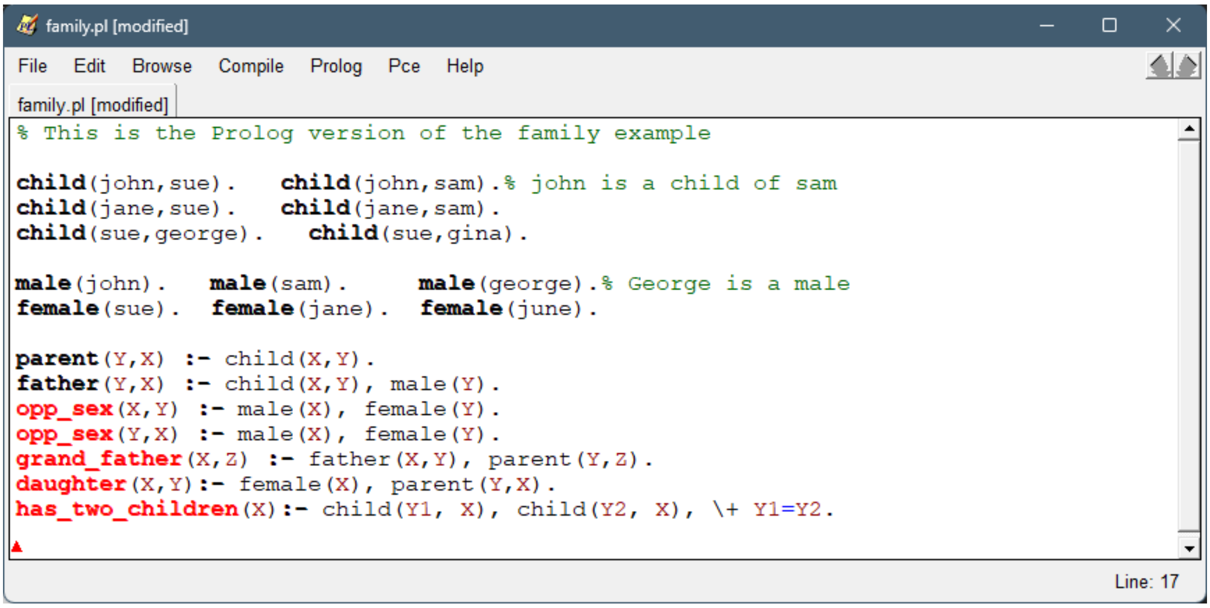
Practical 1: Logical Programming with prolog representation of family relationship

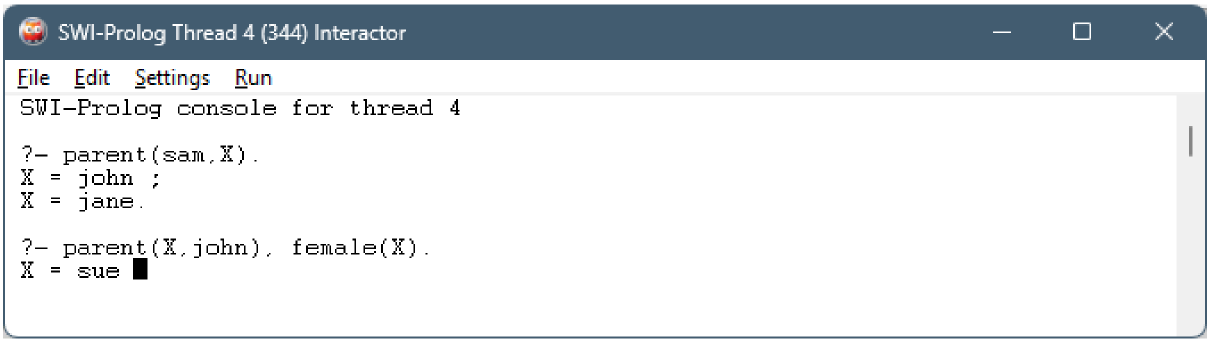
a) Implement family relationships in Prolog as a Family KB using predicates: child, father, mother, male, female, parent, grandfather using Prolog. Make your own assumptions with respect to the needed atomic and conditional sentences. Demonstrate the program by establishing various types of queries pertaining to family relationships.

Knowledge Base:



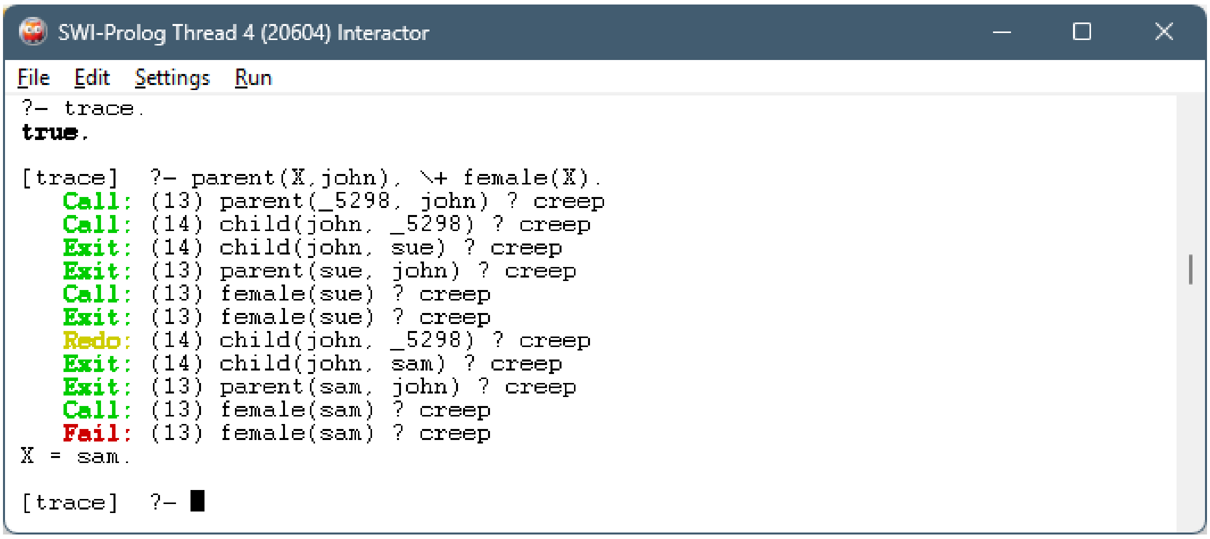
```
% This is the Prolog version of the family example

child(john,sue).   child(john,sam).% john is a child of sam
child(jane,sue).   child(jane,sam).
child(sue,george).   child(sue,gina).

male(john).   male(sam).     male(george).% George is a male
female(sue).  female(jane).  female(june).

parent(Y,X) :- child(X,Y).
father(Y,X) :- child(X,Y), male(Y).
opp_sex(X,Y) :- male(X), female(Y).
opp_sex(Y,X) :- male(X), female(Y).
grand_father(X,Z) :- father(X,Y), parent(Y,Z).
daughter(X,Y):- female(X), parent(Y,X).
has_two_children(X):- child(Y1, X), child(Y2, X), \+ Y1=Y2.
```

Demonstrating the program by establishing various types of queries pertaining to family relationships.



```
SWI-Prolog console for thread 4

?- parent(sam,X).
X = john ;
X = jane.

?- parent(X,john), female(X).
X = sue
```

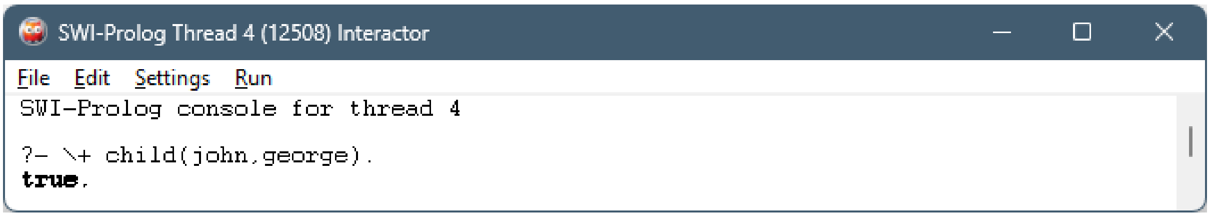
b) Trace the Prolog back chaining on various queries on the Family KB program. Note down the four types of output: Call, Exit, Redo and Fail in the back-chaining trace.



```
?- trace.
true.

[trace]  ?- parent(X,john), \+ female(X).
   Call: (13) parent(_5298, john) ? creep
   Call: (14) child(john, _5298) ? creep
   Exit: (14) child(john, sue) ? creep
   Exit: (13) parent(sue, john) ? creep
   Call: (13) female(sue) ? creep
   Exit: (13) female(sue) ? creep
   Redo: (14) child(john, _5298) ? creep
   Exit: (14) child(john, sam) ? creep
   Exit: (13) parent(sam, john) ? creep
   Call: (13) female(sam) ? creep
   Fail: (13) female(sam) ? creep
X = sam.

[trace]  ?-
```

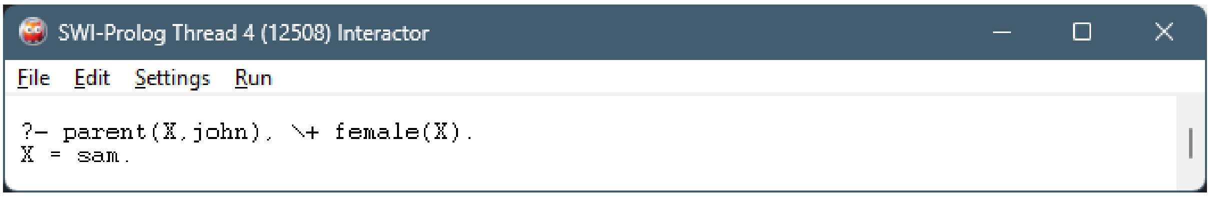
c)Demonstrate negated and conjunctive queries using the Family KB program.

Negated queries:



```
SWI-Prolog console for thread 4

?- \+ child(john,george).
true.
```

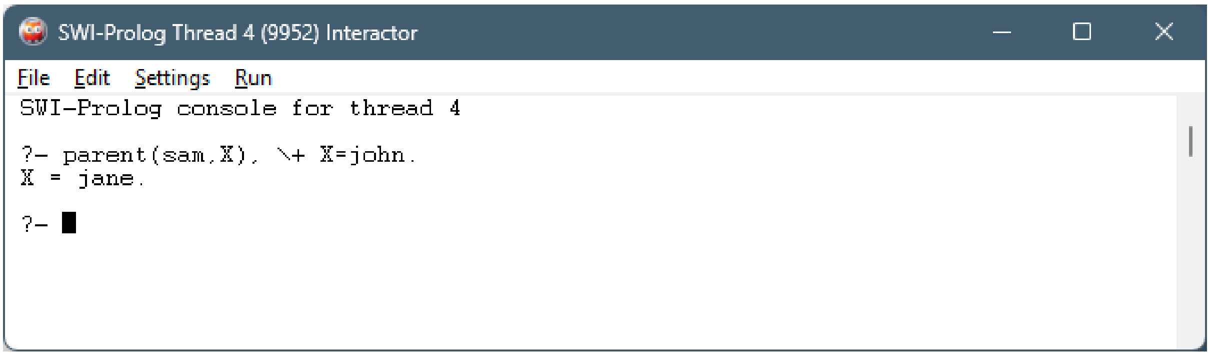
Conjunctive queries:



```
?- parent(X,john), \+ female(X).
X = sam.
```

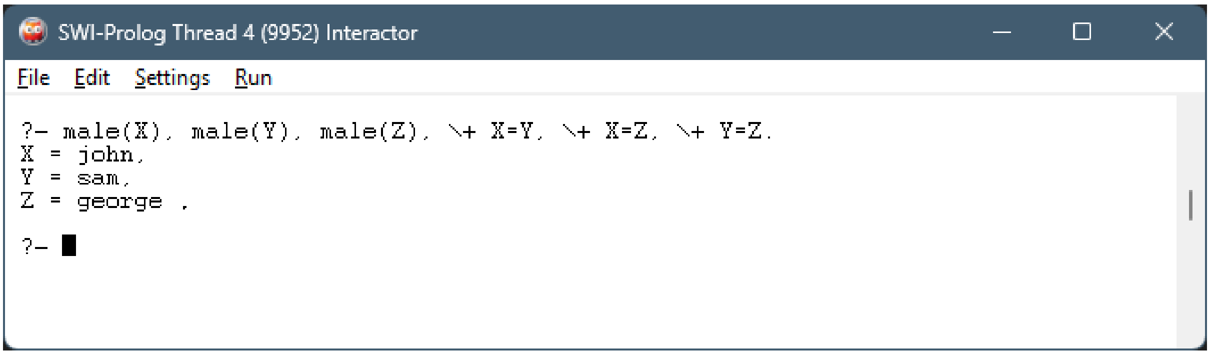
d) Demonstrate equality queries in Prolog. Try to find out three different males in family knowledge base using combination of equality and negation in queries.

Equality queries:



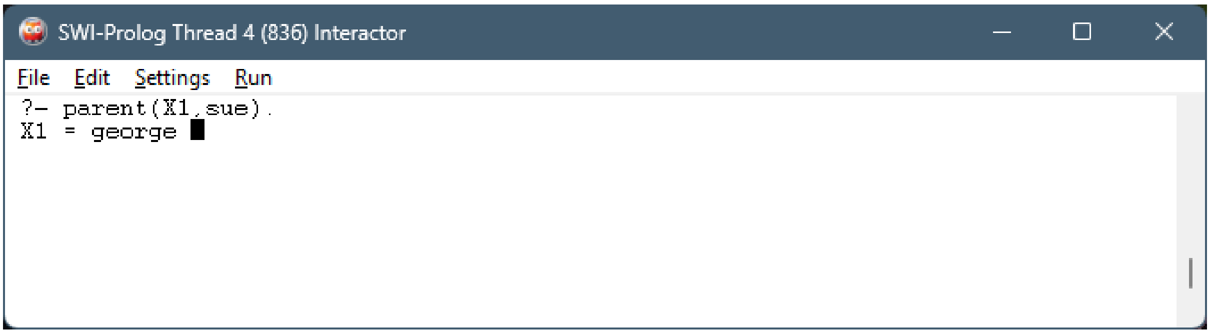
```
SWI-Prolog console for thread 4

?- parent(sam,X), \+ X=john.
X = jane.

?-
```

Try to find out three different males in family knowledge base using combination of equality and negation in queries:



```
?- male(X), male(Y), male(Z), \+ X=Y, \+ X=Z, \+ Y=Z.
X = john,
Y = sam,
Z = george .

?-
```

e) Demonstrate the use of anonymous variables in Prolog queries using the Family KB.

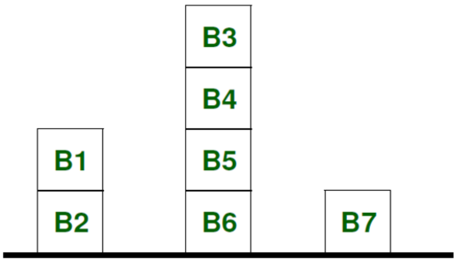


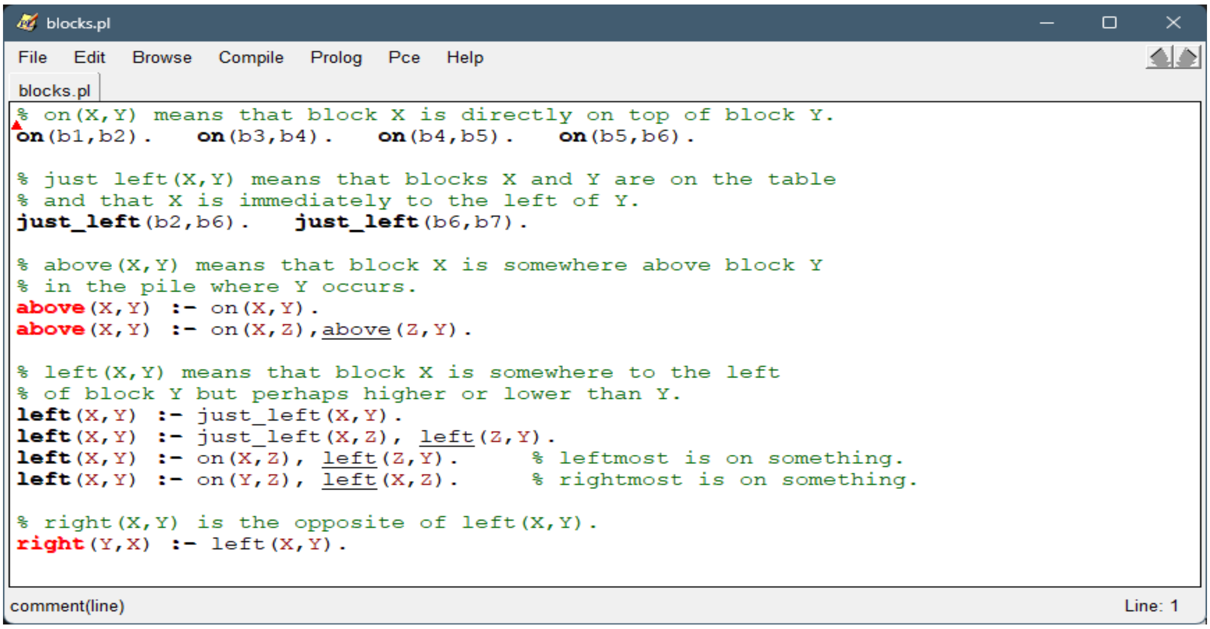
```
?- parent(X1,sue).
X1 = george
```

Practical 2: Problem Solving with Prolog

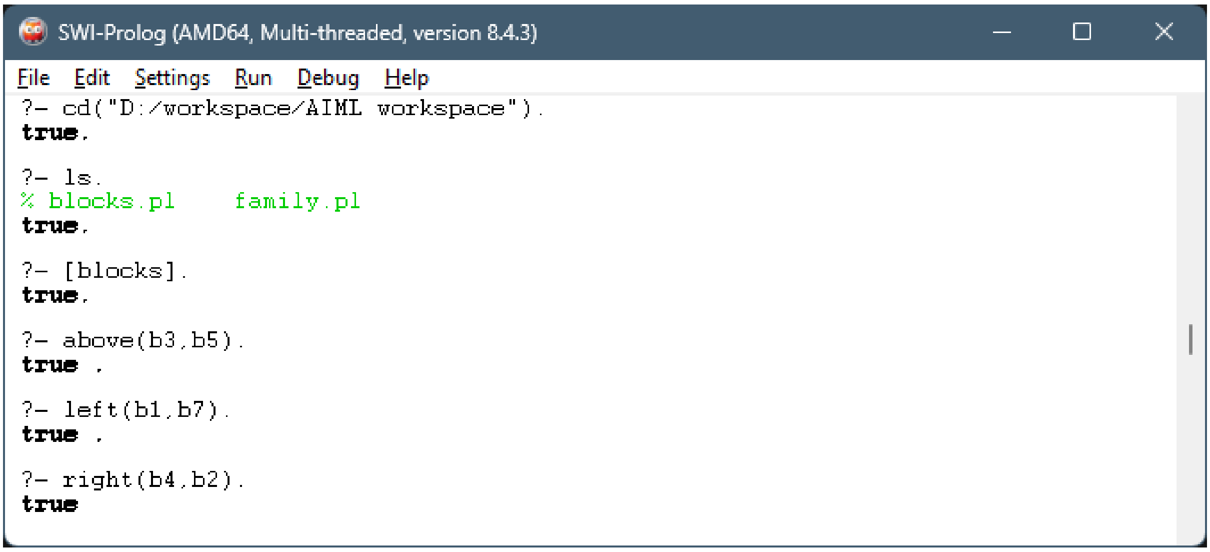
a) Describe the “Blocks World” scene shown below to Prolog such that the following can be determined through Prolog queries:

- Block 3 is above Block 5
- Block 1 is to the left of Block 7
- Block 4 is to the right of Block 2



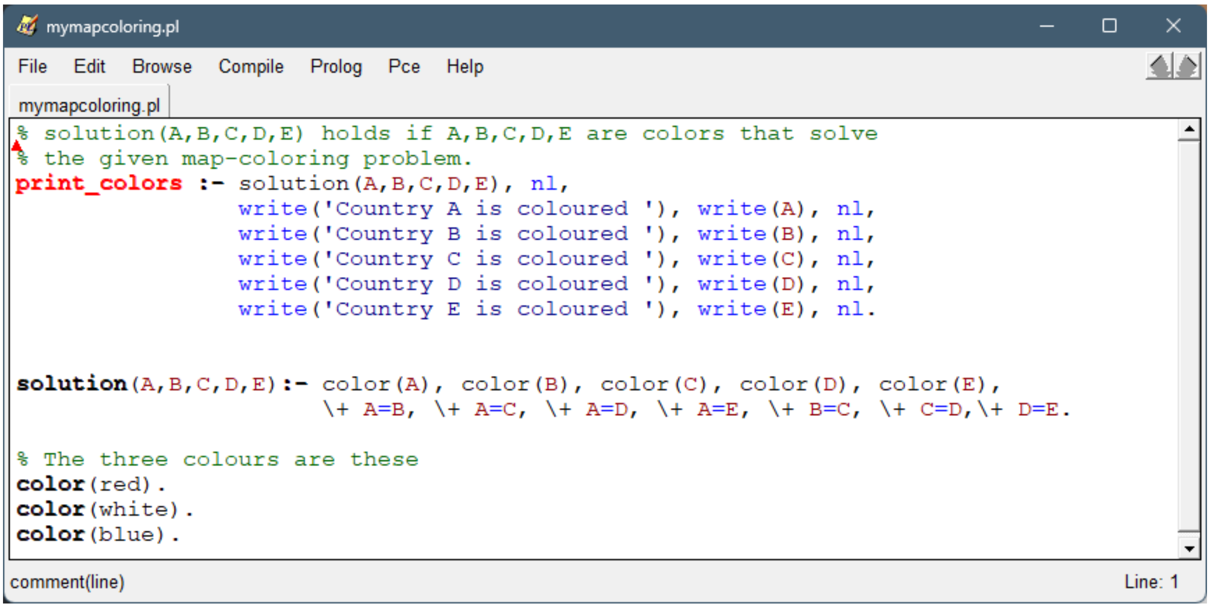


```
% on(X,Y) means that block X is directly on top of block Y.
on(b1,b2).    on(b3,b4).    on(b4,b5).    on(b5,b6).

% just left(X,Y) means that blocks X and Y are on the table
% and that X is immediately to the left of Y.
just_left(b2,b6).    just_left(b6,b7).

% above(X,Y) means that block X is somewhere above block Y
% in the pile where Y occurs.
above(X,Y)  :- on(X,Y).
above(X,Y)  :- on(X,Z),above(Z,Y).

% left(X,Y) means that block X is somewhere to the left
% of block Y but perhaps higher or lower than Y.
left(X,Y)  :- just_left(X,Y).
left(X,Y)  :- just_left(X,Z), left(Z,Y).
left(X,Y)  :- on(X,Z), left(Z,Y).      % leftmost is on something.
left(X,Y)  :- on(Y,Z), left(X,Z).      % rightmost is on something.

% right(X,Y) is the opposite of left(X,Y).
right(Y,X)  :- left(X,Y).
```

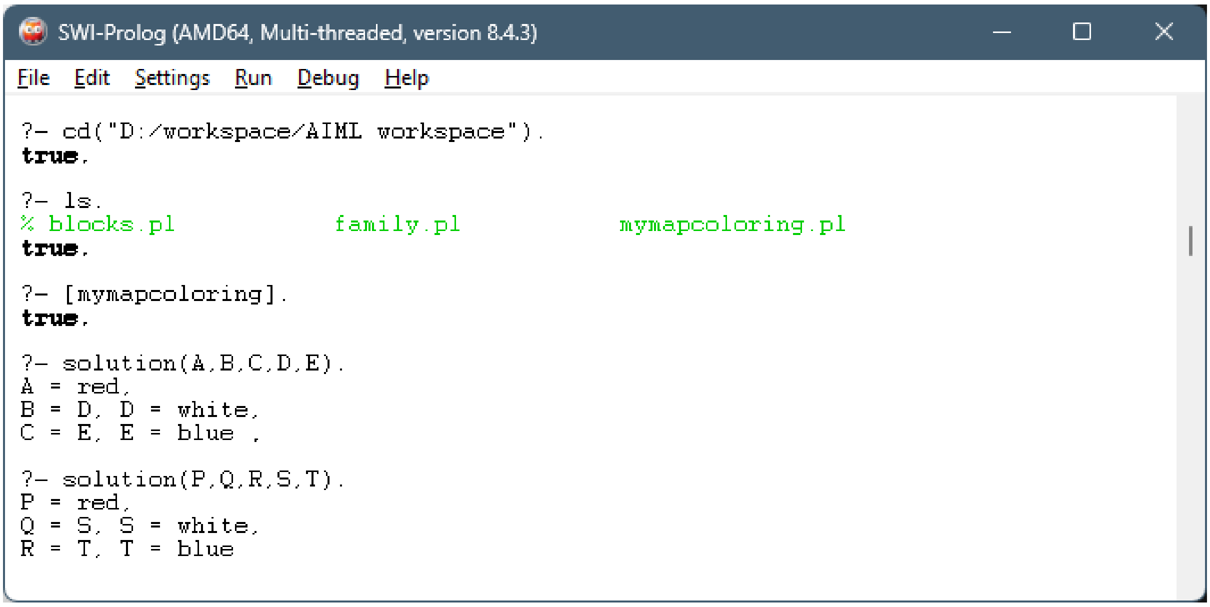

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help

```
?- cd("D:/workspace/AIML workspace").
true.

?- ls.
% blocks.pl     family.pl
true.

?- [blocks].
true.

?- above(b3,b5).
true .

?- left(b1,b7).
true .

?- right(b4,b2).
true
```

b) Map Coloring Problem: Illustrate the solving of the popular constraint satisfaction problem known as Map coloring problem using Prolog.

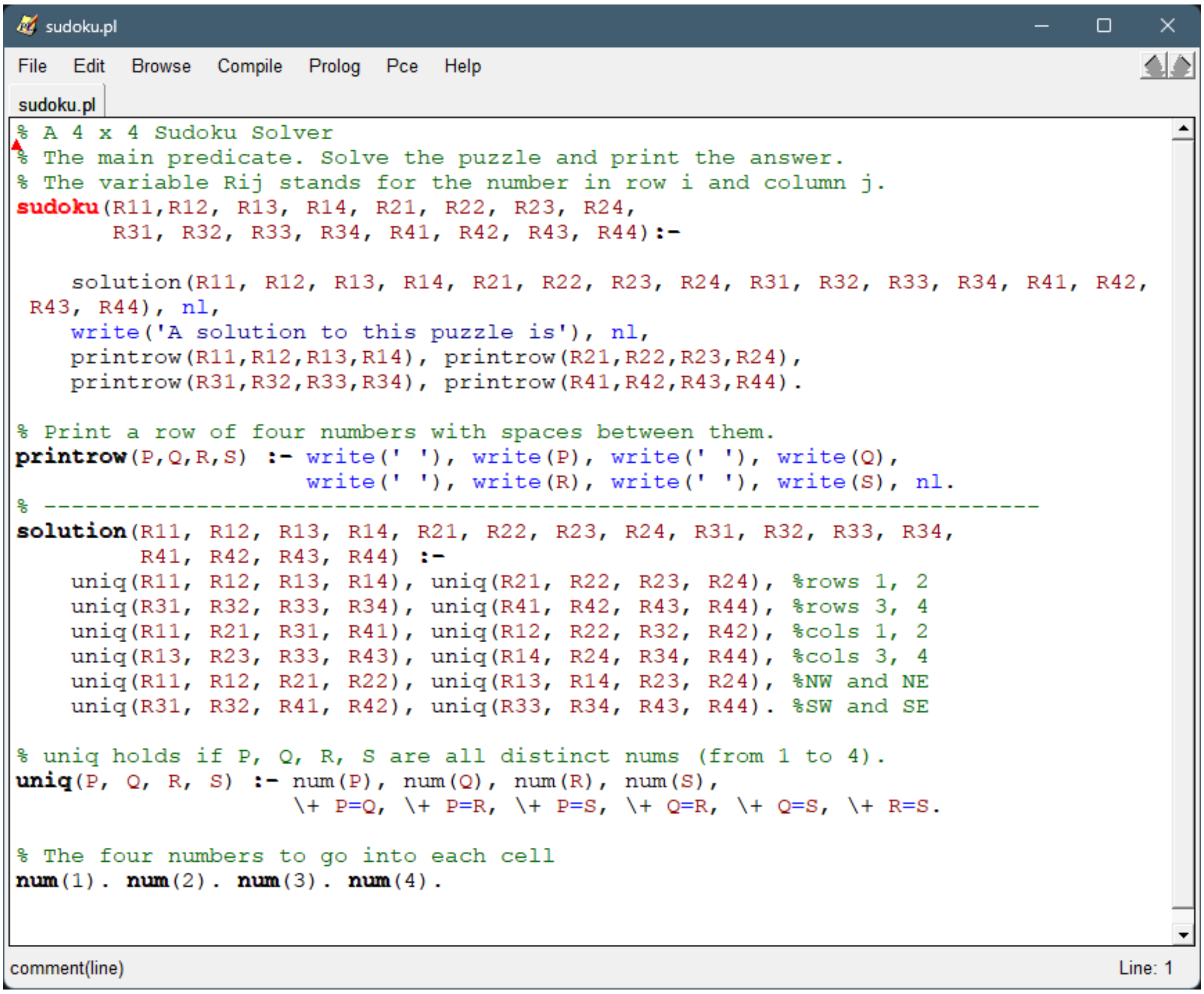


```prolog
% solution(A,B,C,D,E) holds if A,B,C,D,E are colors that solve
% the given map-coloring problem.
print_colors :- solution(A,B,C,D,E), nl,
                write('Country A is coloured '), write(A), nl,
                write('Country B is coloured '), write(B), nl,
                write('Country C is coloured '), write(C), nl,
                write('Country D is coloured '), write(D), nl,
                write('Country E is coloured '), write(E), nl.


solution(A,B,C,D,E):- color(A), color(B), color(C), color(D), color(E),
                      \+ A=B, \+ A=C, \+ A=D, \+ A=E, \+ B=C, \+ C=D,\+ D=E.

% The three colours are these
color(red).
color(white).
color(blue).
```

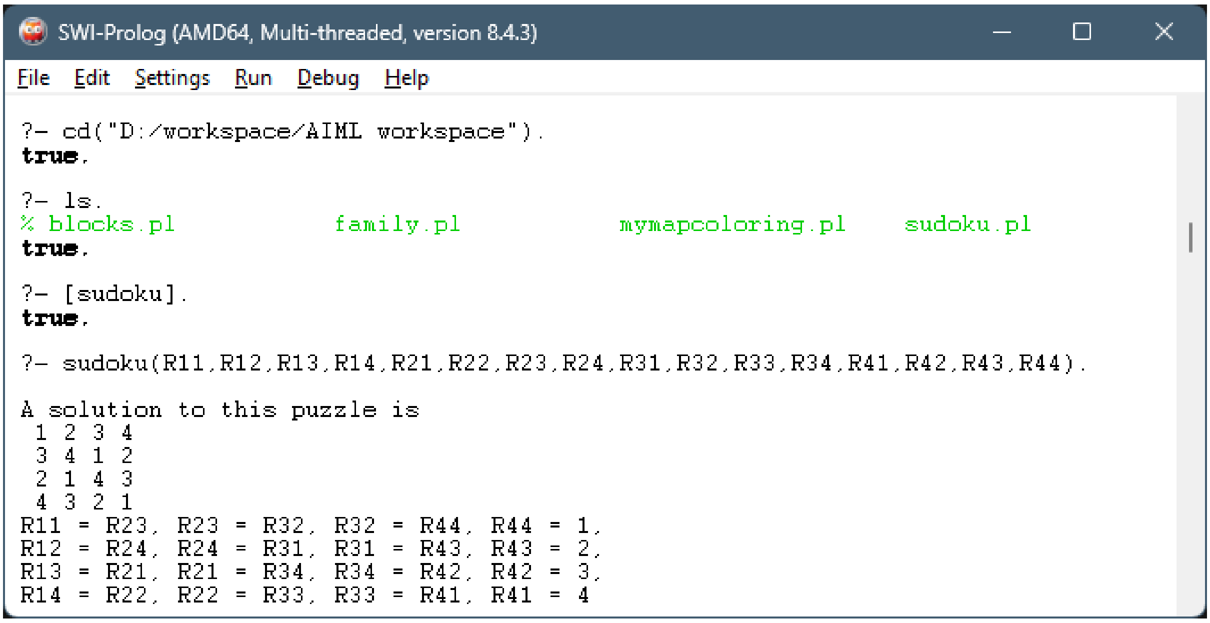

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help

```
?- cd("D:/workspace/AIML workspace").
true.

?- ls.
% blocks.pl              family.pl             mymapcoloring.pl
true.

?- [mymapcoloring].
true.

?- solution(A,B,C,D,E).
A = red,
B = D, D = white,
C = E, E = blue .

?- solution(P,Q,R,S,T).
P = red,
Q = S, S = white,
R = T, T = blue
```

c) Mini Version of Sudoku Puzzle: Illustrate the solving of a 4x4 Sudoku puzzle using prolog. The numbers can be between 1 to 4. Each row, column and quadrant should have distinct numbers.



```
% A 4 x 4 Sudoku Solver
% The main predicate. Solve the puzzle and print the answer.
% The variable Rij stands for the number in row i and column j.
sudoku(R11,R12, R13, R14, R21, R22, R23, R24,
       R31, R32, R33, R34, R41, R42, R43, R44):-

    solution(R11, R12, R13, R14, R21, R22, R23, R24, R31, R32, R33, R34, R41, R42,
 R43, R44), nl,
    write('A solution to this puzzle is'), nl,
    printrow(R11,R12,R13,R14), printrow(R21,R22,R23,R24),
    printrow(R31,R32,R33,R34), printrow(R41,R42,R43,R44).

% Print a row of four numbers with spaces between them.
printrow(P,Q,R,S) :- write(' '), write(P), write(' '), write(Q),
                     write(' '), write(R), write(' '), write(S), nl.
% ---------------------------------------------------------------------------
solution(R11, R12, R13, R14, R21, R22, R23, R24, R31, R32, R33, R34,
         R41, R42, R43, R44) :-
    uniq(R11, R12, R13, R14), uniq(R21, R22, R23, R24), %rows 1, 2
    uniq(R31, R32, R33, R34), uniq(R41, R42, R43, R44), %rows 3, 4
    uniq(R11, R21, R31, R41), uniq(R12, R22, R32, R42), %cols 1, 2
    uniq(R13, R23, R33, R43), uniq(R14, R24, R34, R44), %cols 3, 4
    uniq(R11, R12, R21, R22), uniq(R13, R14, R23, R24), %NW and NE
    uniq(R31, R32, R41, R42), uniq(R33, R34, R43, R44). %SW and SE

% uniq holds if P, Q, R, S are all distinct nums (from 1 to 4).
uniq(P, Q, R, S) :- num(P), num(Q), num(R), num(S),
                    \+ P=Q, \+ P=R, \+ P=S, \+ Q=R, \+ Q=S, \+ R=S.

% The four numbers to go into each cell
num(1). num(2). num(3). num(4).
```



```
?- cd("D:/workspace/AIML workspace").
true.

?- ls.
% blocks.pl           family.pl           mymapcoloring.pl     sudoku.pl
true.

?- [sudoku].
true.

?- sudoku(R11,R12,R13,R14,R21,R22,R23,R24,R31,R32,R33,R34,R41,R42,R43,R44).

A solution to this puzzle is
 1 2 3 4
 3 4 1 2
 2 1 4 3
 4 3 2 1
R11 = R23, R23 = R32, R32 = R44, R44 = 1,
R12 = R24, R24 = R31, R31 = R43, R43 = 2,
R13 = R21, R21 = R34, R34 = R42, R42 = 3,
R14 = R22, R22 = R33, R33 = R41, R41 = 4
```

Practical 3: Introduction to Python Programming:

a) Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn.

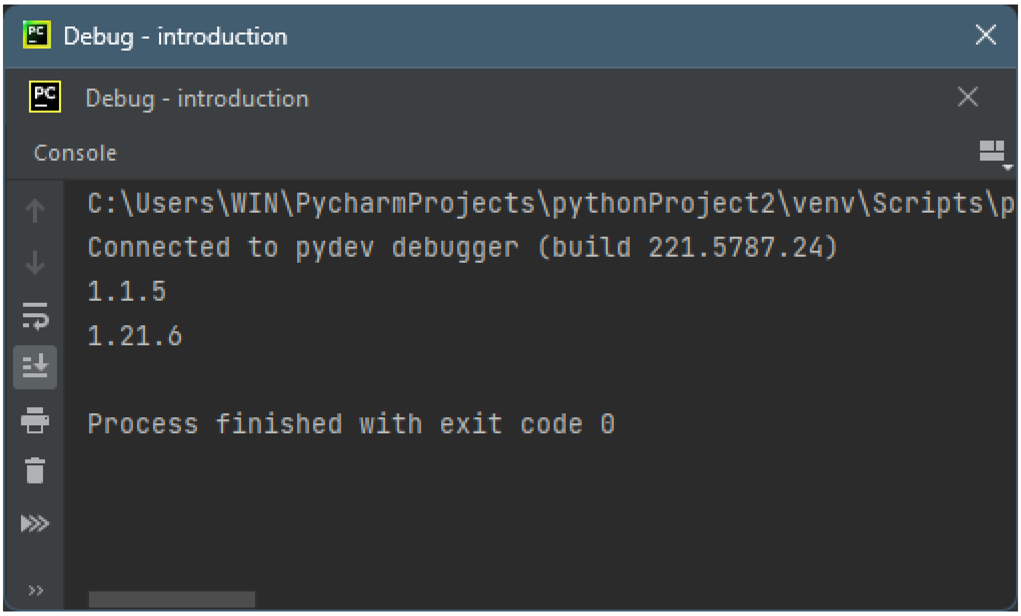
Learning Numpy:

1)Import Numpy, pandas, and display its version.

Source code:

```
import pandas as pd
import numpy as np
print(pd.__version__)
print(np.__version__)
```

Output:

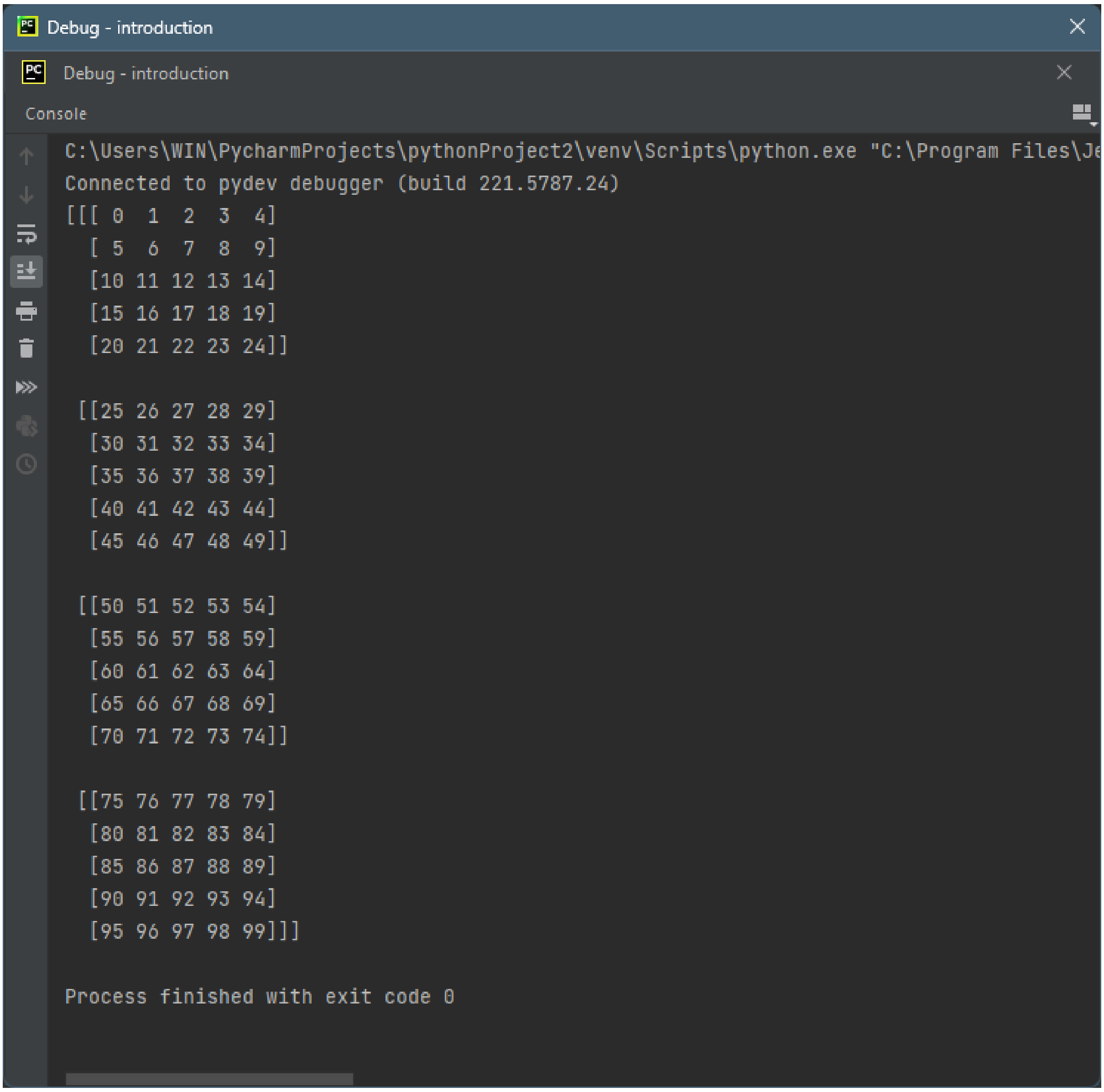


```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\p
Connected to pydev debugger (build 221.5787.24)
1.1.5
1.21.6

Process finished with exit code 0
```

2)Create an array of 0 to 99 numbers and convert the one dimensional array into 3 dimensional array and display array elements on screen.
Source code:

```
import numpy as np
x = np.arange(100).reshape(4,5,5)
x
print(x)
```

Output:



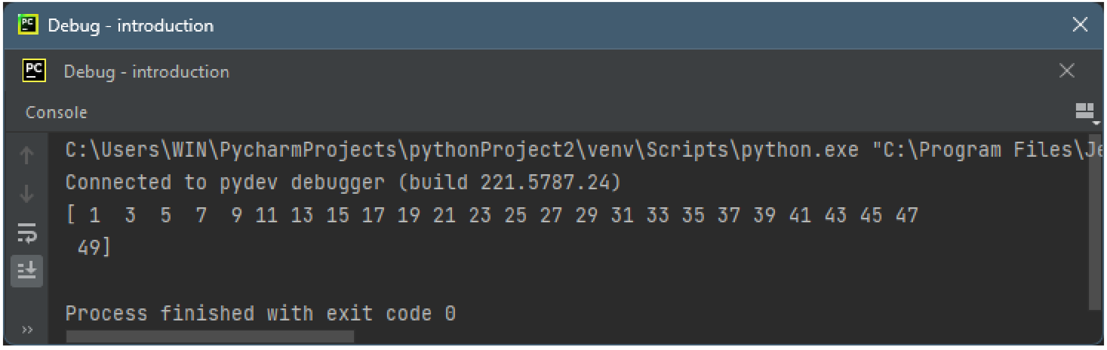
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\Je
Connected to pydev debugger (build 221.5787.24)
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]
  [15 16 17 18 19]
  [20 21 22 23 24]]

 [[25 26 27 28 29]
  [30 31 32 33 34]
  [35 36 37 38 39]
  [40 41 42 43 44]
  [45 46 47 48 49]]

 [[50 51 52 53 54]
  [55 56 57 58 59]
  [60 61 62 63 64]
  [65 66 67 68 69]
  [70 71 72 73 74]]

 [[75 76 77 78 79]
  [80 81 82 83 84]
  [85 86 87 88 89]
  [90 91 92 93 94]
  [95 96 97 98 99]]]

Process finished with exit code 0
```

3)Create an array of 0 to 49 numbers and display only odd numbers from the array.

Source code:

```
import numpy as np
x = np.arange(50)
x = np.array([i for i in range(50) if i%2 != 0])
print(x)
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\Je
Connected to pydev debugger (build 221.5787.24)
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49]

Process finished with exit code 0
```

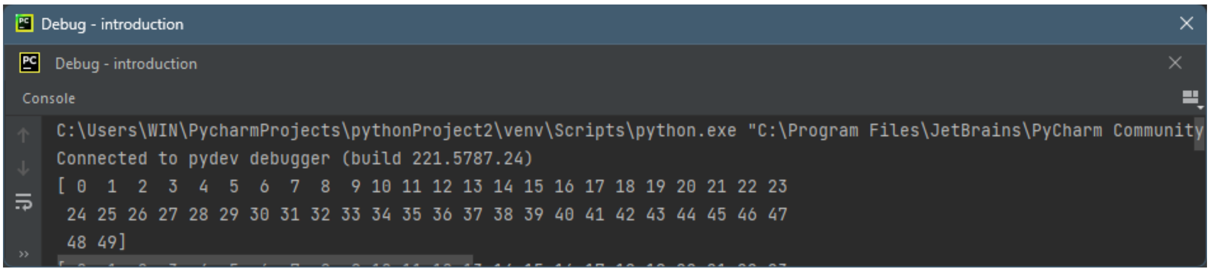
4)Create an array of 0 to 49 numbers and replace all even numbers by 0. Display array elements on screen.

Source code:

```
import numpy as np

x = np.arange(50)
for i in range(50):
  if i % 2 == 0:
    x[i] = 0
print(x)
```

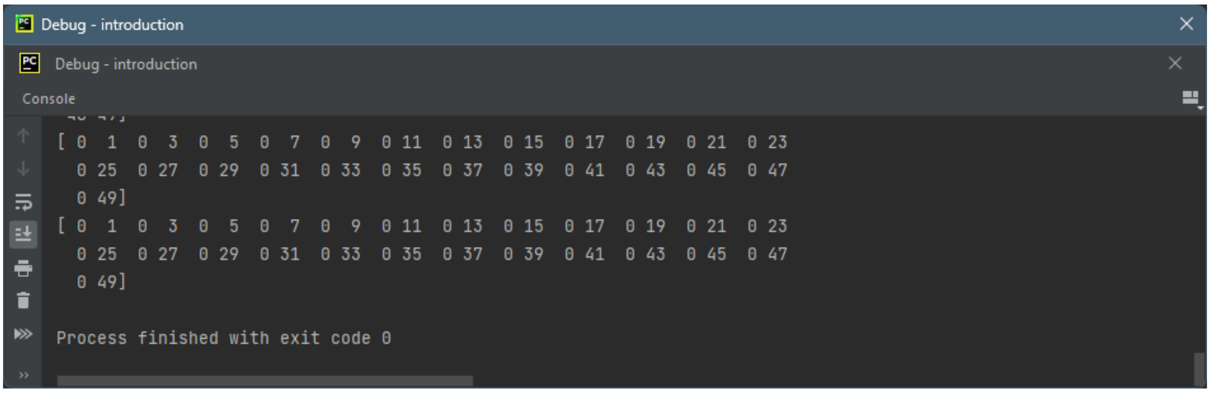
Output:

Before:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community
Connected to pydev debugger (build 221.5787.24)
[ 0  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 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49]
```

After :



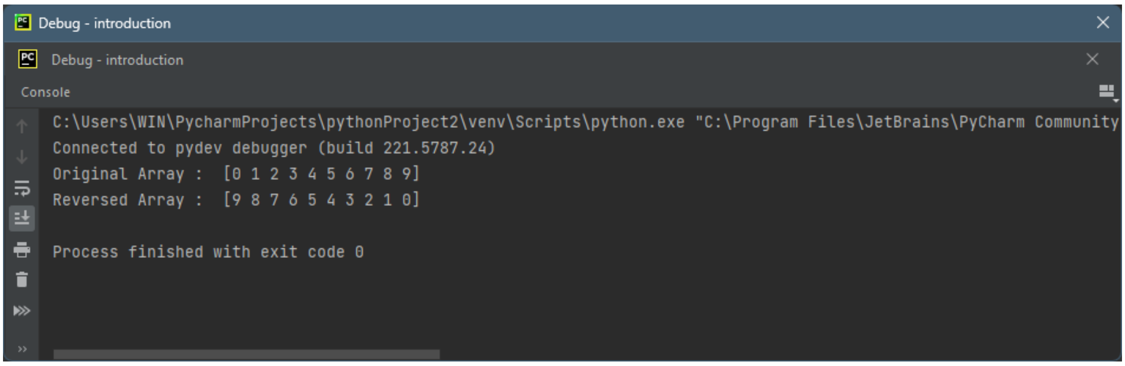
```
[ 0  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19  0 21  0 23
  0 25  0 27  0 29  0 31  0 33  0 35  0 37  0 39  0 41  0 43  0 45  0 47
  0 49]
[ 0  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17  0 19  0 21  0 23
  0 25  0 27  0 29  0 31  0 33  0 35  0 37  0 39  0 41  0 43  0 45  0 47
  0 49]

Process finished with exit code 0
```

5)Create an array of 0 to 9 numbers and reverse it.
e.g. if array is [0,1,2,3,4] then output [4,3,2,1,0].

Source Code:

```
import numpy as np
x = np.arange(10)
print('Original Array : ', x)
print('Reversed Array : ', x[::-1])
```

Output:



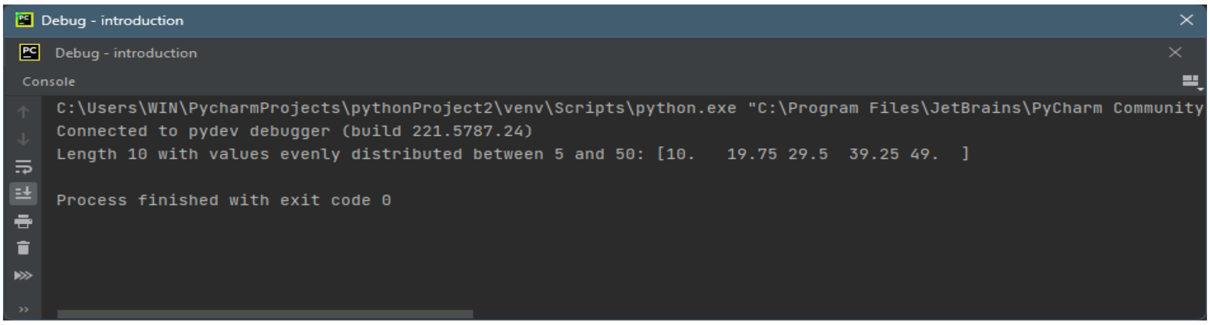
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community
Connected to pydev debugger (build 221.5787.24)
Original Array :  [0 1 2 3 4 5 6 7 8 9]
Reversed Array :  [9 8 7 6 5 4 3 2 1 0]

Process finished with exit code 0
```

6) Write a NumPy code to create an array/ vector of length 10 with values evenly distributed between 5 and 50.

Source Code:

```
import numpy as np
vector = np.linspace(10, 49, 5)
print("Length 10 with values evenly distributed between 5 and 50:", vector)
```

Output:



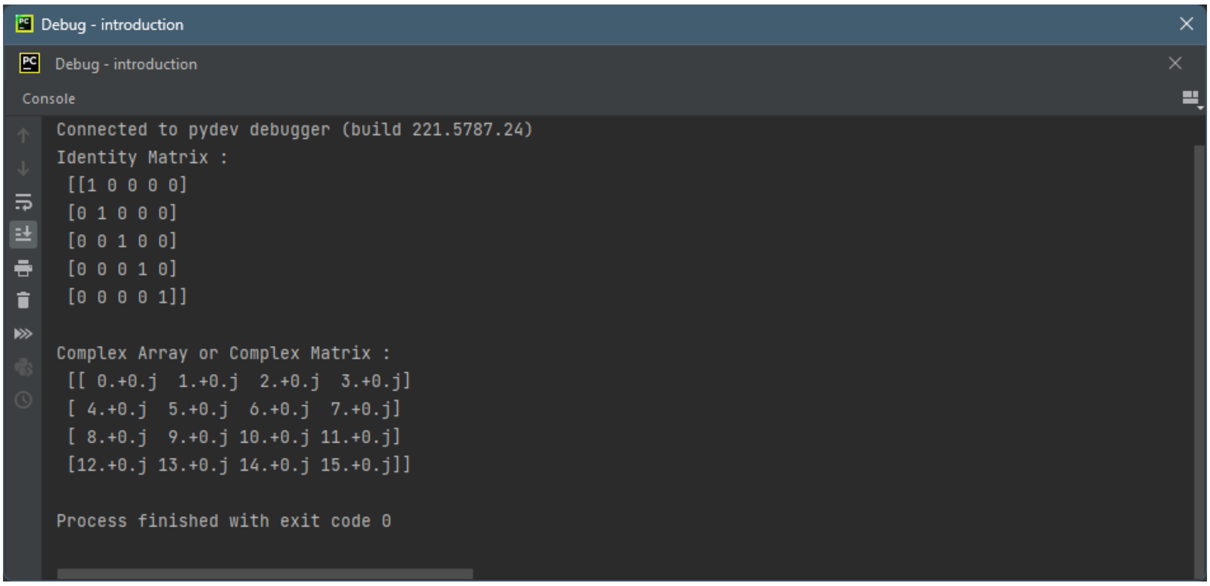
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community
Connected to pydev debugger (build 221.5787.24)
Length 10 with values evenly distributed between 5 and 50: [10.   19.75 29.5  39.25 49.  ]

Process finished with exit code 0
```

7) Create 5X5 identity matrix.
Create 4X4 complex number array.

Source code:

```
import numpy as np
# 5x5 matrix with 1's on main diagonal
i = np.identity(5, dtype = int)
print("Identity Matrix : \n", i)
# 4x4 array with complex numbers
x = np.arange(16, dtype = complex).reshape(4,4)
print("\nComplex Array or Complex Matrix : \n", x)
```

Output:



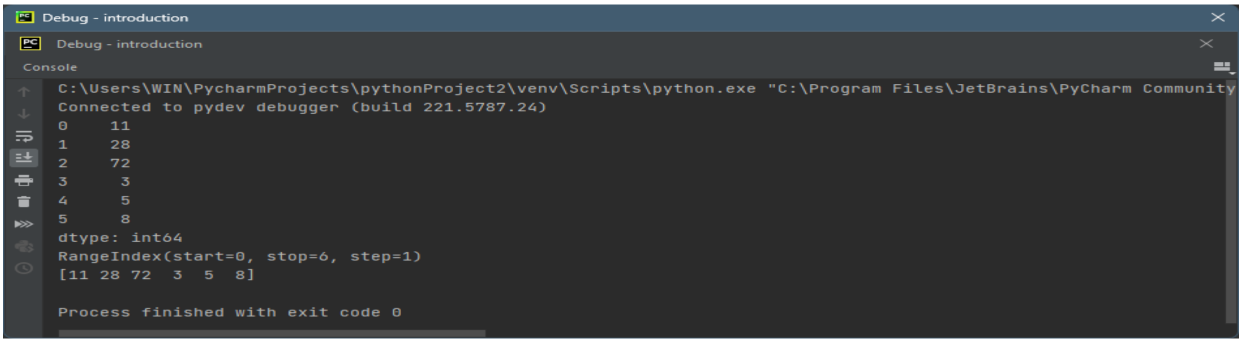
```
Connected to pydev debugger (build 221.5787.24)
Identity Matrix :
 [[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]

Complex Array or Complex Matrix :
 [[ 0.+0.j  1.+0.j  2.+0.j  3.+0.j]
 [ 4.+0.j  5.+0.j  6.+0.j  7.+0.j]
 [ 8.+0.j  9.+0.j 10.+0.j 11.+0.j]
 [12.+0.j 13.+0.j 14.+0.j 15.+0.j]]

Process finished with exit code 0
```

8) Define a simple Series object

Source Code:

```
import pandas as pd
S = pd.Series([11, 28, 72, 3, 5, 8])
print(S)
# We can directly access the index and the values of our Series S:
print(S.index)
print(S.values)
```

Output:



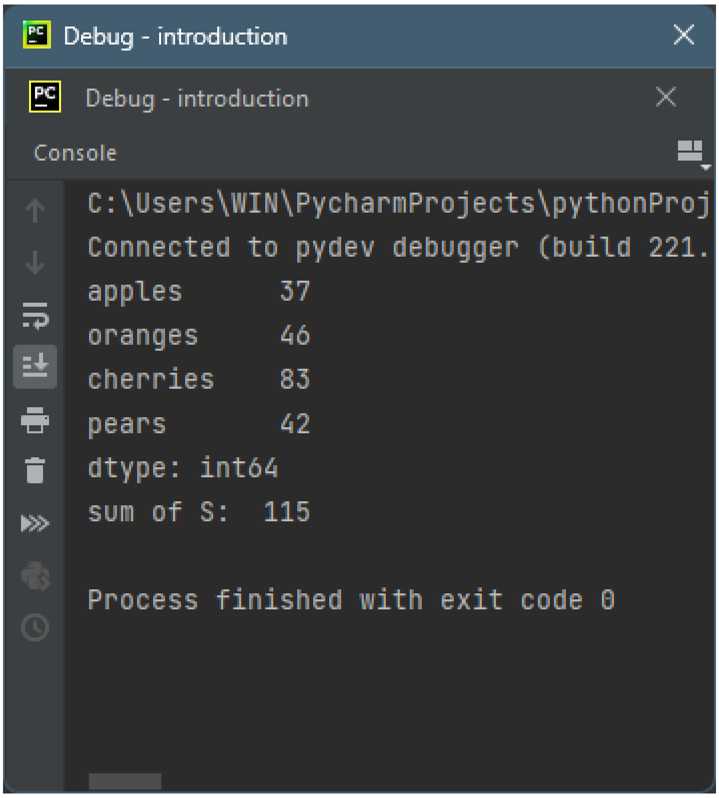
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community
Connected to pydev debugger (build 221.5787.24)
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
RangeIndex(start=0, stop=6, step=1)
[11 28 72  3  5  8]

Process finished with exit code 0
```

9) Add two series with the same indices,
we get a new series with the same index and the correponding values will be added:

Source Code:

```
import pandas as pd
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:



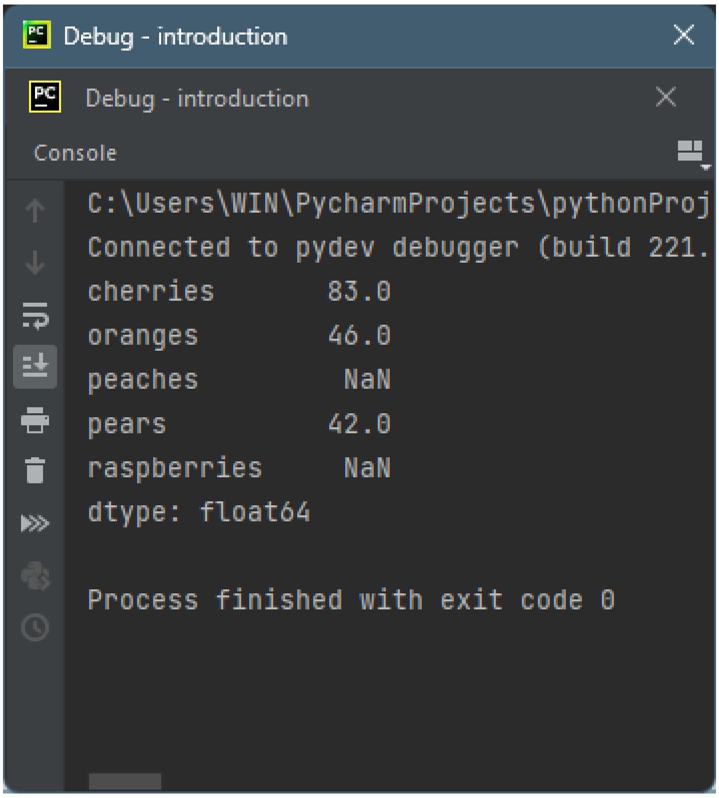
```
C:\Users\WIN\PycharmProjects\pythonProj
Connected to pydev debugger (build 221.
apples      37
oranges     46
cherries    83
pears       42
dtype: int64
sum of S:  115

Process finished with exit code 0
```

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:

Source Code:

```
import pandas as pd
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:



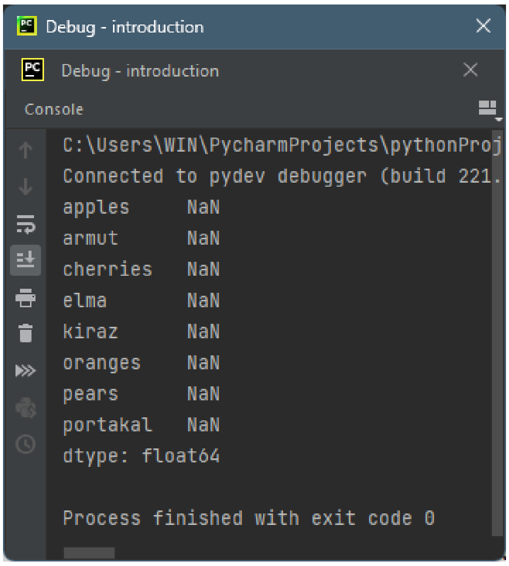
```
C:\Users\WIN\PycharmProjects\pythonProj
Connected to pydev debugger (build 221.
cherries       83.0
oranges        46.0
peaches         NaN
pears          42.0
raspberries     NaN
dtype: float64

Process finished with exit code 0
```

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:

Source Code:

```
import pandas as pd
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:



```
C:\Users\WIN\PycharmProjects\pythonProj
Connected to pydev debugger (build 221.
apples      NaN
armut       NaN
cherries    NaN
elma        NaN
kiraz       NaN
oranges     NaN
pears       NaN
portakal    NaN
dtype: float64

Process finished with exit code 0
```

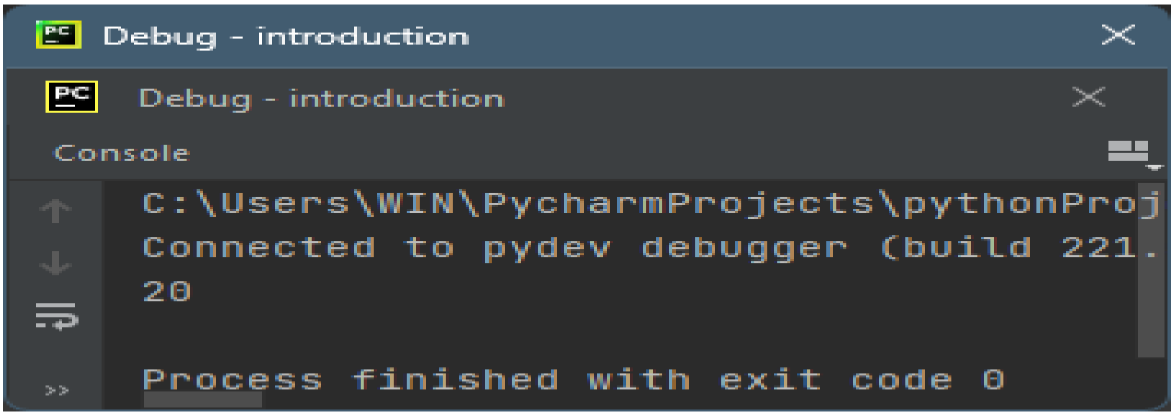
Indexing :

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

Source Code:

```
import pandas as pd
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['apples'])
```

Output:



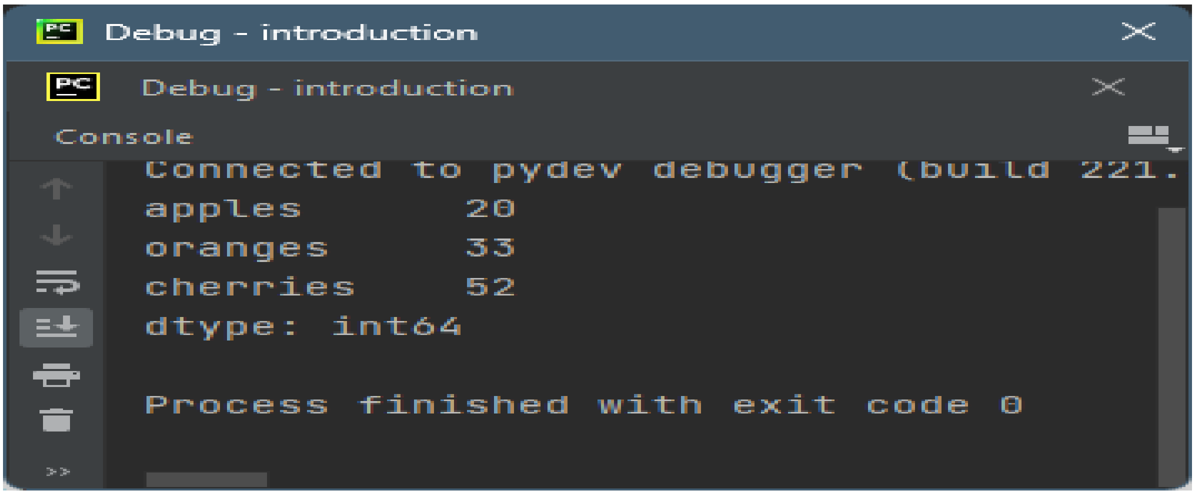
```
C:\Users\WIN\PycharmProjects\pythonProj
Connected to pydev debugger (build 221.
20

Process finished with exit code 0
```

This looks like accessing the values of dictionaries through keys. However, Series objects can also be accessed by multiple indexes at the same time. This can be done by packing the indexes into a list. This type of access returns a Pandas Series again:

Source Code:

```
import pandas as pd
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[['apples', 'oranges', 'cherries']])
```

Output:



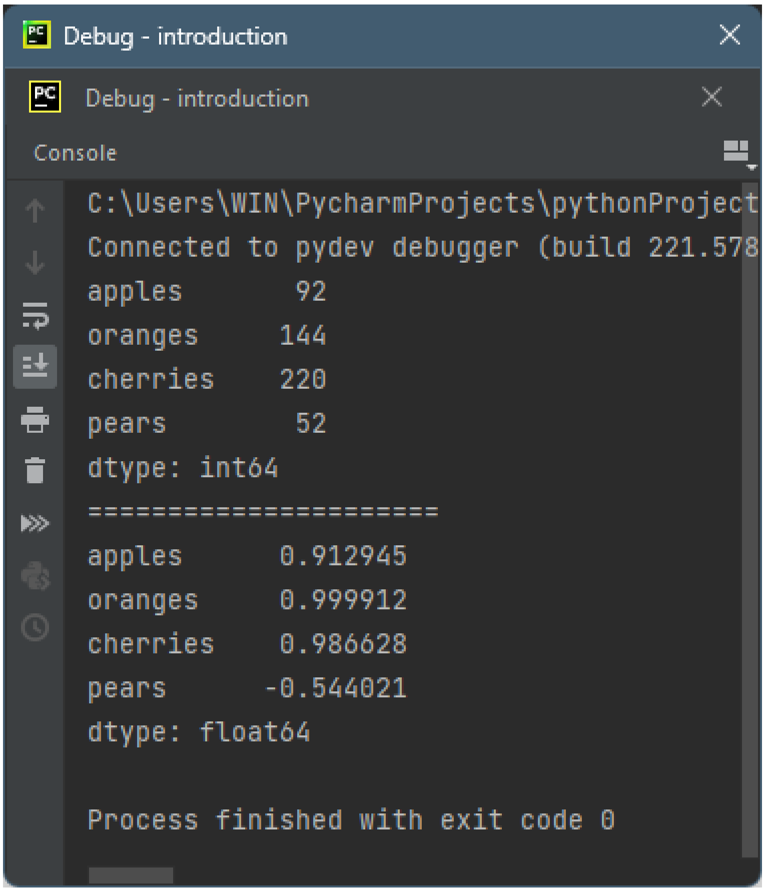
```
Connected to pydev debugger (build 221.
apples        20
oranges       33
cherries      52
dtype: int64

Process finished with exit code 0
```

Similar to Numpy we can use scalar operations or mathematical functions on a series:

Source Code:

```
import numpy as np
import pandas as pd
fruits = ['apples', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
print((S + 3) * 4)
print("======================")
print(np.sin(S))
```

Output:



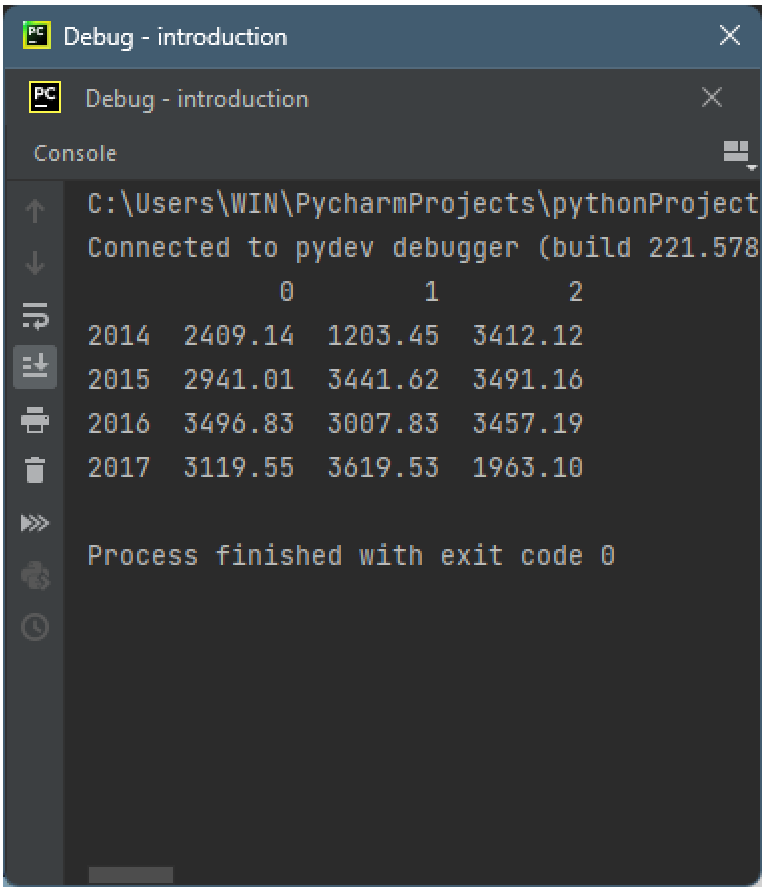
```
C:\Users\WIN\PycharmProjects\pythonProject
Connected to pydev debugger (build 221.578
apples       92
oranges     144
cherries    220
pears        52
dtype: int64
======================
apples      0.912945
oranges     0.999912
cherries    0.986628
pears      -0.544021
dtype: float64

Process finished with exit code 0
```

10)DataFrame with help of series

Source Code:

```
import pandas as pd
years = range(2014, 2018)
shop1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index=years)
shop2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index=years)
shop3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index=years)
shops_df = pd.concat([shop1, shop2, shop3], axis=1)
print(shops_df)
```

Output:



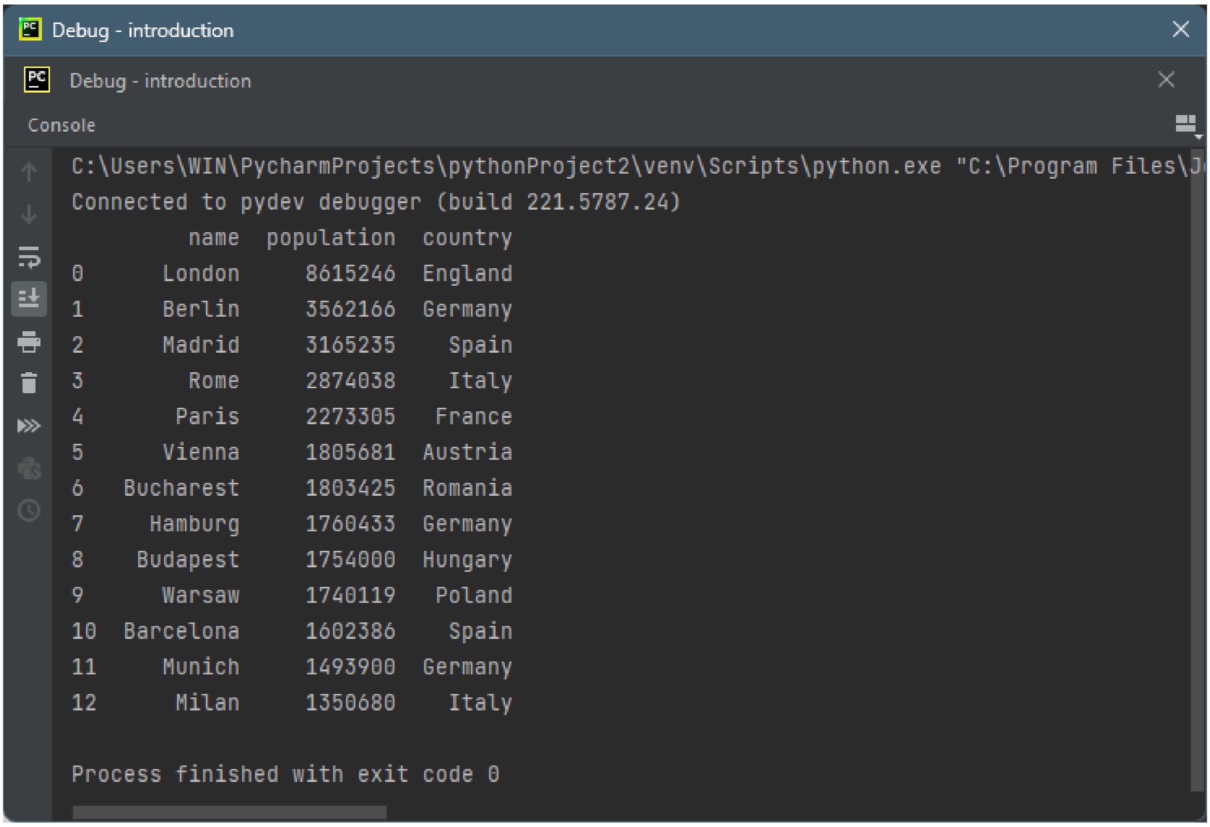
```
C:\Users\WIN\PycharmProjects\pythonProject
Connected to pydev debugger (build 221.578
            0        1        2
2014  2409.14  1203.45  3412.12
2015  2941.01  3441.62  3491.16
2016  3496.83  3007.83  3457.19
2017  3119.55  3619.53  1963.10

Process finished with exit code 0
```

11) DataFrames from Dictionaries and how to read csv file.

Source Code:

```
import pandas as pd
cities = {"name": ["London", "Berlin", "Madrid", "Rome",
"Paris", "Vienna", "Bucharest", "Hamburg",
"Budapest", "Warsaw", "Barcelona",
"Munich", "Milan"],
"population": [8615246, 3562166, 3165235, 2874038,
2273305, 1805681, 1803425, 1760433,
1754000, 1740119, 1602386, 1493900,
1350680],
"country": ["England", "Germany", "Spain", "Italy",
"France", "Austria", "Romania",
"Germany", "Hungary", "Poland", "Spain",
"Germany", "Italy"]}
city_frame = pd.DataFrame(cities)
print(city_frame)
```

Output:



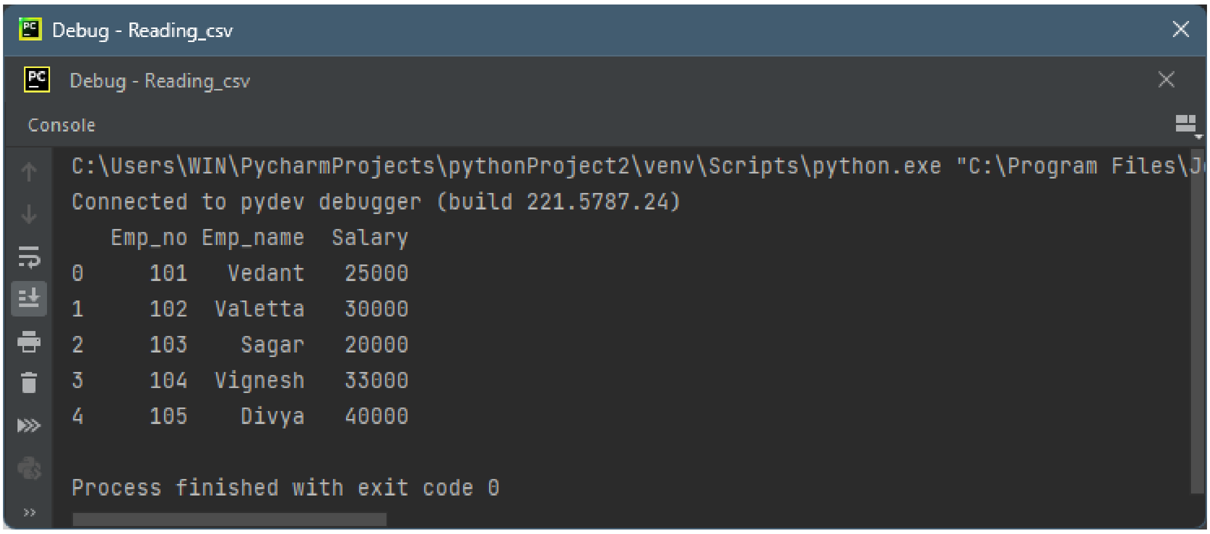
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\J
Connected to pydev debugger (build 221.5787.24)
         name  population  country
0      London     8615246  England
1      Berlin     3562166  Germany
2      Madrid     3165235    Spain
3        Rome     2874038    Italy
4       Paris     2273305   France
5      Vienna     1805681  Austria
6   Bucharest     1803425  Romania
7     Hamburg     1760433  Germany
8    Budapest     1754000  Hungary
9      Warsaw     1740119   Poland
10  Barcelona     1602386    Spain
11     Munich     1493900  Germany
12      Milan     1350680    Italy

Process finished with exit code 0
```

Reading a CSV file

Source Code:

```
import pandas as pd
df = pd.read_csv('D:\C21004\emp_csv.csv')
print(df.to_string())
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\J
Connected to pydev debugger (build 221.5787.24)
   Emp_no Emp_name  Salary
0     101   Vedant   25000
1     102  Valetta   30000
2     103    Sagar   20000
3     104  Vignesh   33000
4     105    Divya   40000

Process finished with exit code 0
```

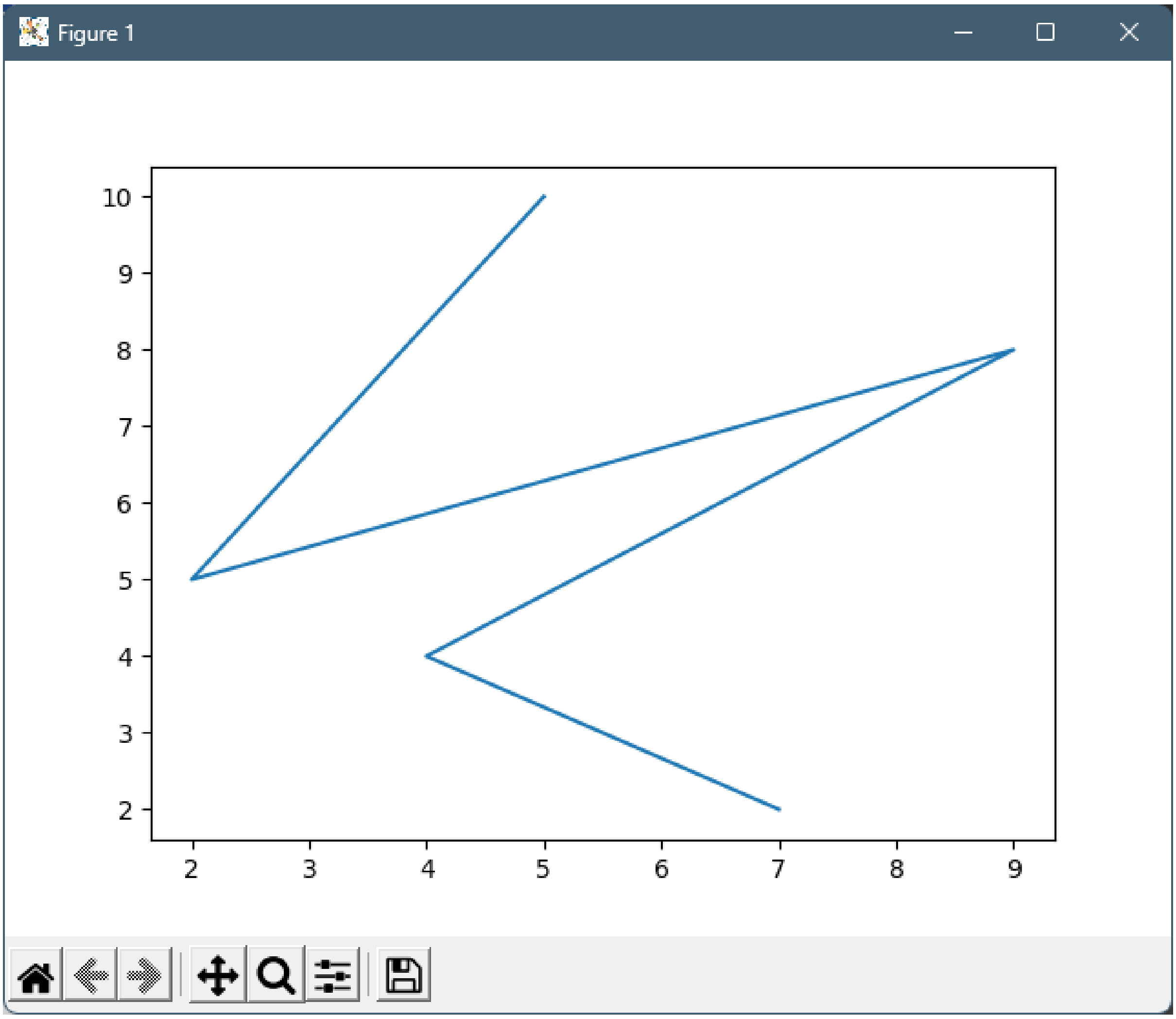
10)Use of Matplotlib

Line plot

Source Code:

```
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot
plt.plot(x, y)
# function to show the plot
plt.show()
```

Output:

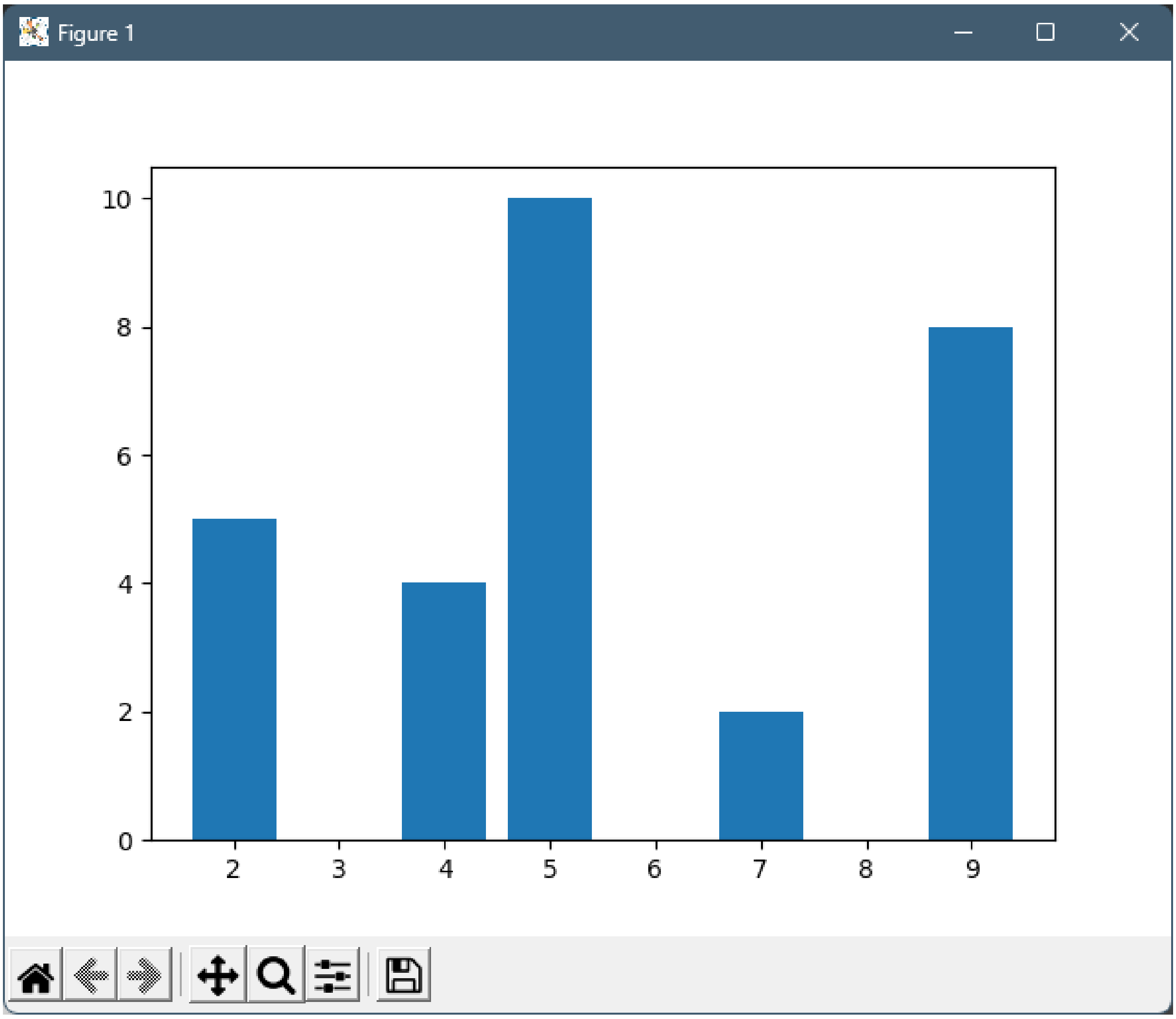


Bar plot

Source Code:

```
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot the bar
plt.bar(x, y)
# function to show the plot
plt.show()
```

Output:

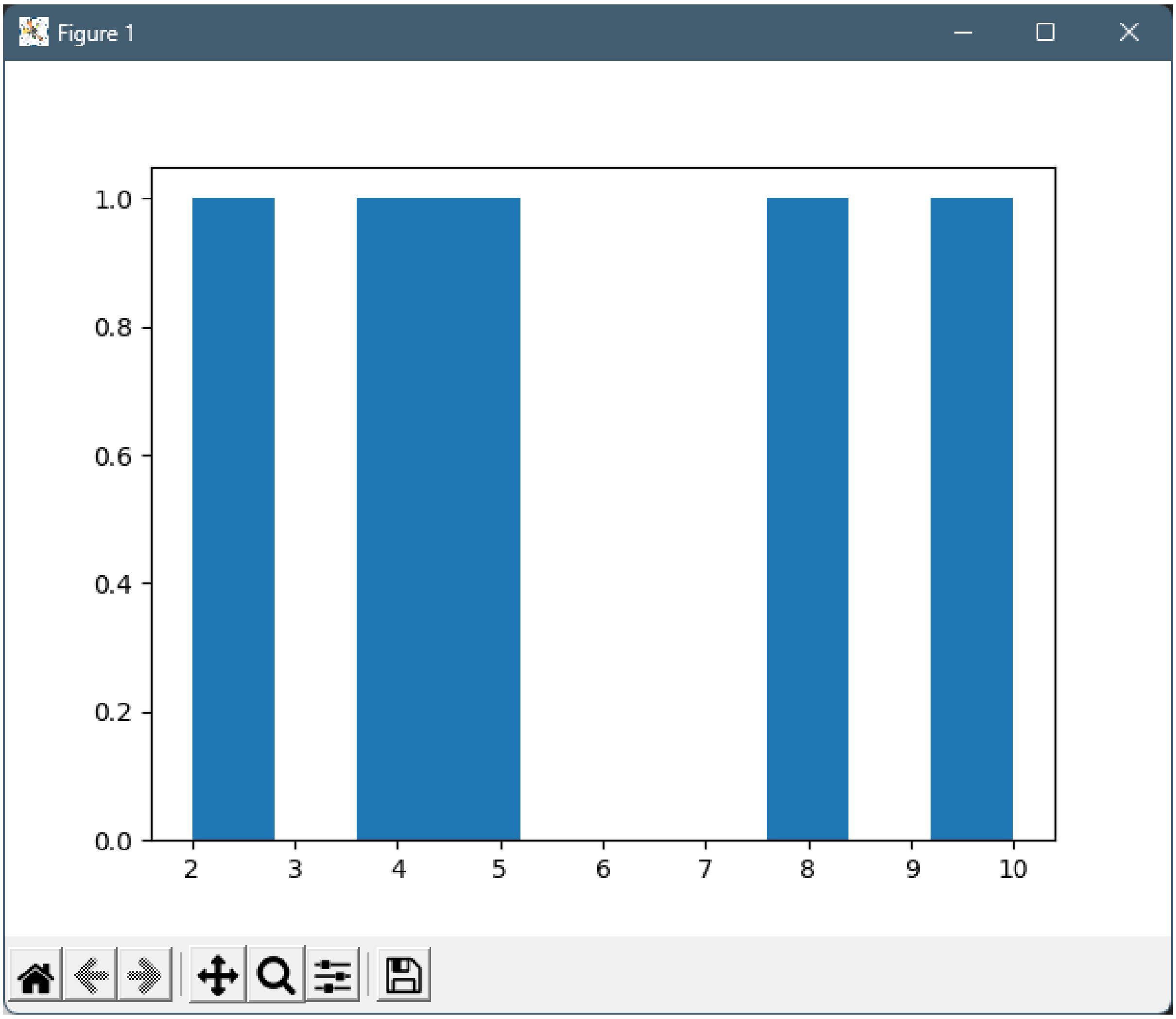


Histogram Plot

Source Code:

```
from matplotlib import pyplot as plt
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot histogram
plt.hist(y)
# Function to show the plot
plt.show()
```

Output:

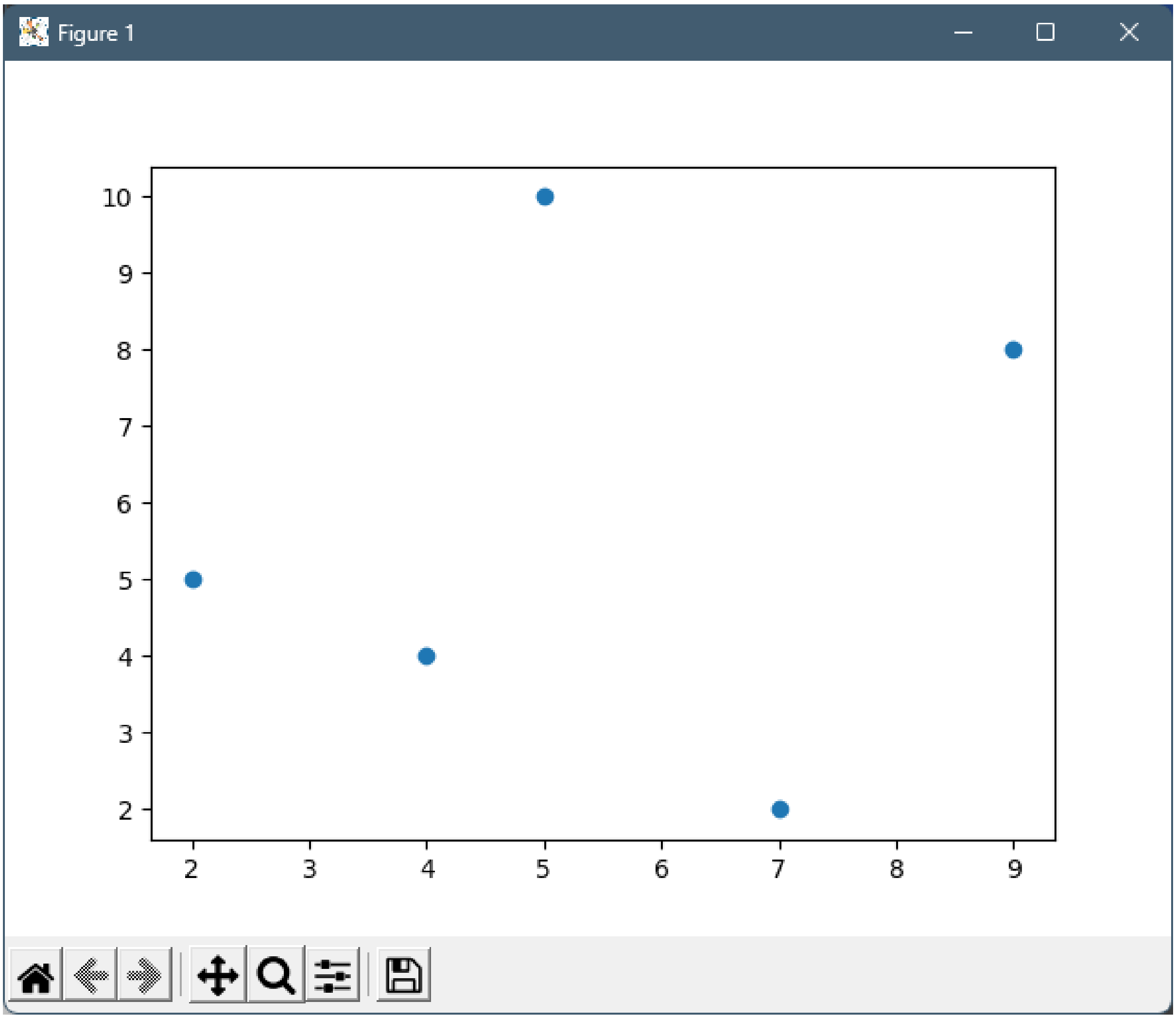


Scatter Plot

Source Code:

```
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot scatter
plt.scatter(x, y)
# function to show the plot
plt.show()
```

Output:

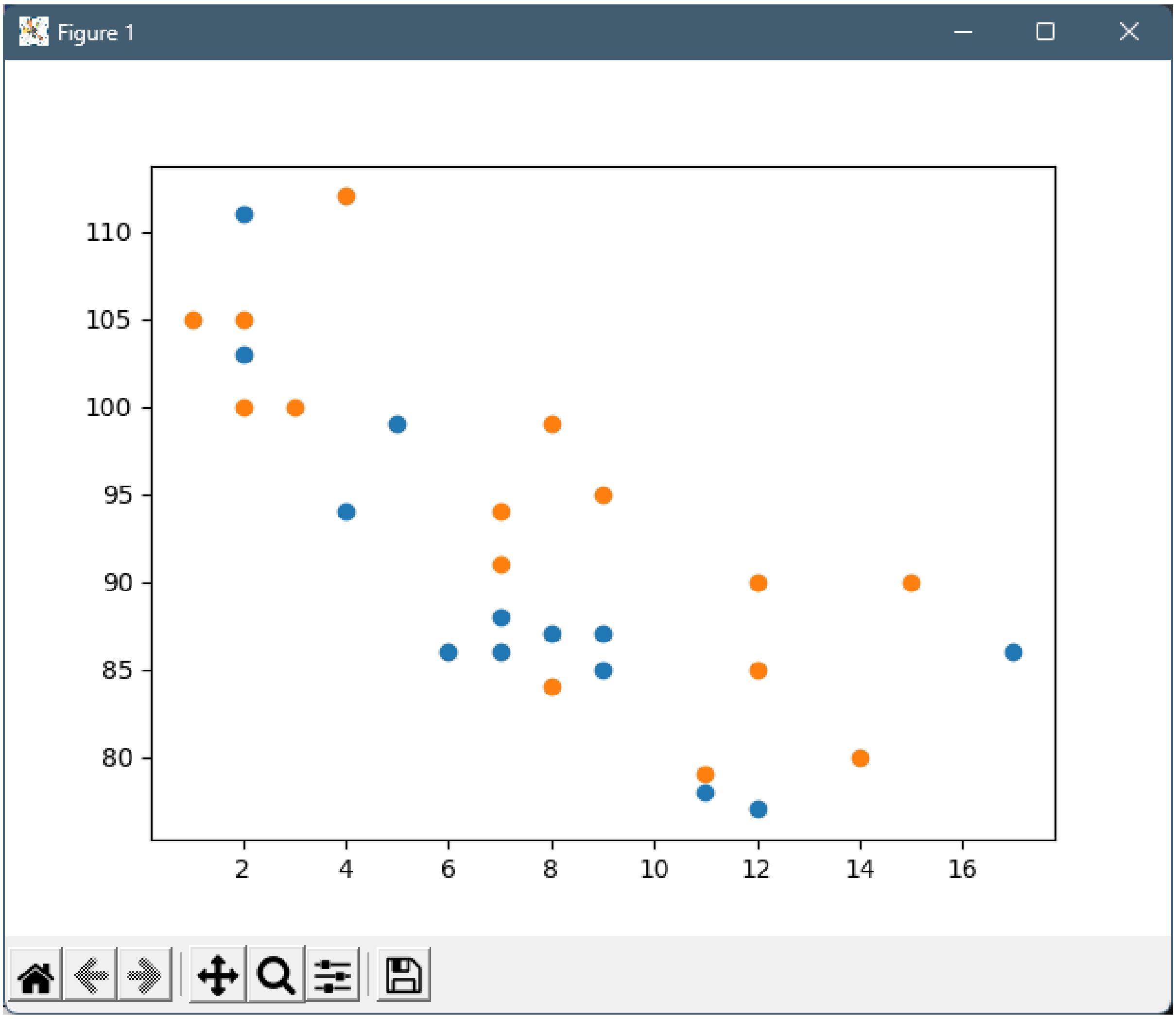


Compare Two scatter plots

Source Code:

```
import matplotlib.pyplot as plt
import numpy as np
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)
plt.show()
```

Output:

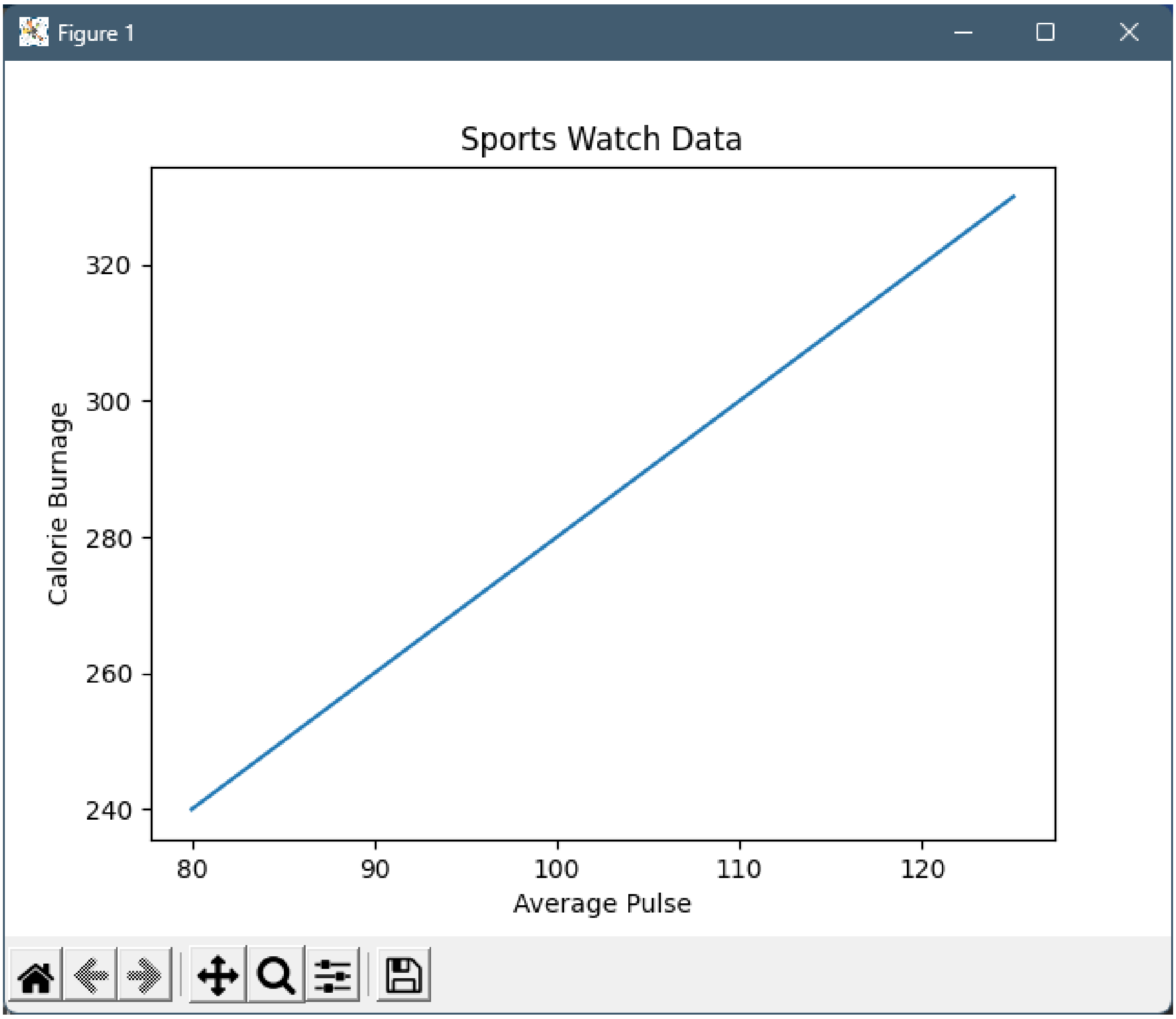


Create Labels and Title for a Plot

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.title("Sports Watch Data")
plt.show()
```

Output:

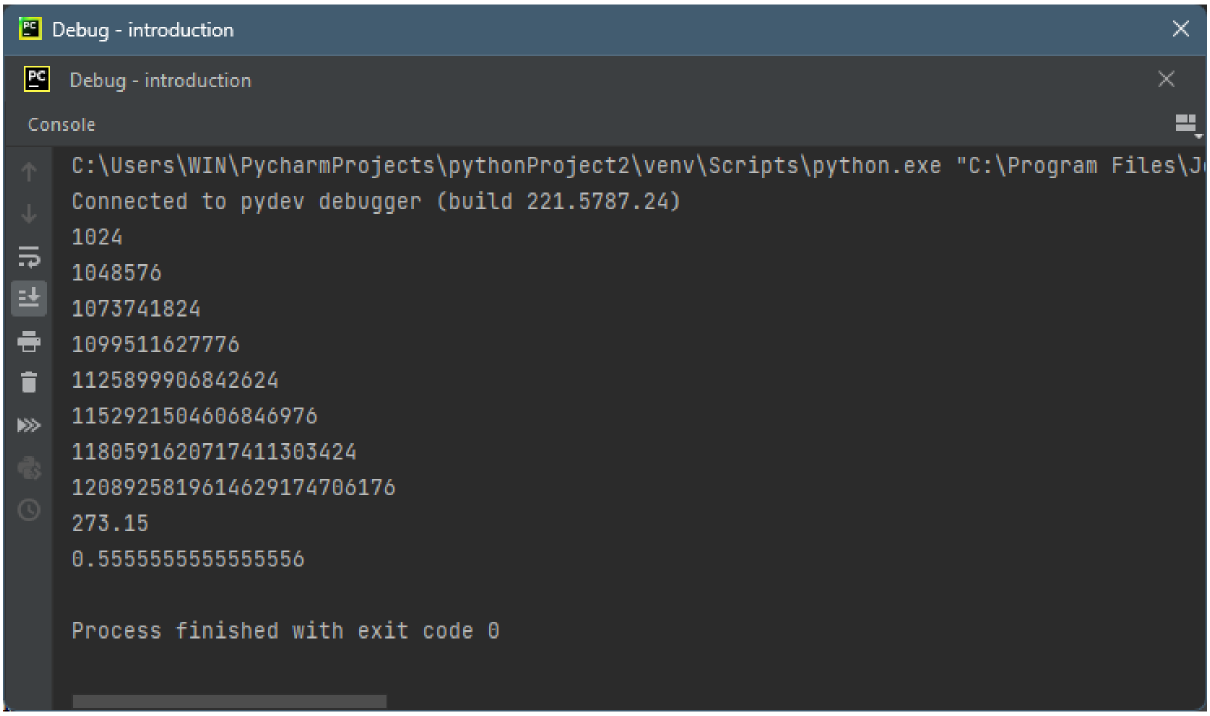


12) Give an example of a scientific constant.

Source Code:

```
from scipy import constants
print(constants.kibi) #1024
print(constants.mebi) #1048576
print(constants.gibi) #1073741824
print(constants.tebi) #1099511627776
print(constants.pebi) #1125899906842624
print(constants.exbi) #1152921504606846976
print(constants.zebi) #1180591620717411303424
print(constants.yobi) #1208925819614629174706176
print(constants.zero_Celsius) #273.15
print(constants.degree_Fahrenheit) #0.5555555555555556
```

Output:



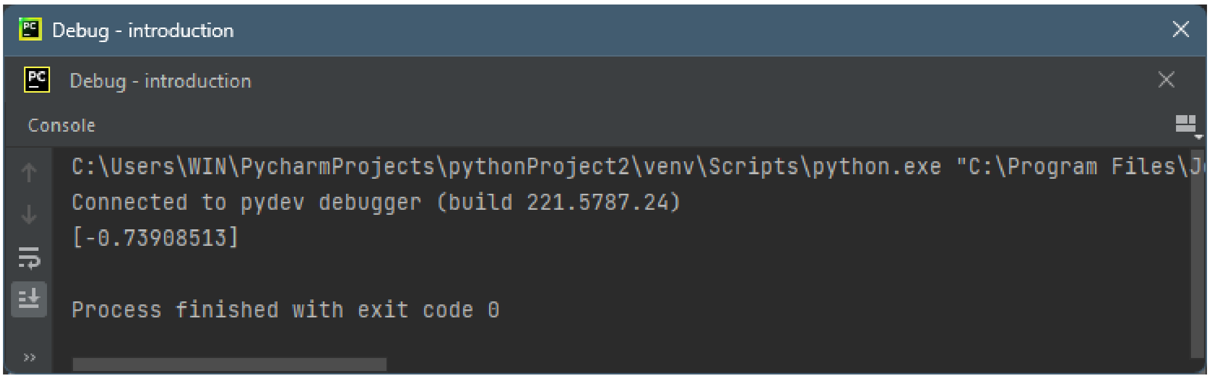
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\J
Connected to pydev debugger (build 221.5787.24)
1024
1048576
1073741824
1099511627776
1125899906842624
1152921504606846976
1180591620717411303424
1208925819614629174706176
273.15
0.5555555555555556

Process finished with exit code 0
```

13) Give an example of a SciPy Optimizers.

Source Code:

```
from scipy.optimize import root
from math import cos
def eqn(x):
   return x + cos(x)
myroot = root(eqn, 0)
print(myroot.x)
```

Output:



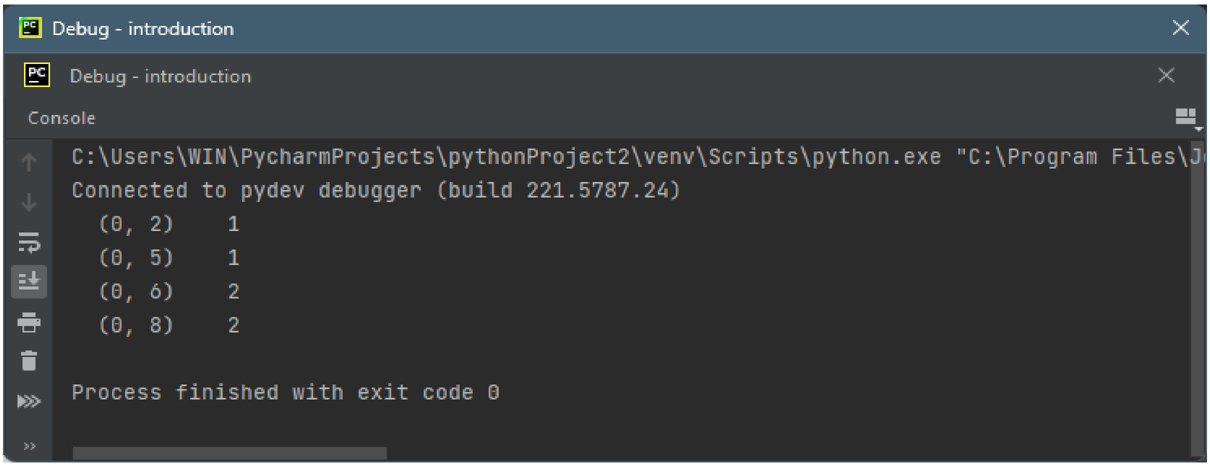
```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\J
Connected to pydev debugger (build 221.5787.24)
[-0.73908513]

Process finished with exit code 0
```

14) Give an example of a SciPy Sparse Data.

Source Code:

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([0, 0, 1, 0, 0, 1, 2, 0, 2])
print(csr_matrix(arr))
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\J
Connected to pydev debugger (build 221.5787.24)
  (0, 2)	1
  (0, 5)	1
  (0, 6)	2
  (0, 8)	2

Process finished with exit code 0
```

15)Scikit-learn example using datasets iris.

A collection of data is called dataset. It is having the following two components –
Features –
The variables of data are called its features. They are also known as predictors, inputs or attributes.
 Feature matrix – It is the collection of features, in case there are more than one.

 Feature Names – It is the list of all the names of the features.

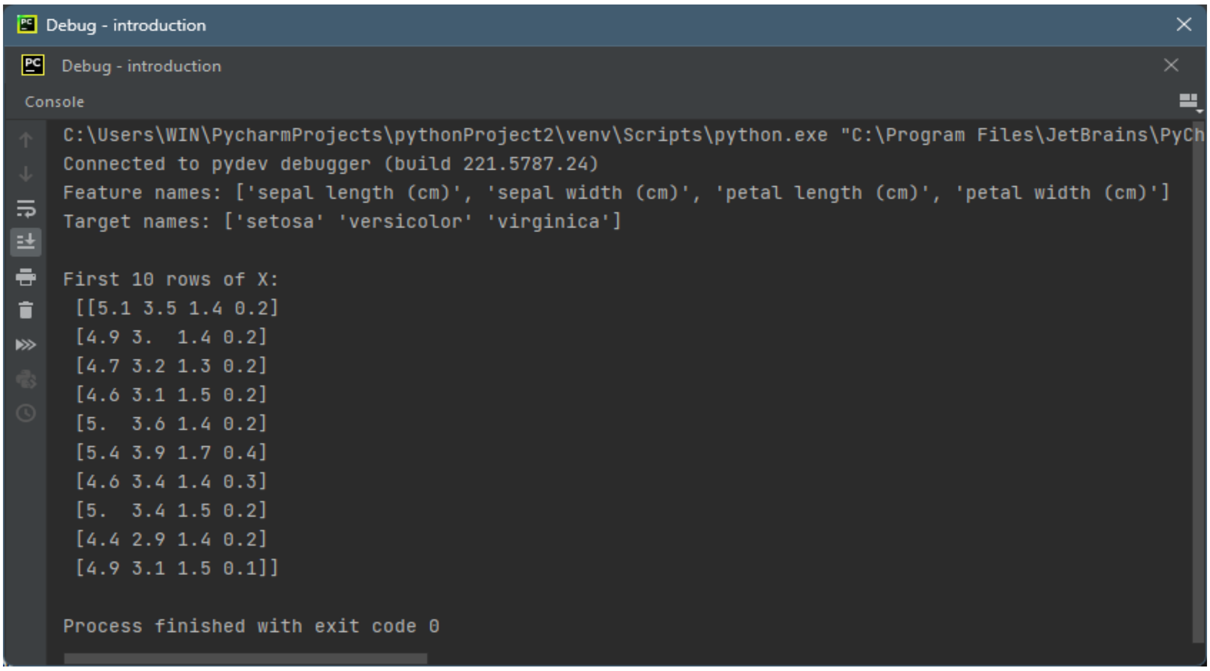
Response –
It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.
 Response Vector – It is used to represent response column. Generally, we have just one response column.

 Target Names – It represent the possible values taken by a response vector.

Source Code:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of X:\n", X[:10])
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCh
Connected to pydev debugger (build 221.5787.24)
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of X:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]

Process finished with exit code 0
```

Some of the most popular groups of models provided by Sklearn are as follows –

Supervised Learning algorithms − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

Unsupervised Learning algorithms − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

Clustering − this model is used for grouping unlabeled data.
Cross Validation − It is used to check the accuracy of supervised models on unseen data.

Dimensionality Reduction − It is used for reducing the number of attributes in data which can be further used for summarization, visualization and feature selection.

Ensemble methods − as name suggest, it is used for combining the predictions of multiple supervised models.

Feature extraction − It is used to extract the features from data to define the attributes in image and text data.

Practical 4: Supervised Learning:

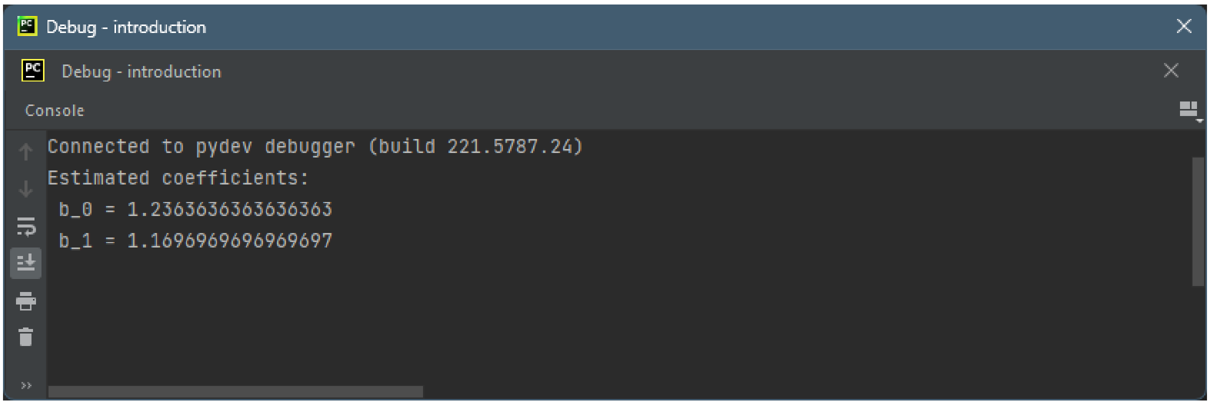
Implementation of Linear Regression, Logistic regression, KNN- classification

Linear regression:

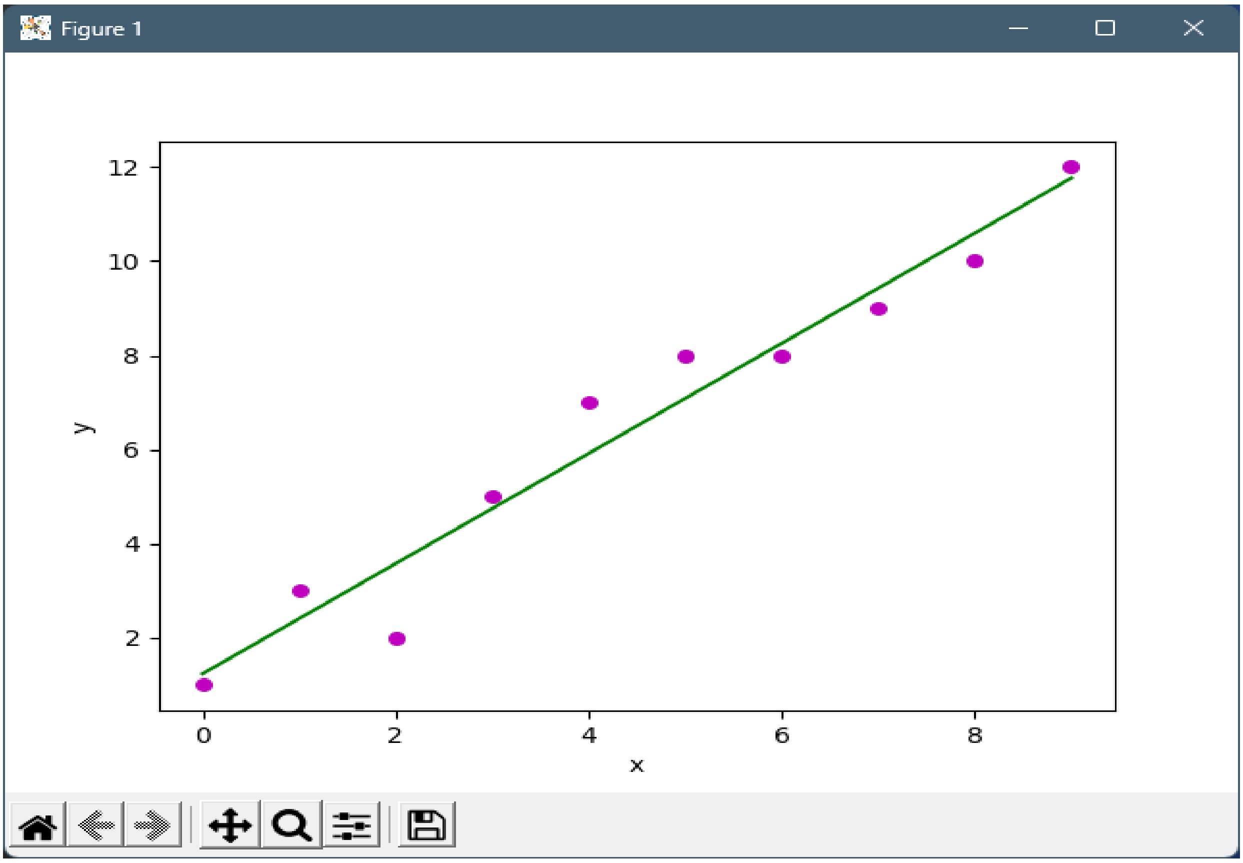
Source code:

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

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    ss_xy = np.sum(y*x) -n*m_y*m_x
    ss_xx = np.sum(x*x) -n*m_x*m_x

    # calculating regression coefficients
    b_1 = ss_xy / ss_xx
    b_0 = m_y-b_1*m_x
    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",marker = "o", s = 30)
    # predicted response vector
    y_pred = b[0] + b[1]*x
    # plotting the regression line
    plt.plot(x, y_pred, color = "g")
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
    # estimating coefficient
    b = estimate_coef(x, y)
    print("Estimated coefficients:\n b_0 = {} \n b_1 = {}".format(b[0], b[1]))
    # plotting regression line
    plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()
```

Output:



```
Connected to pydev debugger (build 221.5787.24)
Estimated coefficients:
 b_0 = 1.2363636363636363
 b_1 = 1.1696969696969697
```

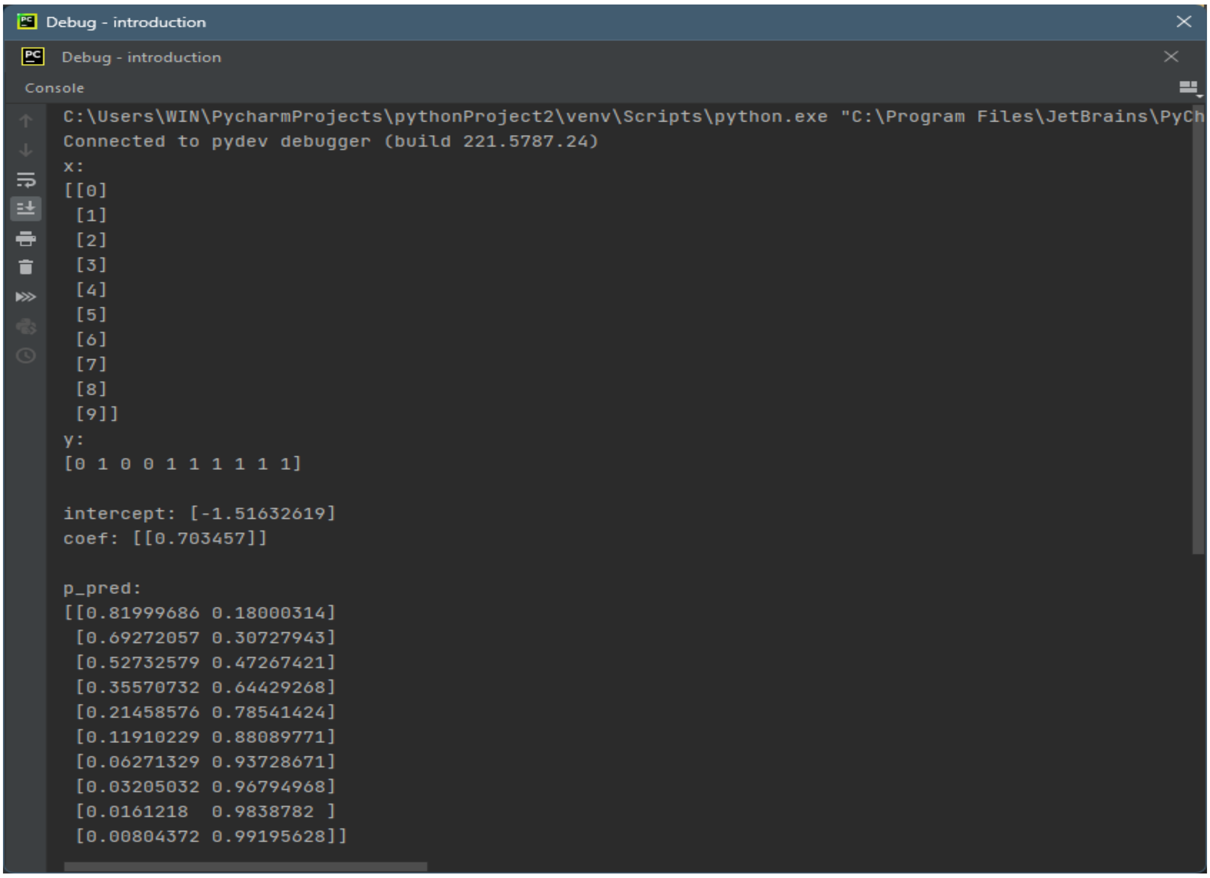


Logistic regression:

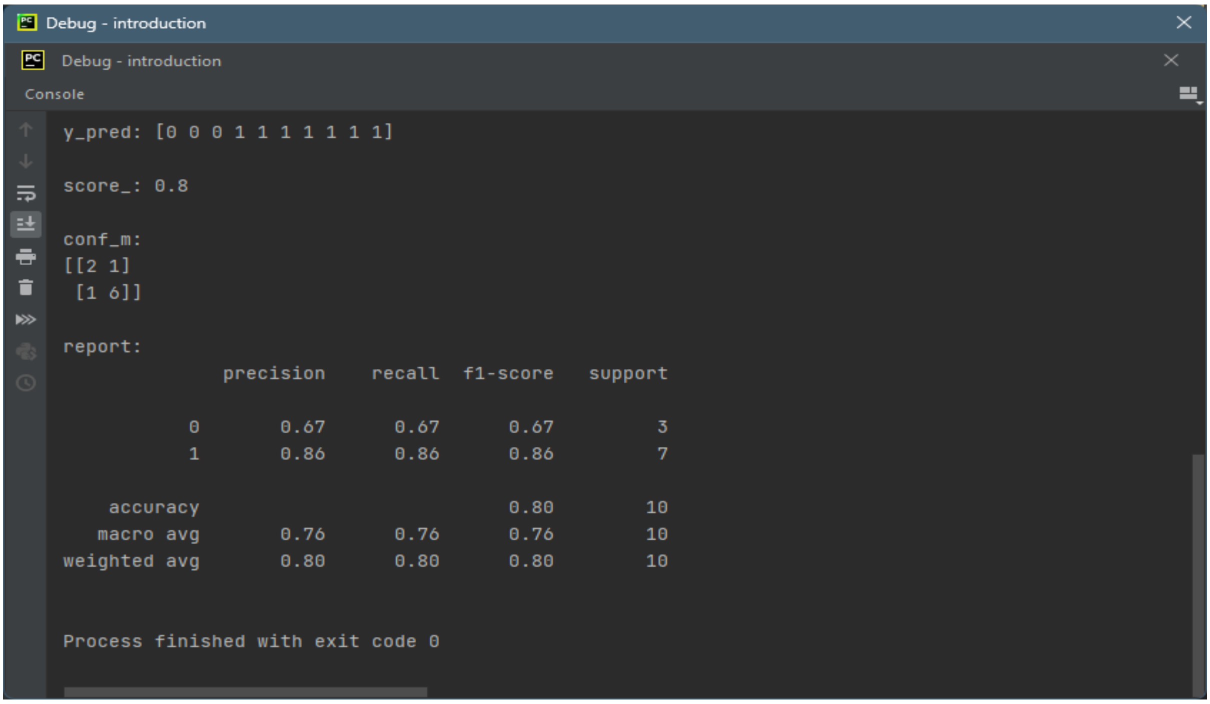
Source Code:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Step 2: Get data
x = np.arange(10).reshape(-1, 1)

y = np.array([0, 1, 0, 0, 1, 1, 1, 1, 1, 1])

# Step 3: Create a model and train it

model = LogisticRegression(solver='liblinear', C=10.0, random_state=0)

model.fit(x, y)

# Step 4: Evaluate the model
p_pred = model.predict_proba(x)
y_pred = model.predict(x)
score_ = model.score(x, y)
conf_m = confusion_matrix(y, y_pred)
report = classification_report(y, y_pred)

print('x:', x, sep='\n')

print('y:', y, sep='\n', end='\n\n')
print('intercept:', model.intercept_)
print('coef:', model.coef_, end='\n\n')
print('p_pred:', p_pred, sep='\n', end='\n\n')
print('y_pred:', y_pred, end='\n\n')
print('score_:', score_, end='\n\n')
print('conf_m:', conf_m, sep='\n', end='\n\n')
print('report:', report, sep='\n')
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCh
Connected to pydev debugger (build 221.5787.24)
x:
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
y:
[0 1 0 0 1 1 1 1 1 1]

intercept: [-1.51632619]
coef: [[0.703457]]

p_pred:
[[0.81999686 0.18000314]
 [0.69272057 0.30727943]
 [0.52732579 0.47267421]
 [0.35570732 0.64429268]
 [0.21458576 0.78541424]
 [0.11910229 0.88089771]
 [0.06271329 0.93728671]
 [0.03205032 0.96794968]
 [0.0161218  0.9838782 ]
 [0.00804372 0.99195628]]
```



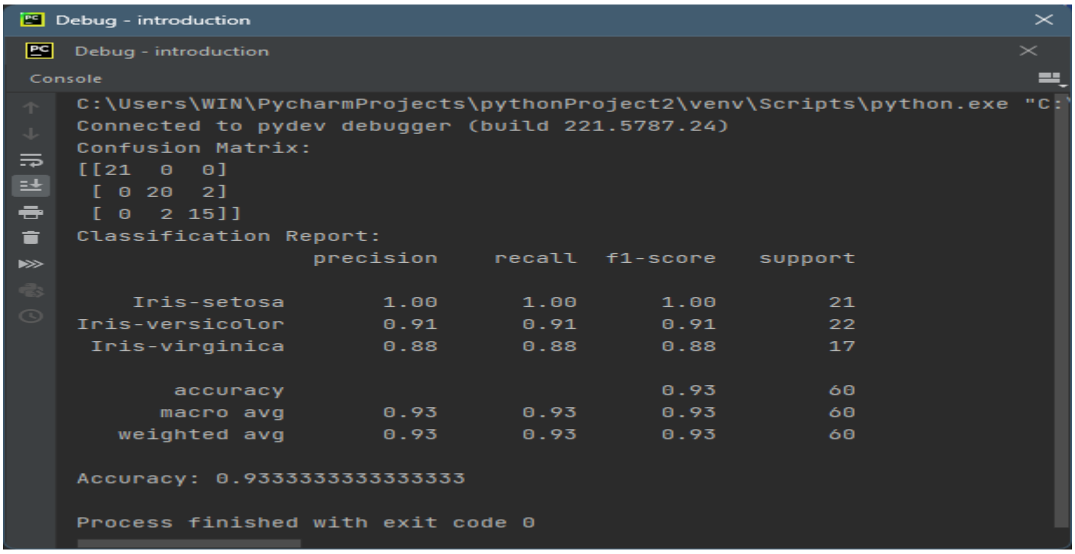
```
y_pred: [0 0 0 1 1 1 1 1 1 1]

score_: 0.8

conf_m:
[[2 1]
 [1 6]]

report:
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         3
           1       0.86      0.86      0.86         7

    accuracy                           0.80        10
   macro avg       0.76      0.76      0.76        10
weighted avg       0.80      0.80      0.80        10


Process finished with exit code 0
```

KNN- classification

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(path, names = headernames)
dataset.head()
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 8)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:
Connected to pydev debugger (build 221.5787.24)
Confusion Matrix:
[[21  0  0]
 [ 0 20  2]
 [ 0  2 15]]
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        21
Iris-versicolor       0.91      0.91      0.91        22
 Iris-virginica       0.88      0.88      0.88        17

       accuracy                           0.93        60
      macro avg       0.93      0.93      0.93        60
   weighted avg       0.93      0.93      0.93        60

Accuracy: 0.9333333333333333

Process finished with exit code 0
```

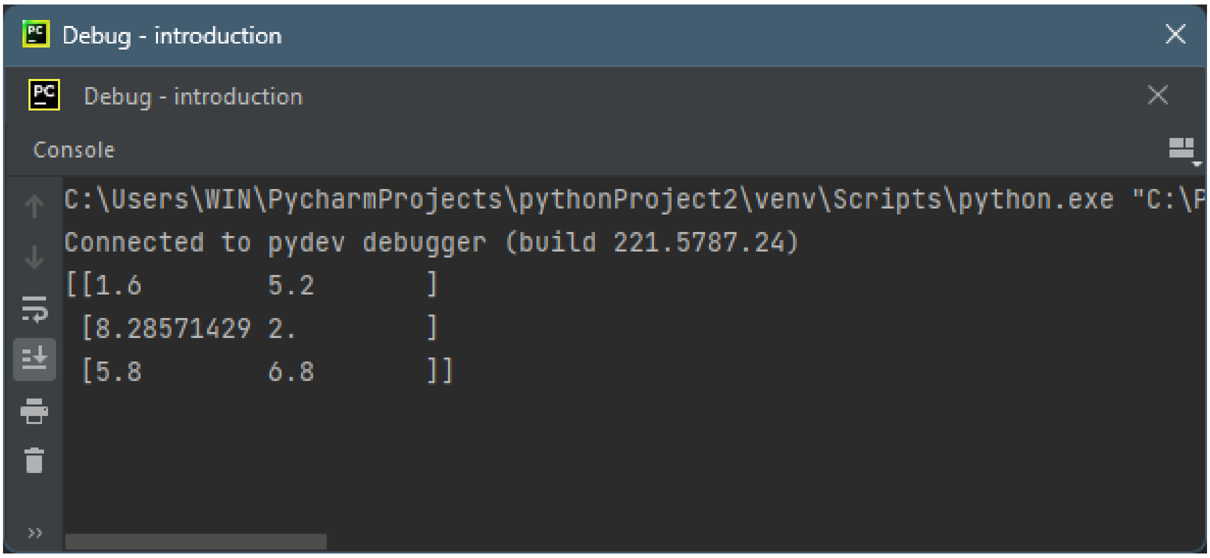
Practical 5: Unsupervised Learning:

Implementation of K-Means clustering algorithm.

Source Code:

```
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'c']
markers = ['o', 'v', 's']
# KMeans algorithm
K = 3
kmeans_model = KMeans(n_clusters=K).fit(X)
print(kmeans_model.cluster_centers_)
centers = np.array(kmeans_model.cluster_centers_)
plt.plot()
plt.title('k means centroids')
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
    plt.xlim([0, 10])
    plt.ylim([0, 10])
plt.scatter(centers[:,0], centers[:,1], marker="x", color='r')
plt.show()
```

Output:



```
C:\Users\WIN\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\P
Connected to pydev debugger (build 221.5787.24)
[[1.6        5.2       ]
 [8.28571429 2.        ]
 [5.8        6.8       ]]
```

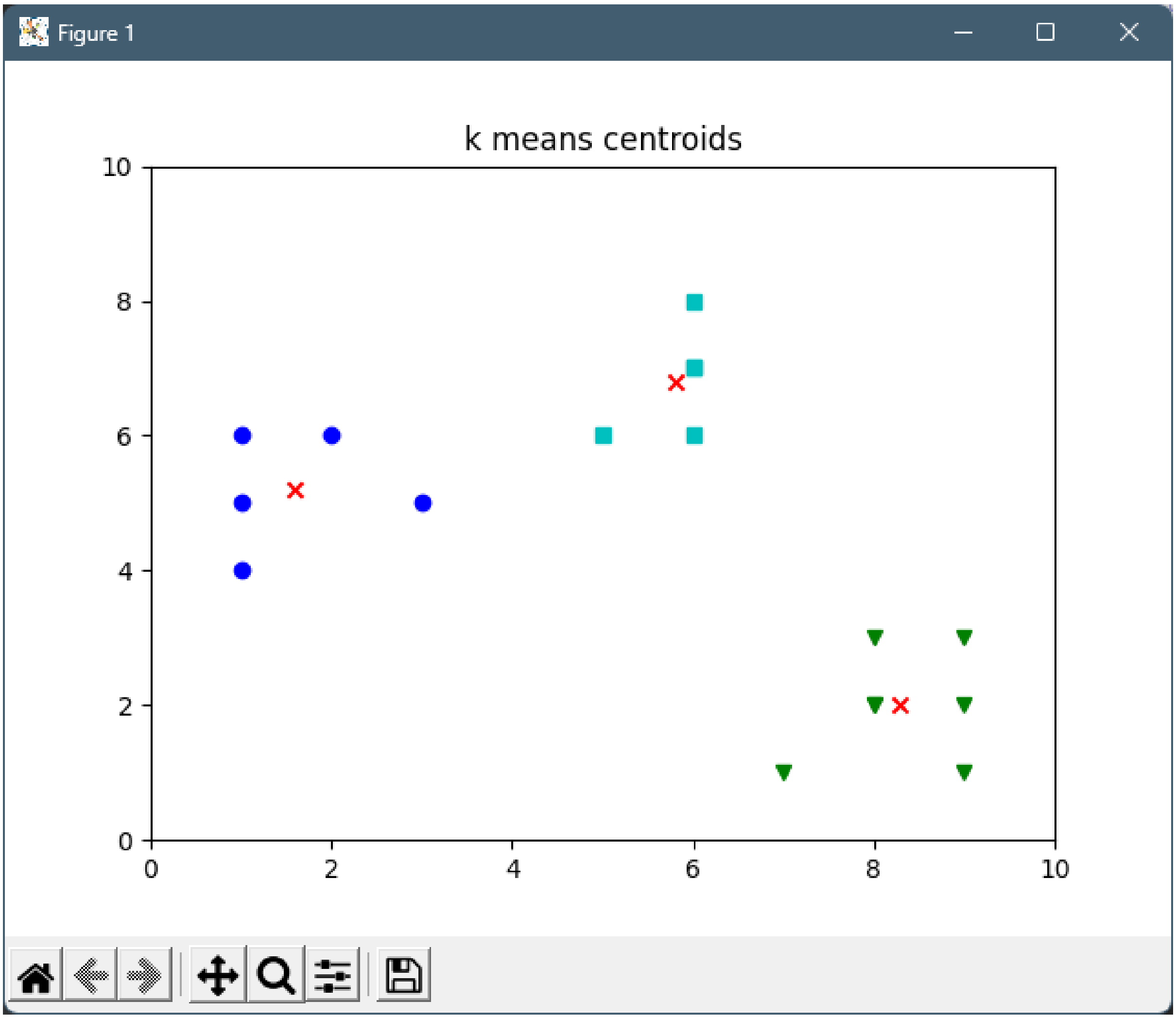


Figure 1
k means centroids
10
8
6
4
2
0
0
2
4
6
8
10

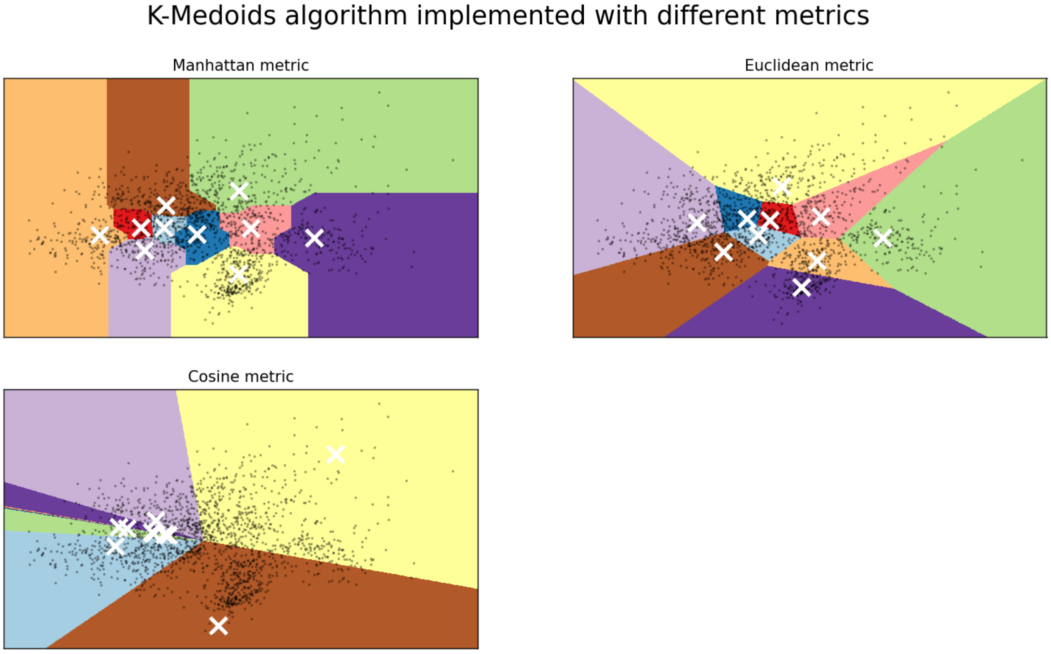
Practical 6: Unsupervised Learning.

Implementation of K-medoid clustering algorithm.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

dataset = load_digits()
# Standardize the data
digit_data = scale(dataset.data)
num_digits = len(np.unique(dataset.target))
red_data = PCA(n_components=2).fit_transform(digit_data)
h = 0.02  # step size of the mesh
# Minimum and maximum x-coordinates
xmin, xmax = red_data[:, 0].min() - 1, red_data[:, 0].max() + 1
# Minimum and maximum y-coordinates
ymin, ymax = red_data[:, 1].min() - 1, red_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(xmin, xmax, h), np.arange(ymin, ymax, h))
models = [
   (
      KMedoids(metric="manhattan", n_clusters=num_digits,
            init="heuristic", max_iter=2), "Manhattan metric",
   ),
   (
      KMedoids(metric="euclidean", n_clusters=num_digits,
            init="heuristic", max_iter=2), "Euclidean metric",
   ),
   (KMedoids(metric="cosine", n_clusters=num_digits, init="heuristic",
            max_iter=2), "Cosine metric",),
]
# number of rows = integer(ceiling(number of model variants/2))
num_rows = int(np.ceil(len(models) / 2.0))
# number of columns
num_cols = 2
#Clear the current figure first (if any)
plt.clf()
#Initialize dimensions of the plot
plt.figure(figsize=(15,10))
for i, (model, description) in enumerate(models):
    # Fit each point in the mesh to the model
    model.fit(red_data)
   #Predict the labels for points in the mesh
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    # Put the result  into a color plot
    Z = Z.reshape(xx.shape)
  #Subplot for the ith model variant
    plt.subplot(num_cols, num_rows, i + 1)
  #Display the subplot
    plt.imshow(
        Z,    #data to be plotted
        interpolation="nearest",
   #bounding box coordinates (left,right,bottom,top)
        extent=(xx.min(), xx.max(), yy.min(), yy.max()),
        cmap=plt.cm.Paired,  #colormap
        aspect="auto", #aspect ratio of the axes
        origin="lower",  #set origin as lower left corner of the axes
    )
    plt.plot(
        red_data[:, 0], red_data[:, 1], "k.", markersize=2, alpha=0.3
    )
    # Plot the centroids as white cross marks
    centroids = model.cluster_centers_
    plt.scatter(
        centroids[:, 0],
        centroids[:, 1],
        marker="x",
        s=169,  #marker’s size (points^2)
        linewidths=3, #width of boundary lines
        color="w",  #white color for centroids markings
        zorder=10,  #drawing order of axes
    )
    #describing text of the tuple will be title of the subplot
    plt.title(description)
    plt.xlim(xmin, xmax)  #limits of x-coordinates
    plt.ylim(ymin, ymax)  #limits of y-coordinates
    plt.xticks(())
    plt.yticks(())
 #Upper title of the whole plot
plt.suptitle(
   #Text to be displayed
    "K-Medoids algorithm implemented with different metrics\n\n",
    fontsize=20,  #size of the fonts
 )
plt.show()
```

Output:


K-Medoids algorithm implemented with different metrics
Manhattan metric
Euclidean metric
Cosine metric