Introduction to Computer Programming

Week 9.1: Matplotlib - Plotting

Matplotlib is a large and versatile package for visualising data.

It is useful for creating graphs and plots.

We will study a submodule of `matlplotlib` called `pyplot`.

`pyplot` is often *renamed* as `plt` on import.

Any functions from `pyplot` can be used by prepending with `plt`

e.g. `plt.plot()`

In [2]:

```python
import matplotlib.pyplot as plt
```

To display plots created using Matplotlib in Jupyter Notebook, the following line of code must be run in the notebook *before* generating the plot:

In [3]:

```python
%matplotlib inline
```

To display plots when running a .py file (e.g. in Spyder), the following line must appear in the programme *after* generating the plot:

In [4]:

```python
plt.show()

```

We will also use numpy today

In [5]:

```python
import numpy as np
```

Line and Scatter Graphs

A sample data set: x with corresponding values of f:

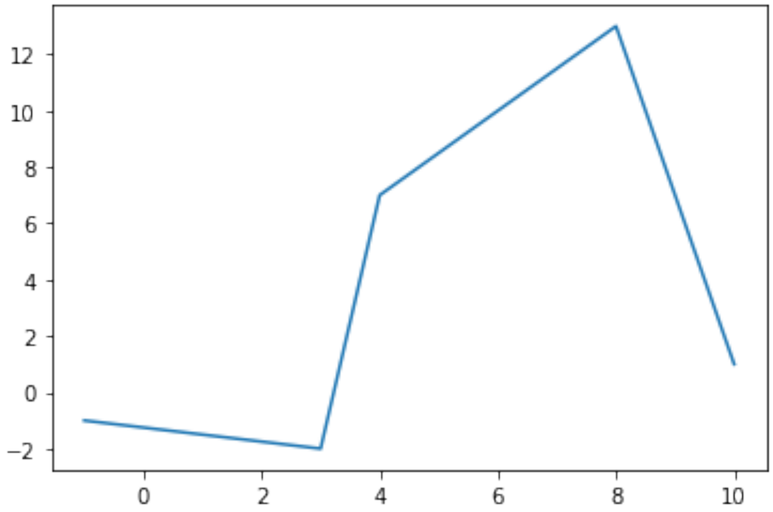
In [6]:

```
x = [-1, 3, 4, 8 , 10]
f = [-1, -2, 7, 13 , 1]

```

Line plot

In [7]:

```
plt.plot(x, f)

plt.show()

```



Printing the statement with format `[<matplotlib.lines.Line2D at 0x30990b0>]` appears each time (the numbers on your computer may look different) can be avoided by including a semicolon after the `plot` function.

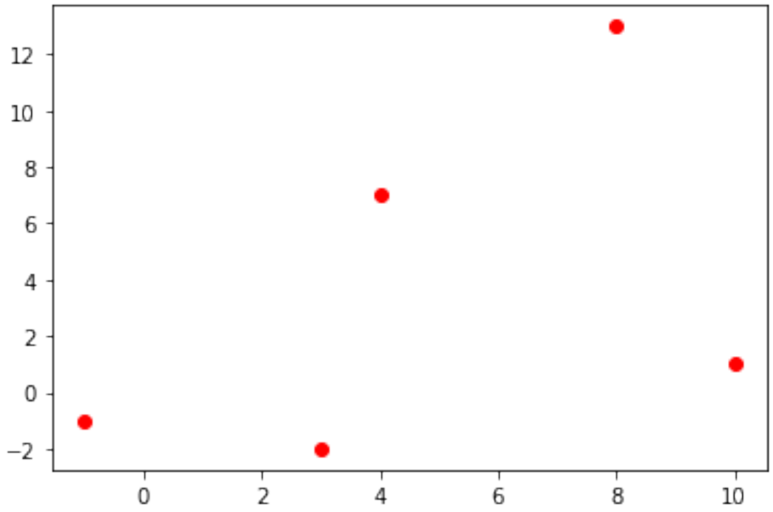
```
plot(x, y);
```

Format string: an *optional* way to define basic formatting

- the colour of the plot (e.g. `r` = red, `k` = black) https://matplotlib.org/2.0.2/api/colors_api.html (https://matplotlib.org/2.0.2/api/colors_api.html)
- the style of the markers (e.g. `o` = points, `*` = stars) https://matplotlib.org/api/markers_api.html (https://matplotlib.org/api/markers_api.html)
- the style of the line (e.g. `--` = dashes, `.` = dots) https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/line_styles_reference.h (https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/line_styles_reference.h

In [8]:

```
plt.plot(x, f, 'or') # scatter, o markers, red

plt.show()
```



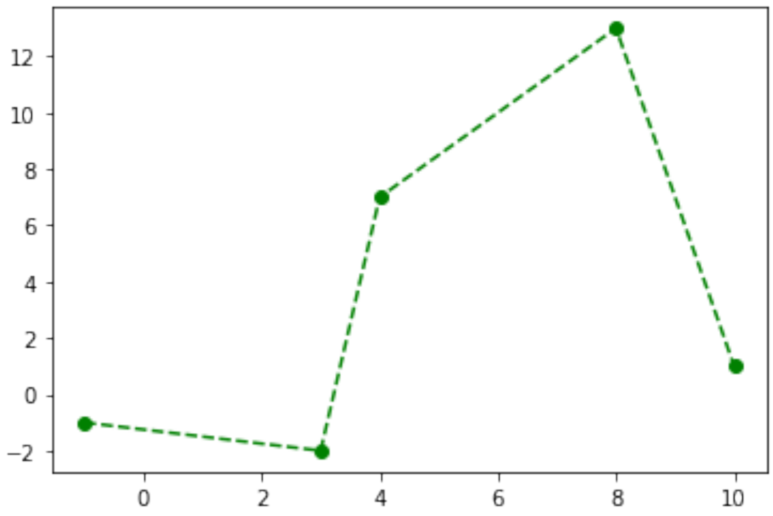
Example 1: Use the format string to change the appearance of the plot of f against x.

In [12]:

```python
plt.plot(x, f, '--og')


```

Out[12]:

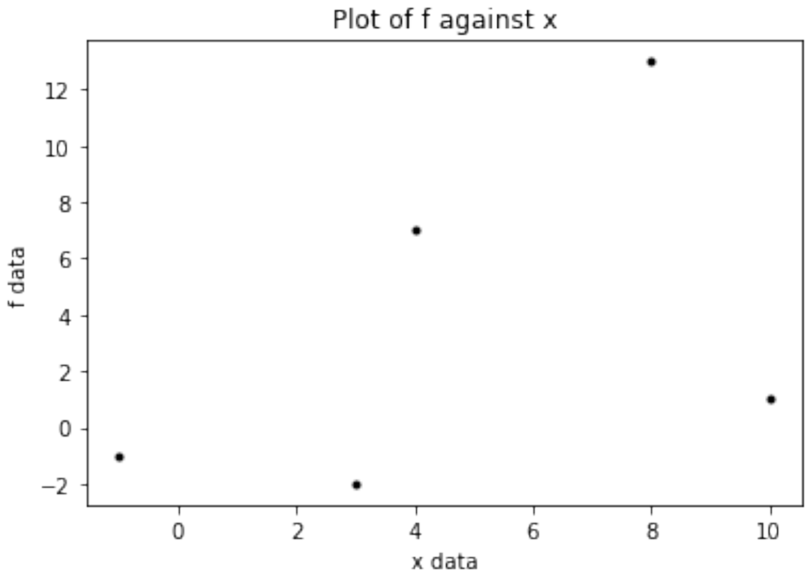
```
[<matplotlib.lines.Line2D at 0x7f907548eaf0>]
```



We can add features like axis labels and a title

In [44]:

```python
plt.plot(x, f, 'k.')

# Axis labels
plt.xlabel('x data')
plt.ylabel('f data')

# title
plt.title('Plot of f against x')

plt.show()
```

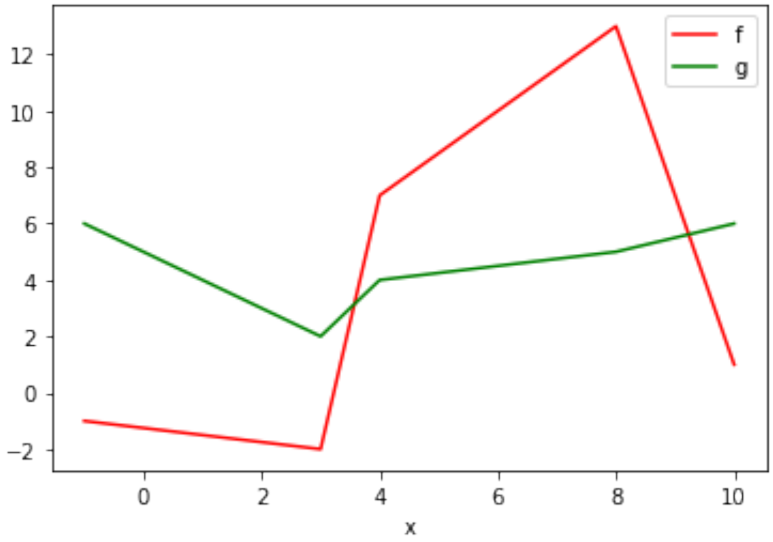


Multiple lines on the same plot

Names can be given to each data series using the `label` named argument when plotting with `plot` .

In [45]:

```
x = [-1, 3, 4, 8 , 10]
f = [-1, -2, 7, 13 , 1]
y = [6, 2, 4, 5, 6]

plt.plot(x, f, '-r', label='f');  # argument 'label' is used in
plt.plot(x, y, '-g', label='g');

plt.xlabel('x');

plt.legend()                      # must be called to display f

plt.show()
```



Bar Charts

Steps to create a bar chart:

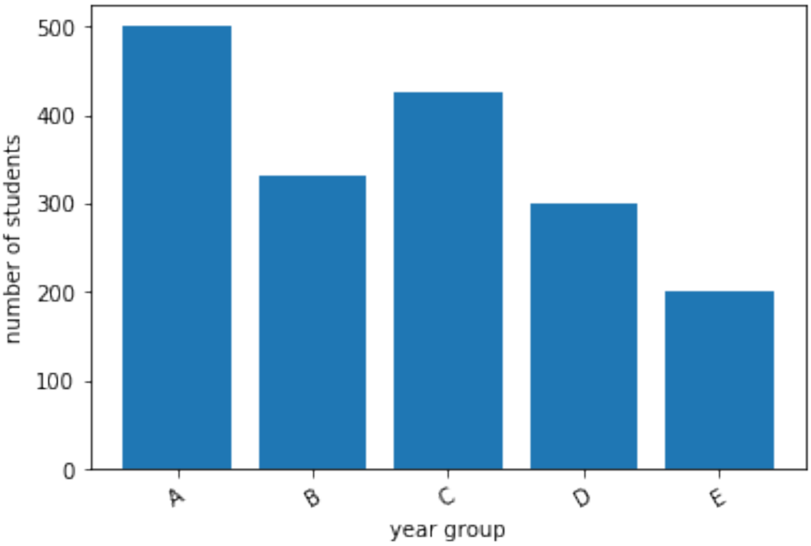
1. Create a numpy array with the same number of positions as bars
2. Generate bar chart using `plt.bar`
3. Replace x ticks with bar names

In [19]:

```
#sample data
groups = ('A', 'B', 'C', 'D', 'E')
num_students = (500, 332, 425, 300, 200)

```

In [47]:

```

# 1. Create a numpy array with the same number of positions as
x_pos = np.arange(len(groups))

# 2. Generate bar chart
plt.bar(x_pos, num_students)

# 3. Replace x ticks with bar names
# (Rotate labels 30 degrees)
plt.xticks(x_pos, groups, rotation=30)

# Add axis labels
plt.xlabel('year group')
plt.ylabel('number of students')

plt.show()

```



Histograms

We can visualise the distribution values using a histogram.

In a histogram, data is sorted into intervals (bins) along one axis.

The number of values that fall within a 'bin' is then displayed on the perpendicular axis.

Example data set of 25 random integers between 0 and 100, generated using `numpy.random.randint`

In [14]:

```
import numpy as np
z = np.random.randint(low=0, high=100, size=25)
```

We can visualise the frequency distribution of `x` by defining bins to hold different ranges of values.

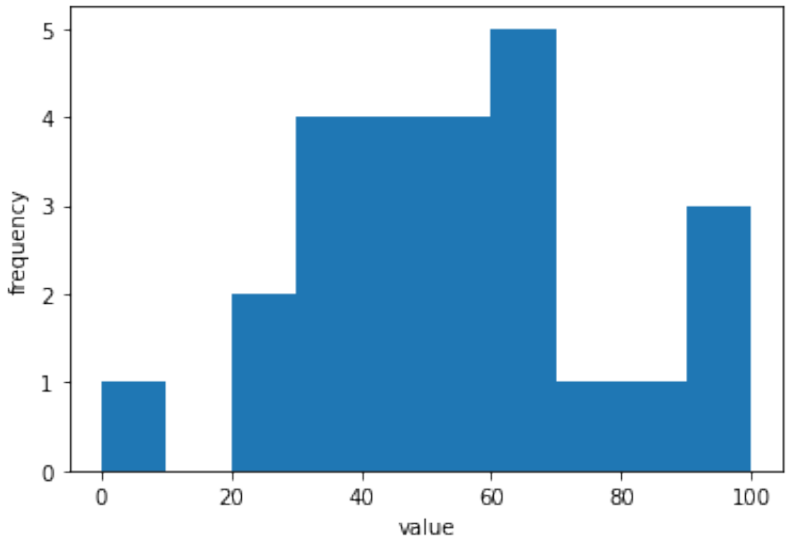
`np.arange()` generates a numpy array of values.

In the example, the edge values of the bins are defined by the array.

In [15]:

```
edges = np.arange(0, 101, 10) # start, stop, step
print(edges)

plt.hist(z, bins=edges)

plt.xlabel('value')
plt.ylabel('frequency')

plt.show()
```

```
[  0  10  20  30  40  50  60  70  80  90 100]
```



Bins are defined by their edge values:

If argument `bins` is:

- **an integer:** defines the number of equal-width bins.
- **a sequence:** (`arange`) defines the bin edges.

Bins *include* left-most value, but *exclude* right-most value.
(Except right-most bin: *includes* left-most and right-most value)

e.g. `bins = [1,2,3,4]`

- first bin [1, 2)
- second bin [2, 3)
- third bin [3, 4]

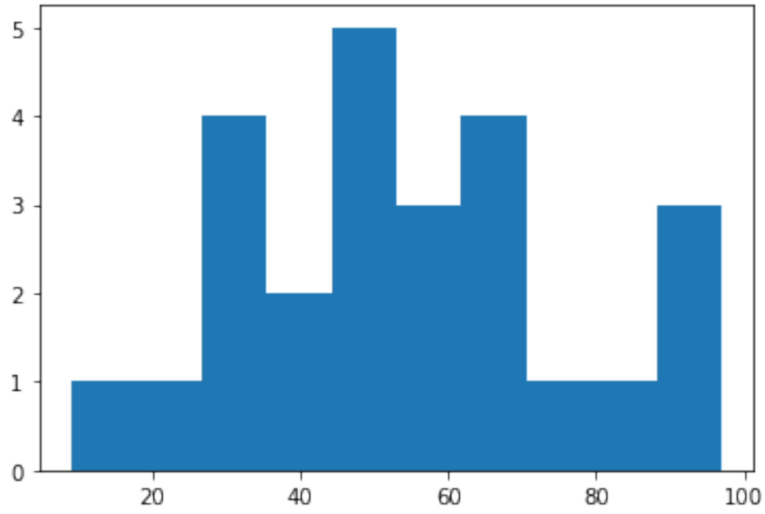
Generating the histogram returns 3 values:

- value of bins
- edges of bins
- graphical data for constructing histogram

In [16]:

```
n, edges, patches = plt.hist(z, bins=10);

print(n)

```

```
[1. 1. 4. 2. 5. 3. 4. 1. 1. 3.]
```



Subplots

Multiple plots can be included in the same figure using `sublplot`

`subplot(nrows ncols index)`

In [51]:

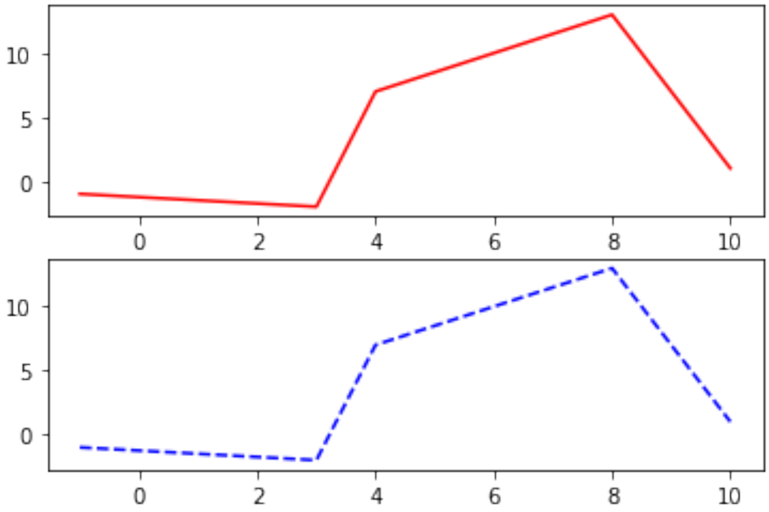
```
x = [-1, 3, 4, 8, 10]
f = [-1, -2, 7, 13 , 1]
```

In [52]:

```
plt.subplot(211)     # 2 rows, 1 column, index 1
plt.plot(x, f, 'r')

plt.subplot(212)     # 2 rows, 1 column, index 2
plt.plot(x, f, 'b--')
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x7fee25c30f10>]
```

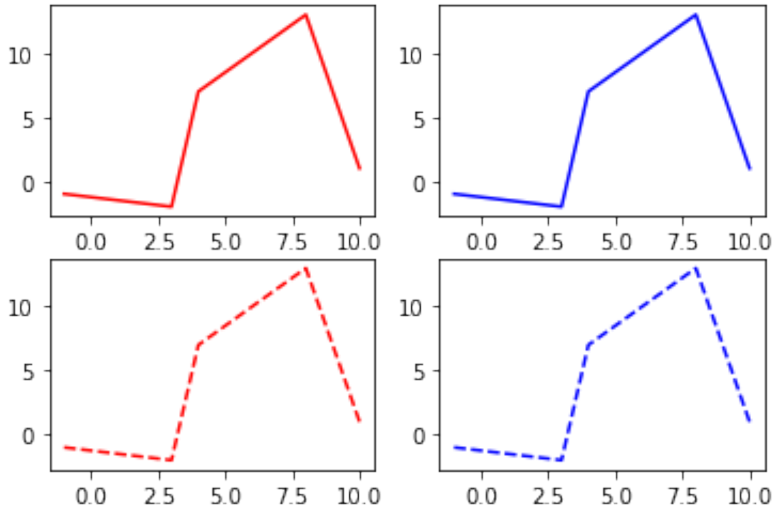


In [53]:

```
plt.subplot(221)    # 2 rows, 2 columns, index 1
plt.plot(x, f, 'r')

plt.subplot(222)    # 2 rows, 2 columns, index 2
plt.plot(x, f, 'b')

plt.subplot(223)    # 2 rows, 2 columns, index 3
plt.plot(x, f, 'r--')

plt.subplot(224)    # 2 rows, 2 columns, index 4
plt.plot(x, f, 'b--')
```

Out[53]:

```
[<matplotlib.lines.Line2D at 0x7fee232d0710>]
```



Alternatively, `subplots(nrows, ncols)` can be used.

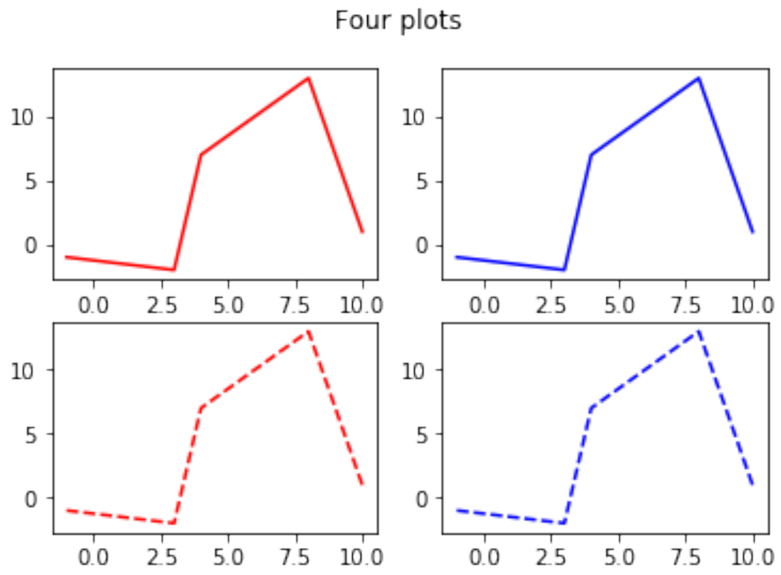
This allows you to control change paraemters at plot or subplot level.

In [54]:

```
fig, ax = plt.subplots(2, 2)

ax[0,0].plot(x, f, 'r')

ax[0,1].plot(x, f, 'b')

ax[1,0].plot(x, f, 'r--')

ax[1,1].plot(x, f, 'b--')

fig.suptitle('Four plots')
```

Out[54]:

```
Text(0.5, 0.98, 'Four plots')
```



Example 2:

Display:

- the bar chart of the students in each group
- the histogram of the frequency distribution of z

as two subplots on the same figure.

`subplots_adjust` can be used to adjust the spacing between plots https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots_adjust.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots_adjust.html)

In [ ]:

Saving plots

Plots can be saved in widely used formats (.png, .pdf etc) specified using the file name extension.

In [27]:
```
plt.savefig('img/my_plots.png')

```

```
<Figure size 432x288 with 0 Axes>
```

Must appear *before*:
`matplotlib.pyplot.show()`
i.e. `plt.show()`

Importing Data with Numpy

`numpy.loadtxt` can be used to import data from delimited text files.
The user can specify parameters including:

- **delimiter** (default = whitespace)
- **data type** (default = float) :
  If data contains items that cannot be expressed as a float, importing will cause an error unless the data-type is specified.
  Mixed data types can be imported as `string` values.

Example: Import data from `sample_data/data.dat`

- **delimiter** : whitespace
- **data type** : float

```
 0.000 1.053 2.105 3.158 4.211
74.452 48.348 68.733 59.796 54.123
```

In [5]:

```
import numpy as np
A = np.loadtxt('sample_data/sample_data.dat')

print(A)        # stored as numpy array

print(A[0][1]) # individual elements can be addressed
```

```
[[ 1.053  2.105  3.158  4.211  6.065]
 [48.348 68.733 59.796 54.123 74.452]]
2.105
```

Regions can be selected, for example to exclude headings and select only numerical data.

`skiprows` skips the first n lines.

`usecols` specifies which columns to read (numbering starts at 0)

- `usecols = (1, 4, 5)` : extracts the 2nd, 5th and 6th columns.
- `usecols = (3, 4)` : extracts the 4th and 5th columns

In [31]:

```
import numpy as np
A = np.loadtxt('sample_data/sample_data.dat',
                skiprows=1,
                usecols=(2,3,4))

print(A)        # stored as numpy array
```

```
[59.796 54.123 74.452]
```

Example 3:
Import height and weight data from `sample_data/sample_student_data.txt` and plot a scatter plot of the data.

How can we change the colour of the markers in the plot?

In [ ]:

In [ ]:

Summary

- Simple line and scatter plots can be customised using a `formatstring`
- Features such as a figure legend and axis labels can be added after generating the plot.
- Steps to generate a bar chart:
    1. Create a numpy array with the same number of positions as bars
    2. Generate bar chart
    3. Replace x ticks with bar name
- Plots can be saved with `matplotlib.pyplot.savefig`

Further reading

- Matplotilib has built-in tools for many more types of plot (scatter, box and whisker, 3D surface, animation etc)
- Matplotlib Gallery (http://matplotlib.org/gallery.html (http://matplotlib.org/gallery.html))
- Github (http://gree2.github.io/python/2015/04/10/python-matplotlib-plotting-examples-and-exercises (http://gree2.github.io/python/2015/04/10/python-matplotlib-plotting-examples-and-exercises))