+1])) )
```

Solution for Problem I: Aldrich's 2D Array Problem

Simply get the input, then calculate the answer.

Solution:

```
dimensions = [int(i) for i in input().split()]
print( dimensions[0] * dimensions[1] % 10007 )
```

Solution for Problem J: Drawn Onward

We can use the All pattern to check if integers at the opposite ends of the list are the same.

To get the element corresponding to the element at index i, we can go backwards from -1, since negative list indices work in Python.

Solution:

```
N = int(input())
sequence = [int(i) for i in input().split()]

valid = True
for i in range(N):
      if sequence[i] != sequence[-1-i]:
            valid = False

if valid:
      print("Yes")
else:
      print("No")
```

Solution for Problem K: Migeee's String Game

One solution is to go through all possible choices of pairs of letters and count how many of the choices are both E's.

```
S1 = input()
S2 = input()

count = 0
for x in S1:
      for y in S2:
            if x == y == "e":
                  count += 1

print(count)
```

However, this method is too slow, since the number of pairs you are checking is

$$\mathbf{25000 \times 25000 = 625,000,000}$$

which is too many operations.

We count the number of E's in S_1 and S_2 then multiply the counts.

This will give us the number of ways to pick one E from S_1 and another E from S_2.

This method will only take roughly

$$25000 + 25000 = 50000 \text{ operations.}$$

Solution:

```
S1 = input()
S2 = input()

count1 = 0
for x in S1:
      if x == "e":
            count1 += 1

count2 = 0
for x in S2:
      if x == "e":
            count2 += 1

print(count1*count2)
```

Solution for Problem L: Hop, Step, Jump!

Let $f(s)$ denote the number of ways to go up s steps.

The number of ways to make it up s steps, when $s > 3$, is

If your first move is a **step**, the number of ways you can complete the remaining steps is $f(s-1)$.

If your first move is a **jump**, the number of ways you can complete the remaining steps is $f(s-2)$.

If your first move is a **hop**, the number of ways you can complete the remaining steps is $f(s-3)$.

Adding up all three cases, we have the following recursive solution.

Solution:

```
def f(n):
      if n == 1:
            return 1
      elif n == 2:
            return 2
      elif n == 3:
            return 4
      else:
            return f(n-1) + f(n-2) + f(n-3)

s = int(input())
print( f(s) )
```

Solution for Problem L: Dungeon Building

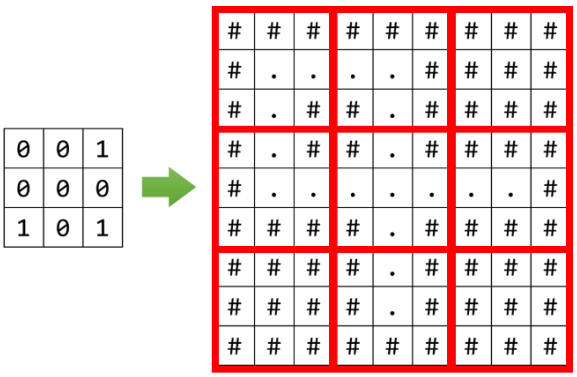
First, we get the input:

```
n = int(input())
A = []
for row in range(n):
      numrow = [int(i) for i in input().split()]
      A.append(numrow)
```

One useful method for this problem is **printing to a character array**. This means, instead of using `System.out.println()` directly to print the correct output, you instead edit a character array first.

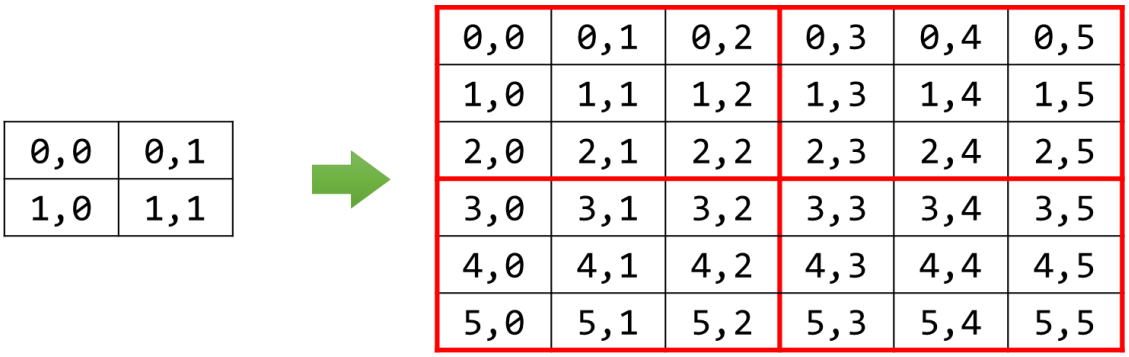
After the character array is ready, you can print it to the console.

We will call the input array A, and the output array B. Each element of A corresponds to a 3×3 subarray of B.

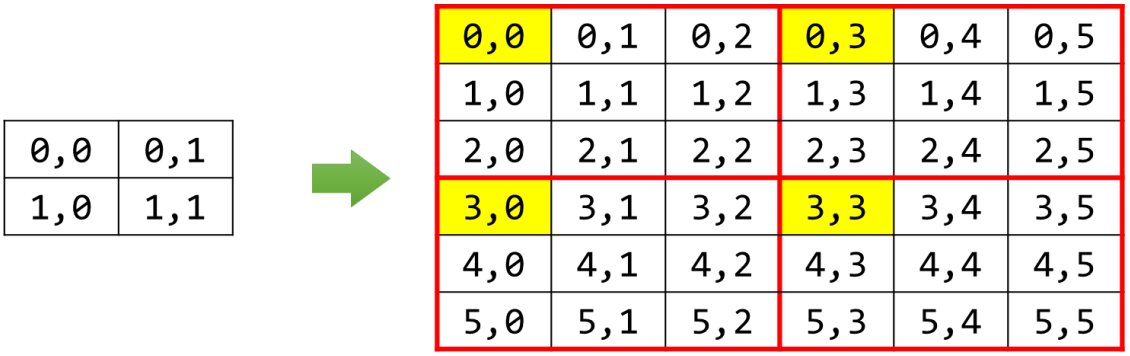


If we want to do this properly, we have to figure out how the index of some element in A relates to the indices in the corresponding subarray in B.

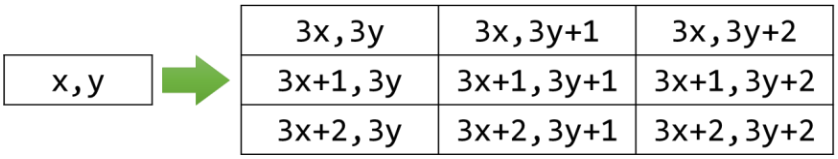
We can have a better idea if we start noting down indices:



We can focus our attention on one of the cells, and see more clearly that every time we move once in grid A, we move by three in grid B.



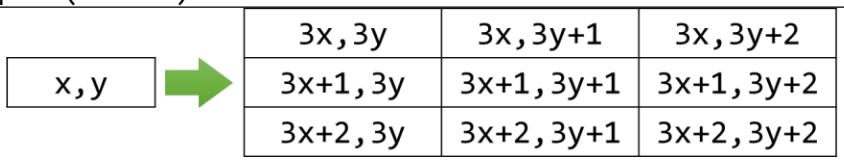
Notice, for example, that if we focus on element (x, y) in grid A, the upper left corner for the subarray in B is $(3x, 3y)$. In general, we have:



We can now start coding.

We go through every cell in A and first, fill all the cells in its subarray in B with the # character.

```
B = []
for row in range(3*n):
      numrow = ["#" for i in range(3*n)]
      B.append(numrow)
```



Then, if the cell is an open room, we set the center character to be a dot:

```
for i in range(n):
      for j in range(n):
            if A[i][j] == 0:
                  B[3*i+1][3*j+1] = '.'
```

Then, we check each of the neighbors to determine which cells we should also turn into dots.

```
for i in range(n):
      for j in range(n):
            if A[i][j] == 0:
                  B[3*i+1][3*j+1] = '.'
                  if i>0 and A[i-1][j]==0:
                        B[3*i][3*j+1] = '.'
                  if i<n-1 and A[i+1][j]==0:
                        B[3*i+2][3*j+1] = '.'
                  if j>0 and A[i][j-1]==0:
                        B[3*i+1][3*j] = '.'
                  if j<n-1 and A[i][j+1]==0:
                        B[3*i+1][3*j+2] = '.'
```

Below is the full solution.

Solution:

```
n = int(input())
A = []
for row in range(n):
      numrow = [int(i) for i in input().split()]
      A.append(numrow)

B = []
for row in range(3*n):
      numrow = ["#" for i in range(3*n)]
      B.append(numrow)

for i in range(n):
      for j in range(n):
            if A[i][j] == 0:
                  B[3*i+1][3*j+1] = '.'
                  if i>0 and A[i-1][j]==0:
                        B[3*i][3*j+1] = '.'
                  if i<n-1 and A[i+1][j]==0:
                        B[3*i+2][3*j+1] = '.'
                  if j>0 and A[i][j-1]==0:
                        B[3*i+1][3*j] = '.'
                  if j<n-1 and A[i][j+1]==0:
                        B[3*i+1][3*j+2] = '.'

for i in range(3*n):
      line = ""
      for j in range(3*n):
            line = line + str(B[i][j])
      print(line)
```