1. Write a Python function that takes a sequence of numbers and determines if all the numbers are different from each other (that is, they are distinct).

2. Write a Python program to remove duplicates from a list. You are allowed to use another list.
Input
[1, 2, 3, 1, 2, 5, 6, 7, 8]
Output
[1, 2, 3, 5, 6, 7, 8]
Input
[1, 2, 3, 1, 2, 5, 6, 7, 8]
Output
[1, 2, 3, 5, 6, 7, 8]

3. Write a Python function primes_lessthan(n) that returns (a list) the number of primes less than a positive integer number 'n' passed to it. primes_lessthan(n) functions calls is_prime(x) in turns, which retruns 1 if x is prime, and returns 0 if x is not prime.
Example, print(primes_lessthan(100))
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

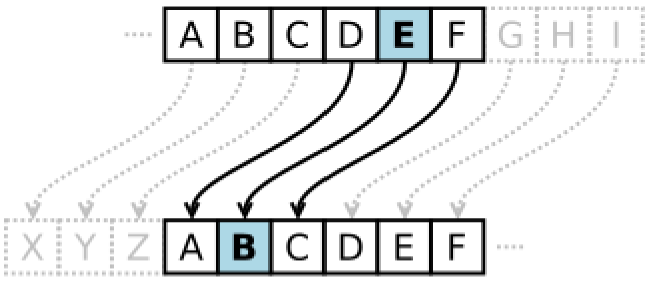
4. Write a Python program that rearranges a sequence of integer values so that all the even values appear before all the odd values.

5. Write a program to o find the mode of n elements. The mode is a value that occurs the highest numbers of time. For Example − assume a set of values 3, 5, 2, 3, 5, 7, 3. The mode of this value set is 3 as it appears more than any other number. The mode may not be unique, in such scenario, your program shout find any one element as mode.

6. Write a Python program for performing a card shuffle of a list of 2n elements. A card shuffle is a permutation where a list A is cut into two sub lists, A1 and A2, where A1 is the first half of A and A2 is the second half of A, and then these two sublists are combine into a single list again by taking the first element in A1, then the first element in A2, followed by the second element in A1, the second element in A2, and so on. Here A1 and A2 are both of size n
Example :
A = {7, 10, 1, 6, 24, 15, 9, 4, 2, 20}
A1 = {7, 10, 1, 6, 24}
A2 = {15, 9, 4, 2, 20}
then the output is B = {7, 15, 10, 9, 1, 4, 6, 2, 24, 20}
Can you write a program which does not use an extra array B?

7. Given an unsorted array of size n, write an algorithm without using sorting to find the maximum difference between any two elements in the array such that smaller element appears before the larger element.
Example:
Input :
a = 2, 8, 1, 6, 10, 4
Output: Maximum difference = 9 (elements are 1 and 10)
Input :
a = 2, 7, 9, 5, 1, 3, 5
Output: Maximum difference = 7 (elements are 2 and 9)
Input :
a = 2, 3, 8, 6, 4, 8, 1
Output: Maximum difference = 6 (elements are 8 and 2)
Input :
a = 7, 9, 5, 6, 13, 2
Output: Maximum difference = 8 (elements are 5 and 13)

8. Caeser Cipher is a type of **substitution cipher** in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.



The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places, equivalent to a right shift of 23 (the shift parameter is used as the key):

```
Plain:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:   XYZABCDEFGHIJKLMNOPQRSTUVW
```

(a) Write a function called CaeserCipher that takes as parameter a string s and an integer k and replaces every letter (upper case or lower case) in s by the letter obtained by shifting the original letter by the amount k. The function should return the encrypted text. Notes: (i) The parameter k may be positive, 0, or negative. (ii) Characters in s that are not letters should remain as they are. (iii) Lower case letters should be replaced by other lower case letters and upper case letters should be replaced by other upper case letters.

(b) Write a program that reads some text and produces as output the corresponding encrypted text. As in earlier problems, the end of the text is specified by an extra enter character. Here is an example of how a user might interact with this program. The first line Enter your text: is the prompt produced by the program. Following this prompt, I entered some text (the first line of President Lincoln's Gettysburg speech) and then typed an extra enter. Notice how the output produced by the program does not disturb the punctuation or the whitespaces. In this example, I used 4 as the Caeser Cipher key. This choice was quite arbitrary.

9. A lucky number is a natural number in a set which is generated by a certain "sieve", eliminates numbers based on their position in the remaining set, instead of their value (or position in the initial set of natural numbers).
Begin with a list of integers starting with 1:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Every second number (all even numbers) in the remaining list is eliminated, leaving only the odd integers:
1 3 5 7 9 11 13 15 17 19 21 23 25
Every third number which remains in the list is eliminated; which starts with 5 (also note that the second term in this sequence is 3):
1 3 7 9 13 15 19 21 25
Every seventh remaining number is eliminated; which starts with 19 (the next surviving number is now 7):
1 3 7 9 13 15 21 25
Write a Python program to find lucky numbers in the range 1-n.
For n=25, Lucky numbers are 1, 3, 7, 9, 13, 15, 21, 25
For n=100, Lucky numbers are 1, 3, 7, 9, 13, 15, 21, 25, 31, 33, 37, 43, 49, 51, 63, 67, 69, 73, 75

10. A "lucky prime" is a lucky number that is prime. First few lucky primes are : 3, 7, 13, 31, 37, 43, 67, 73, 79, 127, 151
Write a Python program to find lucky primes in the range 2-200

11. Write a program to print a given 2D list in spiral form

Examples:

Input:

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

Output:

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

Input:

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

Output:

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

Input:

```
2  4  6  8
5  9  12 16
2  11 5  9
3  2  1  8
```

Output:

```
2 4 6 8 16 9 8 1 2 3 2 5 9 12 5 11
```

12. Write an efficient program to **generate** the following **spiral pattern** for any n, where n is the order of the square matrix.

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

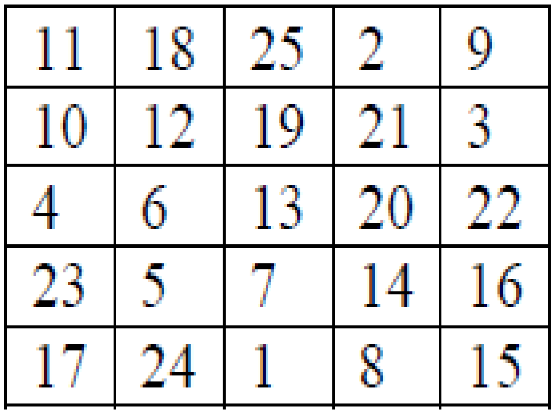
Test your program for n=2,3,4,5,6,7,8,……

13. A magic square is an n×n matrix that have an arrangement of n^2 numbers, usually distinct positive integers, such that the sum of n numbers in all rows, all columns and both diagonals is the same constant called the **magic sum**. A normal magic square contains the integers from 1 to n^2. Normal magic squares exist for all n>2.

The **smallest nontrivial case**, shown below, is of **order 3** and whose **magic sum is 15.**

$$\begin{bmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{bmatrix}$$

Magic square of **order 5, magic sum = 65**



11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Algorithm for generating magic square when n>2 and n is odd
Start with 1 in the middle of the bottom row, then go down and right, assigning numbers in increasing order to empty squares; if you fall off the square imagine the same square as tiling the plane and continue; if a square is occupied, move up instead and continue.
Write an efficient Python program to generate the above Magic Squares of odd order (3,5,7, etc.) up to order 51.

14. Write a efficient Python program to find the value and location of saddle point of a matrix $A_{m\times n}$(if A[i][j] is the **smallest value in the ith row and the largest value in the jth column**) if one exists. A matrix may not have a saddle point. A matrix may have more than one saddle points.
Test cases
Enter the order of the matrix : 3 3
Enter the element of the matrix :
1 2 3
4 5 6
7 8 9
NO SADDLE POINT in row 1
NO SADDLE POINT in row 2
The saddle point at a[3][1]=7

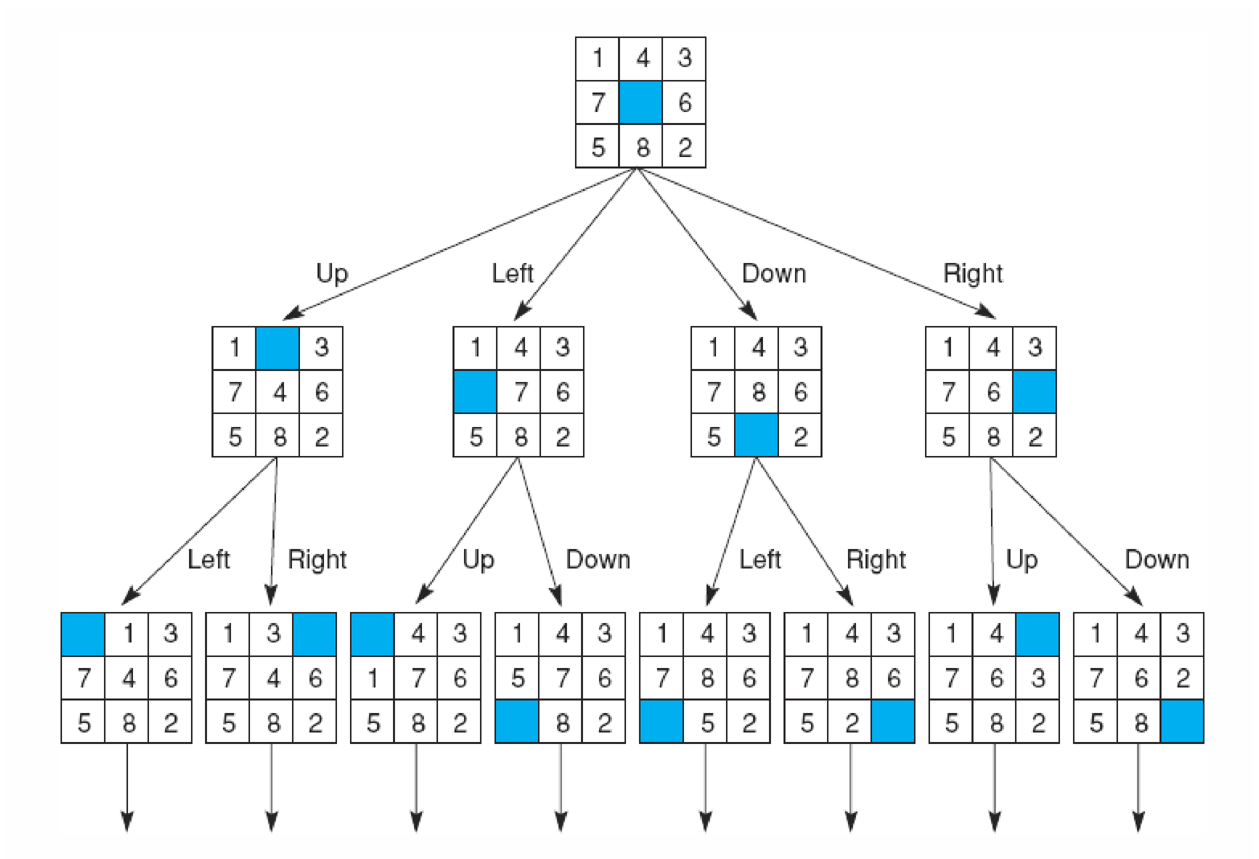
```
Enter the order of the matrix : 3 3
Enter the element of the matrix :
1 2 7
4 5 6
3 8 9
NO SADDLE POINT in row 1
The saddle point at a[2][1]=4
NO SADDLE POINT in row 3
```

```
Enter the order of the matrix : 3 4
Enter the element of the matrix :
11 9 8 7
14 10 9 6
11 9 8 7
The saddle point at a[1][4]=7
The saddle point at a[3][4]=7
```

15. The well-known 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square(empty space). A move consists in sliding a tile to the empty space, or equivalently, moving the empty space horizontally or vertically. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. The objective is to place the numbers on tiles from the given input configuration to match final configuration(row wise sorted) using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.
The following shows a sequence of legal moves from an initial board position (left) to the goal position (right).

```
   1  3        1     3        1  2  3       1  2  3          1  2  3
4  2  5   =>   4  2  5   =>   4     5   =>  4  5       =>     4  5  6
7  8  6        7  8  6        7  8  6       7  8  6           7  8
```

The partial state space tree(for different problem) is as follows



The challenge is to change the random initial configuration into the goal configuration. A solution to the problem is an appropriate sequence of moves, such as “move tiles 5 to the right, move tile 7 to the left , move tile 6 to the down, etc”.
Write a Python program to solve the 8-puzzle problem.
Test your program for the followings input configurations

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

Extend the same for the 15 puzzle.

16. Sudoku is a popular logic-based number-placement classic fun puzzle. The objective is to fill a partially filled 9 × 9 grid (2D list) with digits so that each column, each row, and each of the nine 3 × 3 sub-grids or sectors that compose the grid contains all of the digits from 1 to 9. These constraints are used to determine the missing numbers. A standard Sudoku puzzle contains 81 cells, in a 9 by 9 grid, and has 9 zones, each zone being the intersection of 3 rows and 3 columns. A standard Sudoku contains 81 cells, in a 9×9 grid, and has 9 boxes, each box being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns. Each cell may contain

a number from one to nine, and each number can only occur once in each row, column, and box. Given a partially filled in Sudoku board(some cells containing numbers (clues)), the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9. Proper Sudokus have one and unique solution. In the puzzle below, several sub-grids have missing numbers. Scanning rows (or columns as the case may be) can tell us where to place a missing number in a sector.

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

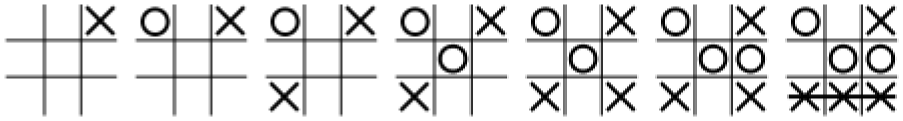
The unique solution is

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

You have given initial configuration(partially filled) of 9x9 grid. Write a program to solve a Sudoku puzzle by filling the empty cells. The grid is represented by a two-dimensional list called grid and a value of 0 means that the location is empty.

17. The simplest two player game, Tic-Tac-Toe(a.k.a noughts and crosses or Xs and Os) game is undoubtedly the most popular childhood games(played using pencil in paper,) in the world, can be played by two players(or A human can play against the computer) where the square block (3 x 3) can be filled with a cross (X) or a circle (O). The game will toggle between the players by

giving the chance for each player to mark their move. In this game either a player win, lose the game or in a significant proportion of the games played, the game ends with a draw(when all the squares of the board/matrix are filled). When one of the players first make a combination of 3 same markers in a horizontal, vertical or diagonal line, own the game.
The following example game is won by the first player, X:



Write a Python program to implement the tic-tac-toe game for two player, that contains a display function and a select function to place the symbol as well as toggle between the symbols allowing each player an alternate turn to play the game. The program will update after each player makes their move and check for the conditions of game as it goes on the program will display which player has won, whether X or O.
You have to output step by step showing whether player ‘0’ win or player ‘X’ win or the game is draw. Print the coordinates of the cell (x, y) of move for the winner.
Input Output Specification
Player 'X' and player 'O' enter their move successively, as shown. The program shall perform input validation.

```
Player 'X', enter your move (row[1-3] column[1-3]): 2 2
   |  |
-----------
   | X |
-----------
   |  |

Player 'O', enter your move (row[1-3] column[1-3]): 1 1
 O |  |
-----------
   | X |
-----------
   |  |

Player 'X', enter your move (row[1-3] column[1-3]): 1 3
 O |  | X
-----------
    |X|
-----------
    | |
Player 'O', enter your move (row[1-3] column[1-3]): 3 1
 O |  | X
-----------
    |X|
-----------
 O | |
Player 'X', enter your move (row[1-3] column[1-3]): 2 2
This move at (2,2) is not valid. Try again...
Player 'X', enter your move (row[1-3] column[1-3]): 2 3
 O |   | X
-----------
   |X| X
-----------
 O |  |

Player 'O', enter your move (row[1-3] column[1-3]): 2 1
 O |  | X
-----------
 O |X| X
-----------
 O |  |

Player 'O' won!
```