1 Installing Python

Read the installation instruction on
https://www.python.org/downloads/

2 Virtualenv

Using Virtualenv, we can make a nice workspace with full admin rights and not have to worry about the system.

You can read more about it
https://virtualenv.pypa.io/en/stable/#

3 Jupyter Notebooks

You can read about Jupyter notebooks here
http://jupyter.org/

You can also try out the notebook in your browser before installing it.
Anaconda (https://www.continuum.io/downloads) will automatically install Jupyter and Python on your machine.

4 Introduction

These exercises are intended to introduce you to the computer programming language Python, as well as give programming tips that will help when working through some of the later tutorials. The concepts that will be covered include * The use of functions, * translation of algorithms to code, * the formatting of code, * tips for methodically writing code, * advice on how to check code.

One important concept in Python not addressed in this tutorial is data types. We will discuss data types in the linear algebra tutorial.
By the end of this tutorial you should have checked that your computer account is working and you will have practiced using the Jupyter notebooks and Python. Along the way, questions will be asked. You should provide answers to these questions in your lab book. Marks for these questions will be awarded based on the work presented in your lab book.
Remember this is a mathematics course. The aim is not for you to prove you can produce some sort of output, you must also show that you understand the output. When you have results in your lab book, it is a good idea to include a small discussion about your output. Are the results what you expected? Why or why not? How did you verify your results? What might be some of the

reasons why the results are incorrect? During the course we will introduce you to terminology and techniques to help you answer these questions. If you just print the output, you will not be awarded full marks.

4.1 Login

The lab computers run Linux. Once in the start menu, just search for Jupyter and start the Jupyter notebook.
If you're using your own machine, then you should be familiar with how to start Jupyter Notebook (but if you aren't, please ask for help).

5 Basic Python

5.1 Python as a calculator

Python can be used as a simple calculator. For example addition, multiplication, exponentiation and others.

```
1+1
```

```
154*9794
```

```
2**3
```

```
2^3
```

The above is the binary XOR operator. It compares the binary digits of the inputs and produces an output (look this up and try to understand it, ask your tutor in the lab to see if you get it).

```
1.0/5.0
```

5.2 Conditional Statements

Often we only need to do something once certain criteria are met. For these situations python has the **if** statements.

```
x=10
if x<30:
    print("x is less than 30")
```

Notice that the line after the **if** statement is indented. Python uses white space to organize code, so you must be careful to use whitespace correctly (often when my code gives an error, it's because of whitespace). You can use tabs or spaces for whitespace, but you must be consistent or problems will occur. Fortunately,

jupyter automatically adds tabs after conditional statements, making life easier for us.

You can also check multiple conditions using **elif** and **else**

```
if x<0:
    print("x is less than 0")
elif x<50:
    print("x is less than 50 and bigger than 0")
else:
    print("Ahhhhhhh")
```

5.3 Loops

Other things you will want/need to do, is to iterate or repeat sections of code. You can do this using a **for** or **while** loop

```
for i in range(10):
    print(i)
```

```
i=0
while i<10:
    print(i)
    i+=1
```

Notice that both cells produce the same output. As you'll come to learn over the course, there's more than one way to do something.

In the above example we used the **range** function (this is a built-in function in python). To learn more about this function (and of course others), we can always use the **help** command which will print a description of the function. You can also always look things up on google.

```
help(range)
```

From the above help section, we can see that the **range** function builds a list given input values of **start**, **stop**, and **step**. Let's try exploring this function a little more.

```
print(range(5))
```

```
print(list(range(5)))
```

```
A = list(range(15))
```

```
print(A)
```

Notice that the upper bound in range, is non-inclusive

```
B=list(range(-5,13,3))
B
```

You can specify sections of a list using slice notation, like so:

```
A[1] #why is this not 0?
```

```
A[3:6]
```

```
A[:7]
```

```
A[-4:]
```

To find the length of a list, use the **len** function

```
len(B)
```

5.4 Functions

To define a function in python that we can reuse, the **def** keyword is used

```
def f(y):
    return y**2

print(f(3), f(4), f(29372))
```

The **return** keyword causes the function to stop and give back a value to whatever called it. In this case, it squares whatever argument is given to it. It always a good idea to document your code and describe what it's doing and why. This allows other people (and yourself later on!!) to understand what the code is doing, and to modify or fix it. To do this add comments to the code by writing a #, then everything till the end of the line is ignored by Python.

As an example of how all this can tie together, here's one way you might calculate the exponential function e^x

```
def ex(x):
    """
    Calculate exponential from the taylor series
    """

    ans=0.0
    n=1.0
    term=1.0
    while ans+term != ans:        # != means "not equal to"
        ans = ans + term
        term = term*(x/n)
        n = n+1
    return ans
```

The comments between the two """ characters are documentation strings and they differ from standard comments in that they provide a way of associating

documentation with a Python function. For example, see what happens when you run the next cell.

```
help(ex)
```

<font, color="blue"> Lab book: Using this program, what is the approximation to e^1, e^{10} and e^{-10}. What are some techniques you could use to check the output?

Lab Book: The Taylor series expansion of the exponential is

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

When writing code to implement Taylor series expansion, we can't sum to ∞, explain what's been done in the above code to get around this problem.

5.5 Python Modules

To make Python more than a calculator while avoiding writing functions to do everything yourself, you use code that other people have written. To do this, you need import modules. Many modules come with Python, others need to be installed, and some you may even have to write yourself.

The first way to import something is with the **import** command.

```
import math
```

To access the functions in the module you have to prefix them with the module name

```
math.exp(1)
```

To avoid excessive typing you can also **import ... as**

```
import math as m
```

```
m.exp(1)
```

if you know exactly which function you want from the module, you can use **from ... import**

```
from math import exp
```

```
exp(1)
```

5.6 Installing python modules (PIP)

Installing modules in python is pretty simple with pip. You can read more info on
https://pip.pypa.io/en/stable/#

One of the most useful python modules (for scientific computing), is NumPy. You can read more about it (and find tutorial and all sorts of resources) here
http://www.numpy.org/
https://docs.scipy.org/doc/

We can use NumPy to define (multidimensional) arrays. This is normally how we do linear algebra in Python with Matrices and Vectors.
For example,

```
import numpy as np

#define a new 3x3 matrix
A = np.array([[1,2,3],[4,5,6],[7,8,9]])

print(A)

#lets define a vector
x = np.array([1,2,3])
```

Lets try and find the product Ax

```
#find the matrix vector product of A and x
b = A*x

print(b)
```

What this has done is multiplied the columns of A with the components of x. To find the matrix vector product Ax we can use NumPy

```
b = np.dot(A,x)

print(b)
```

5.7 The Help Command

Python Modules will usually come with documentation describing what functions are available, what they do, and how to use them. To access this, we use the **help** command (as done above).
To find out what is in a module, simply do **help(modulename)**, for example

```
import math
help(math)
```

The above lists all the function available in the math module. To find out how to use a specific function, do **help(functionname)**, for example

```
help(math.log)
```

Some functions will have arguments in brackets, like so

```
log(x[,base])
```

This means that the *base* argument is optional. If a function lists an argument with and assigned value, it also means it is optional, and that if you do not specify the argument that it will use a default value instead.

If you don't know where to look for a function or what it is called, it might be best just to search the internet or documentation for things like NumPy and MatPlotLib.

6 Example Problems

In the example below, there is a question asked and answered. Try to understand what's being done. Look up any unfamiliar parts of the code. Try to explain what you think is being done to someone in the lab.

6.0.0.1 Multiples of 5

If we list all the natural numbers below 15 that are multiples of 5, we get 5 and 10. The sum of these multiples is 15.

Find the sum of all the multiples of 5 below 1000.

```
def list_function(x):
    my_list=[]
    for i in range(200):
        my_list.append(x*i)
    return my_list

list_function(5)

sum(list_function(5))
```

What if I wanted the sum of multiples of 5 or 7?

What about multiples of 5 and 7?

```
multiples=[]
for i in range(5):
    for j in range(4):
        multiples.append((5**i)*(7**j))
print(multiples)


sum_list=[]
for i in multiples:
    if i<1000:
        sum_list.append(i)
print(sum_list)

print(sum(sum_list))
```

6.0.0.2 Find out if $10,001$ is a prime number.

```
5//2
```

```
5%2
```

```
def isprime(n):
    for i in range(2,n):
        if n%i==0:
            return False
            break
    else:
        return True


isprime(10001)
```

7 Example Application.

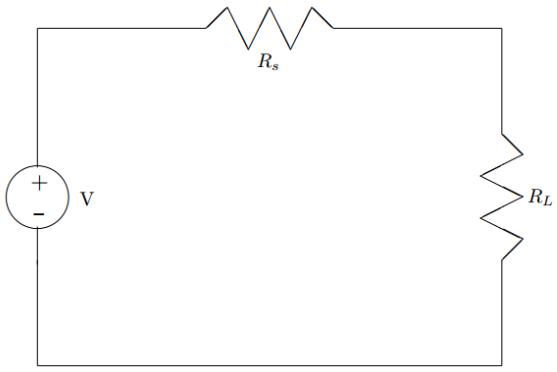

Figure 1: A voltage source with a voltage V and an internal resistance R_S supplying a load of resistance R_L.

Figure 1: Circuit Diagram

7.1 Problem Description

Figure 1 shows a voltage source $V = 120$V with and internal resistance R_s of 50Ω supplying a load of resisteance R_L. Find the value of load resistance R_L that will result in the maximum possible power being supplied by the source load. How much power will be supplied in this case?
Also, plot the power supplied to the load as a function of the load resistance R_L.

7.2 Mathematical Description

In this program we need to vary the load resistance R_L and compute the power supplied to the load at each value of R_L. The power supplied to the load resistance is given by the equation

$$P_L = I^2 R_L \tag{1}$$

where I is the current supplied to the load. The current supplied to the load can be calculated by Ohm's law:

$$I = \frac{V}{R_S + R_L}. \tag{2}$$

7.3 Pseudocode

It is important to always break up our problems down into smaller sections. It is easier to understand, debug and check smaller sectiond of code. This can be done using pseudocode:

Algorithm 1. *Power Supply* 1. Build a list of possible values for the load resistance R_L. The list will vary R_L from 1Ω to 100Ω in 1Ω steps. 2. Calculate the current for each value. 3. Calculate the power supplied to the load for each value of R_L. 4. Plot the power supplied to the load for each value of R_L and determine the value of load resistance resulting in the maximum power.

7.4 Code the function

Lets first write a function that calculated the current using Ohm's law.

```
def current(RL,V,RS):
    """
    Calculate the current given the voltage source with voltage V
    and internal resistance RS supplying a load of resistance RL.

    Input: RL = load of resistance (list)
         : V = Voltage (floating point number)
         : RS = internal resistance (floating point number)

    Output: current (list)
    """

    n= len(RL)
    I = n*[0.0] #initialize I to be a list same size as RL
    for j in range(n):
        I[j]=V/(RL[j]+RS)
    return I
```

NOTE: The use of the **for** statement in examples like this is very inefficient and is in general considered to be poor programming practice. When we talk about data structures we will look at more efficient ways to implement such codes.

We can now test the fucntion as follows

```
RL = list(range(1,6))
current(RL, 1.0, 0.0)
```

<font, color='blue'> Lab Book: Is the above output correct? Why or why not? If it is not correct, what modifications must be made to the code to correct it?

Lets now calculate the power supplied to the load for each value of R_L by implementing the first equation.

```
def power(I, RL):
    """
    Calculate the power given the load of resistance RL
    and I

    Input: RL = load of resistance (list)
         : I = current (list)

    Output: power (list)
    """

    n=len(I)
    P=n*[0.0] #initialize P to be a list same size as I
    for j in range(n):
        P[j]=I[j]**2*RL[j]
    return P
```

<font, color='blue'> Lab Book: Is the above code correct? Why or why not? If it is not correct, what modifications must be made to the code to correct it?

Check the above function using

```
RL = list(range(1, 6))
I = current(RL, 1.0, 0.0)
power(I, RL)
```

Next we define a function to plot our results.

```
from matplotlib.pyplot import plot, show, title, xlabel, ylabel, grid

def power_plot(RL,P):
    """
    Plot the power P versus the load of resistance RL.

    Input: RL = load of resistance (list)
         : P = power (list)
```

```
    Output: none
    """

    plot(RL, P)
    title('Plot of power versus load resistance')
    xlabel('Load resistance (ohms)')
    ylabel('Power (watts)')
    grid('on')
    show()
```

7.5 Putting the code together

We can calculate the power.

Set the internal resistance to 50Ω.

```
RS = 50.0
```

Set voltage source to 120V

```
V = 120.0
```

Creat a list of possible values for the load resistance

```
RL = list(range(1,101))
```

Calculate the current flow for each load resistance

```
I = current(RL, V, RS)
```

Calculate the power supplied to the load

```
P = power(I,RL)
```

Plot the power versus the load resistance

```
power_plot(RL,P)
```

The resulting plot suggests that the maximum power is supplied to the load when the load's resistance is 50Ω. The power supplied to the load at this resistance is 72 Watts.

8 Exercises

If a stationary ball is released at height h_0 above the surface of the Earth with a vertical velocity v_0, the position and velocity of the ball as a function of time is given by the kinematic equations of motion (for linear acceleration)

$$h(t) = \frac{1}{2}gt^2 + v_0t + h_0 v(t) = gt + v_0 \tag{3}$$

where g is the acceleration due to gravity ($-9.81m/s^2$), h is the height above the surface of the Earth (assuming no air friction), and v is the vertical component of the velocity. The aim of this exercise is to write a Python program that takes the initial height of the ball in metres and velocity of the ball in metres per second, and plots the velocity versus height for $0 \leq t \leq 20$.

1. Write a pseudocode (algorithm) that breaks the problem down into smaller subproblems. Similar to what was done in the **Pseudocode** section.
2. Implement each subproblem as a function. I recommend you copy the plotting function from before and make necessary modifications.
   Check that each function is working. Talk about the steps you took to verify the function is working.
3. Produce a plot of the velocity versus height if $v_0 = 100m/s$ and $h_0 = 400m$.

9 Examples of Good Programming Practice

The following lists some examples of what are considered to be good programing practices. They are very useful habits to pick-up and will save you a lot of time and frustration, especially when we look at more complicated programs.

- Use functions. This helps to break the problem down into smaller sections. It is easier to understand and debug a small section of code. It also allows you to reuse parts of your code. If each function is narrowly focused and responsible for one particular aspect of your code, you are less likely to get a cascade of errors, which is very difficult to debug.
- Rerun your code often. When learning a new language you may want to run your code every time you make a change (I do). That way you can easily track down the source of any bugs. Even when you become familiar with the language, it pays to compile and run your code regularly. It helps to narrow down the parts of the code you have to look at when you get syntax errors.
- Use the mathematics. Even if you don't know what the exact solution is, the mathematics will often tell you about some of the properties of the solution. For example, if you get a negative answer when calculating the standard deviation something has gone wrong.
- Write a pseudocode first. This allows you to focus on the logic of the problem. It also helps you structure the flow of information. That is, what do you need to pass into a function and what information do you need to get out it.
- Format your code. You should use spaces, comments, meaningful variable names etc. to make the code easier for a human to read. It will save you a lot of time if you want to ask someone else for help. It also forces you to better structure the code.