Python basics

This handout provides examples of some basic building blocks you might need in the SCO module. Here we will not reproduce elements that will be covered in later sessions. Please work you way through the hand out, copy the code into your python terminal and experiment with the different features.
An excellent and more exhaustive discussion of scientific computing in python can be found here:
http://www.physics.nyu.edu/pine/Teaching.html

Variables

Variables are loosely typed. The *type* of a variable is defined by how it is set, for example:

```
myString = 'hello world'     # string type
myInteger = 1                # integer type
myFloat = 1.0                # floating point type

x = myString + myInteger     # will fail and report an error.

x = myInteger/myFloat

x = myInteger/2              # don't let integer divisions ruin your day!
```

If you want all division to be floating point (this will slow your code down), you can have that by starting your code with

```
from __future__ import division
```

Lists

It is also possible to declare a list:

```
l1 = [0, 1, 2, 3, 4, 5]  # list type
l2 = [10, 11, 12, 13, 14, 15]


l = l1 + l2              # concatenates the lists
```

Arrays

Obviously you might want to do something more useful that simply concatenating lists. For that you can turn to some imported libraries. Most of the functionality that you are used to in matlab can be imported from one of numpy or matplotlib. These are object oriented in design, so when you run import, you obtain an instance of the class.

```
import numpy as np                  # creates an instance named np
import matplotlib.pyplot as plt     # creates an instanced named plt


a1 = np.array([0, 1, 2, 3, 4, 5])   # try also np.zeros() and np.empty()
a2 = np.array([10, 11, 12, 13, 14, 15])

a = a1 + a2
a = a1 * a2

np.dot(a1, a2)
```

It is extremely useful to know that you can type the name of an imported class instance into your console, followed by a dot (e.g. np.), then pressing tab will show you a list of all available functions within that class.

Matrices

Matrices are declared as multi-dimensional arrays

```
M = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
v = np.array([10, 11, 12])

M*v               # this is an elementwise multiply

np.dot(M,v)       # this is a matrix dot product.

np.zeros([3,2]) # create empty matrices to be written
x = np.empty([2,3])

x.shape
x.size
```

Dictionaries

You can associate names with variables in a list.

```
dictionary = {"One":1, "Two":2, "Three":4}

dictionary["One"]
```

Loops

A typical for loop is written:

```
N = 20

for i in range(0, N, 2):

    print 'i = %d' % ( i ) # notice that the indentation defines
                           # which lines are contained in the loop


l = ['the', 'quick', 'brown', 'fox']

for str in l:

    print 'string = %s' % ( str)
```

If you want to have an integer counter in a loop over floats, then you can enumerate your for loop.

```
import math

a = np.array([math.pi, math.pi/2, math.pi/4, math.pi/8])
b = np.empty(len(a))

for idx, element in enumerate(a):
    print element
    b[idx] = element

b == a
```

Alternatively you can use a while loop.

```
x = 0
y = 7
while x <= 10 and y > 4:
    print 'x = %d,y = %d' % (x, y)
    x += 2        # equivalent to x = x + 2.
    y -= 1        # see also  /=, *=
```

Also experiment with break and continue:

```
x = 0
while True:

    print x
    x += 1
    if (x >= 12):
        break

x = 10
while (x >= 0):
    x -= 1
    if (x == 5):
        continue
    print x
```

Conditional statements

There is not much to know about if statements, they look like this:

```
a = 4
b = 'hello'
c = 'world'

if (a == 2):
    print 'No'
elif (b != c):
    print 'Yes'
else:
    print 'Something else'
```

The only thing to note is that 'else if' is NOT a python statement.

Functions

Some inbuilt methods will require you to pass functions as parameters, for example curve fitting, differential equation solving, button callback functions etc. More often it is useful to create a method simply to abstract an individual idea. Good software is more than a list of instructions, it should be a readable text that can be handed on to other people to use, either as whole, or components of it. It is very common to return to a source years after you wrote it and be utterly baffled by it! Comments, whitespace, sensible variables names all help in making code readable, but most of all is abstracting individual concepts into functions.
The simple example below aims to illustrate how the readability and flow of a loop can be enhanced by using functions.

```
p = 10.0
v = 0.0

DT = 1.0
G = 1.0
FREQ = 2.0
OMEGA = 2 * math.pi * FREQ

def reflectVelocity():

    v = wall_v - v

def updateParticlePosition():

    v += DT * G
    r += DT * v

    if (r <= wall_v):
```

```
        reflectVelocity()


t = 0
while t < 1000.0:

    wall_v = sin(OMEGA*t)

    updateParticlePosition()

    t += DT
```

Program functions are also useful for encapsulating mathematical functions, for example:

```
def sinc(x):

    return np.sin(x)/x
```

Plotting

Plotting figures is straightforward using the matplotlib.pyplot library. The plot command takes arguments such as: marker size, mes=3; marker width, mew=3; linewidth lw=2. Plenty more options can be found in the API: http://matplotlib.org/api/pyplot_api.html.

```
import numpy as np
import matplotlib.pyplot as plt
import math

def sinc(x):

    return np.sin(x)/x


x = np.linspace(-2*math.pi, 2*math.pi, 100)


plt.figure(1)

plt.plot(x, sinc(x), 'k-', label='f(x)=sinc(x)')
plt.plot(x, x, 'r--', label='f(x)=x')
plt.legend()

plt.xlabel('x')
plt.ylabel('sinc(x)')

plt.xlim(-3*math.pi, 3*math.pi)
plt.ylim(-1, 2)

plt.show(block=False) # not needed in Spyder.
plt.savefig('tempfig.pdf')
```

User Interface Controls

For some applications is might be useful to enhance your plots with buttons, sliders and keyboard inputs. These can be found in the matplotlib.widgets class.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.widgets as widgets
import math
import sys
```

```
plt.close('all')

initFrequency = 1.0

def sinc(x, frequency):
    omega = 2 * math.pi * frequency
    return np.sin(omega * x)/x


x = np.linspace(-3.0*math.pi, 3.0*math.pi, 1000)

fig, ax1 = plt.subplots(1) # create a new figure with one axis, (also 1 by default)

plt.subplots_adjust(left=0.25, bottom=0.25) # move the position of the axes

axesHandle, = plt.plot(x, sinc(x, 1.0), lw=2, color='green')

# add a slider

def sliderCallback(val):

    localfrequency = sliderHandle.val # get the data from the slider.

    axesHandle.set_ydata(sinc(x, localfrequency)) # reset the y axis data.

    fig.canvas.draw_idle()  # redraw the axes


ax2 = plt.axes([0.25, 0.1, 0.65, 0.03]) # add new axes to the figure

sliderHandle = widgets.Slider(ax2, 'Freq', 0.1, 10.0, valinit=initFrequency)

sliderHandle.on_changed(sliderCallback)

# add a button

def clickCallback(event):

    color = axesHandle.get_color()

    if (color == 'green'):
        axesHandle.set_color('red')
    else:
        axesHandle.set_color('green')

    fig.canvas.draw_idle()  # redraw the axes


ax3 = plt.axes([0.1, 0.75, 0.1, 0.1]) # add new axes to the figure

buttonHandle = widgets.Button(ax3, 'Foo')

buttonHandle.on_clicked(clickCallback)

# add some keypress input

def keyPressCallback(event):

    print event.key

    if event.key == 'right':
        axesHandle.set_ydata(sinc(x, 10.0)) # reset the y axis data.
```

```
    elif event.key == 'left':
        axesHandle.set_ydata(sinc(x, 0.1)) # reset the y axis data.
    elif event.key == 'escape':
        plt.close('all')

    fig.canvas.draw_idle()  # redraw the axes

fig.canvas.mpl_connect('key_press_event', keyPressCallback)

plt.show(block=False)
```

File Handling

Text based data files can be loaded and saved using the inbuilt numpy functions.
Initially create a text file mydata.dat with two columns of numbers in it.

```
data = np.loadtxt('mydata.dat')

# should you wish to extract a column from the data

column0 =  np.array([row[0] for row in data])
column1 =  np.array([row[1] for row in data])
```

Similarly you can save text using :

```
data = np.array([[0, 1, 2, 3, 4], [10, 11, 12, 13, 14]])

np.savetxt('myFile.dat', data)

# or if you want it saving as columns

np.savetxt('myFile.dat', data.transpose())
```