Scientific Computing in Python 1 – Basic Maths

So we can do basic arithmetic in python just like we did in R.

```
2 + 3


7 * 9


3*4 + 5 /91 +1082 * 6 – 72
```

Be careful of integer divisions in python

```
1/2
```

If this returns 0.5 python is set up fine, if it returns 0 then you will have to either run

```
from __future__ import division
```

or the numpy package can force regular devision

In python exponents are done with **

```
4 **2
```

And we can do algebra with things as you know can assign variables:

```
a = 2

b = 6

2*a + 4*b
```

Mathematics and scientific computing are already extremely well supported in python. Many of things we have done in the mathematics course are possible to do in python. We could write our own functions for them or we could just use them from packages that are already created.

There are two primary packages that are used in python for science things.

```
import numpy as np

import scipy as sp
```

You should be familiar with the import function if not the packages themselves by now. These two packages contain an array of incredibly powerful tools that have been added to them by numerous users over many years. Last week scipy left beta and is officially version 1.0 after 16 years of development.

There are a few more packages that are essential.

```
import matplotlib.pyplot as plt

import pandas as pd

import csv

import math

import random
```

Matplotlib is the package used for plotting, hopefully you got introduced to these concepts in the last python week. Pandas is a package that is used for handling data frames like we do in R. CSV is specifically for dealing with csv files. Math is a package that includes a lot of basic math functions that aren't in python like square root etc.

While there are many functions in these that we will use let's start by writing some of our own functions for some of the basic maths things we do.

```
def times(number1, number2):
    return(number1 * number2)
```

You need to indent functions. Look at that, we made a function that multiplies two numbers. Try it out on any numbers. Of course this is incredibly redundant but just an example of the kind of mathematical functions you can create.

Lets make a bit more complicated but cooler function that is going to use some of the maths we learnt in the first week to estimate pi. This function requires some parts of other functions including numpy, math and random (a package to generate random numbers). Make sure you have imported all these packages.

```
#create a function which calls the number of times you want to sample the circle

#and the radius of the circle.

def myPi(sampling, radius):

    #this counter will track the area of the circle based on sampled areas.

    counter = 0
```

```
    for i in range(sampling):
        #select a random point in a square which contains our circle
        point = np.array([random.uniform(0,(2*radius)),random.uniform(0,(2*radius))])
        #the center point is the radius distance from each side.
        center = np.array([radius,radius])
        #working out the distance from the random point to the sample
        sides = point - center
        #getting rid of negatives
        dist = math.sqrt(sides[0]**2+sides[1]**2)
        #if the distance from the point to the center is less than the radius it
        #is inside the circle and we add it to our area counter.
        if dist < radius :
            counter = counter +1
    prop_in = (counter/sampling)
    calc_pi = prop_in/(radius**2) #since the num in the circle is
    #approx the area for large samples, divide area by radius squared to get pi
    print("area of circle with radius", radius, "is approx,", prop_in)
    print("pi is approx",calc_pi)
    print("actual pi is", math.pi)



myPi(1000,0.5)
```

This function is in essence a Bayesian approach to estimating pi:

```
myPi(1000000,0.5)
```

That might take a second to run but it gives a far better estimate, the more you sample the more accurate our approximation is. This is a Bayesian approach.

This is the creation of a function that does some maths for us. We tell it the size of the circle and how many times to sample and it has used that to estimate pi.

We will be doing this same approach to build many of our models, starting with defining our functions that are the model and telling it what parameters we will put into the model, this allows us to vary parameters easily, or even do parameter sweeps.

The rest of this practical and today will be spent on constructing several functions that perform mathematical tasks. The functions will exist in various modules but for the sake of practice in writing them I want you to do it yourself.

You will need to use the demonstrators, code from previous weeks, and definitely the internet (at least to find out whether components have been done for you in modules).

Function 1: create a function that estimate the combined perimeter of all the sides of a cube when you are provided with the volume of said cube.

```
perimeter(volume)
```

You will need to work back from the formula for a volume of a cube to work out the length of a side and then scale that up for perimeter.

Function 2: Create a function that determines the height of trees when you have the distance from it and the angle between the ground and the top.

```
height(distance,angle)
```

Hint: the maths module has various functions for trigonometry, remember your trig from week 1.

Challenge Problem

Function 3: Create a function that computes the given exponent of a number when you put

```
function(number,exponent)
```

This is actually much harder than it appears. Remember roots can be exponents e.g. square root is to the power of ½, cube root is 1/3 . Calculating square root by hand is extremely difficult let alone programming it in. Only tackle this if you are feeling brave, otherwise just do the regular expression for square when it is a root. Check out https://www.wikihow.com/Calculate-a-Square-Root-by-Hand for help.

Hint you may need an if statement to determine if it is a power or a root.