Data Analytics

Graphing

Visualisation/Graphing

It is easy to create visualisations in Python using a module named matplotlib.

We need our data in a list(s) to be able to create a graph.
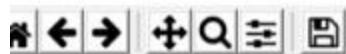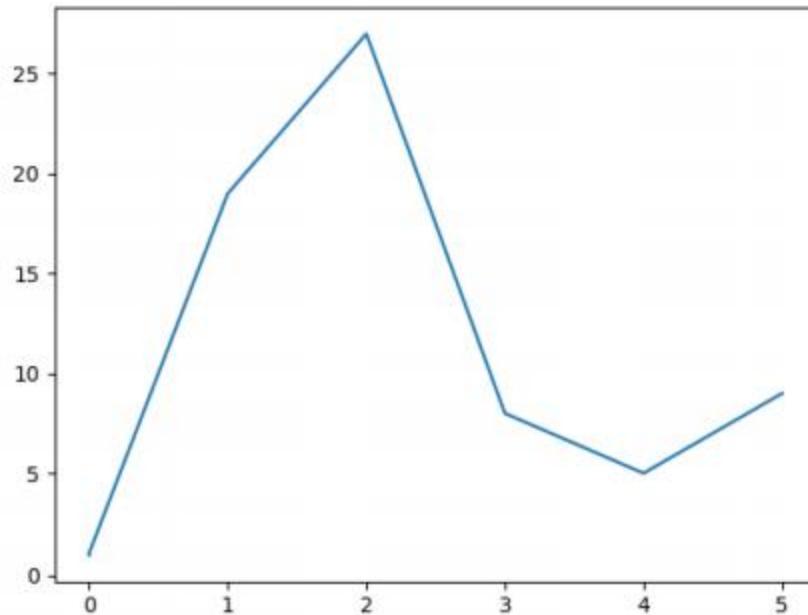
Plotting graphs

```
import matplotlib.pyplot as plt
myList = [1, 19, 27, 8, 5, 9]
plt.plot(myList)
plt.show()
```

The first line uses an interface pyplot, from the matplotlib module. It is mainly used for interactive plots and simple plot generation, which is what we want.

We import as plt so that we don't have to type matplotlib.pyplot every time we want to use it. We can just type plt instead.

The output looks like this

The buttons on the `matplotlib` output are useful.

From left to right:

- The home button resets the view to the original.
- The left and right arrows go back and forward between views, like on a web browser.
- The multi-directional arrows let you grab and move the graph (very useful while zoomed in).
- The magnifying glass, allows you to zoom in on a particular area of the graph, which is useful for detailed graphs like heart rate.
- The sliders allow you to adjust the subplot parameters, such as the location.
- The save icon allows you to save the plot image. This is useful for adding the results to a report or e-portfolio.

Labelling the graph

We can also add labels to the title and the axes of the graph. In the following piece of code, numBS represents the number of brothers and sisters of each student in a group:
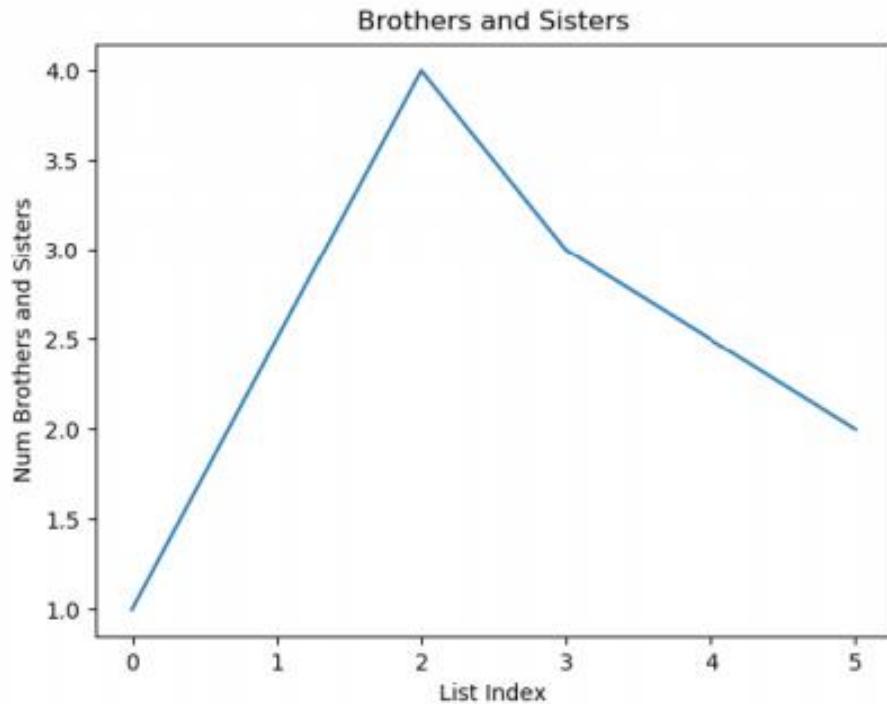
```python
import matplotlib.pyplot as plt
numBS = [1, 2.5, 4, 3, 2.5, 2]
plt.plot(numBS)
plt.title("Brothers and Sisters")
plt.xlabel("List Index")
plt.ylabel("Num Brothers and Sisters")
plt.show()
```

This is the result:

Graphing Options
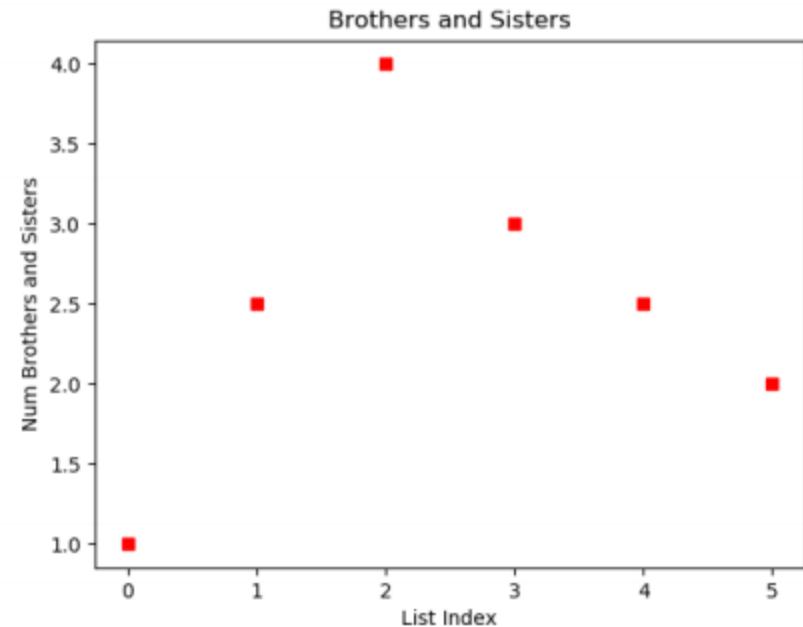
A line graph may not always be suitable. For example, a line between points suggests that they are linked in some way.

The following code plots points, with no lines.

```
plt.plot(numBS,"rs")
```

Resulting in:

The line-style typically consists of two or three characters, the first represents the colour and the second/third indicates the point-style. The default line-style is b–.

The following table gives some more examples of line-styles.

Arguments for line-style	Colour	Point-style
b–	Blue	————————
r––	Red	– – – – – – – – – –
y–.	Yellow	–.–.–.–.–.–.–
b:	Blue	··················
rs	Red	Squares, no line
bo	Blue	Circles, no line

Samples of arguments for line-styles using `matplotlib`

x and y axis
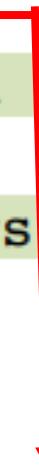
We can show labels on our axis.

Using the following table, we can graph the students and the number of brothers/sisters.

Note: there are two inputs so we need two lists.

names	ages	numBS
Joe Bloggs	17	1
Mary Murphy	18	2.5
Claire Whelan	16	4
Mike Fahey	18	3
Gillian Marks	17	2.5
Arya Quille	17	2

```python
import matplotlib.pyplot as plt
numBS = [1, 2.5, 4, 3, 2.5, 2]
names = ["Joe Bloggs", "Mary Murphy",
"Claire Whelan","Mike Fahey",
"Gillian Marks", "Arya Quille"]
plt.plot(names, numBS)
plt.title("Brothers and Sisters")
plt.xlabel("Student")
plt.ylabel("Num Brothers and Sisters")
plt.show()
```
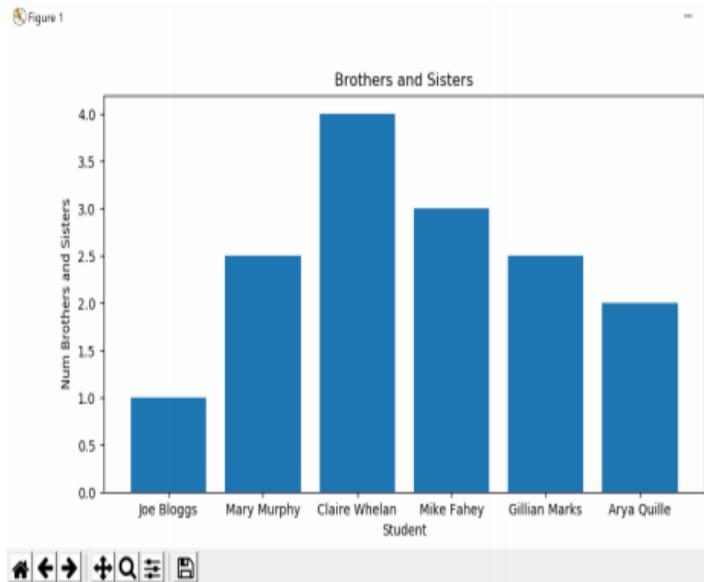
Note: The first argument represents the x-axis and the second argument represents the y-axis

Additional plot types - Bar Chart

The following code outputs a bar chart. Try it:
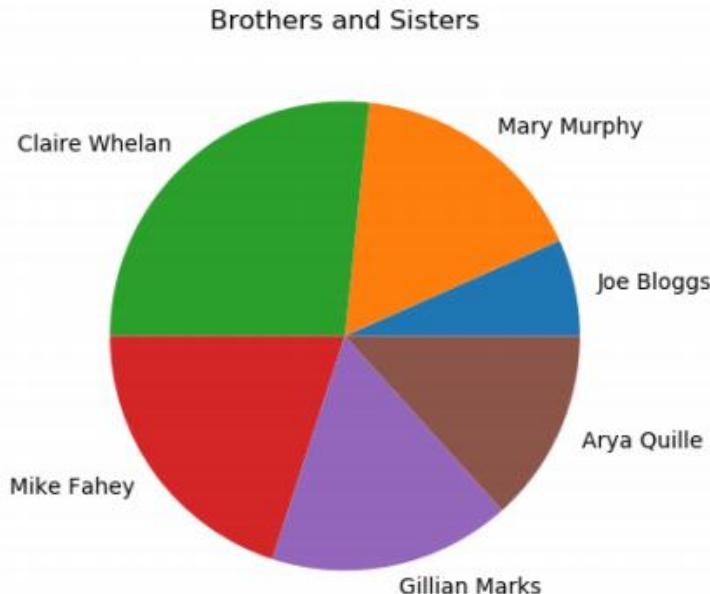
```python
import matplotlib.pyplot as plt
numBS = [1, 2.5, 4, 3, 2.5, 2]
names =  ["Joe Bloggs","Mary Murphy",
"Claire Whelan","Mike Fahey",
"Gillian Marks", "Arya Quille"]
plt.bar(names, numBS)
plt.title("Brothers and Sisters")
plt.xlabel("Student")
plt.ylabel("Num Brothers and Sisters")
plt.show()
```

Pie Chart

```
import matplotlib.pyplot as plt
numBS = [1, 2.5, 4, 3, 2.5, 2]
names = ["Joe Bloggs", "Mary Murphy",
"Claire Whelan", "Mike Fahey",
"Gillian Marks", "Arya Quille"]
plt.pie(numBS, labels=names)
plt.title("Brothers and Sisters")
plt.show()
```
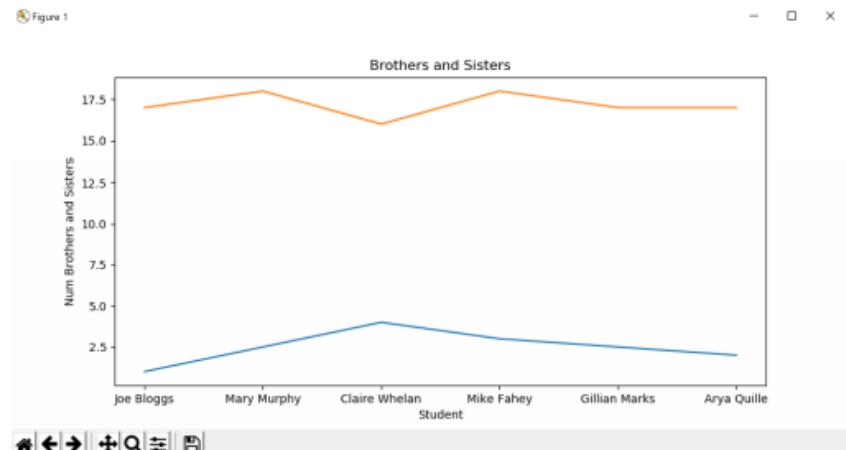
Here is the output:

Displaying two datasets on one graph

```python
import matplotlib.pyplot as plt
numBS = [1, 2.5, 4, 3, 2.5, 2]
ages = [17, 18, 16, 18, 17, 17]
names = ["Joe Bloggs", "Mary Murphy",
"Claire Whelan", "Mike Fahey",
"Gillian Marks", "Arya Quille"]
plt.plot(names, numBS)
plt.plot(ages)
plt.title("Brothers and Sisters")
plt.xlabel("Student")
plt.ylabel("Num Brothers and Sisters")
plt.show()
```

Here is the output:

Note that `matplotlib` used the same y-axis for both sets of data. In our example, ages have greater values than those for the number of brothers and sisters. If the scales were significantly different, what issues might arise? How would you address this?

If you had three or four plots on the same graph you might get confused about what each line represents. In the previous example, it is obvious from its values what each line represents but this may not always be the case.

A solution to this problem is to add a legend. A legend takes a list as its argument, where the order of items in the list corresponds to the order of the lists plotted on the y-axis.

This is the code for adding a legend:

```
plt.legend(["Num Brothers and Sisters",
"Age"])
```

We add the above line to the code as follows:

```python
import matplotlib.pyplot as plt
numBS = [1, 2.5, 4, 3, 2.5, 2]
ages = [17, 18, 16, 18, 17, 17]
names = ["Joe Bloggs", "Mary Murphy",
"Claire Whelan", "Mike Fahey",
"Gillian Marks", "Arya Quille"]
plt.plot(names, numBS)
plt.plot(ages)
plt.legend(["Num Brothers and Sisters",
"Age"])
plt.title("Brothers and Sisters")
plt.xlabel("Student")
plt.show()
```

The output now includes a legend: