Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Unit-04.1

Modules

Prof. Jayesh D. Vagadiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
Jayesh.vagadiya@darshan.ac.in
9537133260

Outline

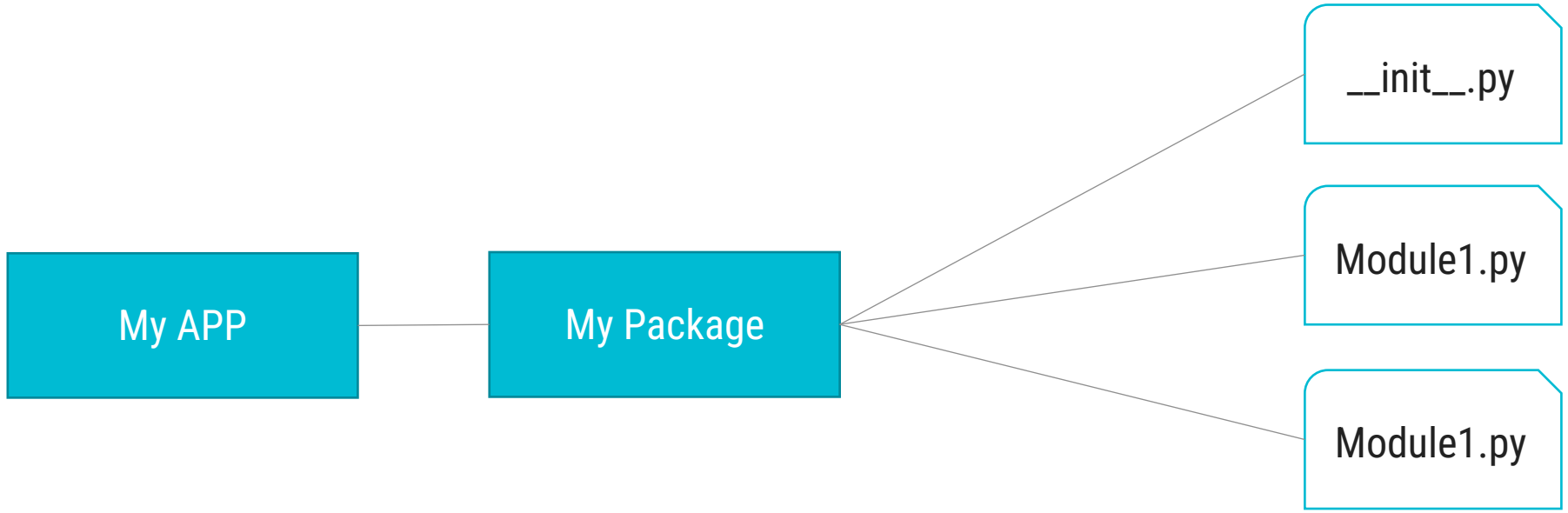
- ✓ Importing a module
- ✓ Math module
- ✓ Random module
- ✓ Datetime module
- ✓ Creating a custom module.
- ✓ Examples

Introduction

- Modules provide a means of collecting sets of functions together so that they can be used by any number of programs.
- A Python module, simply put, is a .py file.
- A module can contain any Python code we like. All the programs we have written so far have been contained in a single .py file, and so they are modules as well as programs.
- The key difference is that programs are designed to be run, whereas modules are designed to be imported and used by programs.
- Not all modules have associated .py files—for example, the sys module is built into Python, and some modules are written in other languages (most commonly, C).
- It makes no difference to our programs what language a module is written in, since all modules are imported and used in the same way.

packages

- A package can contain one or more relevant modules.
- A package is basically a directory with Python files and a file with the name __init__.py.
- This means that every directory inside of the Python path, which contains a file named __init__.py, will be treated as a package by Python.



Importing a module

- import statement is used for importing module and packages.

syntax

```
import importable
import importable1, importable2, ..., importableN
import importable as preferred_name
```

- Here importable is module, package or a module in a package.
- The first syntax is used to import one module at a time and second syntax is used to import multiple module at same time.
- The third syntax allows us to give a name of our choice to the package or module.
- Theoretically this could lead to name clashes, but in practice the as syntax is used to avoid them.
- Renaming is particularly useful when experimenting with different implementations of a module.

Darshan UNIVERSITY
योगः कर्मसु कौशलम्

Importing a module (cont.)

- For example, if we had two modules MyModuleA and MyModuleB that had the same API (Application Programming Interface), we could write import MyModuleA as MyModule in a program, and later on seamlessly switch to using import MyModuleB as MyModule.
- It is common practice to put all the import statements at the beginning of .py files

Example.py

```
1 # import random
2 import random
3
4 list1 = [10,232,23,45,34]
5 print(random.choice(list1))
```

OUTPUT

```
45
```

Random Module

- Python has a built-in module that provide set of function by which we can generate random numbers.
- These are pseudo-random numbers means these are not truly random.

syntax

```
import random
```

- **random()**
  - It is used to generate random numbers in Python.
  - It generate random float number.
  - It generate pseudo-random numbers. That means these randomly generated numbers can be determined.

Example.py

```
1  # import random
2  import random
3
4  print("Random 1=",random.random())
5  print("Random 2=",random.random())
```

OUTPUT

```
Random 1= 0.14738964825362189
Random 2= 0.48730431196440027
```

Darshan UNIVERSITY

Random Module (cont.)

- **seed()**
  - random() function generates numbers for some values. This value is also called seed value.
  - Seed function is used to save the state of a random function, so that it can generate same random numbers on multiple executions of the code on the same machine.

Example.py

```
1  # import random
2  import random
3
4  random.seed(3)
5  print("Random 1=",random.random())
6  print("Random 2=",random.random())
```

OUTPUT

```
Random 1= 0.2379646270918913
Random 2= 0.5442292252959519
```

Random Module (cont.)

randint()

- This method return a random integer number between two range.

Example.py

```
1  # import random
2  import random
3
4  print("Random 1=",random.randint(1, 100))
5  print("Random 2=",random.randint(1, 100))
```

OUTPUT

```
Random 1= 80
Random 2= 39
```

randrange()

- This method return a random integer number from the range created by the start, stop and step arguments.
- The value of start is 0 by default. Similarly, the value of step is 1 by default.

Example.py

```
1  # import random
2  import random
3
4  #random.ranage(start,stop,step)
5  print("Random 1=",random.randrange(0, 100,10))
6  print("Random 2=",random.randrange(1, 100,2))
```

OUTPUT

```
Random 1= 80
Random 2= 39
```

Darshan UNIVERSITY

Random Module (cont.)

- **choice():**
  - Method returns a randomly selected element from a non-empty sequence.

Example.py

```
1  # import random
2  import random
3
4  list1 = [10,232,23,45,34]
5  t1 = (10,232,23,45,34)
6  print(random.choice(list1))
7  print(random.choice(t1))
```

OUTPUT

```
34
23
```

- **shuffle():**
  - Method randomly reorders the elements in a list.

Example.py

```
1  # import random
2  import random
3
4  list1 = [10,232,23,45,34]
5  random.shuffle(list1)
6  print(list1)
```

OUTPUT

```
[45, 34, 23, 232, 10]
```

Darshan UNIVERSITY

Random Module (cont.)

- **uniform()**
  - This method generate random floating point number between two given range.

Example.py

```
1  # import random
2  import random
3
4  #random.ranage(start,stop,step)
5  print("Random 1=",random.uniform(2,9))
6  print("Random 2=",random.uniform(10,20))
```

OUTPUT

```
Random 1= 8.079151436715527
Random 2= 15.767192676676762
```

Math Module

- This module provides access to the mathematical functions.
- This module provides set of methods and constants.
- Constants provided by the math module are
  - Euler's Number
  - Pi
  - Tau
  - Infinity
  - Not a Number (NaN)

Example.py

```
1  import math
2
3  print("e = ", math.e)
4  print("PI = ", math.pi)
5  print("Tau = ", math.tau)
6  print("infinity = ", math.inf)
7  print("NaN = ", math.nan)
```

OUTPUT

```
e =  2.718281828459045
PI =  3.141592653589793
Tau =  6.283185307179586
infinity =  inf
NaN =  nan
```

Darshan UNIVERSITY

Numeric Functions

ceil() and floor()

- Ceil value means the smallest integral value greater than the number and the floor value means the greatest integral value smaller than the number.
- this can be easily calculated using the ceil() and floor() method respectively.

Example.py

```
1  import math
2
3  n = 5.6
4  print("Ceil = ",math.ceil(n))
5  print("Floor = ",math.floor(n))
```

OUTPUT

```
Ceil =  6
Floor =  5
```

factorial()

- It is used to find factorial of given number.

Example.py

```
1  import math
2
3  n = 5
4  fact = math.factorial(n)
5  print("Factorial = ",fact)
```

OUTPUT

```
Factorial =  120
```

Darshan UNIVERSITY

Numeric Functions(cont.)

gcd()

- It is used find greatest common divisor of two number passed as argument.

Example.py

```
1  import math
2  a = 15
3  b = 30
4
5  print("GCD = ",math.gcd(a,b))
```

OUTPUT

```
GCD =  15
```

fabs()

- This function return the absolute value of given number.

Example.py

```
1  import math
2
3  n = -15
4  print("absolute = ",math.fabs(n))
```

OUTPUT

```
absolute =  15.0
```

Darshan UNIVERSITY

Logarithmic and Power Functions

Function	Description	Example	Output
exp(x)	Return *e* raised to the power *x*	math.exp(1)	2.718281828459045
pow(x, y)	Return x raised to the power y (x**y)	math.pow(5,3)	125.0
log(x[, base])	With two arguments, return the logarithm of x to the given base	math.log(10,2)	3.3219280948873626
log2(x)	Return the base-2 logarithm of x	math.log2(10)	3.3219280948873626
log10(x)	Return the base-10 logarithm of *x*	math.log10(10)	1.0
sqrt(x)	Return the square root of *x*	math.sqrt(81)	9.0

Trigonometric and Angular Functions

Function	Description	Example	Output
sin(x)	Return the sine of *x* radians.	math.sin(1)	0.8414709848078965
cos(x)	Return the cosine of x radians.	math.cos(1)	0.5403023058681398
tan(x)	Return the tangent of *x* radians.	math.tan(1)	1.557407724654902
degrees()	Convert angle x from radians to degrees.	math.radians(90)	1.5707963267948966
radians()	Convert angle x from degrees to radians.	math.degrees(1.5)	85.94366926962348

Special Functions

Function	Description	Example	Output
gamma(x)	Return the gamma value of x.	math.gamma(6)	120.0
isinf()	check whether the value is infinity or not.	math.isinf(math.pi)	False
isnan()	check whether the value is NaN or not.	math.isnan(float('nan'))	True
erf(x)	Return the error function at x.	math.erf(5)	0.9999999999984626

Date Time Module

- The datetime module supplies classes for manipulating dates and times.
- These classes provide a number of functions to deal with dates, times and time intervals.
- Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

Classes	Description
date	An idealized naive date. Attributes: year, month, and day.
time	An idealized time. Attributes: hour, minute, second, microsecond, and tzinfo.
datetime	A combination of a date and a time. Attributes: year, month, day, hour, minute, second, microsecond, and tzinfo.
timedelta	A duration expressing the difference between two date, time, or datetime instances
tzinfo	An abstract base class for time zone information objects.
timezone	A class that implements the tzinfo abstract base class as a fixed offset from the UTC.

Date Time Module (cont.)

- The datetime module exports the following constants:
  - **MINYEAR –** value is 1
  - **MAXYEAR –** value is 9999

Darshan UNIVERSITY
योगः कर्मसु कौशलम्

date class

- The date class is used to instantiate date objects in Python. When an object of this class is instantiated, it represents a date in the format YYYY-MM-DD.
- Constructor of this class needs three mandatory arguments year, month and date.

syntax

```
datetime.date(year, month, day)
```

Example.py

```
1  import datetime
2
3  d = datetime.date(1879,4,14)
4  print("Birthday=",d)
```

OUTPUT

```
Birthday 1879-04-14
```

Example.py

```
1  import datetime
2  # print today date
3  d = datetime.date.today()
4  print("Today ",d)
```

OUTPUT

```
Today  2021-12-31
```

Example.py

```
1  import datetime
2  # print today date
3  d = datetime.date.today()
4  print("Date = ",d.day)
5  print("Month = ",d.month)
6  print("Year = ",d.year)
```

OUTPUT

```
Date =   31
Month =   12
Year =   2021
```

time class

- The time class creates the time object which represents local time, independent of any day.

syntax

```
datetime.time(hour, minute, second, microsecond=0, tzinfo=None, *, fold=0)
```

Example.py

```
1  import datetime
2
3  t = datetime.time(10,40,30)
4  print("Time=",t)
```

OUTPUT

```
Time= 10:40:30
```

Example.py

```
1  import datetime
2
3  t = datetime.time(10,40,30)
4  print("hour =",t.hour)
5  print("minute =",t.minute)
6  print("second =",t.second)
7  print("microsecond= ",t.microsecond)
```

OUTPUT

```
hour = 10
minute = 40
second = 30
microsecond = 0
```

Darshan UNIVERSITY

Datetime class

- The DateTime class contains information on both date and time.

syntax

```
datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None, *,fold=0)
```

Example.py

```
1  import datetime
2
3  a = datetime.datetime(1999, 12, 12, 12, 12, 12)
4  print("year =", a.year)
5  print("month =", a.month)
6  print("hour =", a.hour)
7  print("minute =", a.minute)
8  print("timestamp =", a.timestamp())
```

OUTPUT

```
year = 1999
month = 12
hour = 12
minute = 12
timestamp = 944980932.0
```

Example.py

```
1  import datetime
2
3  today = datetime.datetime.now()
4  print("Current ", today)
```

OUTPUT

```
Current  2021-12-31 23:10:42.397856
```

Timedelta class

- Python timedelta class is used for calculating differences in dates and also can be used for date manipulations in Python.

syntax

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)
```

Example.py

```
1  import datetime
2
3  today = datetime.datetime.now()
4  df2 = today + datetime.timedelta(days=2)
5  print("Current ", today)
6  print("After 2 days ", df2)
```

OUTPUT

```
Current  2021-12-31 23:10:42.397856
```

Example.py

```
1  import datetime
2
3  today = datetime.datetime.now()
4  df2 = today + datetime.timedelta(days=2)
5  td = df2 - today
6  print("Current ", today)
7  print("After 2 days ", df2)
8  print("difference ", td)
```

OUTPUT

```
After 2 days  2022-01-02 23:22:24.550069
difference  2 days, 0:00:00
```

Darshan UNIVERSITY

योगः कर्मसु कौशलम्

Format Datetime

- strftime()and strptime() method converts the given date, time or datetime object to the a string representation of the given format or visa versa.

Example.py

```
1  import datetime
2
3  my_string = str(input('Enter date(yyyy-mm-dd): '))
4  date1 = datetime.datetime.strptime(my_string, "%Y-%m-%d")

5  today = datetime.datetime.now()
6  s = date1.strftime("%A %m %Y")
7
8  print('Example 1:', s)
9  s = today.strftime("%a %m %y")
10 print('Example 2:', s)
11 s = today.strftime("%I %p %S")
12 print('Example 3:', s)
13 s = today.strftime("%H:%M:%S")
14 print('Example 4:', s)
15
```

OUTPUT

```
Enter date(yyyy-mm-dd): 1995-3-2
Example 1: Thursday 03 1995
Example 2: Fri 12 21
Example 3: 11 PM 38
Example 4: 23:38:38
```

Creating a custom module

- To create a custom module save the file with .py extension.

MyModule.py

```
1  # module contain Addition function
2
3  def Addition(a,b):
4     return a+b
```

- Now we can use the module we just created, by using the import statement.

Example.py

```
1  import MyModule
2
3  # function call
4  print("Add=",MyModule.Addition(3,4))
```

OUTPUT

```
Add= 7
```

Darshan UNIVERSITY

Variables in module

- The module can contain functions, as already described, but also variables of all types
- We can use them by modulename.variableName in our program.

MyModule.py

```
1  # module contain Addition function
2
3  list = [1,2,3,4,5,6,7,8,9]
4  def Addition(a,b):
5    return a+b
```

Example.py

```
1  import MyModule
2
3  # function call
4  print("Add=",MyModule.Addition(3,4))
5  # print list data
6  print("List=", MyModule.list)
```

OUTPUT

```
Add= 7
List= [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Import From Module

- We can choose to import only parts from a module, by using the from keyword.
- Sometimes we don't required entire module and we want to use only some of the functions from modules.
- This can be archived by from keyword with import.

MyModule.py

```
1  # module contain Addition function
2
3  list = [1,2,3,4,5,6,7,8,9]
4  def Addition(a,b):
5    return a+b
```

Example.py

```
1  from MyModule import list
2
3  # print list data
4  print("List=", list)
```

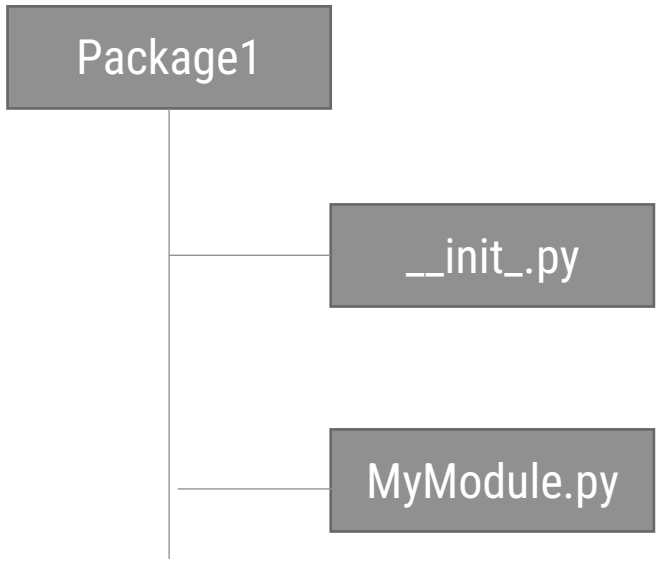
OUTPUT

```
List= [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- When importing using the from keyword, do not use the module name when referring to elements in the module.

Creating custom package

- First, we create a directory and give it a package name, preferably related to its operation
- Then we put the classes and the required functions in it.
- Finally we create an __init__.py file inside the directory, to let Python know that the directory is a package.



MyModule.py

```
1  # module contain Addition function
2
3  list = [1,2,3,4,5,6,7,8,9]
4  def Addition(a,b):
5    return a+b
```

Example.py

```
1  import Package1.MyModule
2
3  # print list data
4  print("List=", Package1.MyModule.list)
```

OUTPUT

```
List= [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Darshan UNIVERSITY

Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Unit-04.2

Matplotlib

Prof. Jayesh D. Vagadiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
Jayesh.vagadiya@darshan.ac.in
9537133260

Outline

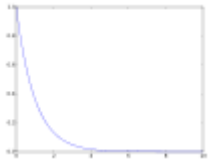
- ✓ Introduction to MatPlotLib
- ✓ Graph
- ✓ Plot
- ✓ Drawing Multiple Lines and Plots
- ✓ Export graphs/plots to Image/PDF/SVG
- ✓ Axis, Ticks ad Grids
- ✓ Line Appearance
- ✓ Labels, Annotation, Legends
- ✓ Types of Graphs
  - ✓ Pie Chart
  - ✓ Bar Chart
  - ✓ Histograms
  - ✓ Boxplots
  - ✓ Scatterplots
  - ✓ Time Series
  - ✓ Plotting Geographical data

Introduction to MatPlotLib

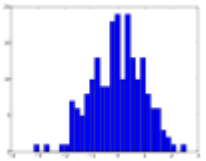
- Most people visualize information better when they see it in graphic versus textual format.
- Graphics help people see relationships and make comparisons with greater ease.
- Fortunately, python makes the task of converting textual data into graphics relatively easy using libraries, one of most commonly used library for this is MatPlotLib.
- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Graph

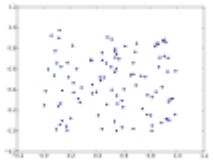
- A Graph or chart is simply a visual representation of numeric data.
- MatPlotLib makes a large number of graph and chart types.
- We can choose any of the common graph such as line charts, histogram, scatter plots etc....



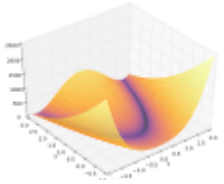
Line Chart



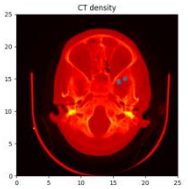
Histogram



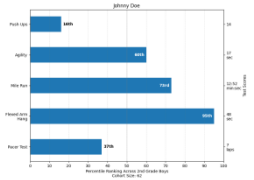
Scatter Plot



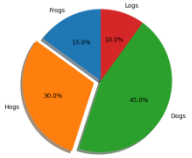
3D Plot



Images



Bar Chart



Pie Chart

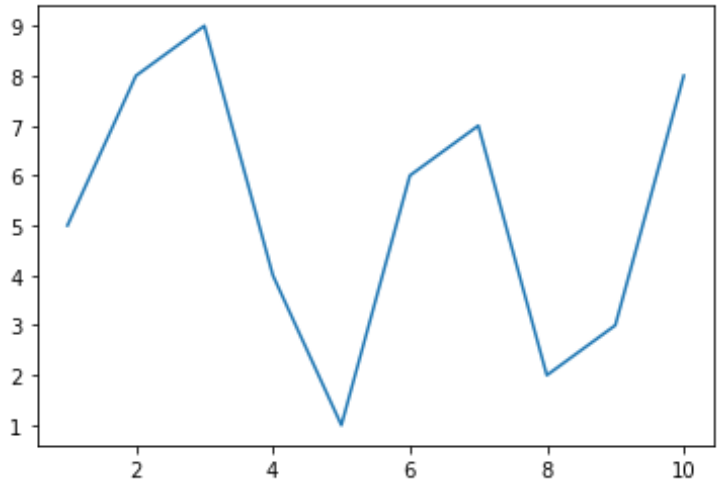
Etc.....

Plot

- To define a **plot**, we need some values, the matplotlib.pyplot module and an idea of what we want to display.

plotDemo1.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 values = [5,8,9,4,1,6,7,2,3,8]
4 plt.plot(range(1,11),values)
5 plt.show()
```



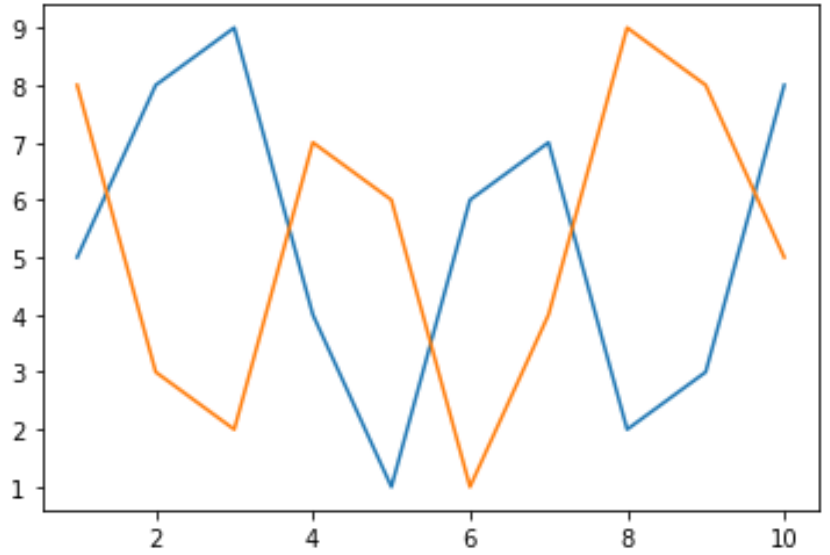
- In this case, the code tells the plt.plot() function to create a plot using x-axis between 1 and 11 and y-axis as per values list.

Plot – Drawing multiple lines

- We can draw multiple lines in a plot by making multiple `plt.plot()` calls.

plotDemo1.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 values1 = [5,8,9,4,1,6,7,2,3,8]
4 values2 = [8,3,2,7,6,1,4,9,8,5]
5 plt.plot(range(1,11),values1)
6 plt.plot(range(1,11),values2)
7 plt.show()
```

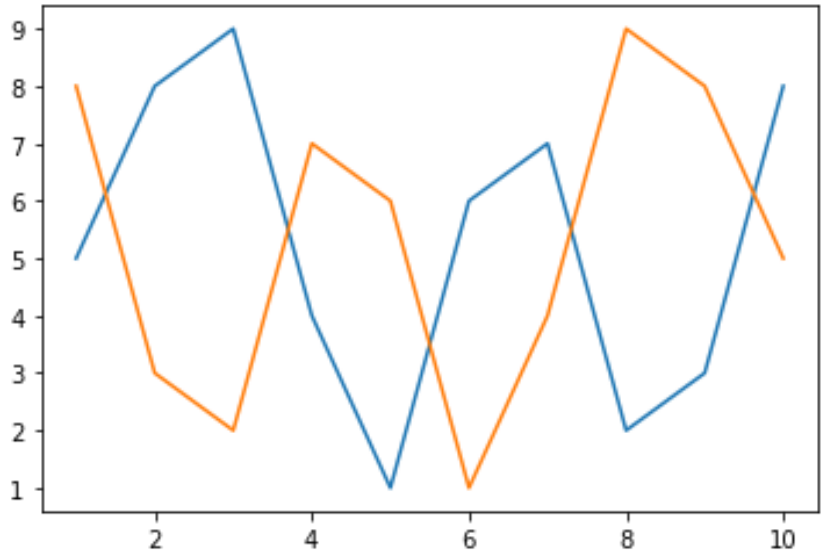


Plot – Export graphs/plots

- We can export/save our plots on a drive using `savefig()` method.

plotDemo1.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 values1 = [5,8,9,4,1,6,7,2,3,8]
4 values2 = [8,3,2,7,6,1,4,9,8,5]
5 plt.plot(range(1,11),values1)
6 plt.plot(range(1,11),values2)
7 #plt.show()
8 plt.savefig('SaveToPath.png',format='png')
```



SaveToPath.png

- Possible values for the format parameters are
  - png
  - svg
  - pdf
  - Etc...

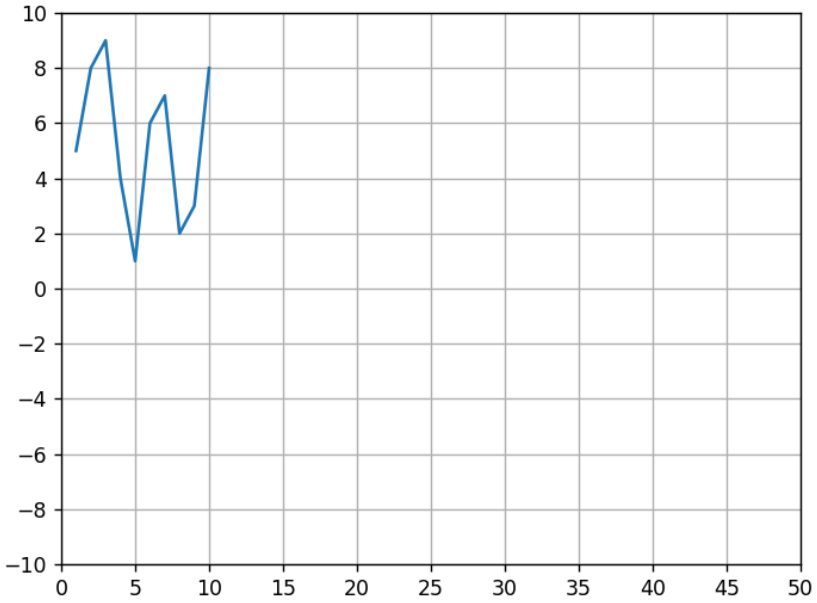
Darshan UNIVERSITY

Plot – Axis, Ticks and Grid

- We can access and format the axis, ticks and grid on the plot using the `axis()` method of the `matplotlib.pyplot.plt`

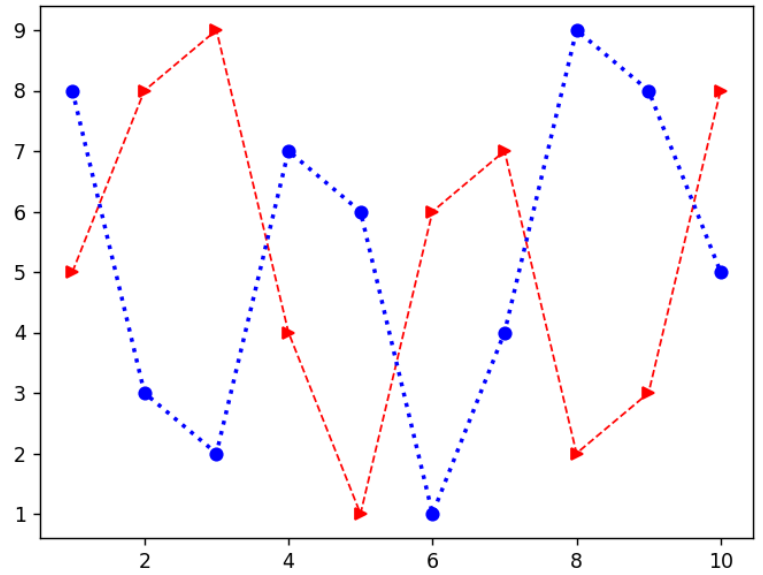
plotDemo1.py

```
1  import matplotlib.pyplot as plt
2  %matplotlib notebook
3  values = [5,8,9,4,1,6,7,2,3,8]
4  ax = plt.axes()
5  ax.set_xlim([0,50])
6  ax.set_ylim([-10,10])
7  ax.set_xticks([0,5,10,15,20,25,30,35,40,45,50])
8  ax.set_yticks([-10,-8,-6,-4,-2,0,2,4,6,8,10])
9  ax.grid()
10 plt.plot(range(1,11),values)
```



Plot – Line Appearance

- We need different line styles in order to differentiate when having multiple lines in the same plot, we can achieve this using many parameters, some of them are listed below.
  - Line style (linestyle or ls)
  - Line width (linewidth or lw)
  - Line color (color or c)
  - Markers (marker)



plotDemo1.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 values1 = [5,8,9,4,1,6,7,2,3,8]
4 values2 = [8,3,2,7,6,1,4,9,8,5]
5 plt.plot(range(1,11),values1,c='r',lw=1,ls='--',marker='>')
6 plt.plot(range(1,11),values2,c='b',lw=2,ls=':',marker='o')
7 plt.show()
```

Plot – Line Appearance (Cont.)

- Possible Values for each parameters are,

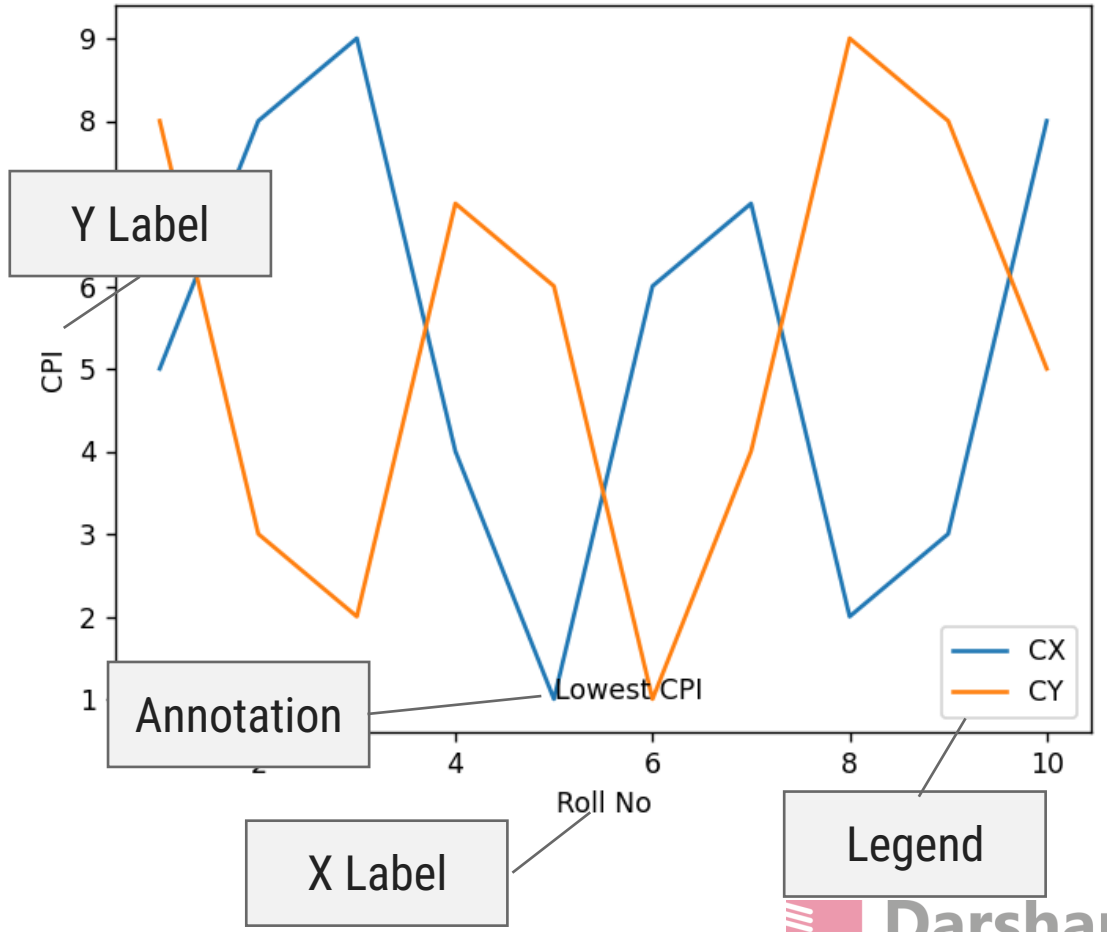
Values	Line Style
'-'	Solid line
'--'	Dashed line
'-.'	Dash-dot line
':'	Dotted line

Values	Color
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Values	Marker
'.'	Point
','	Pixel
'o'	Circle
'v'	Triangle down
'^'	Triangle up
'>'	Triangle right
'<'	Triangle left
'*'	Star
'+'	Plus
'x'	X
Etc.....	

Plot – Labels, Annotation and Legends

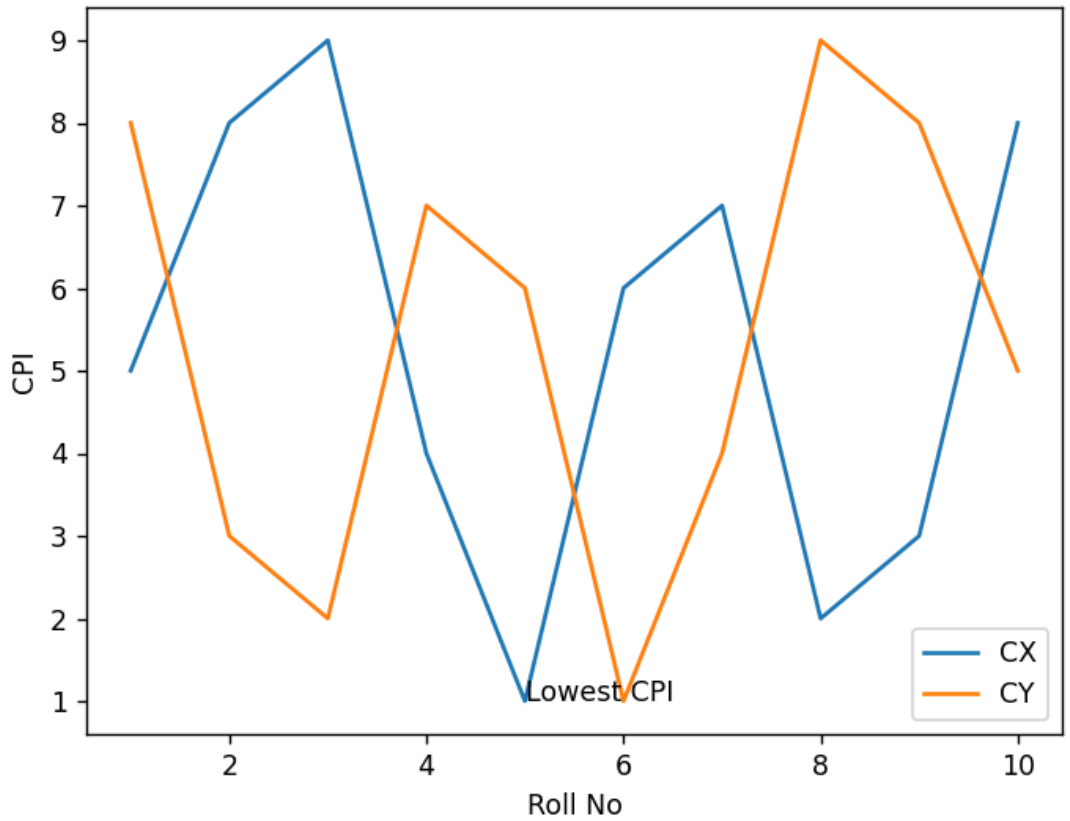
- To fully document our graph, we have to resort the labels, annotation and legends.
- Each of this elements has a different purpose as follows,
  - **Label** : provides identification of a particular data element or grouping, it will make easy for viewer to know the name or kind of data illustrated.
  - **Annotation** : augments the information the viewer can immediately see about the data with notes, sources or other useful information.
  - **Legend** : presents a listing of the data groups within the graph and often provides cues (such as line type or color) to identify the line with the data.



Plot – Labels, Annotation and Legends (Example)

plotDemo1.py

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
3  values1 = [5,8,9,4,1,6,7,2,3,8]
4  values2 = [8,3,2,7,6,1,4,9,8,5]
5  plt.plot(range(1,11),values1)
6  plt.plot(range(1,11),values2)
7  plt.xlabel('Roll No')
8  plt.ylabel('CPI')
9  plt.annotate(xy=[5,1],s='Lowest CPI')
10 plt.legend(['CX','CY'],loc=4)
11 plt.show()
```



Choosing the Right Graph

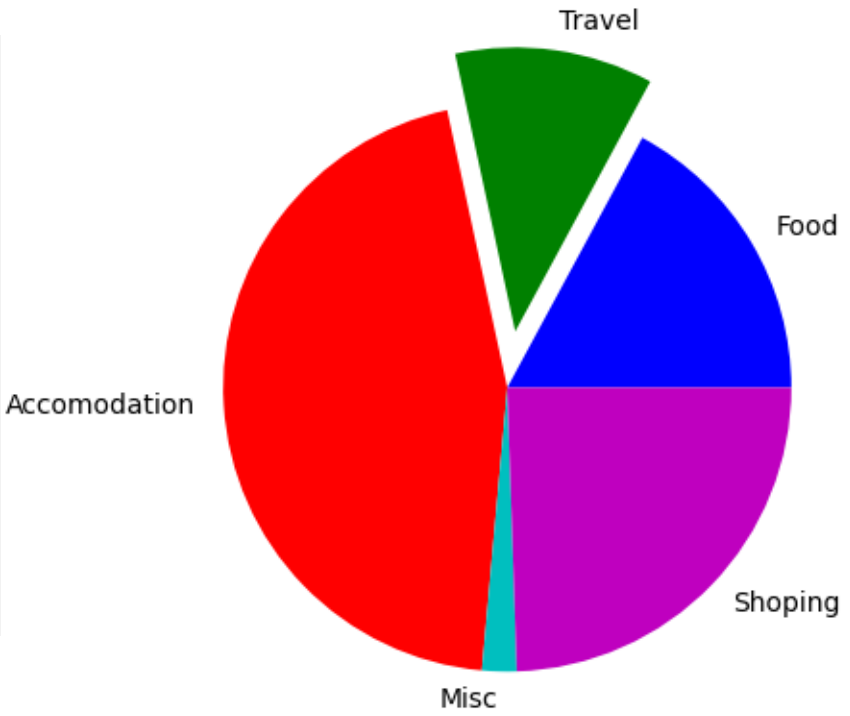
- The kind of graph we choose determines how people view the associated data, so choosing the right graph from the outset is important.
- For example,
  - if we want to show how various data elements **contribute towards a whole**, we should use **pie chart.**
  - If we want to **compare data** elements, we should use **bar chart.**
  - If we want to **show distribution** of elements, we should use **histograms.**
  - If we want to **depict groups** in elements, we should use **boxplots.**
  - If we want to **find patterns** in data, we should use **scatterplots.**
  - If we want to **display trends** over time, we should use **line chart.**
  - If we want to **display geographical** data, we should use **basemap.**
  - If we want to **display network**, we should use **networkx.**
- All the above graphs are there in our syllabus and we are going to cover all the graphs in this Unit.
- We are also going to cover some other types of libraries which is not in the syllabus like seaborn, plotly, cufflinks and choropleth maps etc..

Pie Chart

- Pie chart focus on showing parts of a whole, the entire pie would be 100 percentage, the question is how much of that percentage each value occupies.

pieChartDemo.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib notebook
3 values = [305,201,805,35,436]
4 l =
  ['Food','Travel','Accomodation','Misc','Shoping']
5 c = ['b','g','r','c','m']
6 e = [0,0.2,0,0,0]
7 plt.pie(values,colors=c,labels=l,explode=e)
8 plt.show()
```

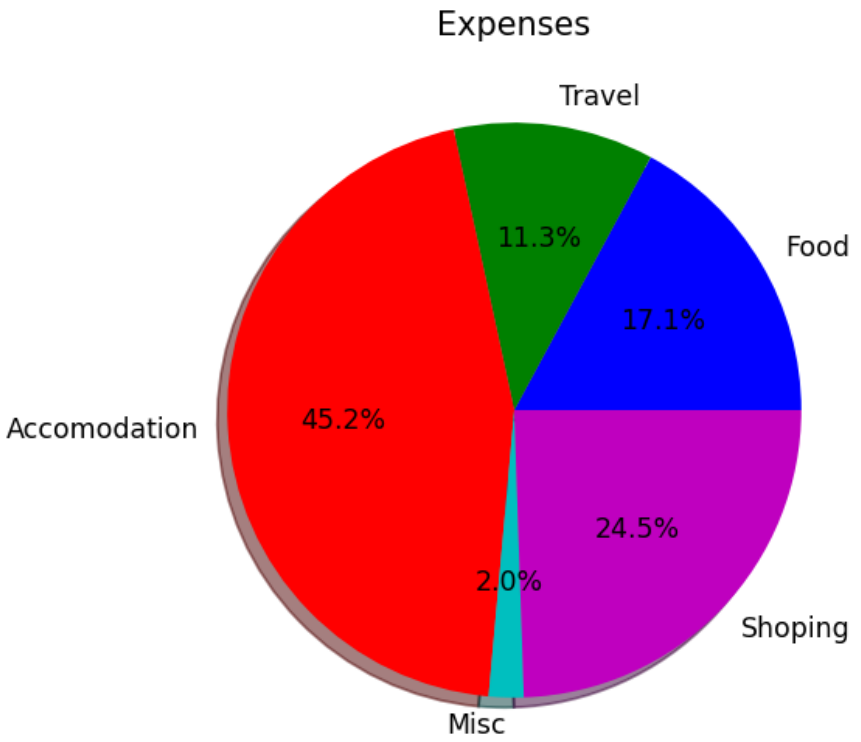


Pie Chart (Cont.)

- There are lots of other options available with the pie chart, we are going to cover two important parameters in this slide.

pieChartDemo.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib notebook
3 values = [305,201,805,35,436]
4 l =
  ['Food','Travel','Accomodation','Misc','Shoping']
5 c = ['b','g','r','c','m']
6 plt.pie(values,colors=c,labels=l,shadow=True,
7         autopct='%1.1f%%')
8 plt.show()
```

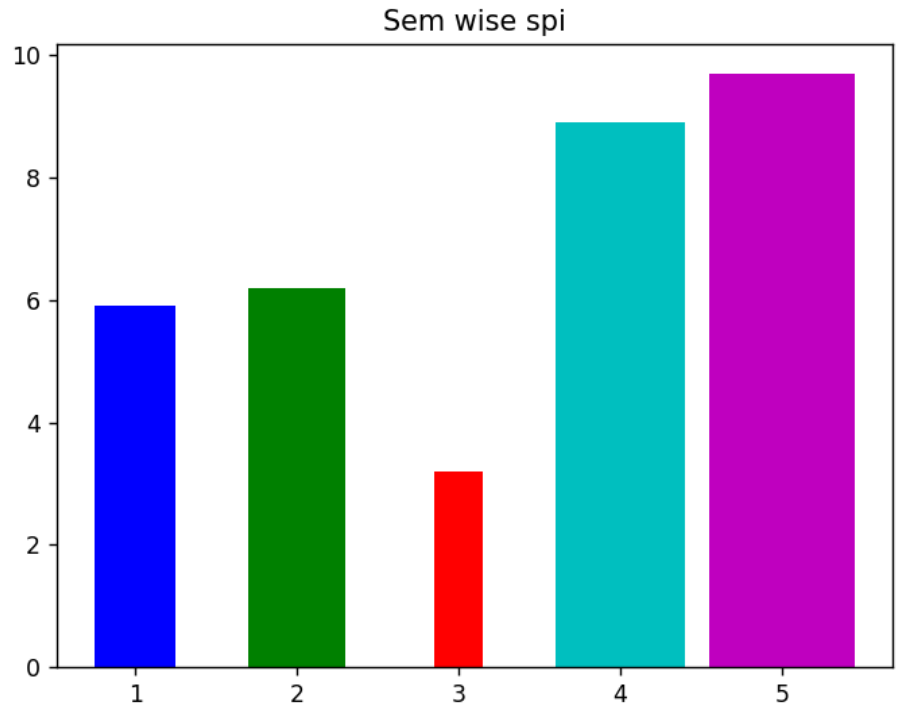


Bar charts

- Bar charts make comparing values easy, wide bars an d segregated measurements emphasize the difference between values, rather that the flow of one value to another as a line graph.

barChartDemo.py

```
1  import matplotlib.pyplot as plt
2  %matplotlib notebook
3  x = [1,2,3,4,5]
4  y = [5.9,6.2,3.2,8.9,9.7]
5  l = ['1st','2nd','3rd','4th','5th']
6  c = ['b','g','r','c','m']
7  w = [0.5,0.6,0.3,0.8,0.9]
8  plt.title('Sem wise spi')
9  plt.bar(x,y,color=c,label=l,width=w)
10 plt.show()
```



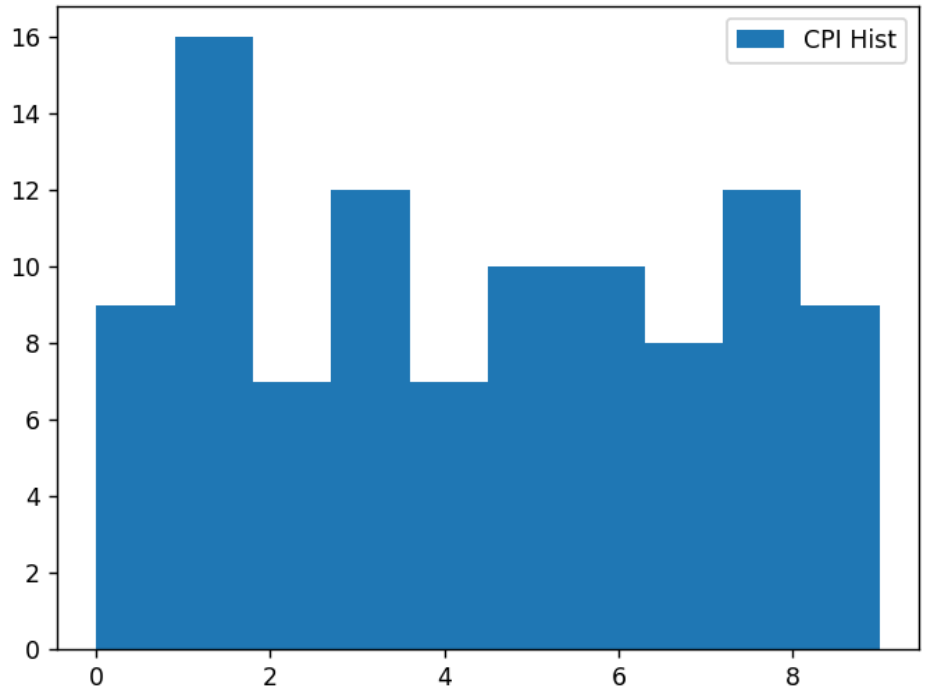
- Note: you can use **barh()** function to generate horizontal bar chart.

Histograms

- Histograms categorize data by breaking it into bins, where each bin contains a subset of the data range.
- A Histogram then displays the number of items in each bin so that you can see the distribution of data and the progression of data from bin to bin.

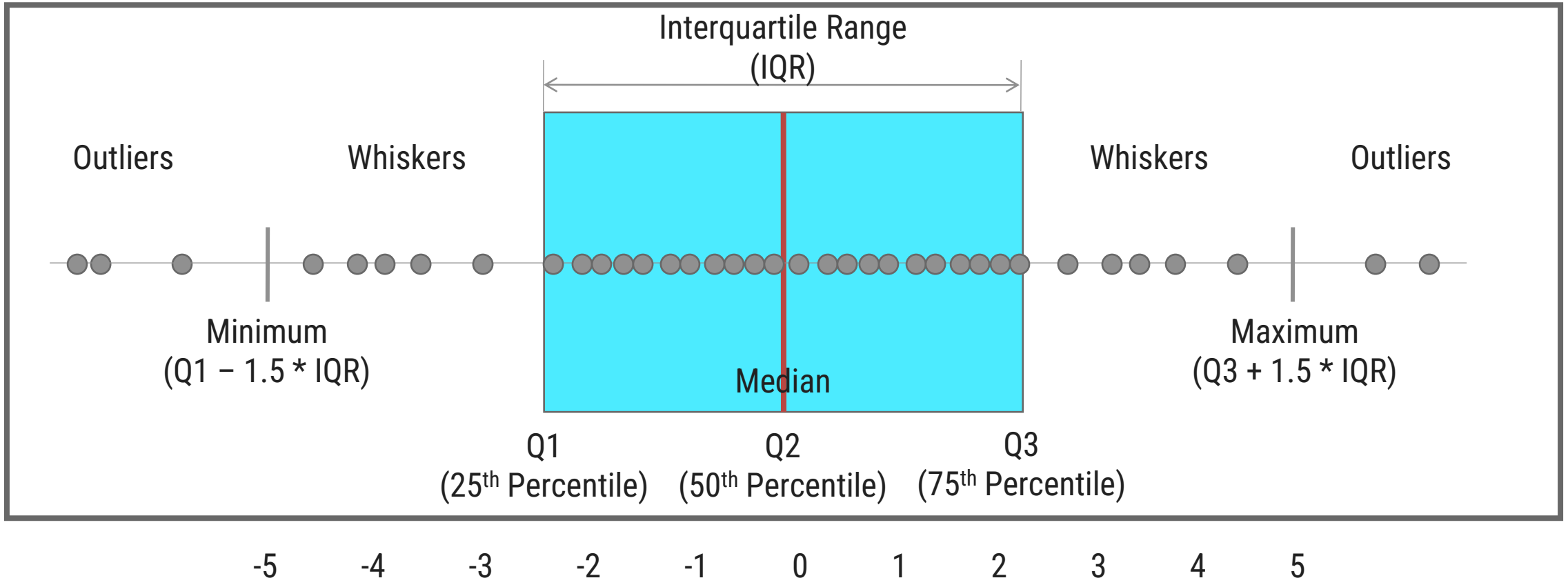
histDemo.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 %matplotlib notebook
4 cpis = np.random.randint(0,10,100)
5 plt.hist(cpis,bins=10,
  histtype='stepfilled',align='mid',label
  ='CPI Hist')
6 plt.legend()
7 plt.show()
```



Boxplots

- Boxplots provide a means of depicting groups of numbers through their quartiles.
- Quartiles means three points dividing a group into four equal parts.
- In boxplot, data will be divided in 4 part using the 3 points (25th percentile, median, 75th percentile)

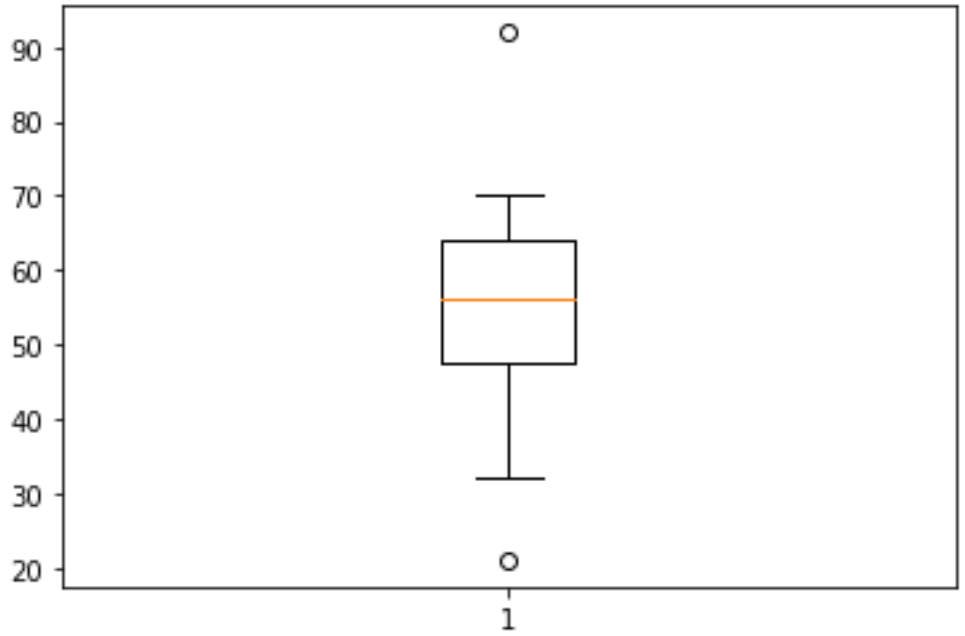


Boxplot (Cont.)

- Boxplot basically used to detect outliers in the data, lets see an example where we need boxplot.
- We have a dataset where we have time taken to check the paper, and we want to find the faculty which either takes more time or very little time to check the paper.

boxDemo.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 plt.boxplot([50,45,52,63,70,21,56,68,54
  ,57,35,62,65,92,32])
4 plt.show()
```



- We can specify other parameters like
  - widths, which specify the width of the box
  - notch, default is False
  - vert, set to 0 if you want to have horizontal graph

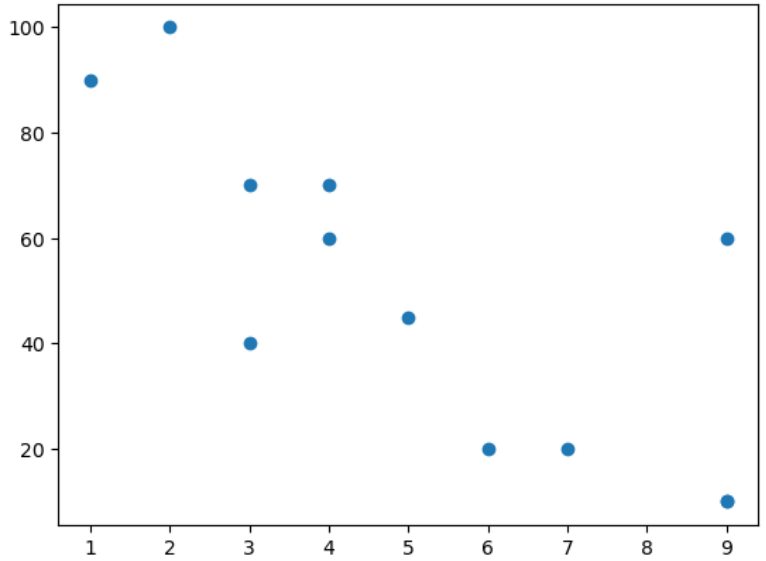
Darshan UNIVERSITY

Scatter Plot

- A scatter plot is a type of plot that shows the data as a collection of points.
- The position of a point depends on its two-dimensional value, where each value is a position on either the horizontal or vertical dimension.
- It is really useful to study the **relationship/pattern** between variables.
- now, Consider one terminal which records the speed of the cars.

histDemo.py

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 carAge = [2,5,7,9,4,3,1,9,4,3,6,9]
4 carspeed =
  [100,45,20,10,60,70,90,60,70,40,20,10]
5 plt.scatter(carAge, carspeed)
6 plt.show()
```

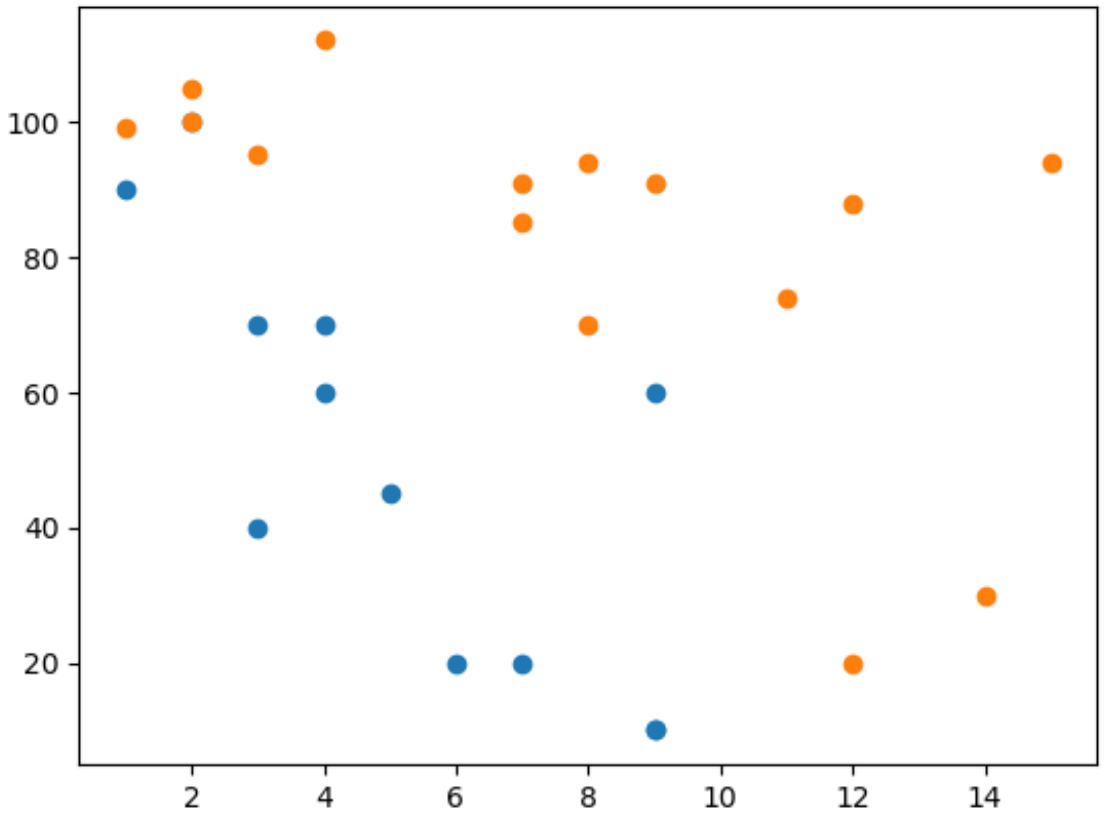


Scatter Plot

- Now take days wise observations.

histDemo.py

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
3  # day -1
4  carAge = [2,5,7,9,4,3,1,9,4,3,6,9]
5  carspeed =
   [100,45,20,10,60,70,90,60,70,40,20,10]
   # day -2
6  carAge1 =
   [2,2,8,1,15,8,12,9,7,3,11,4,7,14,12]
7  carspeed1 =
   [100,105,70,99,94,94,88,91,91,95,74,112
   ,85,30,20]
8  plt.scatter(carAge, carspeed)
9  plt.scatter(carAge1, carspeed1)
10 plt.show()
```



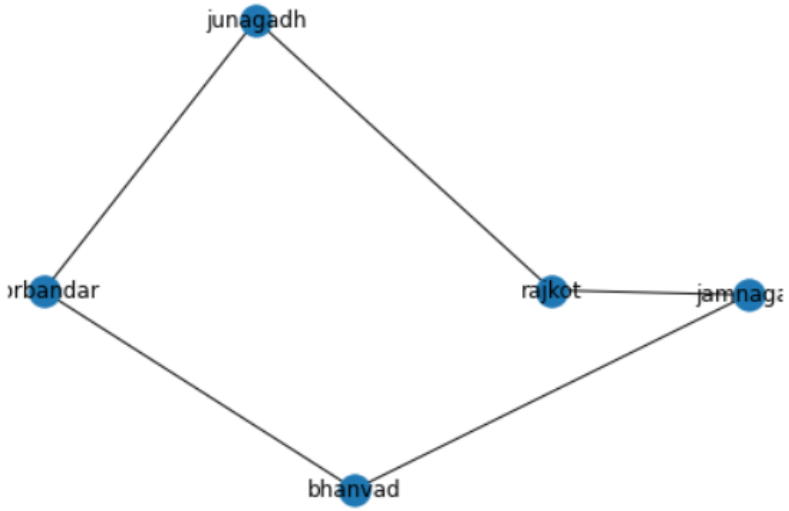
NetworkX

- We can use networkx library in order to deal with any kind of networks, which includes social network, railway network, road connectivity etc….
- Install
  - pip install networkx
  - conda install networkx
- Types of network graph
  - Undirected
  - Directed
  - Weighted graph

NetworkX (example)

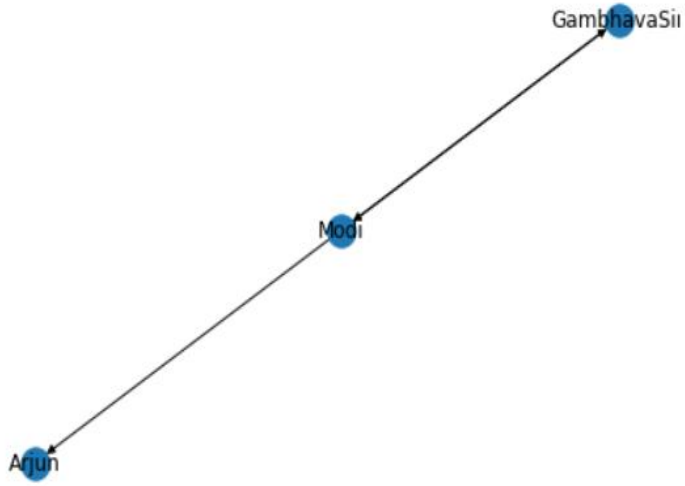
networkxDemo.py

```
1 import networkx as nx
2 g = nx.Graph() # undirected graph
3 g.add_edge('rajkot','junagadh')
4 g.add_edge('junagadh','porbandar')
5 g.add_edge('rajkot','jamnagar')
6 g.add_edge('jamnagar','bhanvad')
7 g.add_edge('bhanvad','porbandar')
8 nx.draw(g,with_labels=True)
```



networkxDemo.py

```
1 import networkx as nx
2 gD = nx.DiGraph() # directed graph
3 gD.add_edge('Modi','Arjun')
4 gD.add_edge('Modi','GambhavaSir')
5 gD.add_edge('GambhavaSir','Modi')
6
7 nx.draw(gD, with_labels=True)
```



NetworkX (cont.)

- We can use many analysis functions available in NetworkX library, some of functions are as below
  - nx.shortest_path(g,'rajkot','porbandar')
    - Will return ['rajkot', 'junagadh', 'porbandar']
  - nx.dijkstra_path(g,'rajkot','porbandar') // will consider weight
    - Will return ['rajkot', 'junagadh', 'porbandar']
  - nx.clustering(g)
    - Will return clustering value for each node
  - nx.degree_centrality(g)
    - Will return the degree of centrality for each node, we can find most popular/influential node using this method.
  - nx.density(g)
    - Will return the density of the graph.
    - The density is 0 for a graph without edges and 1 for a complete graph.
  - nx.info(g)
    - Return a summary of information for the graph G.
    - The summary includes the number of nodes and edges, and their average degree.

Choropleth Maps in Python

- Choropleth maps are used to plot maps with shaded or patterned areas which are proportional to a statistical variable.
- They are composed of colored polygons. They are used for representing spatial variations of a quantity.
- Geometric information:
  - GeoJSON file
  - this can be built-in geometries of plotly – US states and world countries

Choropleth Maps in Python

histDemo.py

```
1 import plotly.express as px
2 fig = px.choropleth(locations=["CA","TX","NY"], locationmode="USA-states",
  color=[1,2,3], scope="usa")
3 fig.show()
```