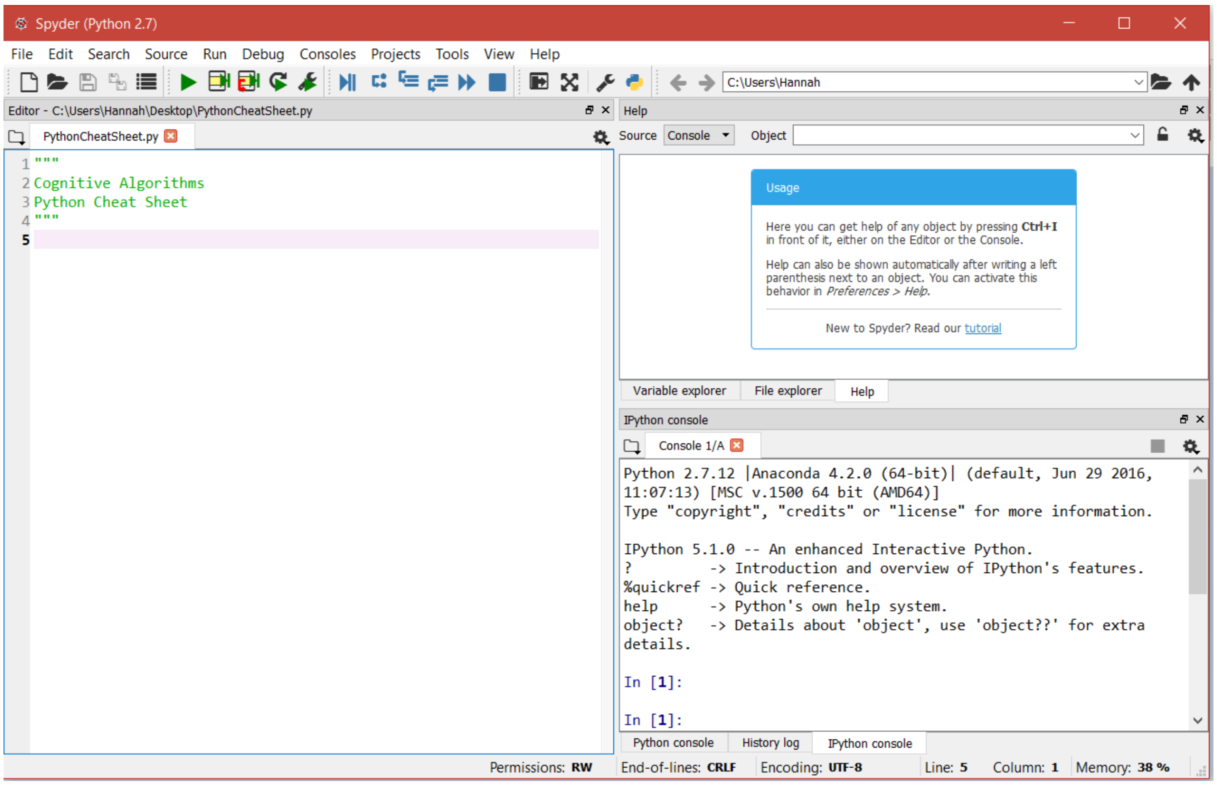
Cognitive Algorithms

Python Introduction

The aim of this document is to familiarize yourself with Python. It introduces you to general concepts that you need to complete the homework. However, it does not cover all aspects and you might attend the course "Python Programming for Machine Learning" for deeper insight.

Throughout this course, we will use Python 2.7 (not Python 3.6). We recommend you install "Anaconda", which is an open source distribution for Python. You can download it for free at https://www.continuum.io/downloads. We will use "Spyder" in this tutorial, which is an IDE and already included in "Anaconda" (see: https://docs.continuum.io/anaconda/ide_integration#spyder for further information). See https://pythonhosted.org/spyder/ for an introduction to "Spyder".



Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Hannah
Editor - C:\Users\Hannah\Desktop\PythonCheatSheet.py
PythonCheatSheet.py
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
Python console History log IPython console

```
"""
Cognitive Algorithms
Python Cheat Sheet
"""
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

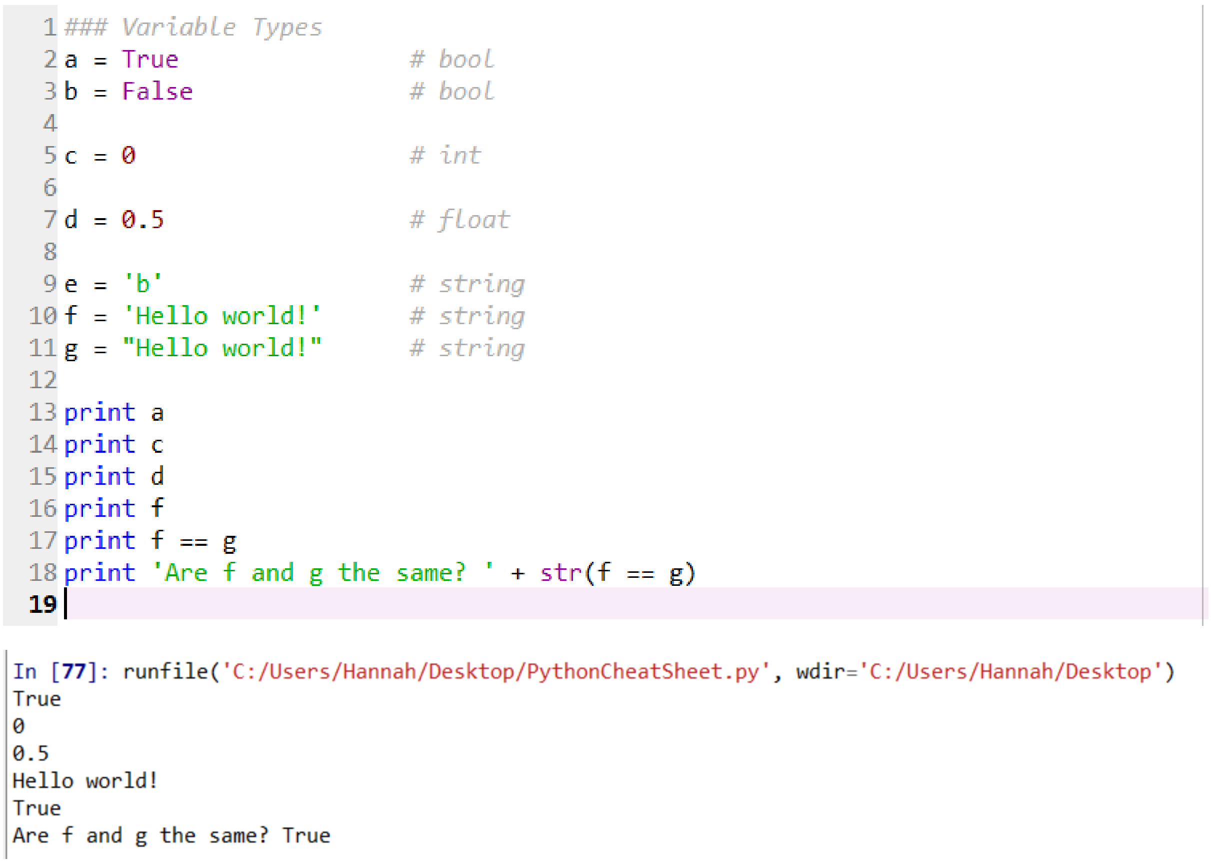
```
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016,
11:07:13) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
details.

In [1]:

In [1]:
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **5** Column: **1** Memory: **38 %**

Figure 1: IDE "Spyder" with editor (left), console (right, lower) and help (right, upper)

Variable Types

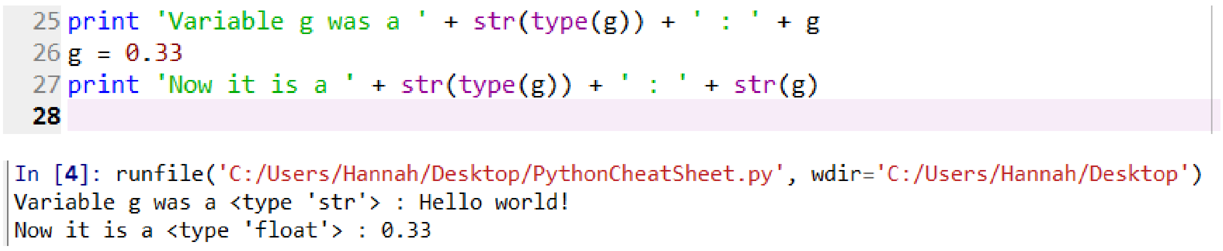
Python has many built-in types like any other programming language - some of them are shown below. You can print the value of a variable on the console using the print command. Two strings can be concatenated using '+'.



```
### Variable Types
a = True                # bool
b = False               # bool

c = 0                   # int

d = 0.5                 # float

e = 'b'                 # string
f = 'Hello world!'      # string
g = "Hello world!"      # string

print a
print c
print d
print f
print f == g
print 'Are f and g the same? ' + str(f == g)

```

```
In [77]: runfile('C:/Users/Hannah/Desktop/PythonCheatSheet.py', wdir='C:/Users/Hannah/Desktop')
True
0
0.5
Hello world!
True
Are f and g the same? True
```

Python uses typed objects but untyped variable names, as shown in this simple example.

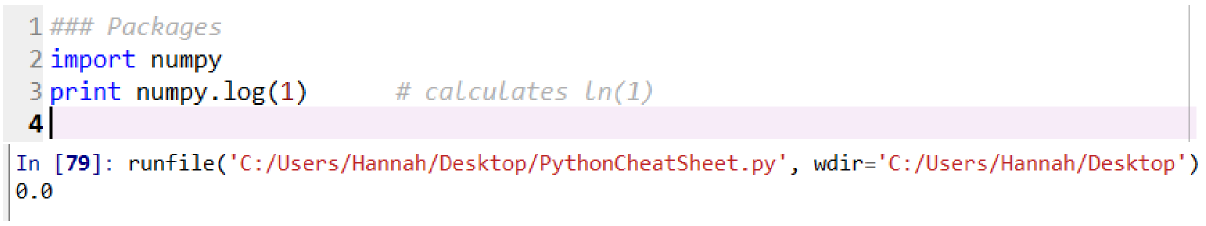


```
print 'Variable g was a ' + str(type(g)) + ' : ' + g
g = 0.33
print 'Now it is a ' + str(type(g)) + ' : ' + str(g)

```

```
In [4]: runfile('C:/Users/Hannah/Desktop/PythonCheatSheet.py', wdir='C:/Users/Hannah/Desktop')
Variable g was a <type 'str'> : Hello world!
Now it is a <type 'float'> : 0.33
```

Standard Modules

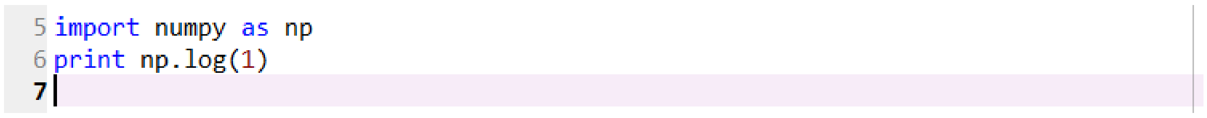
There are many modules and packages that already implement useful functions. We will often need NumPy, SciPy and Matplotlib for the homework. You should put the import commands as first line of your code. Use '.' to indicate that you want to call a function of an imported module.



```
### Packages
import numpy
print numpy.log(1)       # calculates ln(1)

```

```
In [79]: runfile('C:/Users/Hannah/Desktop/PythonCheatSheet.py', wdir='C:/Users/Hannah/Desktop')
0.0
```

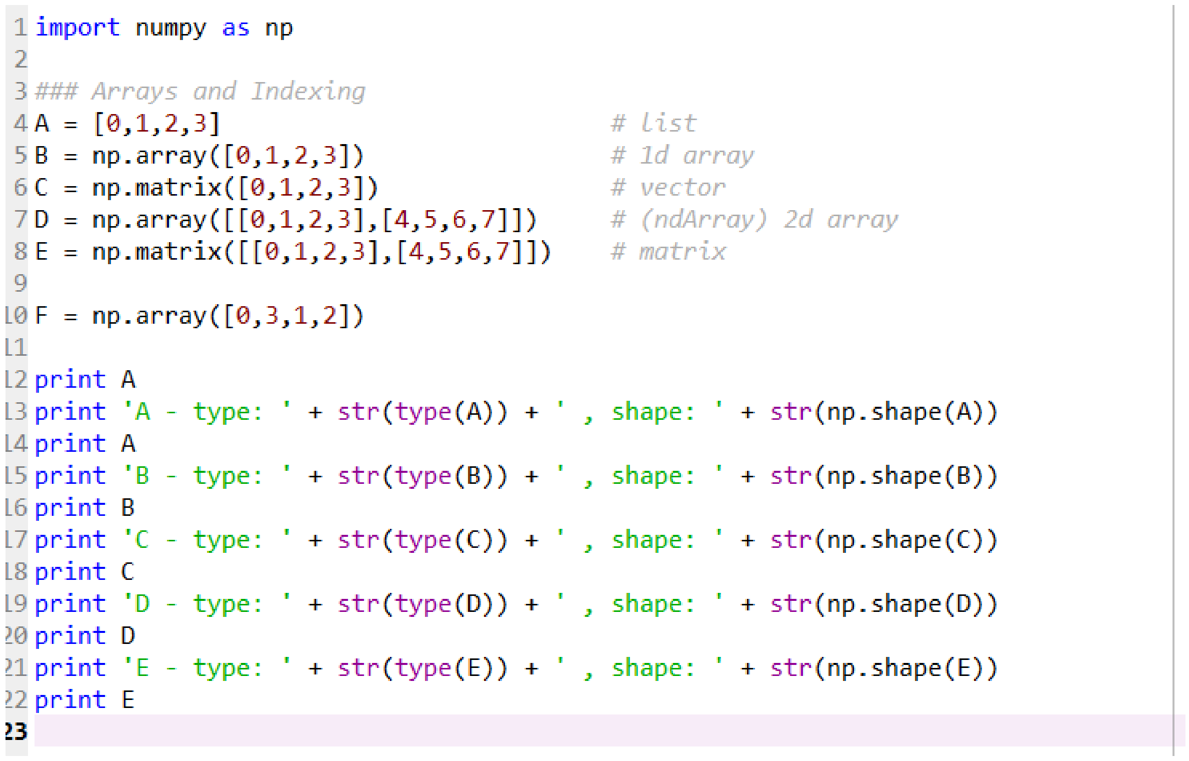
You might declare an abbreviation for the imported module, to shorten the notation



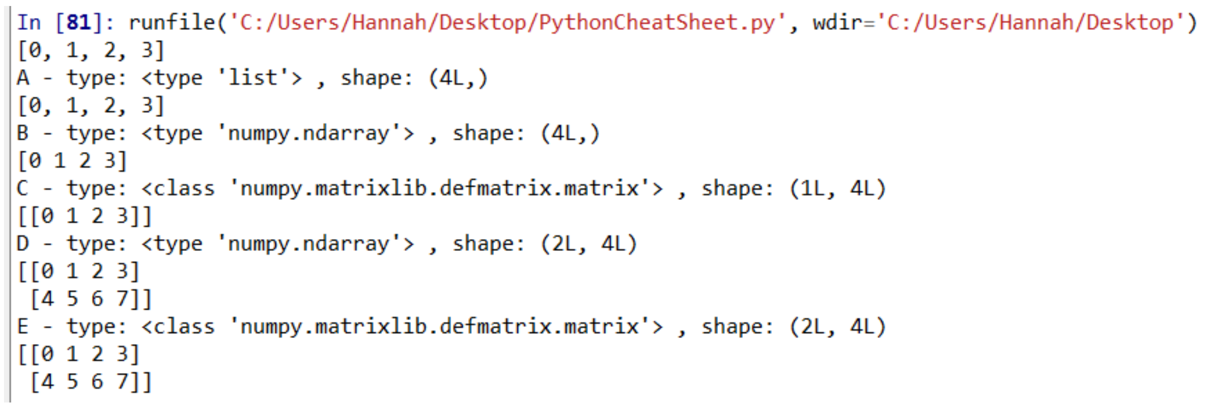
```
import numpy as np
print np.log(1)

```

Arrays and Indexing

Arrays are implemented in NumPy and SciPy. A mathematical matrix can be implemented as numpy.matrix or and 2 dimensional numpy.array. Both types are similar, but not the same. Which one to use, depends on the task. Note, that 1d matrices and arrays differ in shape, although they hold the same data.



```
import numpy as np

### Arrays and Indexing
A = [0,1,2,3]                        # list
B = np.array([0,1,2,3])              # 1d array
C = np.matrix([0,1,2,3])             # vector
D = np.array([[0,1,2,3],[4,5,6,7]])  # (ndArray) 2d array
E = np.matrix([[0,1,2,3],[4,5,6,7]]) # matrix

F = np.array([0,3,1,2])

print A
print 'A - type: ' + str(type(A)) + ' , shape: ' + str(np.shape(A))
print A
print 'B - type: ' + str(type(B)) + ' , shape: ' + str(np.shape(B))
print B
print 'C - type: ' + str(type(C)) + ' , shape: ' + str(np.shape(C))
print C
print 'D - type: ' + str(type(D)) + ' , shape: ' + str(np.shape(D))
print D
print 'E - type: ' + str(type(E)) + ' , shape: ' + str(np.shape(E))
print E

```



```
In [81]: runfile('C:/Users/Hannah/Desktop/PythonCheatSheet.py', wdir='C:/Users/Hannah/Desktop')
[0, 1, 2, 3]
A - type: <type 'list'> , shape: (4L,)
[0, 1, 2, 3]
B - type: <type 'numpy.ndarray'> , shape: (4L,)
[0 1 2 3]
C - type: <class 'numpy.matrixlib.defmatrix.matrix'> , shape: (1L, 4L)
[[0 1 2 3]]
D - type: <type 'numpy.ndarray'> , shape: (2L, 4L)
[[0 1 2 3]
 [4 5 6 7]]
E - type: <class 'numpy.matrixlib.defmatrix.matrix'> , shape: (2L, 4L)
[[0 1 2 3]
 [4 5 6 7]]
```

We get a single entry or multiple entries using [.]. We show examples for the matrix E, but similar can be done for the array D. You are encouraged to try it yourself.

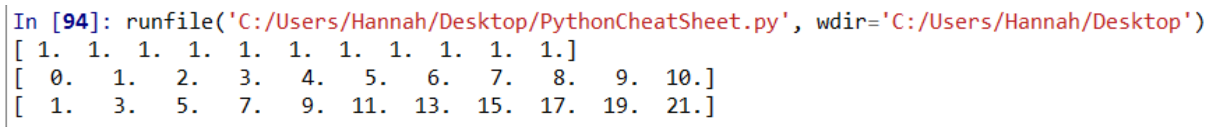
```
In [83]: B[0]
Out[83]: 0

In [84]: E[0,0]
Out[84]: 0

In [85]: E[0]
Out[85]: matrix([[0, 1, 2, 3]])

In [86]: E[0,:]
Out[86]: matrix([[0, 1, 2, 3]])

In [87]: E[:,0]
Out[87]:
matrix([[0],
        [4]])

In [88]: E[:, 1:3]
Out[88]:
matrix([[1, 2],
        [5, 6]])
```

```
In [89]: E[:, 2:]
Out[89]:
matrix([[2, 3],
        [6, 7]])

In [90]: E[:, :2]
Out[90]:
matrix([[0, 1],
        [4, 5]])

In [91]: E[:, F]
Out[91]:
matrix([[0, 3, 1, 2],
        [4, 7, 5, 6]])
```

Functions

We can also write our own functions. Since we have no explicit typing, it is important to write comments for each input and output variable.

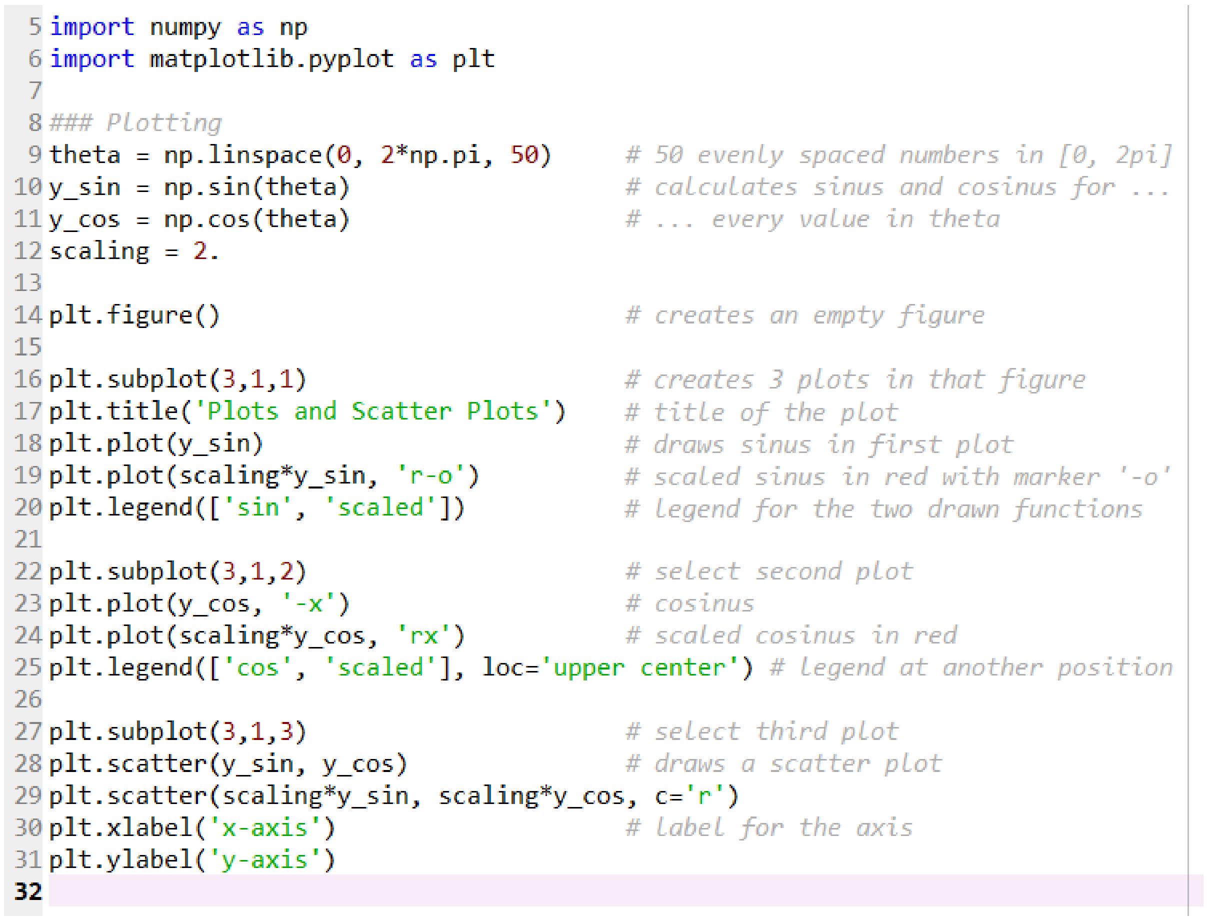
```
import numpy as np

### Functions
def foo(A, B, c):
    '''
    Calculates the sum of A and scaled B. B is scaled using c. X = A + c*B

    Usage:      X = foo(A, B, c)
    Input:      A : (N,) array
                B : (N,) array
                c : scalar
    Returns:    X : (N,) array

    '''
    X = A + c*B
    return X

Y1 = np.ones([11,])            # 1d array filled with ones
Y2 = np.linspace(0, 10, 11)    # [0, 1, ..., 10]
c = 2.
X = foo(Y1,Y2,c)
print Y1
print Y2
print X

```



```
In [94]: runfile('C:/Users/Hannah/Desktop/PythonCheatSheet.py', wdir='C:/Users/Hannah/Desktop')
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
[  1.   3.   5.   7.   9.  11.  13.  15.  17.  19.  21.]
```

Plotting

We often wish to visualize the data. The module matplotlib.pyplot already implements useful functions to draw functional or scatter plots and to individualize those plots.



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

### Plotting
theta = np.linspace(0, 2*np.pi, 50)     # 50 evenly spaced numbers in [0, 2pi]
y_sin = np.sin(theta)                   # calculates sinus and cosinus for ...
y_cos = np.cos(theta)                   # ... every value in theta
scaling = 2.

plt.figure()                            # creates an empty figure

plt.subplot(3,1,1)                      # creates 3 plots in that figure
plt.title('Plots and Scatter Plots')    # title of the plot
plt.plot(y_sin)                         # draws sinus in first plot
plt.plot(scaling*y_sin, 'r-o')          # scaled sinus in red with marker '-o'
plt.legend(['sin', 'scaled'])           # legend for the two drawn functions

plt.subplot(3,1,2)                      # select second plot
plt.plot(y_cos, '-x')                   # cosinus
plt.plot(scaling*y_cos, 'rx')           # scaled cosinus in red
plt.legend(['cos', 'scaled'], loc='upper center') # legend at another position

plt.subplot(3,1,3)                      # select third plot
plt.scatter(y_sin, y_cos)               # draws a scatter plot
plt.scatter(scaling*y_sin, scaling*y_cos, c='r')
plt.xlabel('x-axis')                    # label for the axis
plt.ylabel('y-axis')

```

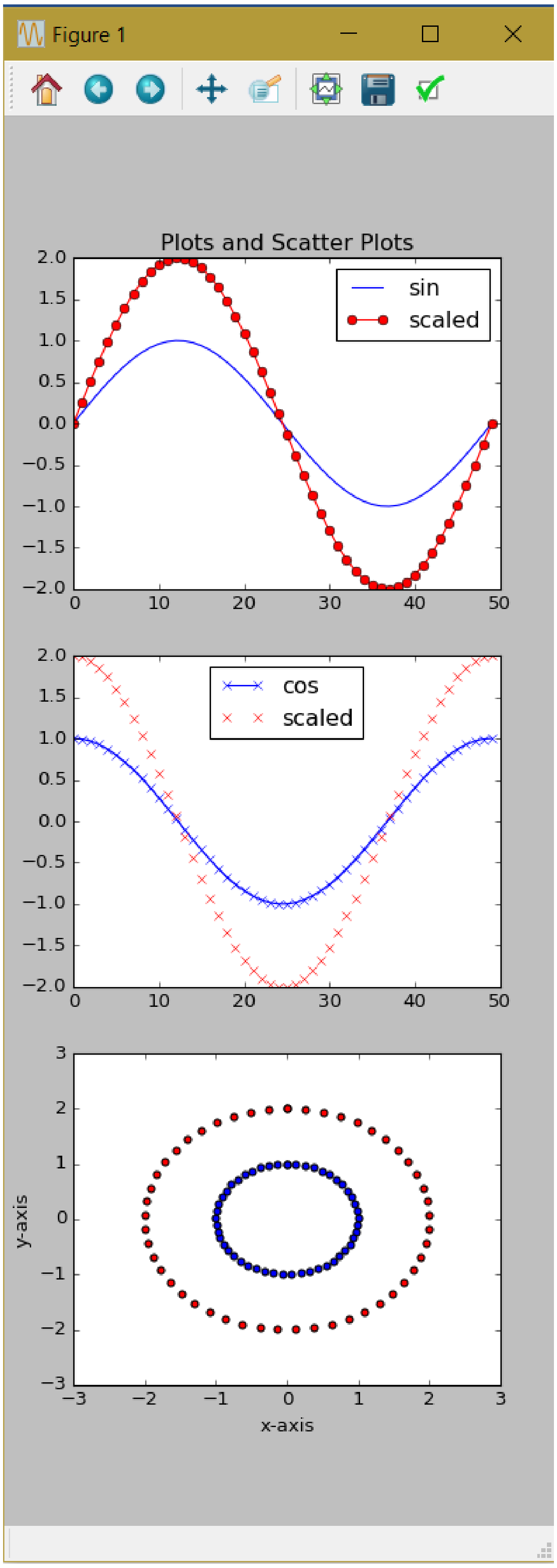
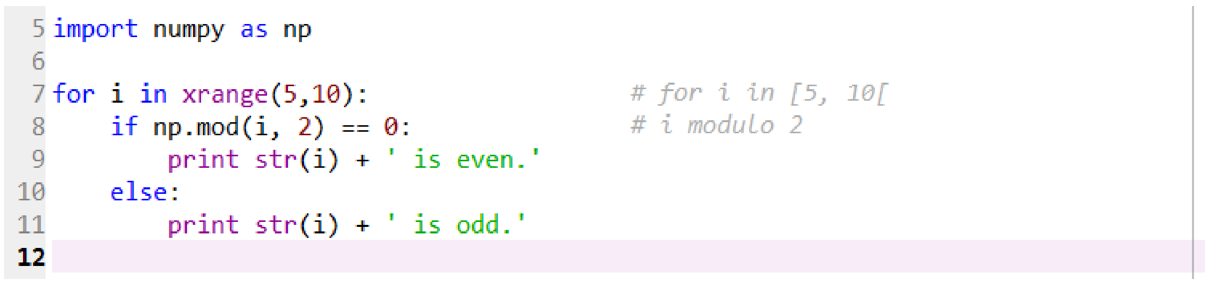


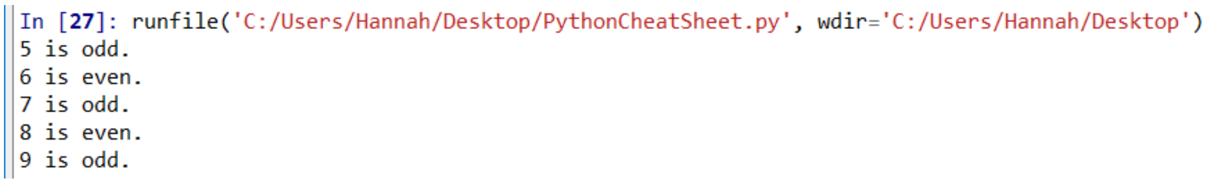
Figure 1
Plots and Scatter Plots
sin
scaled
cos
scaled
y-axis
x-axis

Loops

As we will see in the homework, loops are slow in Python, so you should avoid them and use matrix operands or functions of NumPy or SciPy instead. However, the syntax is straight forward:



```python
import numpy as np

for i in xrange(5,10):                  # for i in [5, 10[
    if np.mod(i, 2) == 0:               # i modulo 2
        print str(i) + ' is even.'
    else:
        print str(i) + ' is odd.'

```



```
In [27]: runfile('C:/Users/Hannah/Desktop/PythonCheatSheet.py', wdir='C:/Users/Hannah/Desktop')
5 is odd.
6 is even.
7 is odd.
8 is even.
9 is odd.
```