

# 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 `matplotlib` called `pyplot`.

A widely-used way to import the `matplotlib` module is by adding the following line at the start of your code.

Any function belonging to the `matplotlib` module can then be accessed by writing, for example, `plt.plot()`

In [2]:

```
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 [1]:

```
%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]:

```
plt.show()
```

We will also use `numpy`

In [ ]:

```
import numpy as np
```

## Line and Scatter Graphs

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

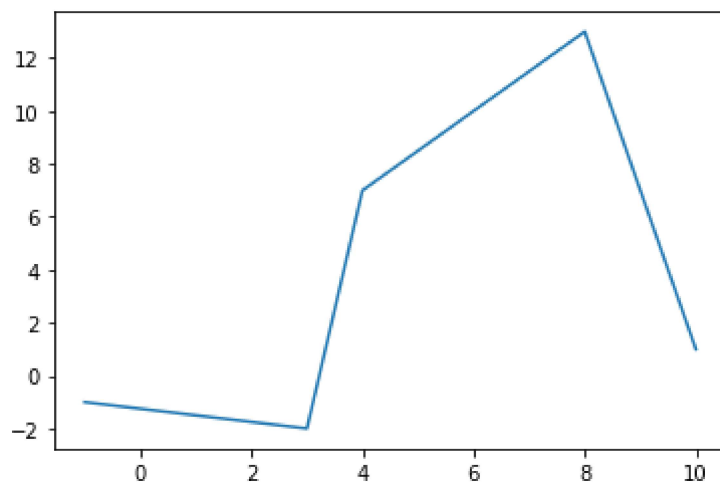
In [2]:

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

Line plot

In [8]:

```
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.

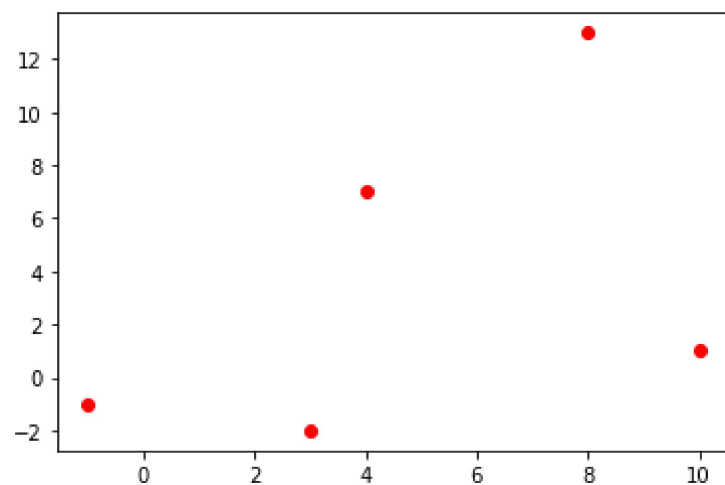
```
`plot(x, y);`
```

**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) ([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) ([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.html](https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/line_styles_reference.html)  
([https://matplotlib.org/devdocs/gallery/lines\\_bars\\_and\\_markers/line\\_styles\\_reference.html](https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/line_styles_reference.html))

In [9]:

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



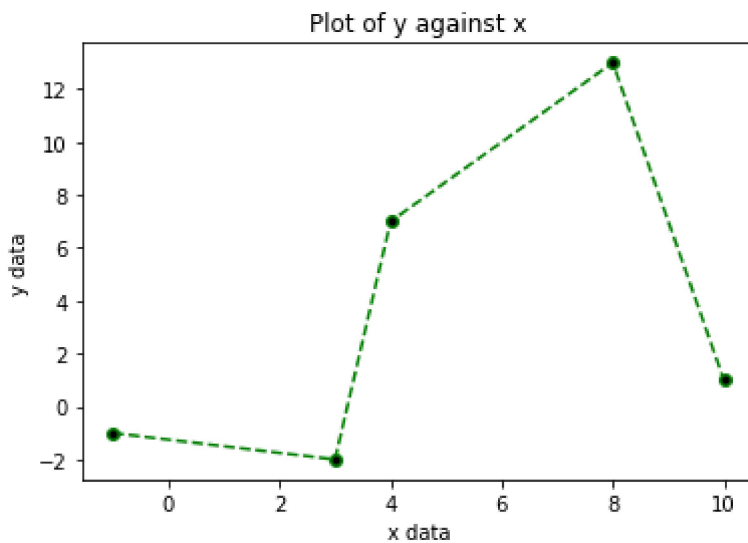
In [10]:

```
#plt.plot(x, f, '--og');  
plt.plot(x, f, '--og');  
#plt.plot(x, f, 'ro');  
plt.plot(x, f, 'k.');
```

*# Axis labels*  
plt.xlabel('x data');  
plt.ylabel('y data');

*# title*  
plt.title('Plot of y against x')

plt.show()



**Multiple lines on the same plot**

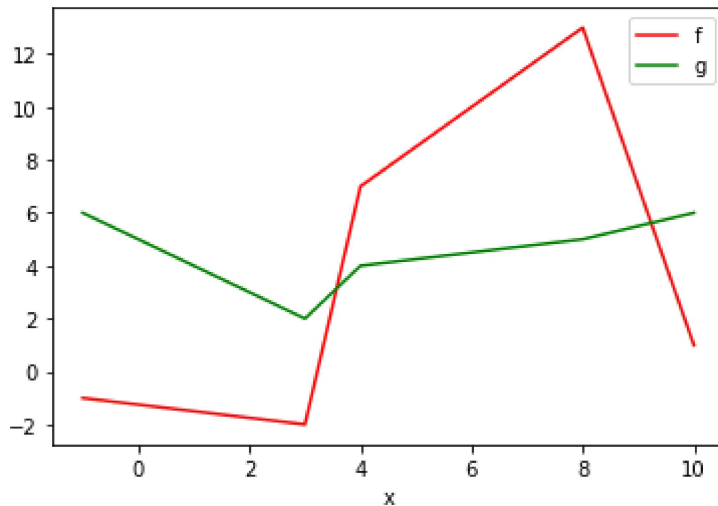
In [11]:

```
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');
plt.plot(x, y, '-g', label='g');

plt.xlabel('x');
plt.legend()

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
4. (Add axis labels)

In [4]:

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

In [5]:

```
# 1. Create a numpy array with the same number of positions as bars
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);

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

plt.show()
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-5-ce9191f99742> in <module>
      1 # 1. Create a numpy array with the same number of positions as bars
----> 2 x_pos = np.arange(len(groups))
      3
      4 # 2. Generate bar chart
      5 plt.bar(x_pos, num_students);

NameError: name 'np' is not defined
```

## 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 [16]:

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

We can visualise how `x` is distributed by determining a set of bins to hold different ranges of values.

In [6]:

```
edges = np.arange(0, 100, 10) # start, stop, step

plt.hist(x, bins=edges);

# Add Label
plt.xlabel('value')
plt.ylabel('frequency')

plt.show()
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-6-e2fc4fcb59a0> in <module>
----> 1 edges = np.arange(0, 100, 10) # start, stop, step
      2
      3 plt.hist(x, bins=edges);
      4
      5 # Add label
```

**NameError:** name 'np' is not defined

Bins are defined by their edge values:

If argument `bins` is:

- **an integer:** defines the number of equal-width bins in the range.
- **a sequence:** defines the bin edges, including the left edge of the first bin and the right edge of the last bin.

Bins *include* left-most value, but *exclude* right-most value.

Apart from 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 [7]:

```
n, edges, patches = plt.hist(x, 21);  
  
print(n)
```

-----  
**NameError** Traceback (most recent call last)

```
<ipython-input-7-2e7a7106e1d4> in <module>  
----> 1 n, edges, patches = plt.hist(x, 21);  
      2  
      3 print(n)
```

**NameError**: name 'plt' is not defined

## Subplots

Multiple plots can be included in the same figure using `subplot` .

`subplot(nrows, ncols, index)`

In [19]:

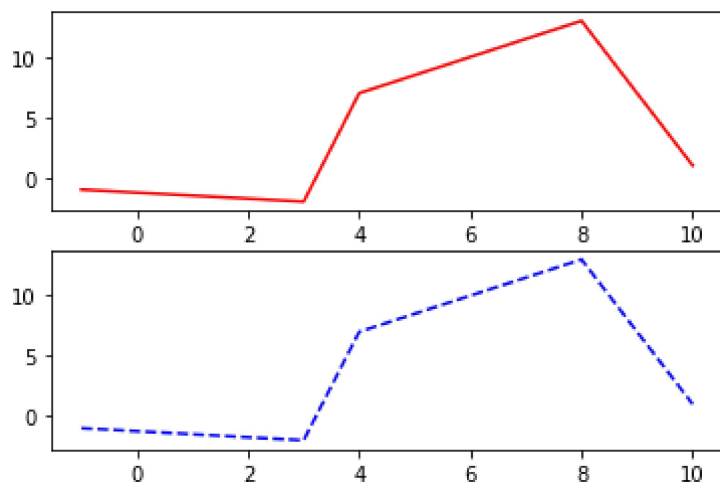
```
x = [-1, 3, 4, 8, 10]  
f = [-1, -2, 7, 13, 1]
```

In [20]:

```
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[20]:

[<matplotlib.lines.Line2D at 0x1130a56d0>]





In [21]:

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

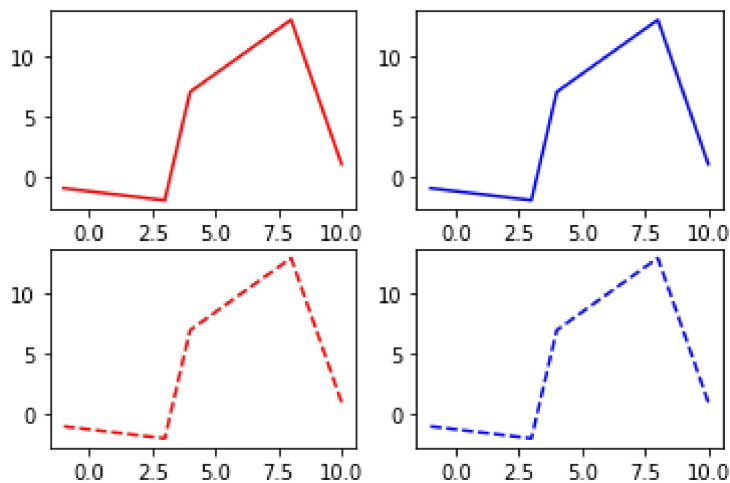
plt.subplot(222)    # 2 rows, 2 columns, index 2
plt.plot(x, f, 'b')

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[21]:

[<matplotlib.lines.Line2D at 0x11323ee10>]



## Saving plots

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

In [23]:

```
plt.savefig('img/four_plots.png');
```

<Figure size 432x288 with 0 Axes>

Must appear before `matplotlib.pyplot.show()` ( `plt.show()` ) in a Python programme.

## 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 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 [28]:

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

print(A)          # stored as numpy array

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

[59.796 54.123 74.452]
59.796
```

## 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

- Matplotlib 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>))

## In-class Demos

### Example 1:

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

Subject (ID)	Sex M/F	DOB dd/mm/yy	Height m	Weight kg	BP mmHg
JW-1	M	19/12/1995	1.82	92.4	119/76
JW-2	M	11/01/1996	1.77	80.9	114/73
JW-3	F	02/10/1995	1.68	69.7	124/79

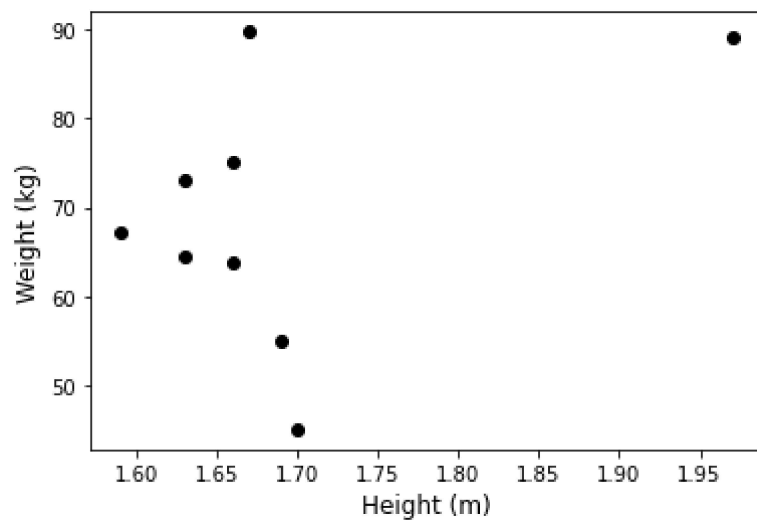
...

In [ ]:

In [46]:

Out[46]:

```
Text(0, 0.5, 'Weight (kg)')
```



### Example 2:

Import data from `sample_data/sample_student_data.txt` and plot a histogram of the height of female students.

In [ ]:

In [ ]:

In [ ]: