# Lecture 4: Libraries and plotting

## Overview

This lecture will give an overview of some of the libraries that are commonly used in Python, particularly those related to numerical computations and plotting. We will finish the lecture by showing some common examples of how to produce plots of your data. But first, let's have a recap of what we learnt in the last lecture:

## Recap from lecture 3

### Lists

A list is simply a collection of data. We can add and remove items, sort the entire list, get maximum and minimum values etc. Here's a review of some simple commands that can be used on a list:

In [1]:

```
my_list = [2, 5, 3, 5, 8, 454]
```

How do we add the value 12 to the end?

In [2]:

```
my_list.append(12)
```

And how do we add the value 4 at index 1?

In [3]:

```
my_list.insert(1,4)
```

And sort and finally print the list?

In [4]:

```
my_list.sort()
print(my_list)
```

```
[2, 3, 4, 5, 5, 8, 12, 454]
```

### Dictionaries

Now to recall the basics of **dictionaries**. Remember to use curly brackets to define a dictionary and remember that a dictionary consists of a set of (key, value) pairs.

Let's create a dictionary where the keys are strings representing material types and the values are lists of strength data readings:

```
In [5]:
```

```
material_data = {'concrete' : [30.4, 34.5, 36.5, 34.2],
                 'steel' : [220.4, 230.4, 212.3, 214.5, 220.3],
                 'timber' : [50.4, 54.6, 54.7, 52.3]}
```

How do we print out the list for 'timber'?

```
In [6]:
```

```
print(material_data['timber'])
```

```
[50.4, 54.6, 54.7, 52.3]
```

and loop over all the keys and values in the dictionary?

```
In [7]:
```

```
for key, val, in material_data.items():
    print(key)
    print(val)
```

```
concrete
[30.4, 34.5, 36.5, 34.2]
steel
[220.4, 230.4, 212.3, 214.5, 220.3]
timber
[50.4, 54.6, 54.7, 52.3]
```

## Libraries

There are many Python libraries at your disposal which can do a whole host of things ranging from plotting, statistics, machine learning, web server functionality and much more. In this course we will focus mainly on the numpy library and a plotting library called matplotlib. Numpy stands for 'Numerical Python' and for those that are interested, you can see all the functionality it offers here https://docs.scipy.org/doc/numpy/reference/routines.html (https://docs.scipy.org/doc/numpy/reference/routines.html)

But let's just look at some basic functionality of Numpy. First, any time you want to use Numpy you need to import it into your notebook like so:

```
In [8]:
```

```
import numpy
```

We could also do it like this, where we use a shortened name 'np' which makes it easier to type later on:

In [9]:

```
import numpy as np
```

## arange(start, stop, [step])

One function which is very useful for plotting in the numpy 'arange' function:

In [10]:

```
xvals = np.arange(0.0, 10.0, 0.1)  # Create a list of values from 0.0 to 10.0 in
 steps of 0.1
print(xvals)

xvals = np.arange(0.0, 10.0)       # The step size is optional
print(xvals)
```

```
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2
1.3  1.4
  1.5  1.6  1.7  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.7
2.8  2.9
  3.   3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.   4.1  4.2
4.3  4.4
  4.5  4.6  4.7  4.8  4.9  5.   5.1  5.2  5.3  5.4  5.5  5.6  5.7
5.8  5.9
  6.   6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9  7.   7.1  7.2
7.3  7.4
  7.5  7.6  7.7  7.8  7.9  8.   8.1  8.2  8.3  8.4  8.5  8.6  8.7
8.8  8.9
  9.   9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
```

Note the 'np.' which preceeds the function since this is a funciton of the numpy library.

To make it clear, the range() function we have used so far only works with integer values and we need to explicity convert it into a list. We use it more commonly for looping over some sort of index.

In [11]:

```
myrange = list(range(0, 10, 2))  # This is what we have used so far. Remember, r
ange() only works with integers
print(myrange)
```

```
[0, 2, 4, 6, 8]
```

## linspace(start, stop, nvals)

If you want to specify the number of values rather than the increment, use linspace() like so:

In [12]:

```
myvals = np.linspace(0.0, 10.0, 15)
print(myvals)
```

```
[  0.           0.71428571   1.42857143   2.14285714   2.85714286
   3.57142857   4.28571429   5.           5.71428571   6.42857143
   7.14285714   7.85714286   8.57142857   9.28571429  10.        ]
```

## Other common numpy functions

There are many mathematical functions which are useful, including trigonometric functions, exponential functions and many more. See a full list here https://docs.scipy.org/doc/numpy/reference/routines.math.html#handling-complex-numbers (https://docs.scipy.org/doc/numpy/reference/routines.math.html#handling-complex-numbers) E.g.

## $\pi$

In [13]:

```
precise_pi = np.pi
print(precise_pi)
```

```
3.141592653589793
```

### Trig functions

In [14]:

```
theta = np.pi * 0.5  # Pi / 2 (i.e. 90 degrees)

c = np.cos(theta)
s = np.sin(theta)
t = np.tan(theta)

print(c,s,t)
```

```
6.12323399574e-17 1.0 1.63312393532e+16
```

### Complex numbers

In [15]:

```
a = 5 + 5j   # No need for numpy library just yet
```

In [16]:

```
myangle = np.angle(a)  # What is the angle this complex number forms
print(myangle)

myangle_degrees = np.rad2deg(myangle)  # And we can convert radians to degrees
print(myangle_degrees)

print(np.real(a))
```

```
0.785398163397
45.0
5.0
```

## Working with lists and numpy

All the previous functions were shown for a single number, but the real power of these functions are that they can operator on lists. For example:

In [17]:

```
pi = np.pi
xvals = np.linspace(0.0, 2.0 * pi, 10)  # Create list of xvalues between [0,2pi)
print(xvals)

sin_vals = np.sin(xvals)  # Sin values for the above list
print(sin_vals)
```

```
[ 0.          0.6981317   1.3962634   2.0943951   2.7925268   3.4906
585
  4.1887902   4.88692191  5.58505361  6.28318531]
[  0.00000000e+00   6.42787610e-01   9.84807753e-01   8.66025404e-01
   3.42020143e-01  -3.42020143e-01  -8.66025404e-01  -9.84807753e-01
  -6.42787610e-01  -2.44929360e-16]
```

Or maybe you even created your own function and you want to compute lists of x and y values:

In [18]:

```
def myfunction(x):
    return np.log(x) * 2.0

myxvals = np.arange(0.1, 10.0, 0.5)
myyvals = myfunction(myxvals)
```

In [19]:

```
print(myxvals)
print(myyvals)
```

```
[ 0.1  0.6  1.1  1.6  2.1  2.6  3.1  3.6  4.1  4.6  5.1  5.6  6.1
6.6  7.1
  7.6  8.1  8.6  9.1  9.6]
[-4.60517019 -1.02165125  0.19062036  0.94000726  1.48387469  1.9110
2289
  2.26280422  2.56186769  2.82197395  3.05211261  3.25848108  3.4455
332
  3.61657754  3.7741393   3.92018957  4.05629649  4.18372812  4.3035
2441
  4.41654883  4.5235262 ]
```

## Basic statistics

We can now do very basic statistics using the numpy library like so:

In [20]:

```
myreadings = [220.0, 221.4, 225.0, 225.4, 228.9, 221.8, 223.9]

my_mean = np.mean(myreadings)
my_std = np.std(myreadings)
my_90_percentile = np.percentile(myreadings, 90)

print('My steel specimens have a mean of {} std of {} and 90th percentile of {}'
.format(my_mean, my_std, my_90_percentile))
```

```
My steel specimens have a mean of 223.77142857142857 std of 2.782965
6769683084 and 90th percentile of 226.8
```

# Plotting

Now we have most of the knowledge to start creating plots in Python. But we first need to import another library before we use any of the plotting functionality:

In [21]:

```
import matplotlib.pyplot
```

or, as I prefer

In [22]:

```
import matplotlib.pyplot as ppt
```

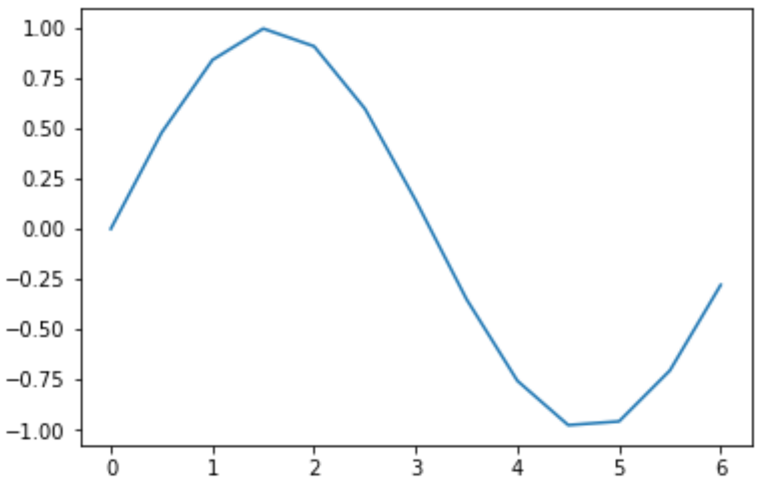
which means we now only need to type 'ppt' before any of the plotting functions.

## Line plots

Let's plot $\sin(x)$ between 0 and $2\pi$

```
In [23]:
```

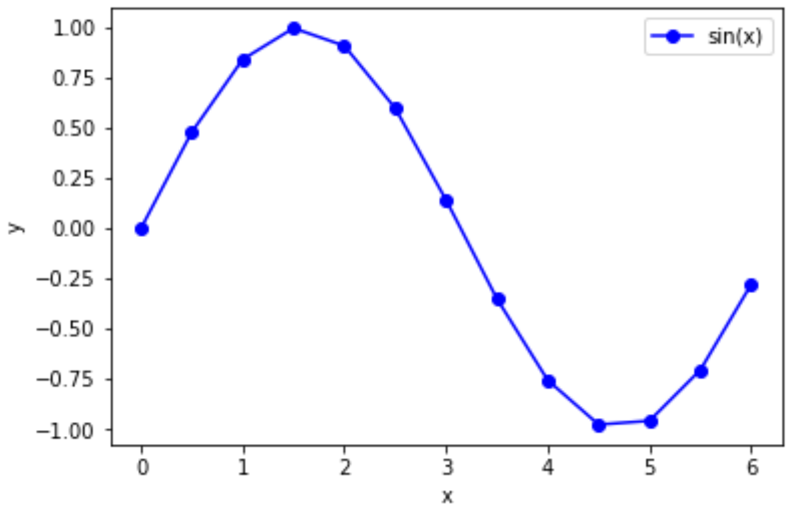
```
xvals = np.arange(0.0, 2.0 * pi, 0.5)
yvals = np.sin(xvals)

ppt.plot(xvals, yvals)
ppt.show()
```



We can make this look a bit nicer like so:

```
In [24]:
```

```
ppt.plot(xvals, yvals, 'bo-', label='sin(x)') # bo- stands for blue, dots and a
 line connecting the dots
ppt.legend(loc='upper right')
ppt.xlabel('x')
ppt.ylabel('y')
ppt.show()
```



Have a look here for all the different formats you can use
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html
(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html)

## Scatter plots

Here we might have two lists which define names and values for those names. E.g.

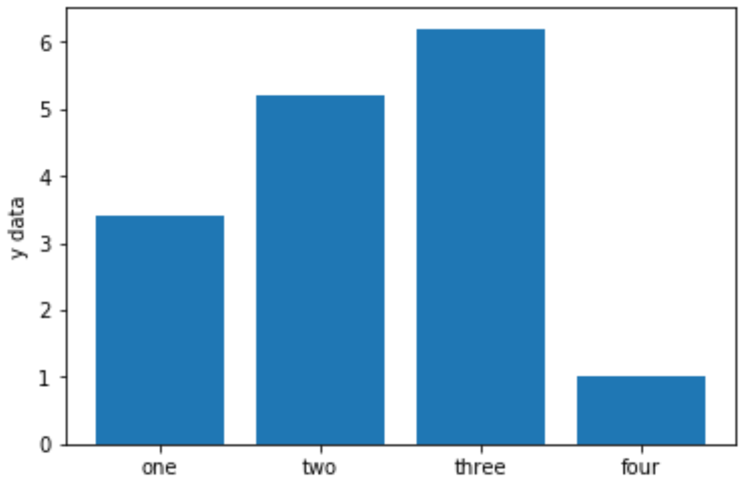
In [25]:

```
data_names = ['one', 'two', 'three', 'four']
data = [3.4, 5.2, 6.2, 1.0]
```

A scatter plot is created like so:

In [26]:

```
x = range(len(data))      # First create a list for the xaxis which is basically
 [0,1,2,3]

ppt.bar(x, data, align='center')
ppt.xticks(x, data_names)
ppt.ylabel('y data')
ppt.show()
```



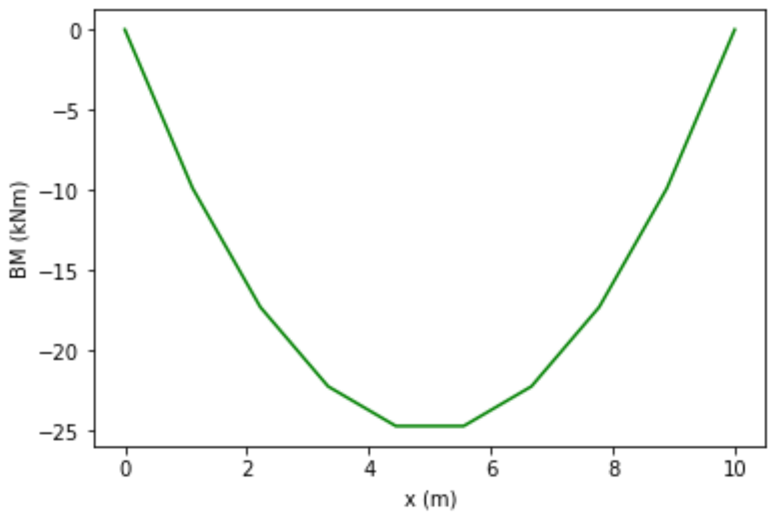
## Class exercise (10mins)

Write some Python code that allows a user to input both the u.d.l. of a simply supported beam and its length. Plot the bending moment diagram for the beam.

```
In [27]:
```

```
w = float(input('What is your u.d.l. in kN/m? '))
L = float(input('What is beam length in m? '))

def bm(x,w,L):
    return -w * L * x * 0.5 + w * x**2 * 0.5

npoints = 10
xvals = np.linspace(0.0, L, npoints)
yvals = bm(xvals, w, L)

ppt.plot(xvals, yvals, 'g-')
ppt.xlabel('x (m)')
ppt.ylabel('BM (kNm)')
ppt.show()
```

```
What is your u.d.l. in kN/m? 2
What is beam length in m? 10
```



## Plotting discontinuous functions (tricky)

Often we want to plot functions which gradients that change suddely at points. Here we need to be careful. For example, try plotting the function of the bending moment of a simply supported beam with a point load at midspan.

In [30]:

```
L = float(input('Please enter length of beam (m) --> '))
P = float(input('Please enter load at midspan (kN) --> '))

def bendingMoment(x, L, P):
    if x <= L / 2:
        return -P * x / 2
    else:
        return -(L - x) * P / 2

xvals = np.arange(0, L, 0.01)
yvals = bendingMoment(xvals, L, P)  # This doesn't work!!!
```

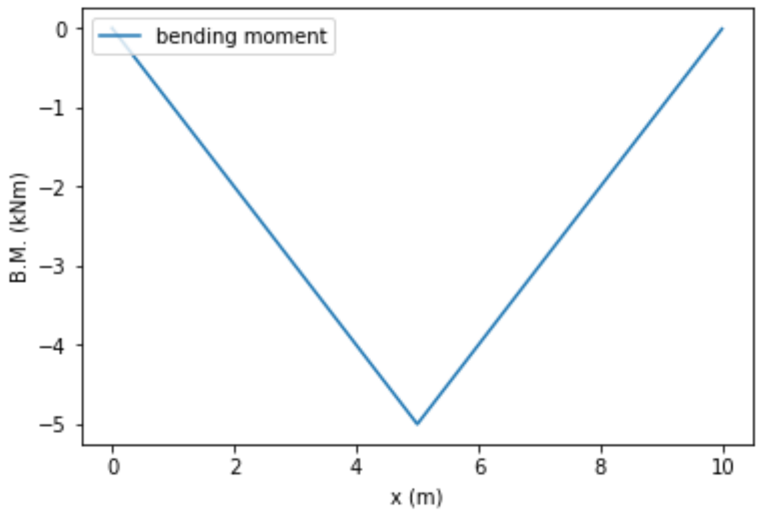
```
Please enter length of beam (m) --> 10
Please enter load at midspan (kN) --> 2

---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-30-52698ed478be> in <module>()
      9
     10 xvals = np.arange(0, L, 0.01)
---> 11 yvals = bendingMoment(xvals, L, P)  # This doesn't work!!!

<ipython-input-30-52698ed478be> in bendingMoment(x, L, P)
      3
      4 def bendingMoment(x, L, P):
----> 5     if x <= L / 2:
      6         return -P * x / 2
      7     else:

ValueError: The truth value of an array with more than one element i
s ambiguous. Use a.any() or a.all()
```

In [31]:

```
bm_vec = np.vectorize(bendingMoment)   # You need to do this for anything in gen
eral with if statements in the function
yvals = bm_vec(xvals, L, P)
```

In [32]:

```
ppt.plot(xvals, yvals, label='bending moment')
ppt.legend(loc='upper left')
ppt.xlabel('x (m)')
ppt.ylabel('B.M. (kNm)')
ppt.show()
```



Voila!