INDEX

Artificial Intelligence Programs Using PROLOG

Week-1

Aim:

Study of PROLOG Programming language and its Functions. Write simple facts for the statements using PROLOG.

OBJECTIVE: Study of PROLOG.

PROLOG-PROGRAMMING IN LOGIC

PROLOG stands for Programming, In Logic — an idea that emerged in the early 1970's to use logic as programming language. The early developers of this idea included Robert Kowaiski at Edinburgh (on the theoretical side), Marrten van Emden at Edinburgh (experimental demonstration) and Alian Colmerauer at Marseilles (implementation).

David D.H. Warren's efficient implementation at Edinburgh in the mid -1970's greatly contributed to the popularity of PROLOG. PROLOG is a programming language centred around a small set of basic mechanisms, Including pattern matching, tree based data structuring and automatic backtracking. This Small set constitutes a surprisingly powerful and flexible programming framework. PROLOG is especially well suited for problems that involve objects- in particular, structured objects- and relations between them.

SYMBOLIC LANGUAGE

PROLOG is a programming language for symbolic, non-numeric computation. It is especially well suited for solving problems that involve objects and relations between objects.

For example, it is an easy exercise in PROLOG to express spatial relationship between objects, such as the blue sphere is behind the green one. It is also easy to state a more general rule: if object X is closer to the observer than object Y. and object Y is closer than Z, then X must be closer than Z. PROLOG can reason about the spatial relationships and their consistency with respect to the general rule. Features like this make PROLOG a powerful language for ArtJIcia1 LanguageA1,) and non- numerical programming.

There are well-known examples of symbolic computation whose implementation in other standard languages took tens of pages of indigestible code, when the same algorithms were implemented in PROLOG, the result was a crystal-clear program easily fitting on one page.

FACTS, RULES AND QUERIES

Programming in PROIOG is accomplished by creating a database of facts and rules about objects, their properties, and their relationships to other objects. Queries then can be posed about the objects and valid conclusions will be determined and returned by the program Responses to user queries are determined through a form of inference control known as resolution.

FOR EXAMPLE:

a. Ram likes mango.

b. Seema is a girl.

c. Bill likes Cindy.

d. Rose is red.

e. John owns gold.

Program:

```
Clauses
likes(ram ,mango).
girl(seema).
red(rose).
likes(bill ,cindy).
owns(john ,gold).
```

Output:

```
queries
?-likes(ram,What).
What= mango

?-likes(Who,cindy).
Who= cindy
?-red(What).
What= rose
?-owns(Who,What).
Who= john
What= gold.
```

Viva Questions:

1- First create a source file for the genealogical logicbase application. Start by adding a few members of your family tree. It is important to be accurate, since we will be exploring family relationships. Your own knowledge of who your relatives are will verify the correctness of your Prolog programs.

2- Enter a two-argument predicate that records the parent-child relationship. One argument represents the parent, and the other the child. It doesn't matter in which order you enter the arguments, as long as you are consistent. Often Prolog programmers adopt the convention that parent(A,B) is interpreted "A is the parent of B".

3- Create a source file for the customer order entry program. We will begin it with three record types (predicates). The first is customer/3 where the three arguments are

arg1

Customer name

arg2

City

arg3

Credit rating (aaa, bbb, etc)

4- Next add clauses that define the items that are for sale. It should also have three arguments

arg1

Item identification number

arg2

Item name

arg3

The reorder point for inventory (when at or below this level, reorder)

5- Next add an inventory record for each item. It has two arguments.

arg1

Item identification number (same as in the item record)

arg2

Amount in stock

Assignment:

Aim: Write facts for following:

1. Ram likes apple.

2. Ram is taller then Mohan.

3. My name is Subodh.

4. Apple is fruit.

5. Orange is fruit.

6. Ram is male.

AIM: Write simple queries for following facts.

Simple Queries

Now that we have some facts in our Prolog program, we can consult the program in the listener and query, or call, the facts. This chapter, and the next, will assume the Prolog program contains only facts. Queries against programs with rules will be covered in a later chapter.

Prolog queries work by pattern matching. The query pattern is called a goal. If there is a fact that matches the goal, then the query succeeds and the listener responds with 'yes.' If there is no matching fact, then the query fails and the listener responds with 'no.'

Prolog's pattern matching is called unification. In the case where the logicbase contains only facts, unification succeeds if the following three conditions hold.

- The predicate named in the goal and logic base are the same.
- Both predicates have the same arity.
- All of the arguments are the same.

Before proceeding, review figure 3.1, which has a listing of the program so far.
The first query we will look at asks if the office is a room in the game. To pose this, we would enter that goal followed by a period at the listener prompt.

?- room(office).
yes
Prolog will respond with a 'yes' if a match was found. If we wanted to know if the attic was a room, we would enter that goal.
?- room(attic).
No

Solution:-
clauses
likes(ram ,mango).

girl(seema).
red(rose).
likes(bill ,cindy).
owns(john ,gold).

queries
?-likes(ram,What).
What= mango
?-likes(Who,cindy).
Who= cindy
?-red(What).
What= rose
?-owns(Who,What).
Who= john
What= gold
Viva Questions:

1- Consider the following Prolog logic base
easy(1).
easy(2).
easy(3).
gizmo(a,1).
gizmo(b,3).
gizmo(a,2).
gizmo(d,5).
gizmo(c,3).
gizmo(a,3).
gizmo(c,4).
and predict the answers to the queries below, including all alternatives when the semicolon (;) is entered after an answer.
?- easy(2).
?- easy(X).
?- gizmo(a,X).
?- gizmo(X,3).
?- gizmo(d,Y).
?- gizmo(X,X).
2- Consider this logicbase,

harder(a,1).

harder(c,X).

harder(b,4).

harder(d,2).

and predict the answers to these queries.

?- harder(a,X).

?- harder(c,X).

?- harder(X,1).

?- harder(X,4).

3- Enter the listener and reproduce some of the example queries you have seen against location/2. List or print location/2 for reference if you need it. Remember to respond with a semicolon (;) for multiple answers. Trace the query.

4- Pose queries against the genealogical logicbase that:

- Confirm a parent relationship such as parent(dennis, diana)
- Find someone's parent such as parent(X, diana)
- Find someone's children such as parent(dennis, X)
- List all parent-children such as parent(X,Y)

5- If parent/2 seems to be working, you can add additional family members to get a larger logicbase. Remember to include the corresponding male/1 or female/1 predicate for each individual added.

Week-2

Aim:

Implementation of Depth First Search for Water Jug problem.

Theory/Description:

In the water jug problem in Artificial Intelligence, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

So, to solve this problem, following set of rules were proposed:

Production rules for solving the water jug problem

Solution:

The state space for this problem can be described as the set of ordered pairs of integers (x,y)

Where,

X represents the quantity of water in the 4-gallon jug X= 0,1,2,3,4

Y represents the quantity of water in 3-gallon jug Y=0,1,2,3

Start State: (0,0)

Goal State: (2,0)

Generate production rules for the water jug problem

Production Rules:

Rule	State	Process
1	(X,Y | X<4)	(4,Y) {Fill 4-gallon jug}
2	(X,Y |Y<3)	(X,3) {Fill 3-gallon jug}
3	(X,Y |X>0)	(0,Y) {Empty 4-gallon jug}
4	(X,Y | Y>0)	(X,0) {Empty 3-gallon jug}
5	(X,Y | X+Y>=4 ^ Y>0)	(4,Y-(4-X)) {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
6	(X,Y | X+Y>=3 ^X>0)	(X-(3-Y),3) {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}
7	(X,Y | X+Y<=4 ^Y>0)	(X+Y,0) {Pour all water from 3-gallon jug into 4-gallon jug}
8	(X,Y | X+Y <=3^ X>0)	(0,X+Y) {Pour all water from 4-gallon jug into 3-gallon jug}
9	(0,2)	(2,0) {Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

Initialization:

Start State: (0,0)

Apply Rule 2:

(X,Y | Y<3) ->

(X,3)

{Fill 3-gallon jug}

Now the state is (X,3)

Iteration 1:

Current State: (X,3)

Apply Rule 7:

(X,Y | X+Y<=4 ^Y>0)

(X+Y,0)

{Pour all water from 3-gallon jug into 4-gallon jug}

Now the state is (3,0)

Iteration 2:

Current State : (3,0)

Apply Rule 2:

(X,Y | Y<3) ->

(3,3)

{Fill 3-gallon jug}

Now the state is (3,3)

Iteration 3:

Current State:(3,3)

Apply Rule 5:

(X,Y | X+Y>=4 ^ Y>0)

(4,Y-(4-X))

{Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}

Now the state is (4,2)

Iteration 4:

Current State : (4,2)

Apply Rule 3:

(X,Y | X>0)

(0,Y)

{Empty 4-gallon jug}

Now state is (0,2)

Iteration 5:

Current State : (0,2)

Apply Rule 9:

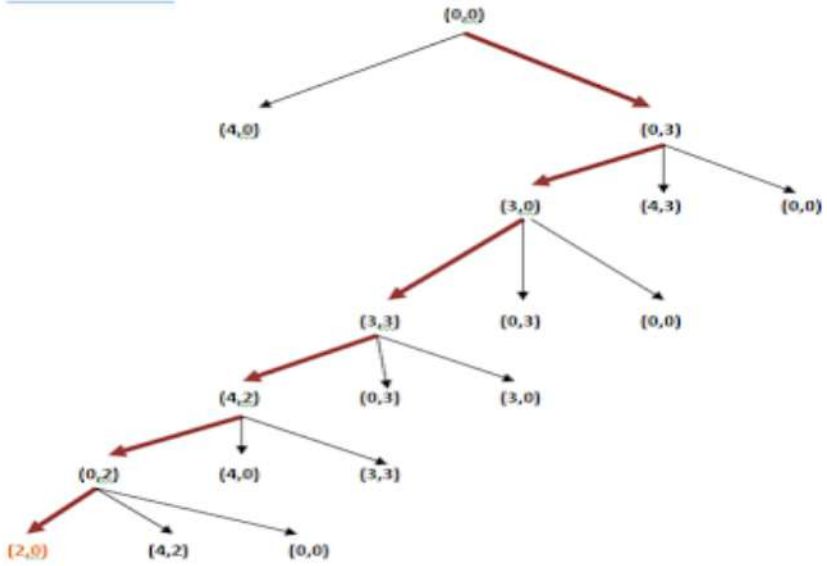
(0,2)

(2,0)

{Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

Now the state is (2,0)

Goal Achieved.



Progaram:

```
solve_dfs(State, History, []) :- final_state(State).
solve_dfs(State, History, [Move|Moves]) :-
    move(State, Move),
    update(State, Move, State1),
    legal(State1),
    not(member(State1, History)),
    solve_dfs(State1, [State1|History], Moves).

test_dfs(Problem, Moves) :-
    initial_state(Problem, State), solve_dfs(State, [State], Moves).

capacity(1, 10).
capacity(2, 7).

initial_state(jugs, jugs(0, 0)).
final_state(jugs(6, 0)).
%final_state(jugs(4, 0)).
```

```
legal(jugs(V1, V2)).

move(jugs(V1, V2), fill(1)) :- capacity(1, C1), V1 < C1, capacity(2, C2), V2 < C2.
move(jugs(V1, V2), fill(2)) :- capacity(2, C2), V2 < C2, capacity(1, C1), V1 < C1.

move(jugs(V1, V2), empty(1)) :- V1 > 0.
move(jugs(V1, V2), empty(2)) :- V2 > 0.
move(jugs(V1, V2), transfer(1, 2)).
move(jugs(V1, V2), transfer(2, 1)).

adjust(Liquid, Excess, Liquid, 0) :- Excess =< 0.
adjust(Liquid, Excess, V, Excess) :- Excess > 0, V is Liquid - Excess.

update(jugs(V1, V2), fill(1), jugs(C1, V2)) :- capacity(1, C1).
update(jugs(V1, V2), fill(2), jugs(V1, C2)) :- capacity(2, C2).
update(jugs(V1, V2), empty(1), jugs(0, V2)).
update(jugs(V1, V2), empty(2), jugs(V1, 0)).

update(jugs(V1, V2), transfer(1, 2),jugs(NewV1, NewV2)) :-
   capacity(2, C2),
   Liquid is  V1 + V2,
   Excess is Liquid - C2,
   adjust(Liquid, Excess, NewV2, NewV1).

update(jugs(V1, V2), transfer(2, 1),jugs(NewV1, NewV2)) :-
   capacity(1, C1),
   Liquid is  V1 + V2,
   Excess is Liquid - C1,
   adjust(Liquid, Excess, NewV1, NewV2).
```

Output:

```
Query: test_dfs(jugs, Moves).
Moves = [fill(1),
transfer(1,2),
```

empty(1),
transfer(2,1),
fill(2),
transfer(2,1),
empty(1),
transfer(2,1),
fill(2),
transfer(2,1),
empty(1),
transfer(2,1),
fill(2),
transfer(2,1),
fill(2),
transfer(2,1),
empty(1),
transfer(2,1),
fill(2),
transfer(2,1),
empty(1),
transfer(2,1),
fill(2),
transfer(2,1),
fill(2),
transfer(2,1),
empty(1),
transfer(2,1)]

Week-3

Aim:

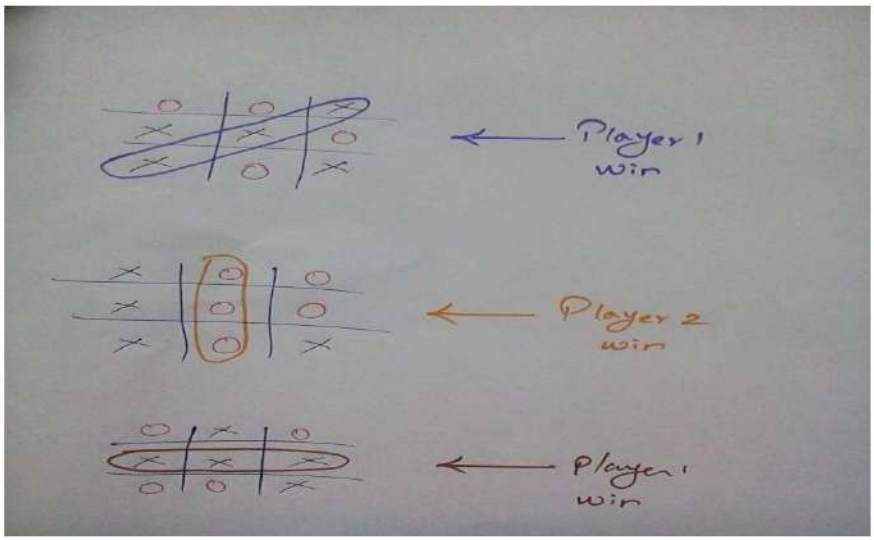
Implementation of Breadth First Search for Tic-Tac-Toe problem.

Theory/Description:

The game **Tic Tac Toe** is also known as **Noughts** and **Crosses** or **X**s and **O**s ,the player needs to take turns marking the spaces in a 3x3 grid with their own marks,if 3 consecutive marks (**Horizontal**, **Vertical**,**Diagonal**) are formed then the player who owns these moves get won.

Assume ,

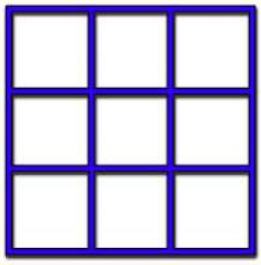
Player 1 - X
Player 2 - O



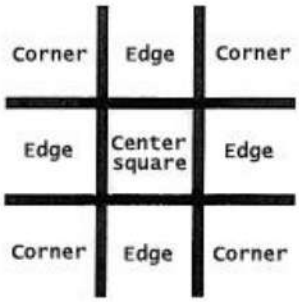
So,a player who gets 3 consecutive marks first,they will win the game .

Let's have a discussion about how a board's data structure looks and how the Tic Tac Toe algorithm works.

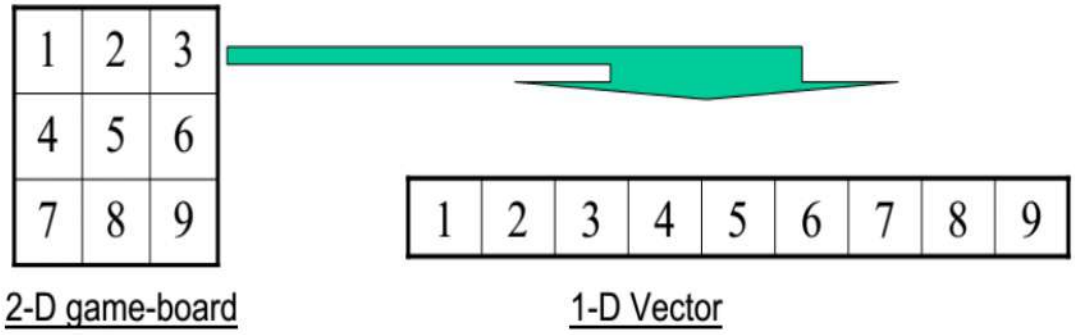
Board's Data Structure:



The cells could be represent as Center square,Corner,Edge as like below



Number each square starts from 1 to 9 like following image



Consider a Board having nine elements vector.Each element will contain

- 0 for blank
- 1 indicating X player move
- 2 indicating O player move

Computer may play as X or O player. First player is always X.

Move Table

It is a vector of 3^9 elements, each element of which is a nine element vector representing board position.

Total of 3^9(19683) elements in move table

Move Table

Index	Current Board position	New Board position
0	000000000	000010000
1	000000001	020000001
2	000000002	000100002
3	000000010	002000010

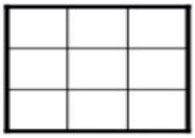
.
.

Algorithm

To make a move, do the following:

1. View the vector (board) as a ternary number and convert it to its corresponding decimal number.
2. Use the computed number as an index into the move table and access the vector stored there.
3. The vector selected in step 2 represents the way the board will look after the move that should be made. So set board equal to that vector.

Let's start with empty board



Step 1: Now our board looks like **000 000 000 (**tenary number**)** convert it into decimal no.The decimal no is **0**

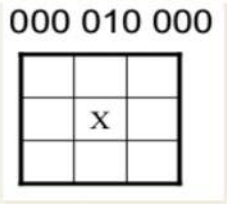
Step 2: Use the computed number ie **0** as an index into the move table and access the vector stored in

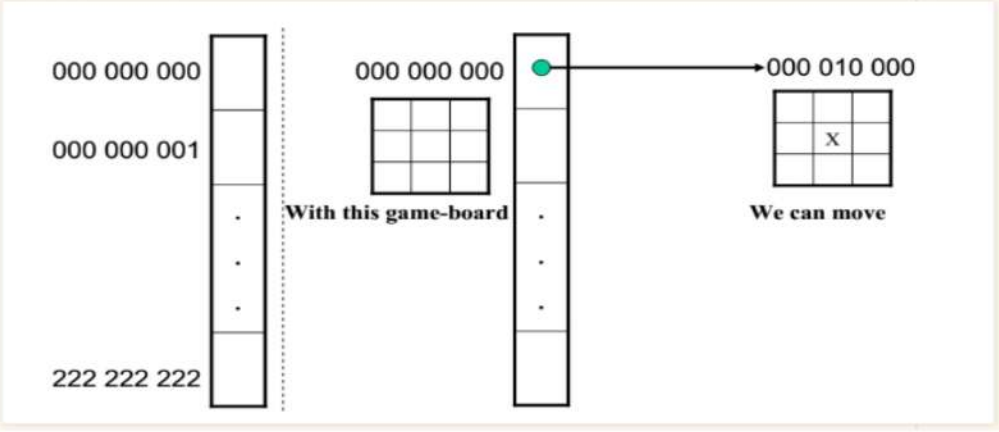
New Board Position.

The new board position is **000 010 000**

Step 3: The vector selected in step 2(**000 010 000**) represents the way the board will look after the move that should be made. So set board equal to that vector.

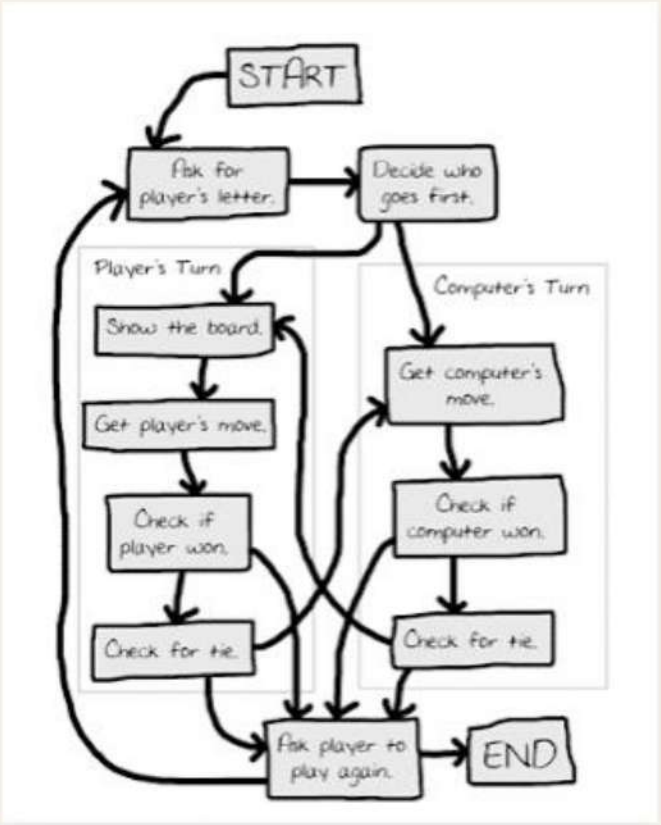
After complete the 3rd step your board looks like\





This process continues until the player get win or tie.

Flowchart:



Program:

```
win(Board, Player) :- rowwin(Board, Player).
win(Board, Player) :- colwin(Board, Player).
win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,_,_,_,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,Player,Player,Player,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,_,_,_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,_,_,Player,_,_,Player,_,_].
colwin(Board, Player) :- Board = [_,Player,_,_,Player,_,_,Player,_].
colwin(Board, Player) :- Board = [_,_,Player,_,_,Player,_,_,Player].

diagwin(Board, Player) :- Board = [Player,_,_,_,Player,_,_,_,Player].
diagwin(Board, Player) :- Board = [_,_,Player,_,Player,_,Player,_,_].
```

```
% Helping predicate for alternating play in a "self" game:

other(x,o).
other(o,x).

game(Board, Player) :- win(Board, Player), !, write([player, Player, wins]).
game(Board, Player) :-
  other(Player,Otherplayer),
  move(Board,Player,Newboard),
  !,
  display(Newboard),
  game(Newboard,Otherplayer).

move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).
move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).
move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).
move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).
move([A,B,C,D,b,F,G,H,I], Player, [A,B,C,D,Player,F,G,H,I]).
move([A,B,C,D,E,b,G,H,I], Player, [A,B,C,D,E,Player,G,H,I]).
move([A,B,C,D,E,F,b,H,I], Player, [A,B,C,D,E,F,Player,H,I]).
move([A,B,C,D,E,F,G,b,I], Player, [A,B,C,D,E,F,G,Player,I]).
move([A,B,C,D,E,F,G,H,b], Player, [A,B,C,D,E,F,G,H,Player]).

display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
 write([G,H,I]),nl,nl.

selfgame :- game([b,b,b,b,b,b,b,b,b],x).

% Predicates to support playing a game with the user:

x_can_win_in_one(Board) :- move(Board, x, Newboard), win(Newboard, x).

% The predicate orespond generates the computer's (playing o) reponse
% from the current Board.

orespond(Board,Newboard) :-
  move(Board, o, Newboard),
  win(Newboard, o),
  !.
```

```
orespond(Board,Newboard) :-
  move(Board, o, Newboard),
  not(x_can_win_in_one(Newboard)).
orespond(Board,Newboard) :-
  move(Board, o, Newboard).
orespond(Board,Newboard) :-
  not(member(b,Board)),
  !,
  write('Cats game!'), nl,
  Newboard = Board.

% The following translates from an integer description
% of x's move to a board transformation.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).
xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).
xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).
xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).
xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).
xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).
xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).
xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).
xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).
xmove(Board, _, Board) :- write('Illegal move.'), nl.

% The 0-place predicate playo starts a game with the user.

playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).

explain :-
  write('You play X by entering integer positions followed by a period.'),
  nl,
  display([1,2,3,4,5,6,7,8,9]).

playfrom(Board) :- win(Board, x), write('You win!').
playfrom(Board) :- win(Board, o), write('I win!').
playfrom(Board) :- read(N),
  xmove(Board, N, Newboard),
  display(Newboard),
  orespond(Newboard, Newnewboard),
```

Program-5

In [19]:
```
# Import math Library
import math

# Check whether some values are NaN or not
print (math.isnan (56))
print (math.isnan (-45.34))
print (math.isnan (+45.34))
print (math.isnan (math.inf))
print (math.isnan (float("nan")))
print (math.isnan (float("inf")))
print (math.isnan (float("-inf")))
print (math.isnan (math.nan))
```

```
False
False
False
False
True
False
False
True
```

Program-6

In [25]:
```
# Import math Library
import math

# Print the square root of different numbers
print (math.sqrt(10))
print (math.sqrt (12))
print (math.sqrt (68))
print (math.sqrt (100))

# Round square root downward to the nearest integer
print (math.isqrt(10))
print (math.isqrt (12))
print (math.isqrt (68))
print (math.isqrt (100))
```

```
3.1622776601683795
3.4641016151377544
8.246211251235321
10.0
3
3
8
10
```

- **Python Numpy Library**

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication

For any scientific project, NumPy is the tool to know. It has been built to work with the N- dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated toC/C++ and Fortran.

NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top ofthe arrays and matrices, NumPy supports a large number of mathematical operations.

NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than anyother library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikit learn to use NumPy arrayto compute the matrix multiplication in the back end.

- **Arrays in NumPy:** NumPy's main object is the homogeneous multidimensional array.

  - It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positiveintegers.
  - In NumPy dimensions are called axes. The number of axes is rank.
  - NumPy's array class is called **ndarray**. It is also known by the alias **array**.

  We use python numpy array instead of a list because of the below three reasons:
  1. Less Memory
  2. Fast
  3. Convenient

- **Numpy Functions**

  Numpy arrays carry attributes around with them. The most important ones are:
  ndim: The number of axes or rank of the array
  shape: A tuple containing the length in each dimension
  size: The total number of elements

Program-1

```
In [27]: import numpy        #DEPT OF SoCSE4
         x = numpy.array([[1,2,3], [4,5,6], [7,8,9]]) # 3x3 matrix
         print(x.ndim) # Prints 2
         print(x.shape) # Prints (3L, 3L)
         print(x.size) # Prints 9

         2
         (3, 3)
         9
```

Can be used just like Python lists
x[1] will access the second element
x[-1] will access the last element

Program-2

Arithmetic operations apply element wise

```
In [32]: a = numpy.array( [20,30,40,50,60] )
         b = numpy.arange( 5 )
         c = a-b     #DEPT OF SoCSE4
         #c => array([20, 29, 38, 47])
         c

Out[32]: array([20, 29, 38, 47, 56])
```

- **Built-in Methods**

Many standard numerical functions are available as methods out of the box:

Program-3

```
In [34]: x = numpy.array([1,2,3,4,5])
         avg = x.mean()     #DEPT OF SoCSE4
         sum = x.sum()
         sx = numpy.sin(x)
         sx

Out[34]: array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427])
```

- **Python Scipy Library**

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy also pronounced as "Sigh Pi."

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

Numpy VS SciPyNumpy:

1. Numpy is written in C and use for mathematical or numeric calculation.
2. It is faster than other Python Libraries
3. Numpy is the most useful library for Data Science to perform basic calculations.
4. Numpy contains nothing but array data type which performs the most basic operation like sorting,shaping, indexing, etc.

SciPy:

1. SciPy is built in top of the NumPy
2. SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
3. Most new Data Science features are available in Scipy rather than Numpy.

Linear Algebra with SciPy

1. Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
2. Performance of Linear Algebra is very fast compared to BLAS and LAPACK.
3. Linear algebra routine accepts two-dimensional array object and output is also a two-dimensional array.

Now let's do some test with **scipy.linalg,**

Calculating **determinant** of a two-dimensional matrix,

Program-1

```
from scipy import linalg
import numpy as np #define square matrix
two_d_array = np.array([ [4,5], [3,2] ]) #pass values to det() function
linalg.det( two_d_array )
```

```
-7.0
```

Eigenvalues and Eigenvector – scipy.linalg.eig()

- The most common problem in linear algebra is eigenvalues and eigenvector which can beeasily solved using **eig()** function.
- Now lets we find the Eigenvalue of (**X**) and correspond eigenvector of a two-dimensionalsquare matrix.

Program-2

```
from scipy import linalg
import numpy as np
#define two dimensional array
arr = np.array([[5,4],[6,3]]) #pass value into function
eg_val, eg_vect = linalg.eig(arr) #get eigenvalues
print(eg_val) #get eigenvectors print(eg_vect)
```

```
[ 9.+0.j -1.+0.j]
```

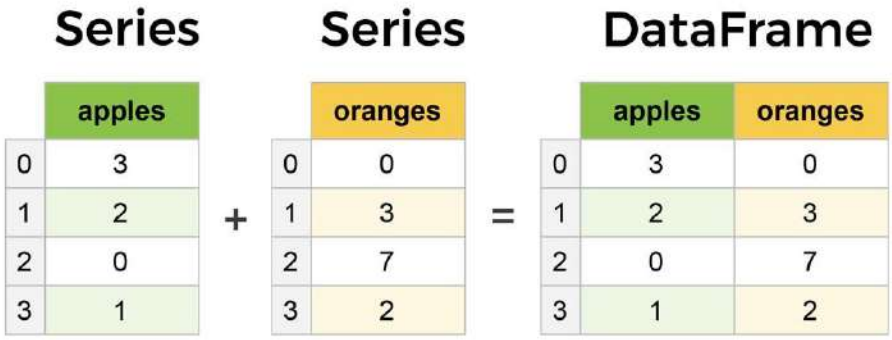
Exercise programs:

1. consider a list datatype then reshape it into 2d,3d matrix using numpy
2. genrate random matrices using numpy
3. find the determinant of a matrix using scipy
4. find eigenvalue and eigenvector of a matrix using scipy

b) Implementation of Python Libraries for ML application such as Pandas and Matplotlib.

- **Pandas Library**

The primary two components of pandas are the Series and DataFrame.
A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of acollection of Series.
DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.



- **Reading data from CSVs**

With CSV files all you need is a single line to load in the data:

```
df =
pd.read_csv('purchases.csv')df
```

Let's load in the IMDB movies dataset to begin:

```
movies_df = pd.read_csv("IMDB-Movie-Data.csv", index_col="Title")
```

We're loading this dataset from a CSV and designating the movie titles to be our index.

- **Viewing your data**

The first thing to do when opening a new dataset is print out a few rows to keep as a visualreference. We accomplish this with .head():

```
movies_df.head()
```

Another fast and useful attribute is .shape, which outputs just a tuple of (rows, columns):

```
movies_df.shape
```

Note that .shape has no parentheses and is a simple tuple of format (rows, columns). So we have1000 rows and 11 columns in our movies DataFrame.
You'll be going to .shape a lot when cleaning and transforming data. For example, you might filtersome rows based on some criteria and then want to know quickly how many rows were removed.

Program-1

```
import pandas as pd
S = pd.Series([11, 28, 72, 3, 5, 8])
S
```

```
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
```

We haven't defined an index in our example, but we see two columns in our output: The right column contains our data, whereas the left column contains the index. Pandas created a default index starting with 0 going to 5, which is the length of the data minus 1.

Program-2

We can directly access the index and the values of our Series S:

```
print(S.index)
print(S.values)
```

```
RangeIndex(start=0, stop=6, step=1)
[11 28 72  3  5  8]
```

Program-3

If we compare this to creating an array in numpy, we will find lots of similarities:

```
import numpy as np
X = np.array([11, 28, 72, 3, 5, 8])
print(X)
print(S.values)
# both are the same type:
print(type(S.values), type(X))
```

```
[11 28 72  3  5  8]
[11 28 72  3  5  8]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

So far our Series have not been very different to ndarrays of Numpy. This changes, as soon as we start defining Series objects with individual indices:

Program-4

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
quantities = [20, 33, 52, 10]
S = pd.Series(quantities, index=fruits)
S
```

```
apples      20
oranges     33
cherries    52
pears       10
dtype: int64
```

Program-5

A big advantage to NumPy arrays is obvious from the previous example: We can use arbitrary indices. If we add two series with the same indices, we get a new series with the same index and the correponding values will be added:

```
fruits = ['apples', 'oranges', 'cherries', 'pears']

S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits)
print(S + S2)
print("sum of S: ", sum(S))
```

OUTPUT:

```
apples     37
oranges    46
cherries   83
pears      42
dtype: int64
sum of S:  115
```

Program-6

The indices do not have to be the same for the Series addition. The index will be the "union" of both indices. If an index doesn't occur in both Series, the value for this Series will be NaN:

```
fruits = ['peaches', 'oranges', 'cherries', 'pears']
fruits2 = ['raspberries', 'oranges', 'cherries', 'pears']

S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits2)
print(S + S2)
```

OUTPUT:

```
cherries      83.0
oranges       46.0
```

```
peaches        NaN
pears         42.0
raspberries    NaN
dtype: float64
```

Program-7

In principle, the indices can be completely different, as in the following example. We have two indices. One is the Turkish translation of the English fruit names:

```
fruits = ['apples', 'oranges', 'cherries', 'pears']

fruits_tr = ['elma', 'portakal', 'kiraz', 'armut']

S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits_tr)
print(S + S2)
```

OUTPUT:

```
apples     NaN
armut      NaN
cherries   NaN
elma       NaN
kiraz      NaN
oranges    NaN
pears      NaN
portakal   NaN
dtype: float64
```

Program-8

Indexing

It's possible to access single values of a Series.

```
print(S['apples'])
```

OUTPUT:

```
20
```

- **Matplotlib Library**

  Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.

  - **Make a simple plot**

    import matplotlib.pyplot as pltimport
    numpy as np

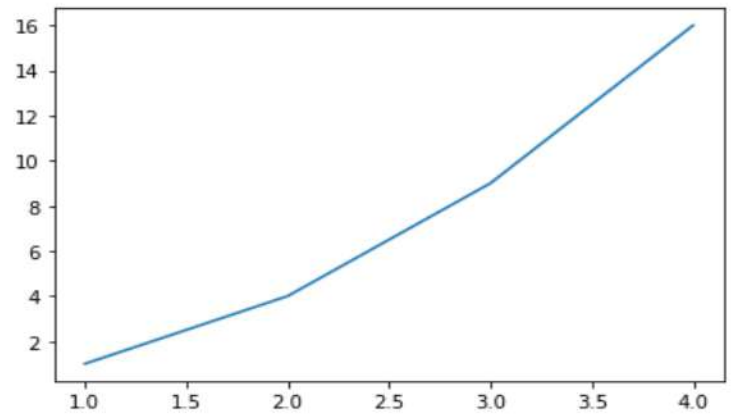
    List of all the methods as they appeared.

    - plot(x-axis values, y-axis values) — plots a simple line graph with x-axis values against y-axis values
    - show() — displays the graph
    - title(‒string‖) — set the title of the plot as specified by the string
    - xlabel(‒string‖) — set the label for x-axis as specified by the string
    - ylabel(‒string‖) — set the label for y-axis as specified by the string
    - figure() — used to control a figure level attributes
    - subplot(nrows, ncols, index) — Add a subplot to the current figure
    - suptitle(‒string‖) — It adds a common title to the figure specified by the string
    - subplots(nrows, ncols, figsize) — a convenient way to create subplots, in a single call. It returns a tuple of a figure and number of axes.
    - set_title(‒string‖) — an axes level method used to set the title of subplots in a figure
    - bar(categorical variables, values, color) — used to create vertical bar graphs
    - barh(categorical variables, values, color) — used to create horizontal bar graphs
    - legend(loc) — used to make legend of the graph
    - xticks(index, categorical variables) — Get or set the current tick locations and labels of the x-axis
    - pie(value, categorical variables) — used to create a pie chart
    - hist(values, number of bins) — used to create a histogram
    - xlim(start value, end value) — used to set the limit of values of the x-axis
    - ylim(start value, end value) — used to set the limit of values of the y-axis
    - scatter(x-axis values, y-axis values) — plots a scatter plot with x-axis values against y-axis values
    - axes() — adds an axes to the current figure
    - set_xlabel(‒string‖) — axes level method used to set the x-label of the plot specified as a string
    - set_ylabel(‒string‖) — axes level method used to set the y-label of the plot specified as a string
    - scatter3D(x-axis values, y-axis values) — plots a three-dimensional scatter plot with x-axis values against y-axis values
    - plot3D(x-axis values, y-axis values) — plots a three-dimensional line graph with x-axis values against y-axis values

    Here we import Matplotlib‘s Pyplot module and Numpy library as most of the data that we will be working with will be in the form of arrays only.

Program-1

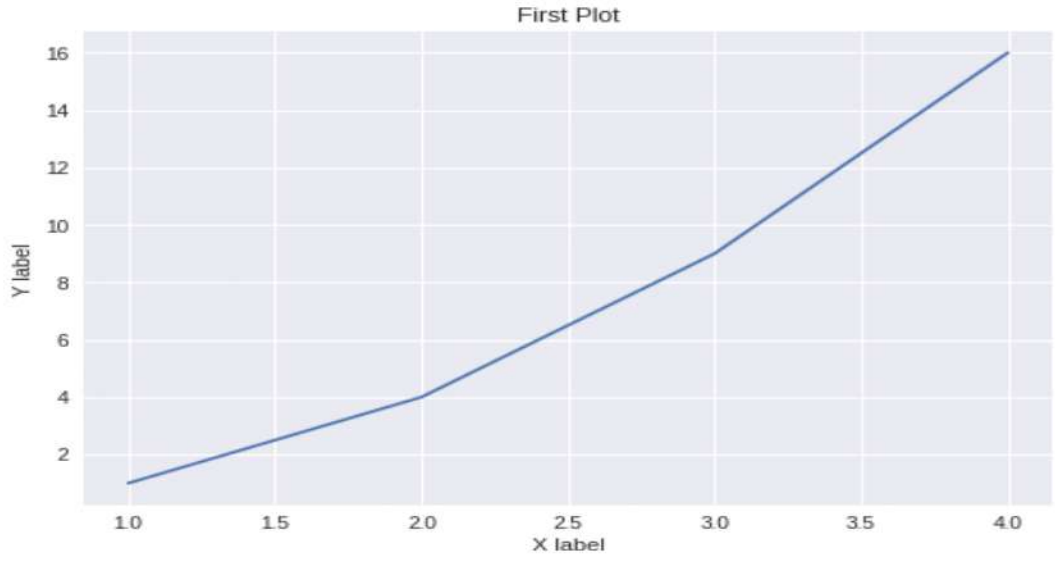
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```



We pass two arrays as our input arguments to Pyplot's plot() method and use show() method to invoke the required plot. Here note that the first array appears on the x-axis andsecond array appears on the y-axis of the plot. Now that our first plot is ready, let us add the title, and name x-axis and y-axis using methods title(), xlabel() and ylabel() respectively.

Program-2

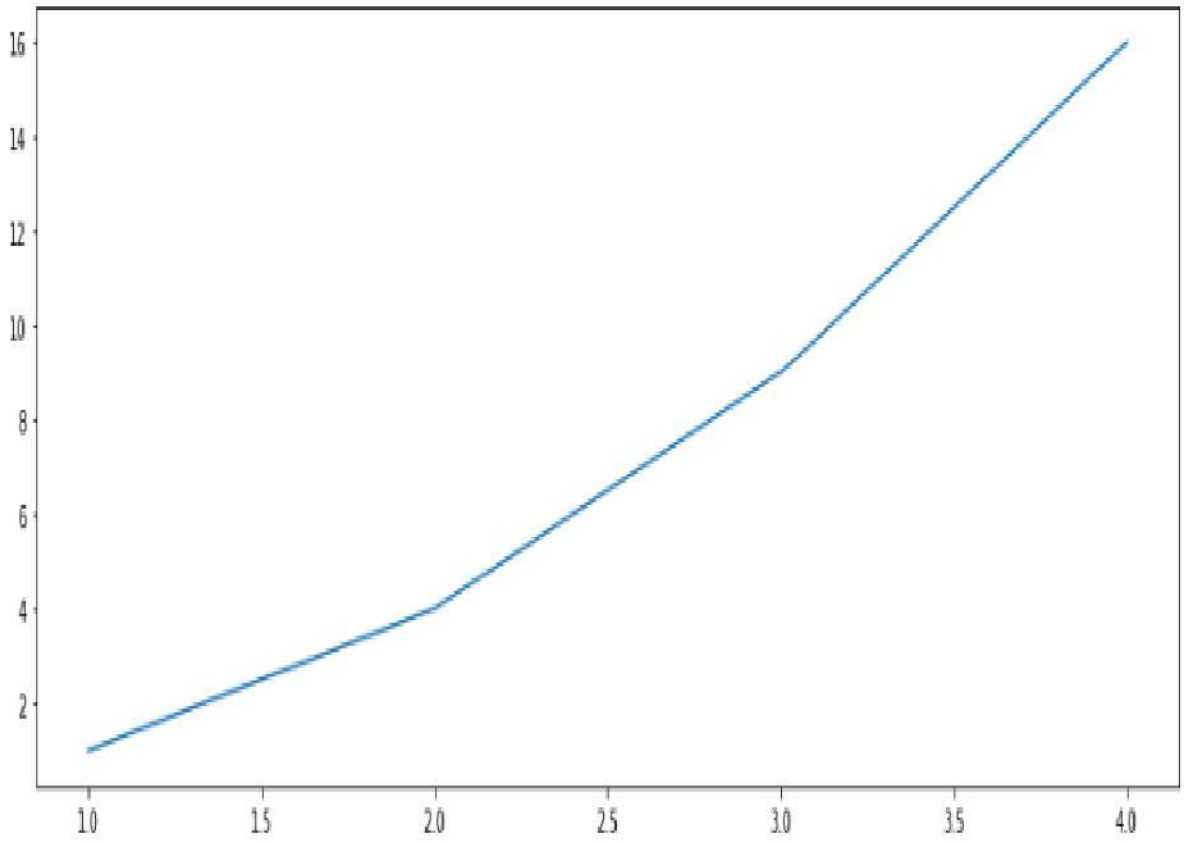
```
plt.plot([1,2,3,4],[1,4,9,16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



Program-3

We can also specify the size of the figure using method figure()and passing the valuesas a tuple of the length of rows and columns to the argument figsize

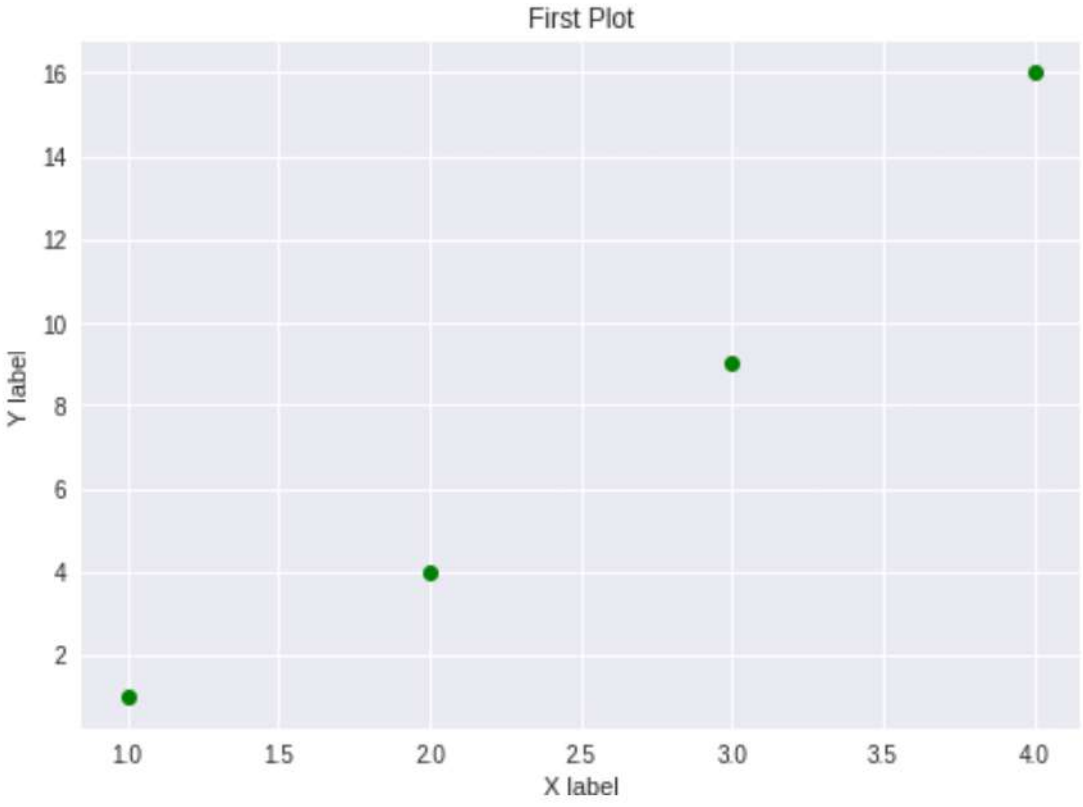
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(15,5))
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```



Program-4

With every X and Y argument, you can also pass an optional third argument in the formof a string which indicates the colour and line type of the plot. The default format is **b-** which means a solid blue line. In the figure below we use **go** which means green circles.Likewise, we can make many such combinations to format our plot.

```
plt.plot([1,2,3,4],[1,4,9,16],"go")
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



WEEK-2

a) Creation and Loading different datasets in Python

Program-1

Method-I

```
# Import pandas package
import pandas as pd

# Assign data
data = {'Name': ['Jai', 'Princi', 'Gaurav',
                 'Anuj', 'Ravi', 'Natasha', 'Riya'],
        'Age': [17, 17, 18, 17, 18, 17, 17],
        'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],
        'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}

# Convert into DataFrame
df = pd.DataFrame(data)

# Display data
df
```

	Name	Age	Gender	Marks
0	Jai	17	M	90
1	Princi	17	F	76
2	Gaurav	18	M	NaN
3	Anuj	17	M	74
4	Ravi	18	M	65
5	Natasha	17	F	NaN
6	Riya	17	F	71

Program-2

Method-II:

```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
print(boston_dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

Program-3 **Uploading csv file:**

Method-III:

```
import pandas as pd

df = pd.read_csv (r'E:\ml datasets\Machine-Learning-with-Python-master\Datasets\loan_data.csv')
print (df.head())
```

```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

    dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737       5639.958333      28854        52.1               0
1  14.29   707       2760.000000      33623        76.7               0
2  11.63   682       4710.000000       3511        25.6               1
3   8.10   712       2699.958333      33667        73.2               1
4  14.97   667       4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

b) Write a python program to compute Mean, Median, Mode, Variance, Standard Deviation using Datasets

- **Python Statistics library**

  This module provides functions for calculating mathematical statistics of numeric (Real-valued) data. The statistics module comes with very useful functions like: Mean, median, mode, standard deviation,and variance.

  The four functions we'll use in this post are common in statistics:

  1. mean - average value
  2. median - middle value
  3. mode - most often value
  4. standard deviation - spread of values

- **Averages and measures of central location**

  These functions calculate an average or typical value from a population or

sample.mean()	Arithmetic mean (–average‖) of data.
harmonic_mean()	Harmonic mean of data.
median()	Median (middle value) of
data.median_low()	Low median of data.
median_high()	High median of data.
median_grouped()	Median, or 50th percentile, of grouped
data.mode()	Mode (most common value) of discrete
data.	

- **Measures of spread**

  These functions calculate a measure of how much the population or sample tends to deviate fromthe typical or average values.

pstdev()	Population standard deviation of data.
pvariance()	Population variance of data.
stdev()	Sample standard deviation of data.
variance()	Sample variance of data.

Program-1

```
# Import statistics Library
import statistics

# Calculate average values
print(statistics.mean([1, 3, 5, 7, 9, 11, 13]))
print(statistics.mean([1, 3, 5, 7, 9, 11]))
print(statistics.mean([-11, 5.5, -3.4, 7.1, -9, 22]))
```

```
7
6
1.8666666666666667
```

Program-2

```
# Import statistics Library
import statistics

# Calculate middle values
print(statistics.median([1, 3, 5, 7, 9, 11, 13]))
print(statistics.median([1, 3, 5, 7, 9, 11]))
print(statistics.median([-11, 5.5, -3.4, 7.1, -9, 22]))
```

```
7
6.0
1.05
```

Program-3

```
# Import statistics Library
import statistics

# Calculate the mode
print(statistics.mode([1, 3, 3, 3, 5, 7, 7, 9, 11]))
print(statistics.mode([1, 1, 3, -5, 7, -9, 11]))
print(statistics.mode(['red', 'green', 'blue', 'red']
```

```
3
1
red
```

Program-4

```
# Import statistics Library
import statistics

# Calculate the standard deviation from a sample of data
print(statistics.stdev([1, 3, 5, 7, 9, 11]))
print(statistics.stdev([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.stdev([-11, 5.5, -3.4, 7.1]))
print(statistics.stdev([1, 30, 50, 100]))
```

```
3.7416573867739413
0.6925797186365384
8.414471660973929
41.67633221226008
```

Program-5

```
# Import statistics Library
import statistics

# Calculate the variance from a sample of data
print(statistics.variance([1, 3, 5, 7, 9, 11]))
print(statistics.variance([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.variance([-11, 5.5, -3.4, 7.1]))
print(statistics.variance([1, 30, 50, 100]))
```

```
14
0.4796666666666667
70.80333333333334
1736.9166666666667
```

c) Write a python program to compute reshaping the data, Filtering the data , merging the data and handling the missing values in datasets.

Program-1

Reshaping the data:

Method-I

```
import numpy as np
array1 = np.arange(8)
print("Original array : \n", array1)

# shape array with 2 rows and 4 columns
array2 = np.arange(8).reshape(2,4)
print("\narray reshaped with 2 rows and 4 columns : \n",array2)

# shape array with 4 rows and 2 columns
array3 = np.arange(8).reshape(4, 2)
print("\narray reshaped with 4 rows and 2 columns : \n",array3)

# Constructs 3D array
array4 = np.arange(8).reshape(2, 2, 2)
print("\nOriginal array reshaped to 3D : \n",array4)
```

```
Original array :
 [0 1 2 3 4 5 6 7]

array reshaped with 2 rows and 4 columns :
 [[0 1 2 3]
 [4 5 6 7]]

array reshaped with 4 rows and 2 columns :
 [[0 1]
 [2 3]
 [4 5]
 [6 7]]

Original array reshaped to 3D :
 [[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

Program-2

Method:II

Assigning the data:

```
 #Import pandas package
import pandas as pd

# Assign data
data = {'Name': ['Jai', 'Princi', 'Gaurav',
                 'Anuj', 'Ravi', 'Natasha', 'Riya'],
        'Age': [17, 17, 18, 17, 18, 17, 17],
        'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],
        'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}

# Convert into DataFrame
df = pd.DataFrame(data)

# Display data
df
```

	Name	Age	Gender	Marks
0	Jai	17	M	90
1	Princi	17	F	76
2	Gaurav	18	M	NaN
3	Anuj	17	M	74
4	Ravi	18	M	65
5	Natasha	17	F	NaN
6	Riya	17	F	71

Program-3

```
# Categorize gender
df['Gender'] = df['Gender'].map({'M': 0,
                                 'F': 1, }).astype(float)

# Display data
df
```

	Name	Age	Gender	Marks
0	Jai	17	0.0	90
1	Princi	17	1.0	76
2	Gaurav	18	0.0	NaN
3	Anuj	17	0.0	74
4	Ravi	18	0.0	65
5	Natasha	17	1.0	NaN
6	Riya	17	1.0	71