Python Tutorial

This tutorial is adapted from the Python tutorial session of MS&E 111/211 given by Professor Benjamin Van Roy. The script in this tutorial follows the syntax of Python 2.7. If you want to run it in a Python 3 environment, the commands "print XXX" should be replaced by "print(XXX)" while all the other commands should be compatible.

1 Python installation

We would recommend you to install Anaconda python . Anaconda is a free and open source distribution of the Python and its aim is to simplify the module management and development. The installation link is:

`https://www.anaconda.com/download/`

You may install both python 2.7 or python 3 for this course. If you want both: for those of you who have already installed python 3 on your computer, you may create a new environment for python 2 by running (in terminal)

```
conda create -n python2 python=2.7 anaconda
```

activate the environment with

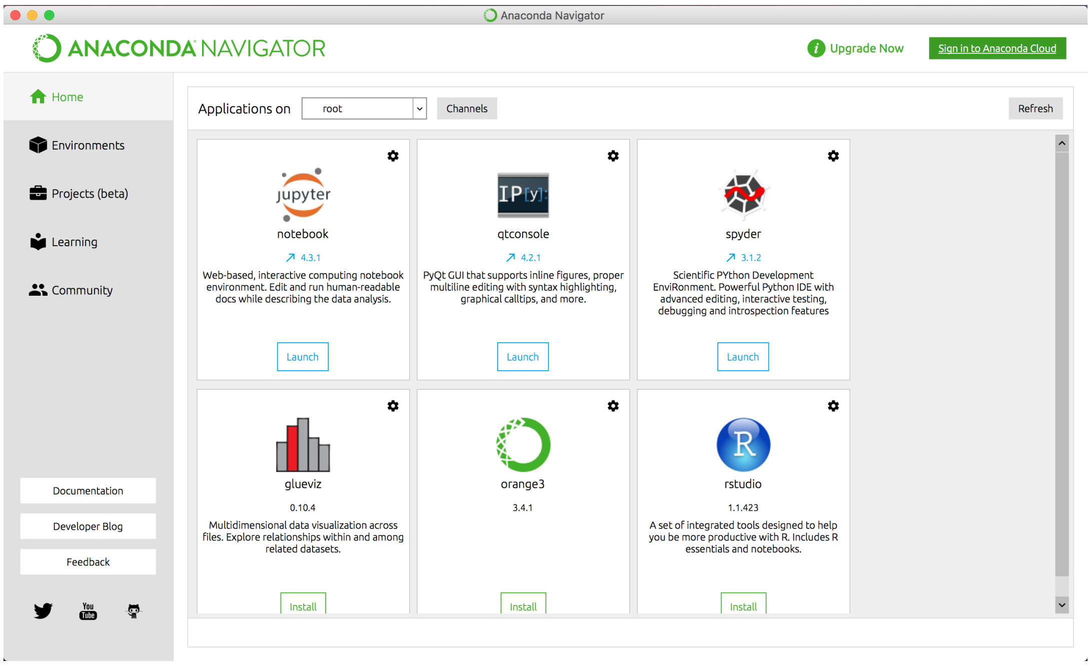
```
source activate python2
```

and deactivate the environment with

```
source deactivate
```

2 Jupyter ipython notebook

We also recommend writing code with Jupyter ipython notebook. Jupyter notebooks provide an interactive programming environment and rich text/figure support. If you are familiar with Matlab or R, Jupyter enables us to program Python interactively as Matlab or R. In the current version of Anaconda, Jupyter Notebooks should be automatically installed and can be found in the home page of the navigator (on the top left).



notebook 4.3.1
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

qtconsole 4.2.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder 3.1.2
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

glueviz 0.10.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.

orange3 3.4.1

rstudio 1.1.423
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

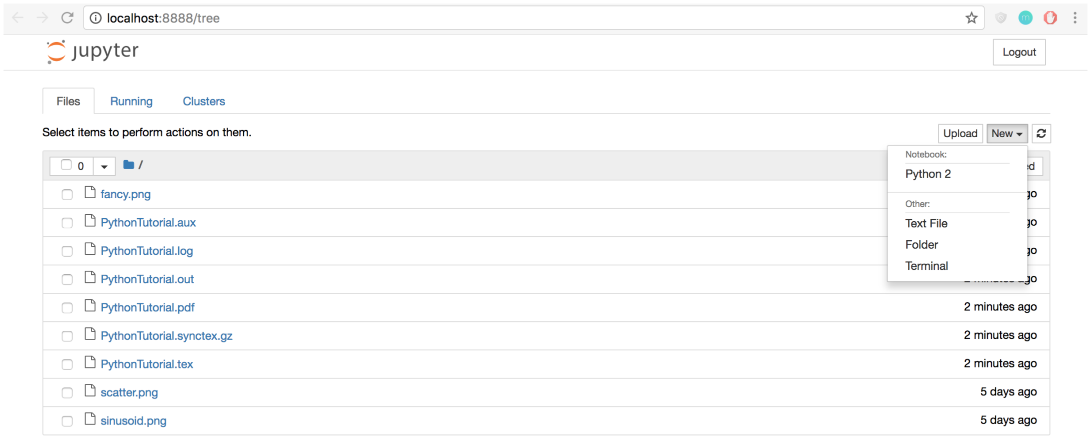
However, if it isn't installed. The installation of Jupyter notebook can be done with command in terminal once after you installed the Anaconda python. The details can be found

```
http://jupyter.org/install
```

To launch the notebook, run the following command at the Terminal (Mac/Linux) or Command Prompt/Anaconda Prompt (Windows):

```
jupyter notebook
```

Then you will launch a Jupyter iPython notebook in your browser:



Select items to perform actions on them.

fancy.png	
PythonTutorial.aux	
PythonTutorial.log	
PythonTutorial.out	
PythonTutorial.pdf	2 minutes ago
PythonTutorial.synctex.gz	2 minutes ago
PythonTutorial.tex	2 minutes ago
scatter.png	5 days ago
sinusoid.png	5 days ago

It will automatically detect and link to the active python environment in your computer. To create a new python notebook, you can click the "new" button on the upper-right corner. The default working directory is the directory in which you launch the notebook. For example, if you launch the notebook

```
/home/111_python_tutorial Ben$ jupyter notebook
```

Then all the files in the directory `/home/111_python_tutorial` will appear in the notebook interface.

3 Basics

After you launch the Jupyter notebook, you may create a new notebook and try the following code.

```
In [1]: print 'hello world'
hello world

In [2]:
```

The above transcript includes an instruction at the first ipython command prompt to print “hello world.”

It is straightforward to carry out basic arithmetic and variable assignments:

```
In [2]: 2 + 5
Out[2]: 7

In [3]: 2.1 * 10
Out[3]: 21.0

In [4]: a = 2

In [5]: b = 5

In [6]: c = a + b

In [7]: print c
7
```

One thing to be careful for, though, is that integers and floats are distinct types in python, and arithmetic operations are defined differently for them. The following example illustrates problems that can arise:

```
In [8]: 2.0/3.0
Out[8]: 0.6666666666666666

In [9]: 2/3
Out[9]: 0
```

Note that integer two divided by integer three is zero because integer division truncates fractions. When integers and floats are mixed in a calculation, python converts all the numbers to float:

```
In [10]: 2/3.0
Out[10]: 0.6666666666666666
```

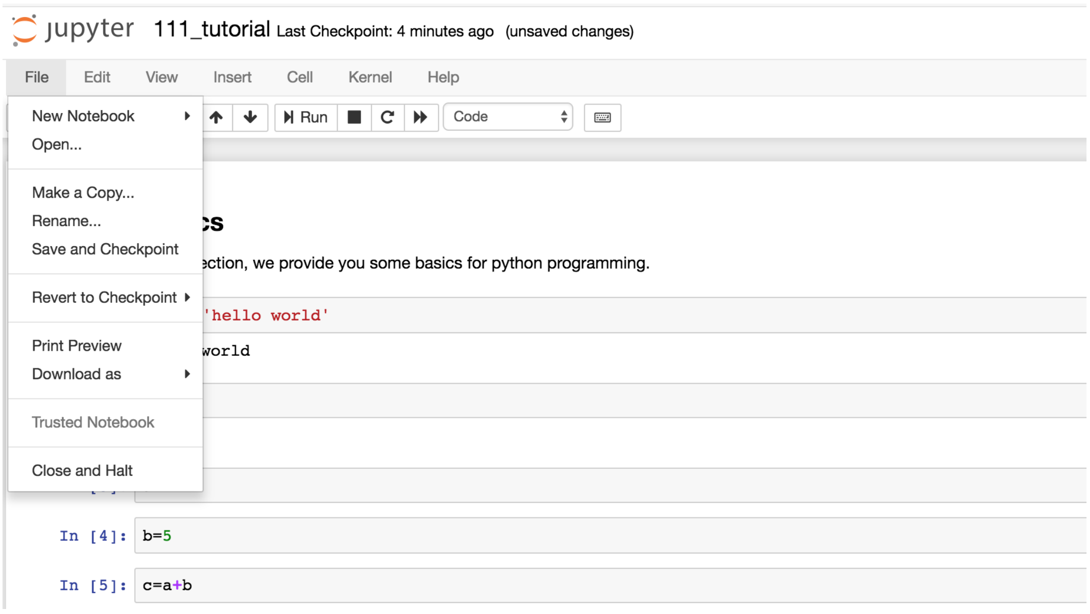
The `type` command can be used to check the type of a variable. Further, python accommodates type conversions, as the following example illustrates:

```
In [11]: type(c)
Out[11]: int

In [12]: d = float(c)
In [13]: d
Out[13]: 7.0

In [14]: type(d)
Out[14]: float
```

The way you exit the current notebook file is simply to save and exit just like what you do for a Word or Excel file. It is done by clicking the "save and checkpoint" button in the below figure or simply "command+s"/"ctrl+s" in mac/windows.



ction, we provide you some basics for python programming.

```
'hello world'
world
In [4]: b=5
In [5]: c=a+b
```

4 Scripts

To save sequences of instructions and to facilitate debugging, it is often helpful to write instructions in a script, which is a text file consisting of python instructions. To create a script, click the "new" button on the upper right corner of the Jupyter interface and create a new text file. As an example, consider a the following instructions:

```
print 'hello world'
a = 2
b = 5
print(a + b)
```

After saving this file as `/home/111_python_tutorial/script_example.py`, we can execute the script from the terminal command line:

```
/home/111_python_tutorial Ben$ python script_example.py
hello world
7
/home/111_python_tutorial Ben$
```

Scripts can also be executed from ipython notebok:

```
In [1]: run script_example.py
hello world
7
```

One can insert text comments in a script on any line after the # symbol. Any text appearing to the right of the # is ignored by the python interpreter.

5 Data structures

Let us now discuss a few data structures that we will use regularly in computational assignments.

5.1 Lists

A list consists of a sequence of elements of any type. Let us illustrate construction, inspection, and manipulation of lists:

```
In [1]: alist = ['elephant', 532, 0.001, 'zebra', 50.05]

In [2]: alist
Out[2]: ['elephant', 532, 0.001, 'zebra', 50.05]

In [3]: alist[0]
Out[3]: 'elephant'

In [4]: alist[1]
Out[4]: 532

In [5]: alist[4]
Out[5]: 50.05

In [6]: alist[-1]
Out[6]: 50.05

In [7]: alist[-2]
Out[7]: 'zebra'

In [8]: blist = [1,2,3,4]

In [9]: alist + blist
Out[9]: ['elephant', 532, 0.001, 'zebra', 50.05, 1, 2, 3, 4]
```

Note that we can create lists of lists:

```
In [10]: clist = [[1,2], [3,4,5], 6]

In [11]: clist[0]
Out[11]: [1, 2]

In [12]: clist[1]
Out[12]: [3, 4, 5]

In [13]: clist[2]
Out[13]: 6
```

List comprehensions offer a useful tool for manipulating lists, as the following example illustrates:

```
In [1]: alist = range(10)

In [2]: alist
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [3]: blist = [2*a for a in alist]
In [4]: blist
Out[4]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [5]: [2*a for a in range(10)]
Out[5]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

The `range` function produces a list of integers; in this case, a list ranging from 0 to 9. The content within the square brackets used to define `blist` is interpreted as a loop that iterates over elements of `alist`, generating a new list by multiplying each element by 2.

5.2 Tuples

A tuple is similar to a list except that it is immutable. In other words, it cannot be changed in any way once it is created. While lists are constructed using square brackets, tuples are constructed using parentheses. The following example illustrates how lists can be modified while tuples can not:

```
In [14]: clist
Out[14]: [[1, 2], [3, 4, 5], 6]

In [15]: clist[0] = 101

In [16]: clist
Out[16]: [101, [3, 4, 5], 6]

In [17]: ctuple = ((1,2), (3,4,5), 6)

In [18]: ctuple[0] = 101
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-d5dcef241166> in <module>()
----> 1 ctuple[0] = 101

TypeError: 'tuple' object does not support item assignment

In [19]: ctuple
Out[19]: ((1, 2), (3, 4, 5), 6)
```

5.3 Numpy matrices

Numpy is a python software package that facilitates numerical computation. Among other things, numpy offers tools to work with matrices. The following example illustrates the construction of a matrix:

```
In [1]: A = np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

In [2]: A
Out[2]:
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])
```

Note that `np` is an abbreviation for numpy, and we produced the matrix using numpy's `matrix` function, providing it with a list of lists of numbers (a tuple of tuples would also work). This list can be thought of as a list of rows, where each row is a list of matrix entries. We can access rows, columns, or other subsets of elements:

```
In [1]: A = np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

In [2]: A
Out[2]:
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])

In [3]: A[0,:]
Out[3]: matrix([[1, 2, 3, 4]])

In [4]: A[:,1]
Out[4]:
matrix([[ 2],
        [ 6],
        [10]])

In [5]: A[1,1:2]
Out[5]: matrix([[6]])

In [6]: A[1,1:3]
Out[6]: matrix([[6, 7]])

In [7]: A[[0,2],[1,3]]
Out[7]: matrix([[ 2, 12]])
```

Note that a range of the form `m:n` includes the $n - m$ indices $m, m + 1, n - 1$, while excluding n. Transposition is applied via `.T`:

```
In [8]: A.T
Out[8]:
matrix([[ 1,  5,  9],
        [ 2,  6, 10],
        [ 3,  7, 11],
        [ 4,  8, 12]])
```

Row and column vectors are simply matrices with single rows or columns. Vector components can be specified by a list of numbers rather than a list of lists of numbers:

```
In [1]: np.matrix([1,2,3])
Out[1]: matrix([[1, 2, 3]])

In [2]: np.matrix([1,2,3]).T
Out[2]:
matrix([[1],
        [2],
        [3]])
```

Rows or columns of a matrix can also be converted to vectors:

```
In [1]: A = np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

In [2]: A[1,:]
Out[2]: matrix([[5, 6, 7, 8]])

In [3]: A[:,2]
Out[3]:
matrix([[ 3],
        [ 7],
        [11]])
```

Matrix arithmetic is straightforward:

```
In [1]: A = np.matrix([[1,2],[3,4]])

In [2]: B = np.matrix([[5,6],[7,8]])

In [3]: A+B
Out[3]:
matrix([[ 6,  8],
        [10, 12]])

In [4]: A*B
Out[4]:
matrix([[19, 22],
        [43, 50]])

In [5]: np.multiply(A,B)
Out[5]:
matrix([[ 5, 12],
        [21, 32]])
```

Note that `np.multiply` and `np.divide` carry out element-wise multiplication and division for matrices of identical dimensions. On the topic of dimensions, the dimensions of a matrix can be obtained in the form of a tuple as follows:

```
In [6]: A.shape
Out[6]: (2, 2)
```

Matrices can be concatenated if their dimensions are compatible:

```
In [8]: np.concatenate((A,B),axis=0)
Out[8]:
matrix([[1, 2],
        [3, 4],
        [5, 6],
        [7, 8]])

In [9]: np.concatenate((A,B),axis=1)
Out[9]:
matrix([[1, 2, 5, 6],
        [3, 4, 7, 8]])

In [10]: np.concatenate((A,B,B,A),axis=1)
Out[10]:
matrix([[1, 2, 5, 6, 5, 6, 1, 2],
        [3, 4, 7, 8, 7, 8, 3, 4]])
```

Note that the first argument to `np.concatenate` is a tuple of matrices, while the axis argument specifies whether the concatenation should be done vertically or horizontally.

There are dedicated functions for producing particular kinds of matrices:

```
In [1]: np.identity(3)
Out[1]:
array([[ 1.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  0.,  1.]])

In [2]: np.zeros((2,3))
Out[2]:
array([[ 0.,  0.,  0.],
        [ 0.,  0.,  0.]])

In [3]: np.ones((2,3))
Out[3]:
array([[ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])

In [4]: np.random.random([2,3])
Out[4]:
array([[ 0.8925401 ,  0.62319141,  0.23885902],
        [ 0.95715811,  0.27609046,  0.32808951]])

In [5]: np.random.normal([2,3])
Out[5]:
array([[ 0.22242987,  0.51574683,  0.59468841],
        [-1.78117514, -0.01304228, -1.16989767]])
```

The functions `np.random.random` and `np.random.normal` return random matrices. In the first case, each entry is drawn from a uniform distribution over the interval $[0, 1]$. In the second case, each entry is drawn from a Gaussian distribution with mean zero and variance one.

There is also a function that inverts matrices:

```
In [6]: A = np.random.random((2,2))

In [7]: A_inv = np.linalg.inv(A)

In [8]: A
Out[8]:
array([[-0.45949096,  1.55846699],
        [-0.25718063,  3.08378464]])

In [9]: A_inv
Out[9]:
array([[-3.0347323 ,  1.53367718],
        [-0.25308977,  0.45218205]])

In [10]: A * A_inv
Out[10]:
array([[  1.00000000e+00,   0.00000000e+00],
        [ -1.11022302e-16,   1.00000000e+00]])
```

Note that the product of the matrix and its inverse is virtually equal to the identity matrix; the small difference is due to errors stemming from limited numerical precision.

6 Conditional and Iterative Statements

As with other common programming languages, python supports syntax for conditional and iterative statements.

6.1 if...else

This is an example of a simple conditional statement:

```
if x == 3:
    x = x + 1
    print "new value of x is 4"
print "done"
```

If the value of the variable x is 3, the value of x will increase by one and then the message will print; otherwise, the two lines following the `if` statement will be skipped. A distinctive element of python is its use of white space to distinguish code blocks. In the above example, the two lines following the `if` statement make up a code block, while the line that prints "done" is not part of the code block, and as such is executed regardless of the value of x. The code block is distinguished by a common level of indentation relative to the `if` statement. For consistency, we recommend using a tab or four spaces for identation as in the example above.

Here is a more complicated conditional statement:

```
if x < 3:
    print "x is less than 3"
elif (x >= 3) and (x < 6):
    print "x is greater or equal to 3 and smaller than 6"
else:
    print "x is greater or equal to 6"
```

A conditional statement can have multiple `elif` statements but it can have at most one `else` statement.

6.2 for...in

Below is an example of looping in python:

```
In [11]: alist = ['elephant', 532, 0.001, 'zebra', 50.05]
In [12]: for i in alist:
    ...:     print i
elephant
532
0.001
zebra
50.05
```

Python allows iterating over lists. Similar to conditional statements, indentation is required to distinguish the code block that is repeatedly executed.

To iterate over a list of consecutive integers, one can use the `range` function:

```
In [13]: for i in range(4):
    ...:     print i
0
1
2
3
```

7 Modules and packages

A module is a file containing python code. When a module is loaded in ipython, its code is executed, and variables and functions defined in by the module are available for use. To distinguish names used in the module from other work, when accessing a function or variable from a loaded module, the module name must be used as a prefix. For example, numpy is a module. To use a function from numpy in ipython, one would load it using the `import` command:

```
In [1]: import numpy

In [2]: numpy.sqrt(2)
Out[2]: 1.4142135623730951

In [3]: numpy.nan
Out[3]: nan

In [4]: numpy.inf
Out[4]: inf
```

The names `sqrt`, `nan`, and `inf` are defined in numpy. The first is a square-root function. The latter represent undefined or infinite outcomes to computations carried out using numpy tools. For example:

```
In [5]: numpy.divide(7.0,0.0)
Out[5]: inf

In [6]: numpy.divide(0.0,0.0)
Out[6]: nan
```

Every time when you want to call functions in the numpy, you need to import the module first. Further, an alias of np is assigned. To assign an alias when loading a module, one can do this:

```
In [1]: import numpy as np
```

A script that makes use of a module must load the module within the script before it is used. This is true even if the module has already been loaded in ipython and the script is being executed from ipython.

A package is a collection of Python modules. Once you successfully installed the Anaconda, you can install packages by command line. For example, by running

```
pip install cvxopt
```

and

```
pip install cvxpy
```

you successfully install the packages of “cvxopt” and “cvxpy”. Also, when you encountered error message like below, it is usually caused by packages not yet installed.

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-93-a649b509054f> in <module>()
----> 1 import tabulate

ImportError: No module named tabulate
```

To resolve the issue, you just need to run

```
pip install tabulate
```

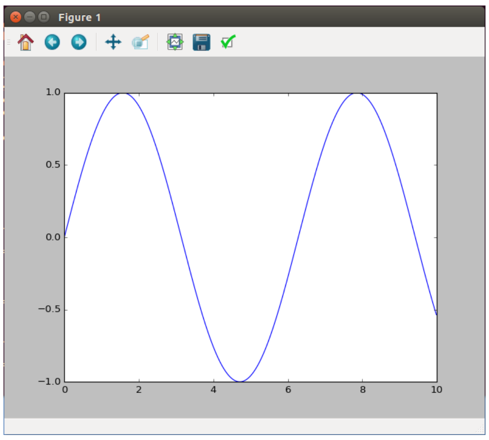


Figure 1: sinusoid

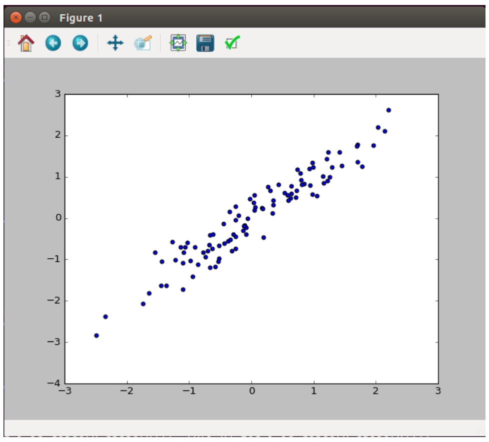


Figure 2: scatter plot

8 Plotting

The `matplotlib.pyplot` module provides useful tools for plotting data. This module is automatically loaded when you start ipython. Here is an example where we plot a sinusoid:

```
In [1]: x = [z/100.0 for z in range(1000)]

In [2]: y = [np.sin(z) for z in x]

In [3]: plt.plot(x, y)
Out[3]: [<matplotlib.lines.Line2D at 0x7f7d1411d7d0>]

In [4]: plt.show(block=False)
```

Figure 1 shows the popup window that appears with the execution of `plt.show(block=False)`. Without the `block=False` argument, ipython freezes until the plot window is closed. The `range` function here produces a list of integers from 0 to 999. Note that the variables `x` and `y` are defined by list comprehensions.

We can also produce a scatter plot, as depicted in Figure 2:

```
In [1]: x = np.random.normal(0,1,[100,1])

In [2]: y = x + 0.3 * np.random.normal(0,1,[100,1])

In [3]: plt.scatter([x[i,0] for i in range(x.shape[0])], [y[i,0] for i in range(y.shape[0])])
Out[3]: <matplotlib.collections.PathCollection at 0x7ffb116cea10>

In [4]: plt.show(block=False)
```

More sophisticated things can also be done, like plotting multiple data sets and adding labels and legends. The following script generates Figure 3:

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

## generate data sets
x1 = np.random.normal(0,1,[100,1])
y1 = x1 + 0.3 * np.random.normal(0,1,[100,1])
x1 = [x1[i,0] for i in range(x1.shape[0])]
```

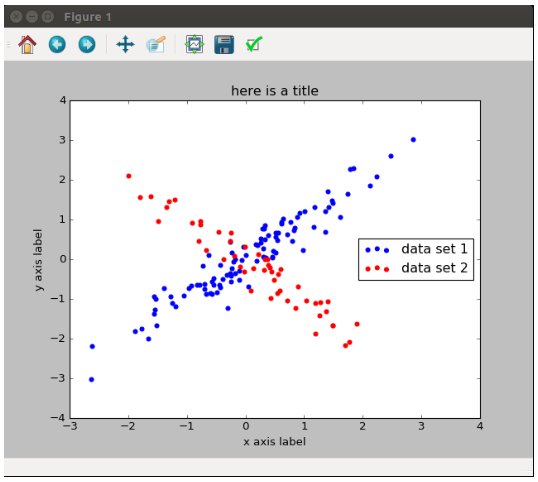


Figure 3: a more sophisticated scatter plot

```
y1 = [y1[i,0] for i in range(y1.shape[0])]
x2 = np.random.normal(0,1,[50,1])
y2 = -x2 + 0.3 * np.random.normal(0,1,[50,1])
x2 = [x2[i,0] for i in range(x2.shape[0])]
y2 = [y2[i,0] for i in range(y2.shape[0])]

## produce figure and plots
myfig = plt.figure()
myplot = myfig.add_subplot(111)
scatter1 = myplot.scatter(x1, y1, color='blue')
scatter2 = myplot.scatter(x2, y2, color='red')
myplot.set_title('here is a title')
myplot.set_xlabel('x axis label')
myplot.set_ylabel('y axis label')
myplot.legend((scatter1, scatter2), ('data set 1', 'data set 2'), loc='right')
myfig.show()
```