Python tutorial

Lascon 2016

Ludmila Brochini

Neuromat

Why python?

Python is an intepreted object oriented programming language

- elegant syntax, simple to read
- extensive documentation and huge comunity
- nice modules for scientific computing/data analysis/vizualization
- good starting point for begginer-level programmers

Goal: Learn enough python so you can start using python using neuron simulations - Must have python 2 installed

Topics

- Data types
- Flow control (if statement, for loops)
- Functions and Classes
- Modules
    - Numpy (scientific computing)
    - Matplotlib (2D plots)

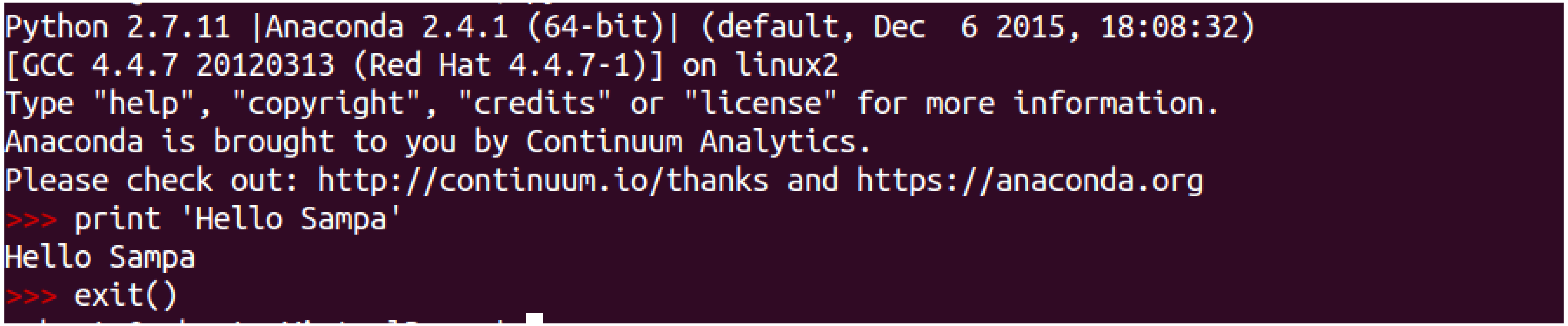
Handout:pytut.py (try it out)

Getting started

1- interacting with the interpreter (prompt mode)

In a terminal:

```
$python
```



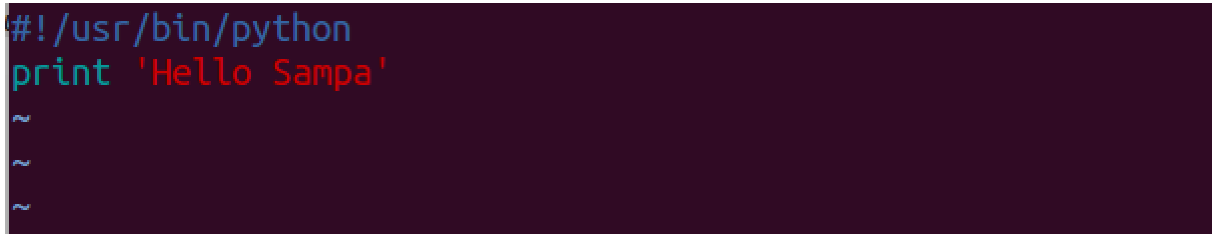
```
Python 2.7.11 |Anaconda 2.4.1 (64-bit)| (default, Dec  6 2015, 18:08:32)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> print 'Hello Sampa'
Hello Sampa
>>> exit()
```

You just wrote your first line of code in python

Getting started

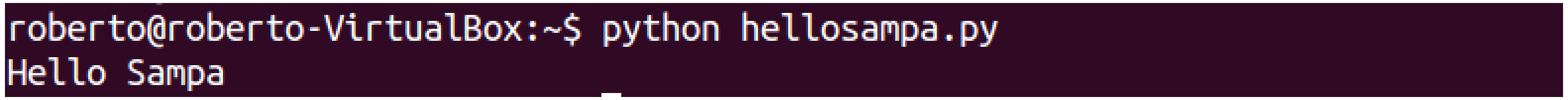
2- Source file mode

Create a file *filename.py* that contains *



```
#!/usr/bin/python
print 'Hello Sampa'
~
~
~
```

$ python *filename.py*



```
roberto@roberto-VirtualBox:~$ python hellosampa.py
Hello Sampa
```

...You can also try in IDE (pycharm, eclipse, spyder)

*about print: no parenthesis in python 2

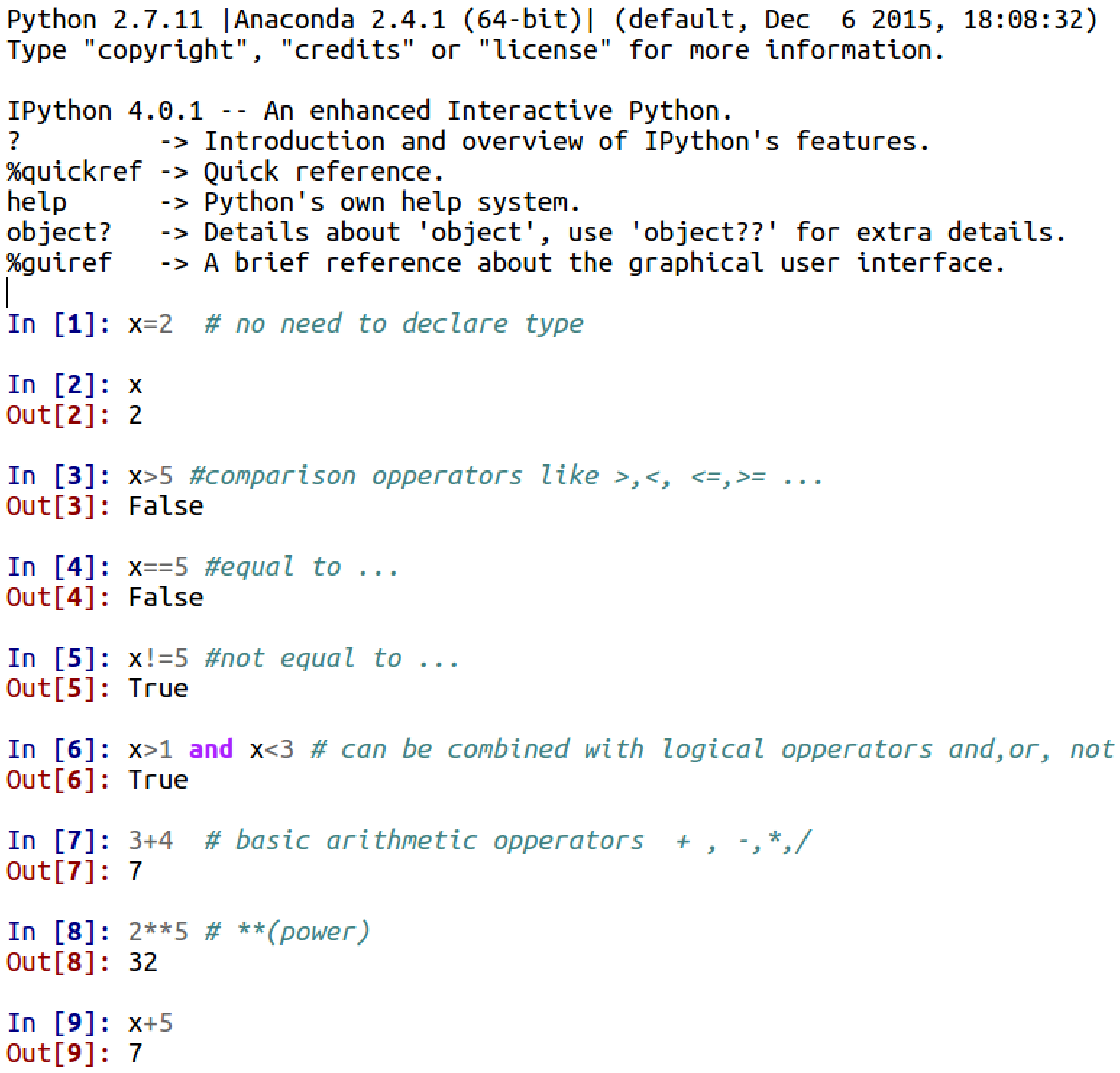
Data types

Standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Numbers

Integer, float, boolean

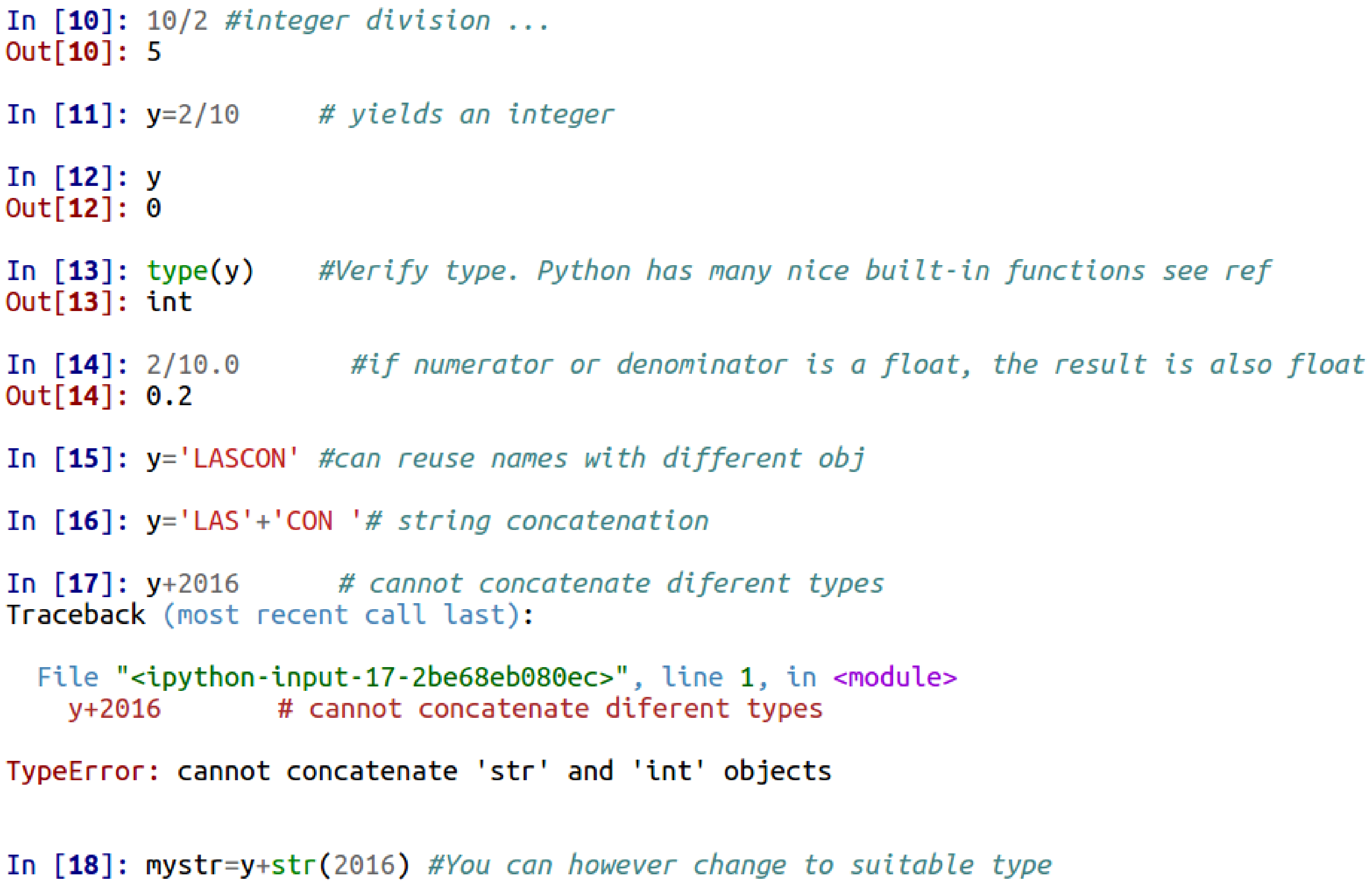


```
Python 2.7.11 |Anaconda 2.4.1 (64-bit)| (default, Dec  6 2015, 18:08:32)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.1 -- 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.
%guiref   -> A brief reference about the graphical user interface.

In [1]: x=2  # no need to declare type

In [2]: x
Out[2]: 2

In [3]: x>5 #comparison opperators like >,<, <=,>= ...
Out[3]: False

In [4]: x==5 #equal to ...
Out[4]: False

In [5]: x!=5 #not equal to ...
Out[5]: True

In [6]: x>1 and x<3 # can be combined with logical opperators and,or, not
Out[6]: True

In [7]: 3+4  # basic arithmetic opperators  + , -,*,/
Out[7]: 7

In [8]: 2**5 # **(power)
Out[8]: 32

In [9]: x+5
Out[9]: 7
```

String

Characters in between

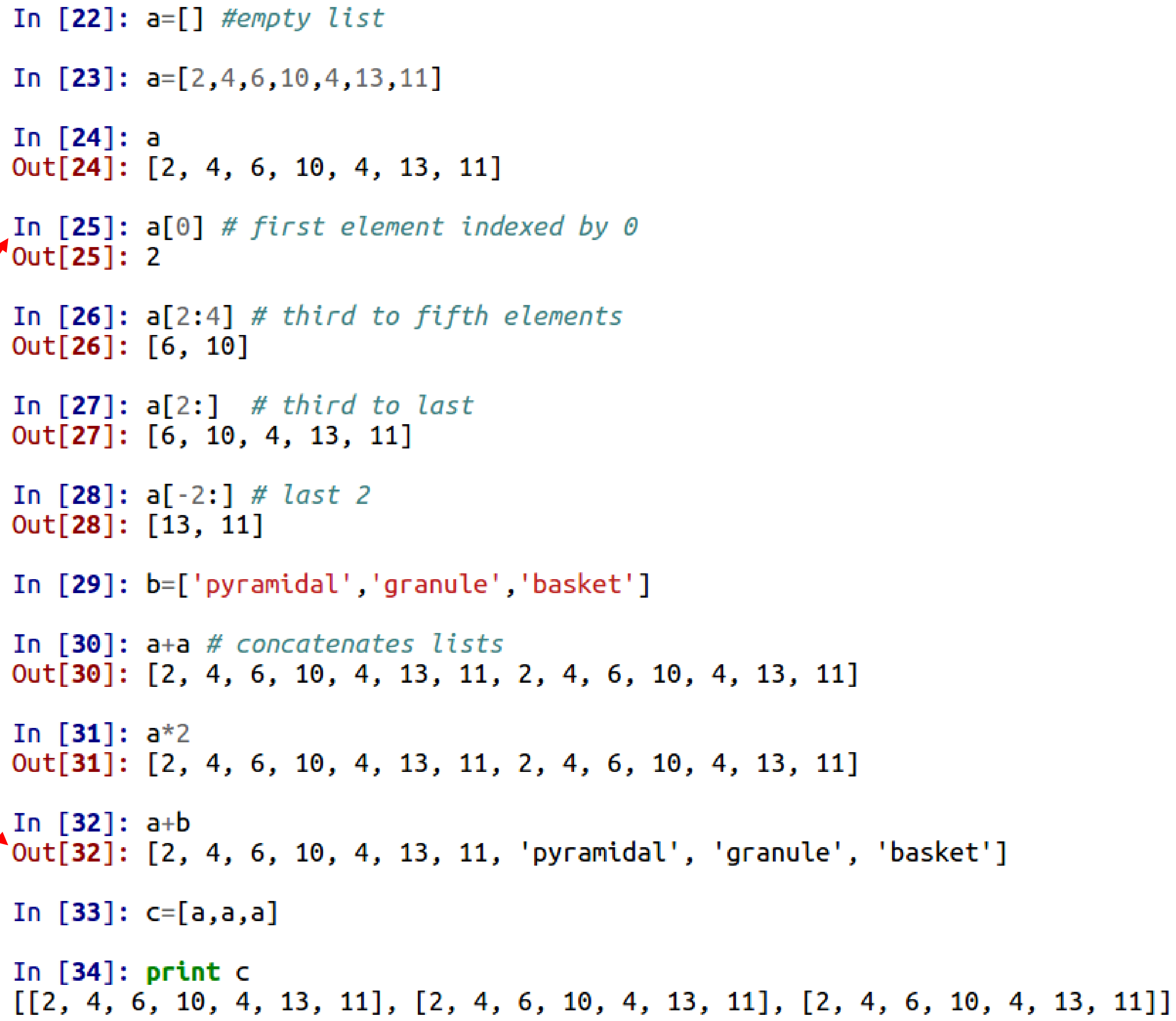
quotes



```
In [10]: 10/2 #integer division ...
Out[10]: 5

In [11]: y=2/10     # yields an integer

In [12]: y
Out[12]: 0

In [13]: type(y)    #Verify type. Python has many nice built-in functions see ref
Out[13]: int

In [14]: 2/10.0       #if numerator or denominator is a float, the result is also float
Out[14]: 0.2

In [15]: y='LASCON' #can reuse names with different obj

In [16]: y='LAS'+'CON '# string concatenation

In [17]: y+2016      # cannot concatenate diferent types
Traceback (most recent call last):

  File "<ipython-input-17-2be68eb080ec>", line 1, in <module>
    y+2016      # cannot concatenate diferent types

TypeError: cannot concatenate 'str' and 'int' objects


In [18]: mystr=y+str(2016) #You can however change to suitable type
```

List

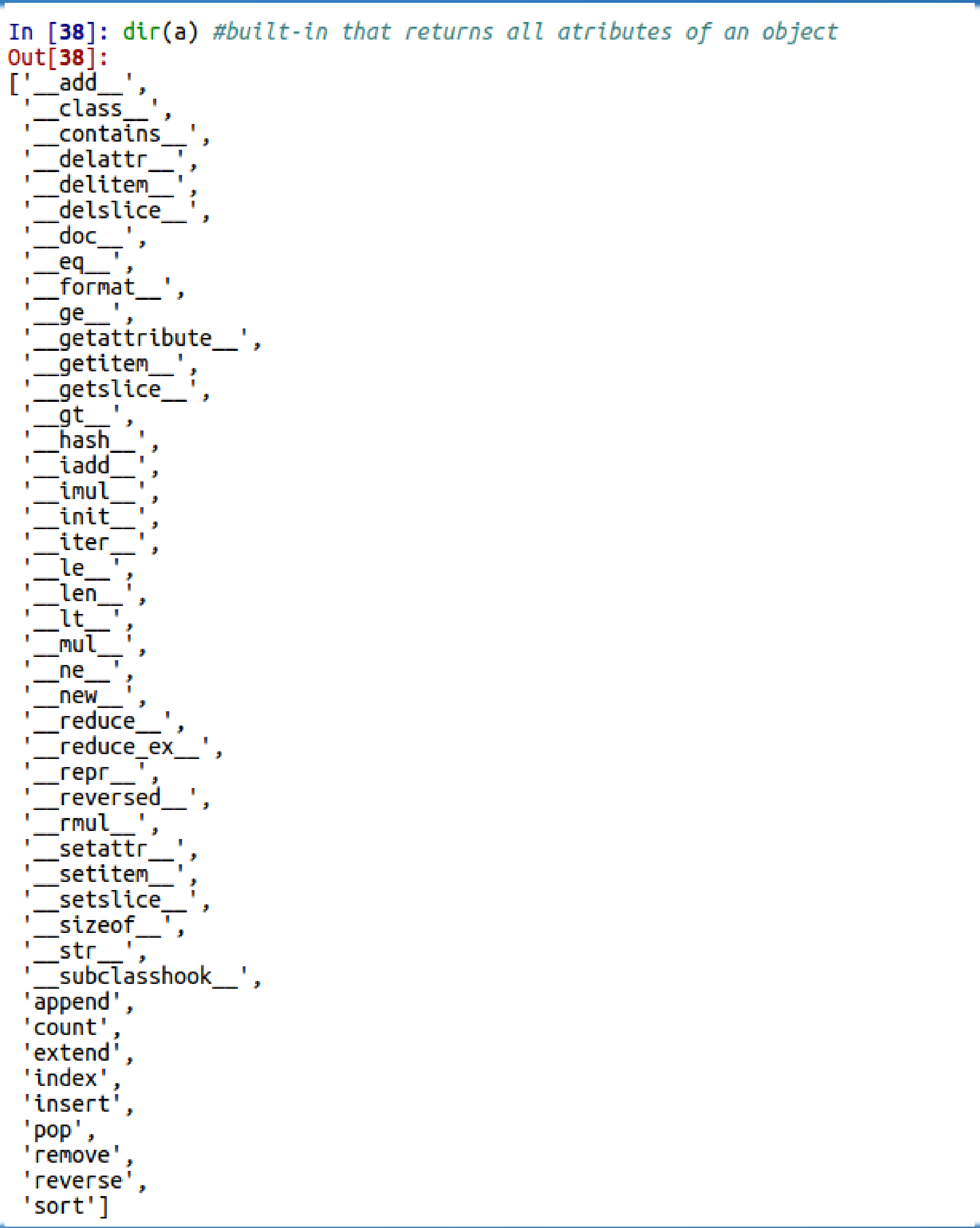
- Sequence of elements defined in between square backets, separated by commas.
- An element is accesed by its position in the list
- Allows mixed data types



```
In [22]: a=[] #empty list

In [23]: a=[2,4,6,10,4,13,11]

In [24]: a
Out[24]: [2, 4, 6, 10, 4, 13, 11]

In [25]: a[0] # first element indexed by 0
Out[25]: 2

In [26]: a[2:4] # third to fifth elements
Out[26]: [6, 10]

In [27]: a[2:]  # third to last
Out[27]: [6, 10, 4, 13, 11]

In [28]: a[-2:] # last 2
Out[28]: [13, 11]

In [29]: b=['pyramidal','granule','basket']

In [30]: a+a # concatenates lists
Out[30]: [2, 4, 6, 10, 4, 13, 11, 2, 4, 6, 10, 4, 13, 11]

In [31]: a*2
Out[31]: [2, 4, 6, 10, 4, 13, 11, 2, 4, 6, 10, 4, 13, 11]

In [32]: a+b
Out[32]: [2, 4, 6, 10, 4, 13, 11, 'pyramidal', 'granule', 'basket']

In [33]: c=[a,a,a]

In [34]: print c
[[2, 4, 6, 10, 4, 13, 11], [2, 4, 6, 10, 4, 13, 11], [2, 4, 6, 10, 4, 13, 11]]
```

dir(obj) function returns all atributes of an obj

Lists methods:
append
count
pop
sort
etc

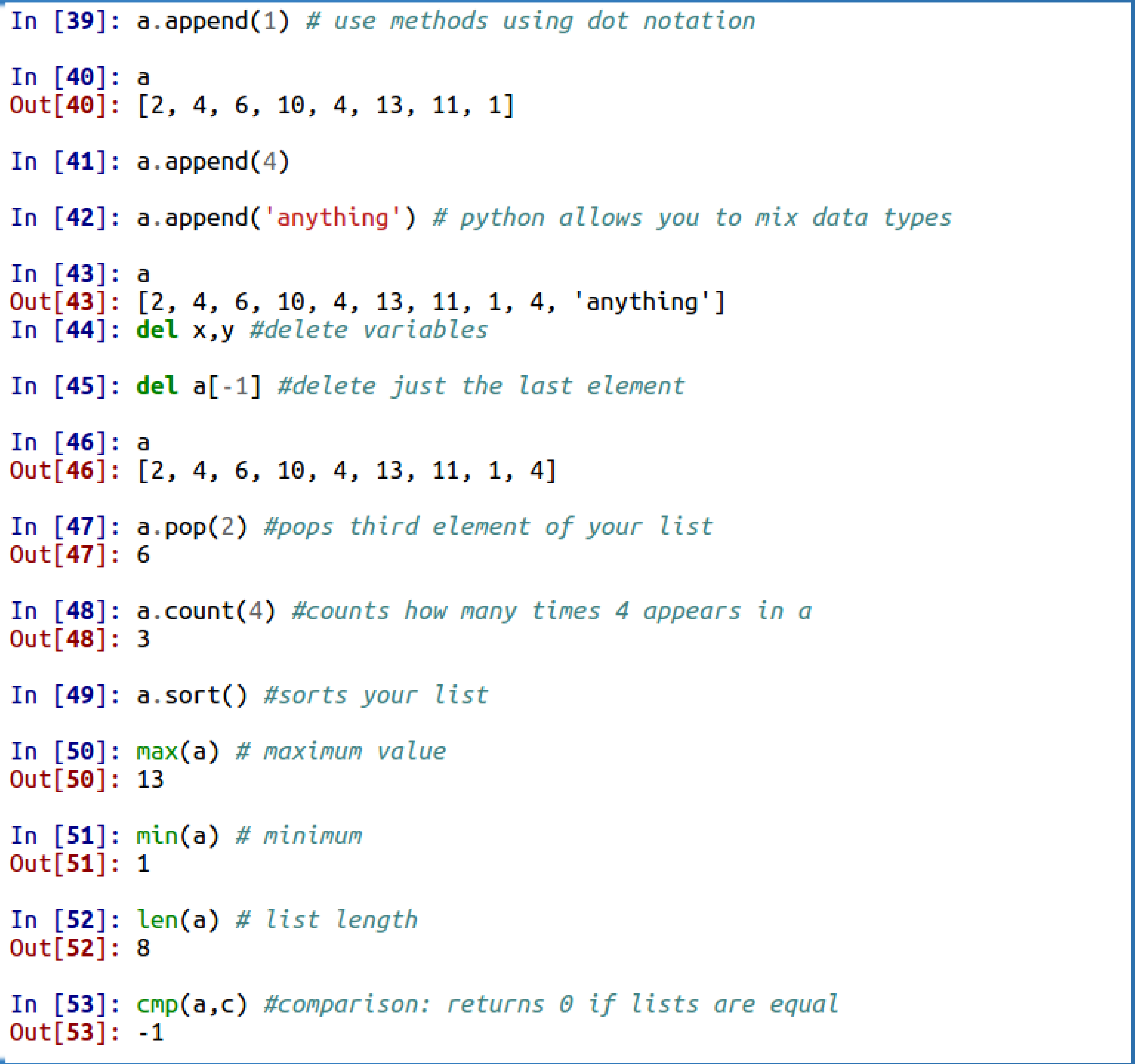


```
In [38]: dir(a) #built-in that returns all atributes of an object
Out[38]:
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__delslice__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getslice__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__setslice__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

Notation: obj.method(args)

dir(obj) function returns all atributes of an obj

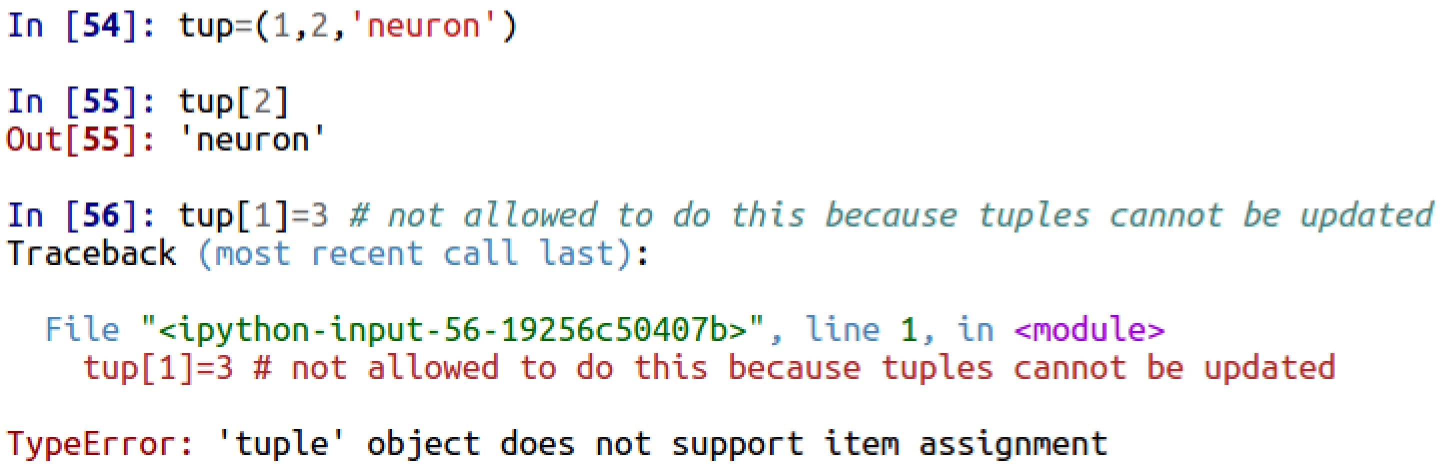
Lists methods:
append
count
pop
sort
etc



```
In [39]: a.append(1) # use methods using dot notation

In [40]: a
Out[40]: [2, 4, 6, 10, 4, 13, 11, 1]

In [41]: a.append(4)

In [42]: a.append('anything') # python allows you to mix data types

In [43]: a
Out[43]: [2, 4, 6, 10, 4, 13, 11, 1, 4, 'anything']
In [44]: del x,y #delete variables

In [45]: del a[-1] #delete just the last element

In [46]: a
Out[46]: [2, 4, 6, 10, 4, 13, 11, 1, 4]

In [47]: a.pop(2) #pops third element of your list
Out[47]: 6

In [48]: a.count(4) #counts how many times 4 appears in a
Out[48]: 3

In [49]: a.sort() #sorts your list

In [50]: max(a) # maximum value
Out[50]: 13

In [51]: min(a) # minimum
Out[51]: 1

In [52]: len(a) # list length
Out[52]: 8

In [53]: cmp(a,c) #comparison: returns 0 if lists are equal
Out[53]: -1
```

Tuples

Sequence of elements.
Immutable type (‘read only’)
Uses parenthesis

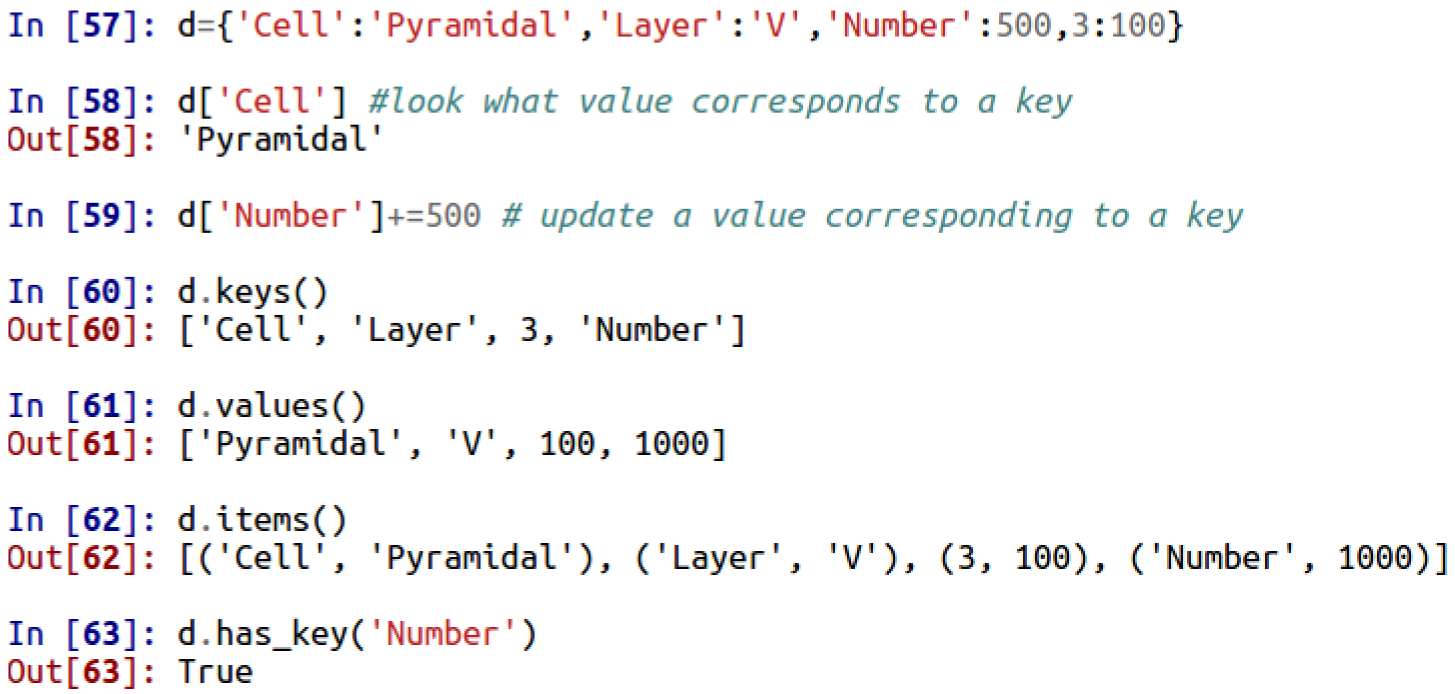


```
In [54]: tup=(1,2,'neuron')

In [55]: tup[2]
Out[55]: 'neuron'

In [56]: tup[1]=3 # not allowed to do this because tuples cannot be updated
Traceback (most recent call last):

  File "<ipython-input-56-19256c50407b>", line 1, in <module>
    tup[1]=3 # not allowed to do this because tuples cannot be updated

TypeError: 'tuple' object does not support item assignment
```

Dictionary

- type where a value (python object) is associated to a key (python type, like a string or number).
- defined between curly braces dic={key1:value1,ke2:value2…}

-Not a sequence:There is no element position or order like in a list



```
In [57]: d={'Cell':'Pyramidal','Layer':'V','Number':500,3:100}

In [58]: d['Cell'] #look what value corresponds to a key
Out[58]: 'Pyramidal'

In [59]: d['Number']+=500 # update a value corresponding to a key

In [60]: d.keys()
Out[60]: ['Cell', 'Layer', 3, 'Number']

In [61]: d.values()
Out[61]: ['Pyramidal', 'V', 100, 1000]

In [62]: d.items()
Out[62]: [('Cell', 'Pyramidal'), ('Layer', 'V'), (3, 100), ('Number', 1000)]

In [63]: d.has_key('Number')
Out[63]: True
```

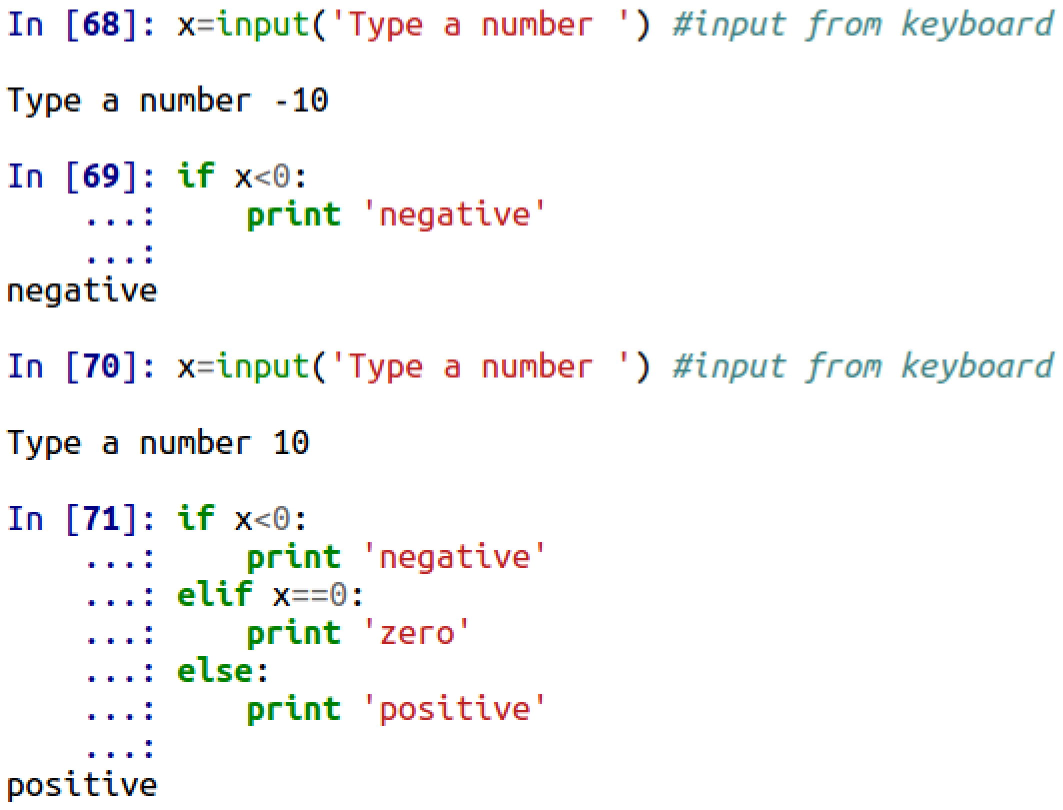
Making decisions with if

NO space

```
if condition:
            statement1
            statement2
else:
            statement3
```

space

INDENTATION IS PARAMOUNT



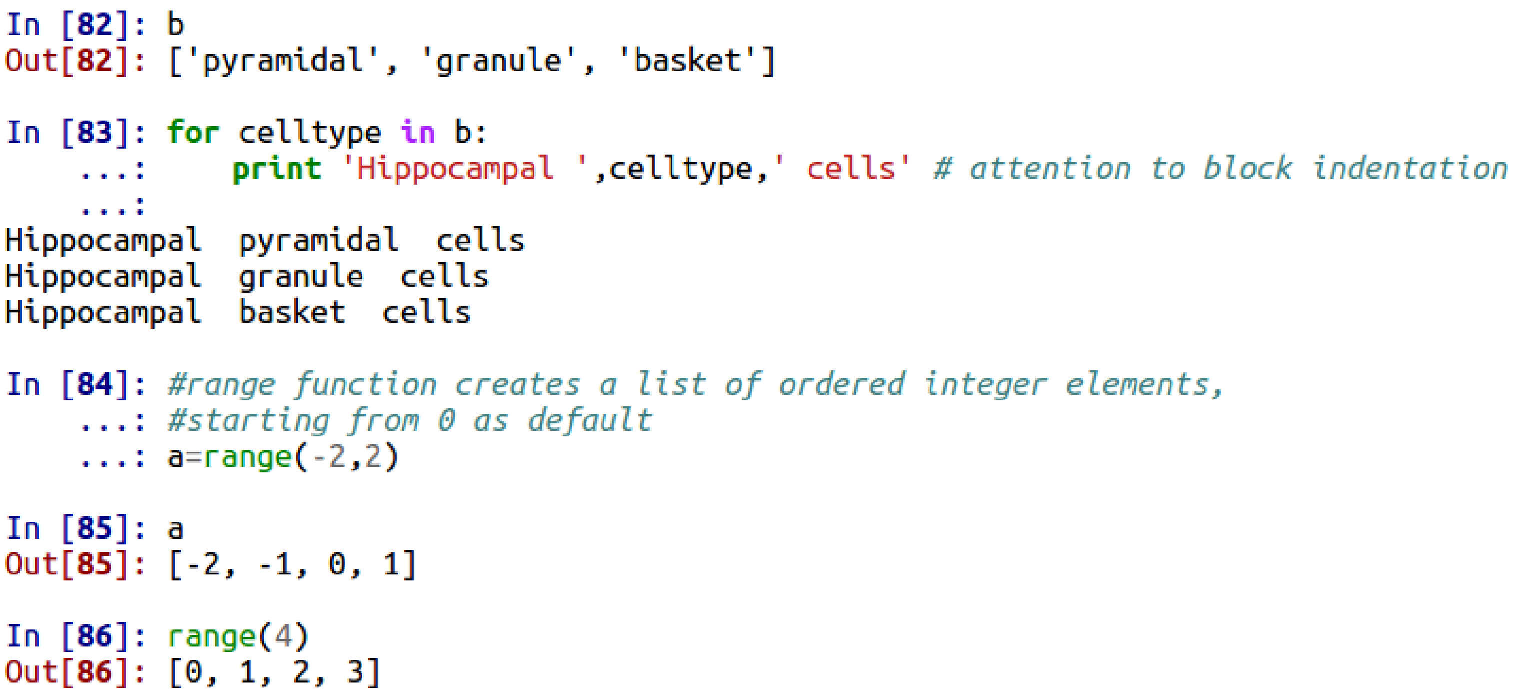
```
In [68]: x=input('Type a number ') #input from keyboard

Type a number -10

In [69]: if x<0:
    ...:     print 'negative'
    ...:
negative

In [70]: x=input('Type a number ') #input from keyboard

Type a number 10

In [71]: if x<0:
    ...:     print 'negative'
    ...: elif x==0:
    ...:     print 'zero'
    ...: else:
    ...:     print 'positive'
    ...:
positive
```

Iterations: list

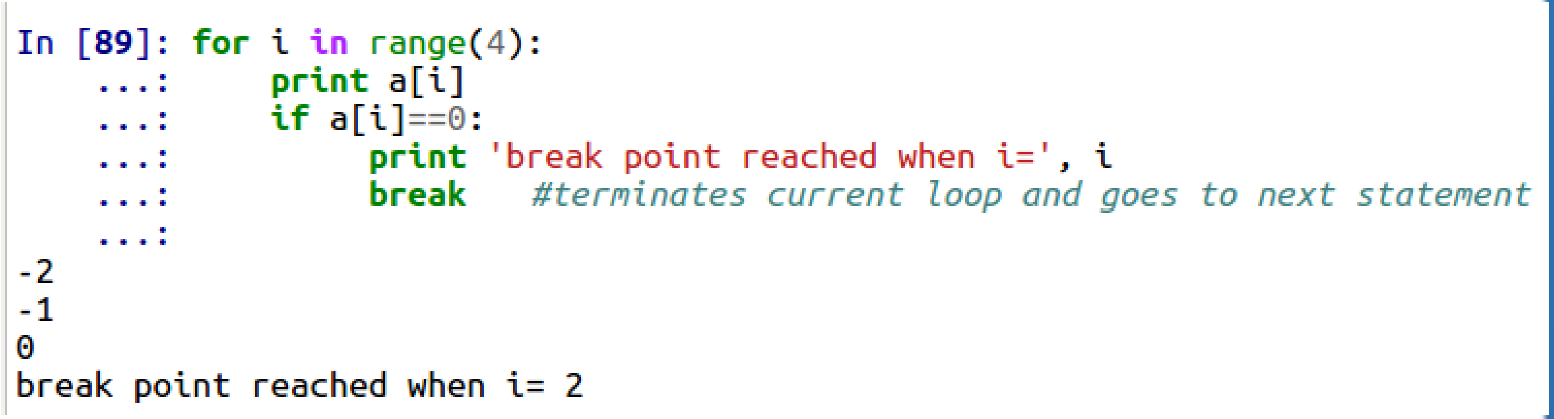
Iterating with for

```
for element in sequence:
        statement1
        statement2

for element in sequence:
        if statement1:
                statement2
                break
```

INDENTATION IS PARAMOUNT



```
In [82]: b
Out[82]: ['pyramidal', 'granule', 'basket']

In [83]: for celltype in b:
    ...:     print 'Hippocampal ',celltype,' cells' # attention to block indentation
    ...:
Hippocampal  pyramidal  cells
Hippocampal  granule  cells
Hippocampal  basket  cells

In [84]: #range function creates a list of ordered integer elements,
    ...: #starting from 0 as default
    ...: a=range(-2,2)

In [85]: a
Out[85]: [-2, -1, 0, 1]

In [86]: range(4)
Out[86]: [0, 1, 2, 3]
```

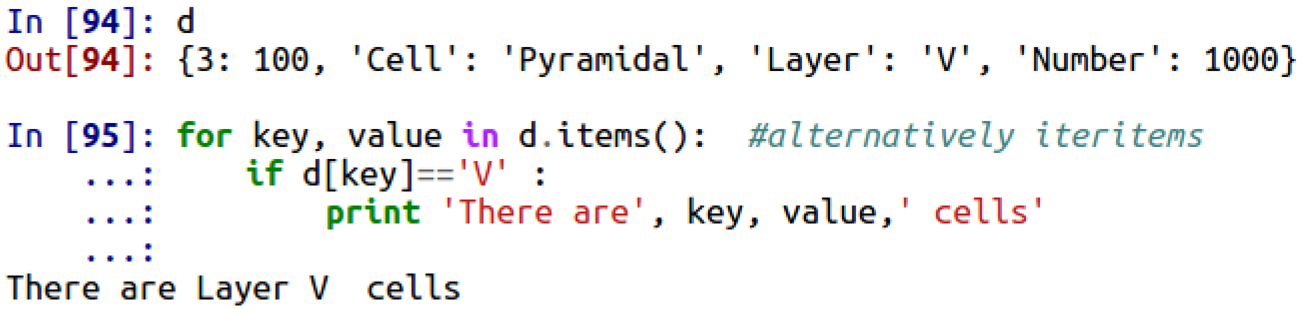


```
In [89]: for i in range(4):
    ...:     print a[i]
    ...:     if a[i]==0:
    ...:         print 'break point reached when i=', i
    ...:         break   #terminates current loop and goes to next statement
    ...:
-2
-1
0
break point reached when i= 2
```

Iterations: dictionary

Iterating with for

```
for element in sequence:
        statement1
        statement2

for element in sequence:
        if statement1:
                statement2
                break
```



```
In [94]: d
Out[94]: {3: 100, 'Cell': 'Pyramidal', 'Layer': 'V', 'Number': 1000}

In [95]: for key, value in d.items():  #alternatively iteritems
    ...:     if d[key]=='V' :
    ...:         print 'There are', key, value,' cells'
    ...:
There are Layer V  cells
```

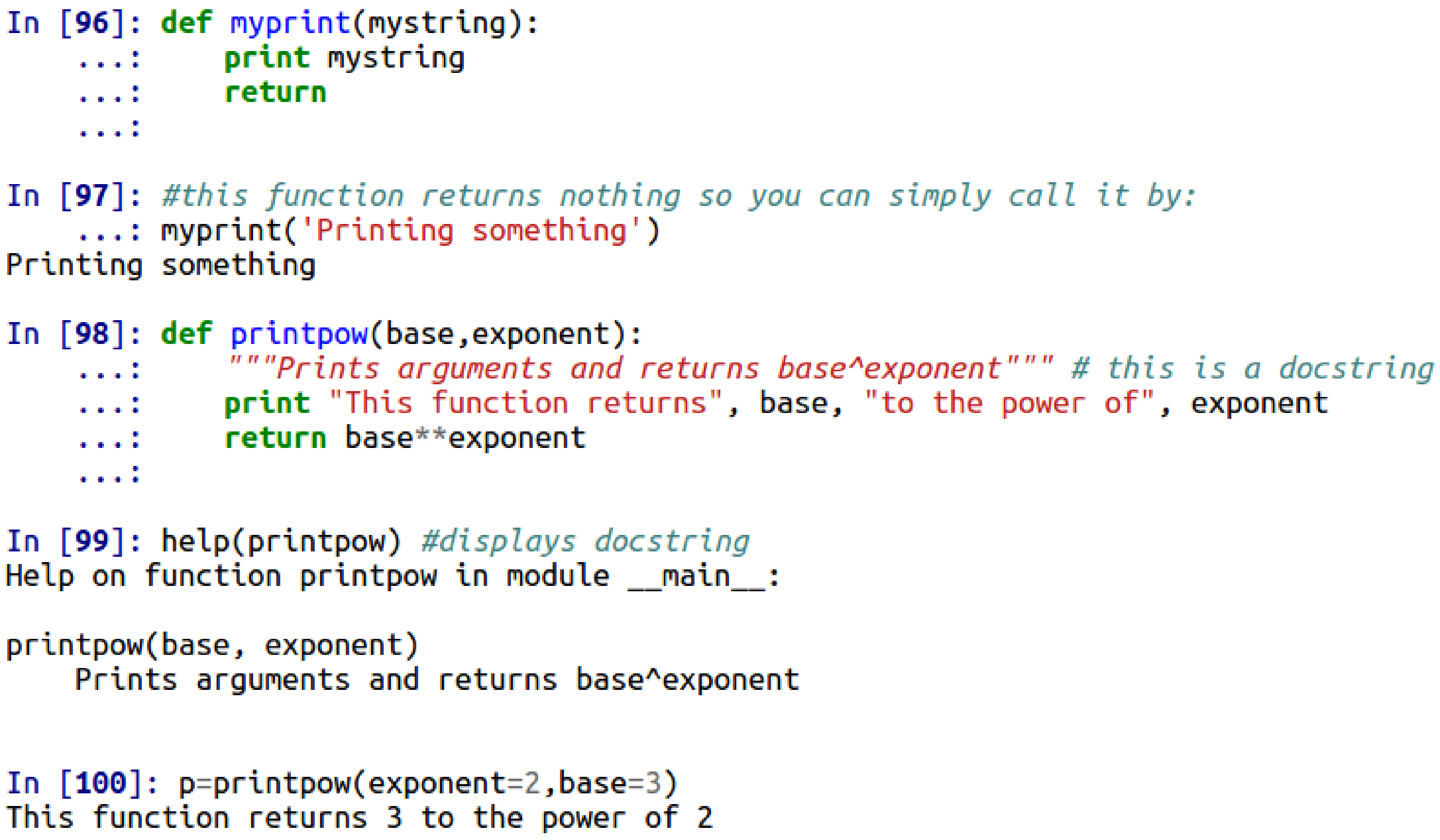
Functions

Block of code that you can reuse.
Step 1- Define your function

```
>> def myfunction (arguments):
          instructions
          return (something)
```

Step 2- call myfunction whenever you want (after you’ve defined it)

```
>> x=myfunction(argument=value)
```



```
In [96]: def myprint(mystring):
    ...:     print mystring
    ...:     return
    ...:

In [97]: #this function returns nothing so you can simply call it by:
    ...: myprint('Printing something')
Printing something

In [98]: def printpow(base,exponent):
    ...:     """Prints arguments and returns base^exponent""" # this is a docstring
    ...:     print "This function returns", base, "to the power of", exponent
    ...:     return base**exponent
    ...:

In [99]: help(printpow) #displays docstring
Help on function printpow in module __main__:

printpow(base, exponent)
    Prints arguments and returns base^exponent


In [100]: p=printpow(exponent=2,base=3)
This function returns 3 to the power of 2
```

Classes

A class is a blueprint or recipe of object

-Define a class

```
class ClassName:
        method1
        method2
```

Cake

-instantiate(create na object)

```
>obj=ClassName(args)
```

```
mycake=Cake(chocolate,big)
```

-access its atributes and methods via dot notation

```
>obj.method1
```

```
>mycake.flavour
chocolate
>mycake.size
big
>mycake.cutapiece()
```

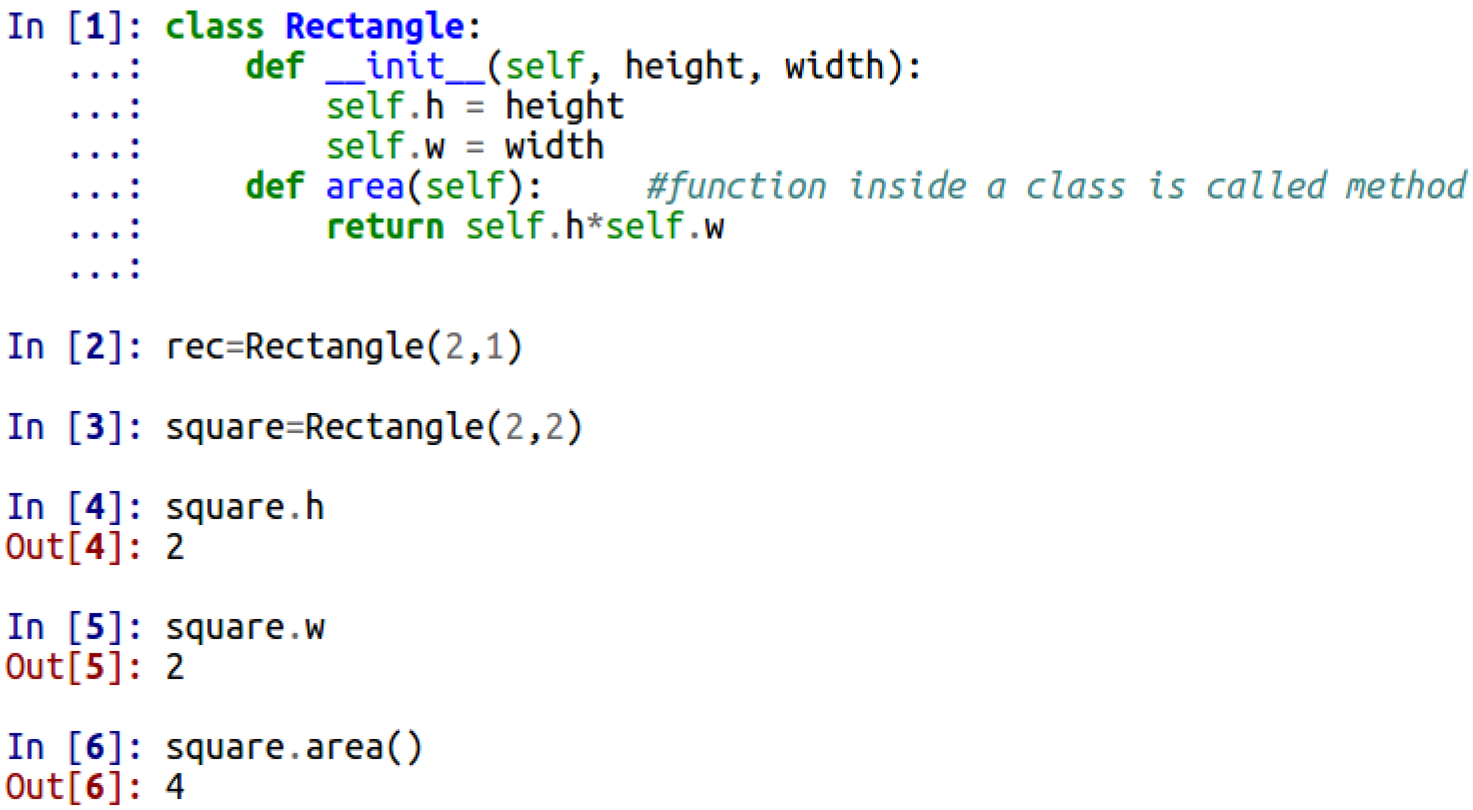
Classes

A class is a blueprint or recipe of object

-Define a class
class ClassName:
            method1
            method2

-instantiate(create na object)
>obj=ClassName(args)

-access its atributes and methods via dot notation
>obj.method1



```
In [1]: class Rectangle:
   ...:     def __init__(self, height, width):
   ...:         self.h = height
   ...:         self.w = width
   ...:     def area(self):     #function inside a class is called method
   ...:         return self.h*self.w
   ...:

In [2]: rec=Rectangle(2,1)

In [3]: square=Rectangle(2,2)

In [4]: square.h
Out[4]: 2

In [5]: square.w
Out[5]: 2

In [6]: square.area()
Out[6]: 4
```

Modules

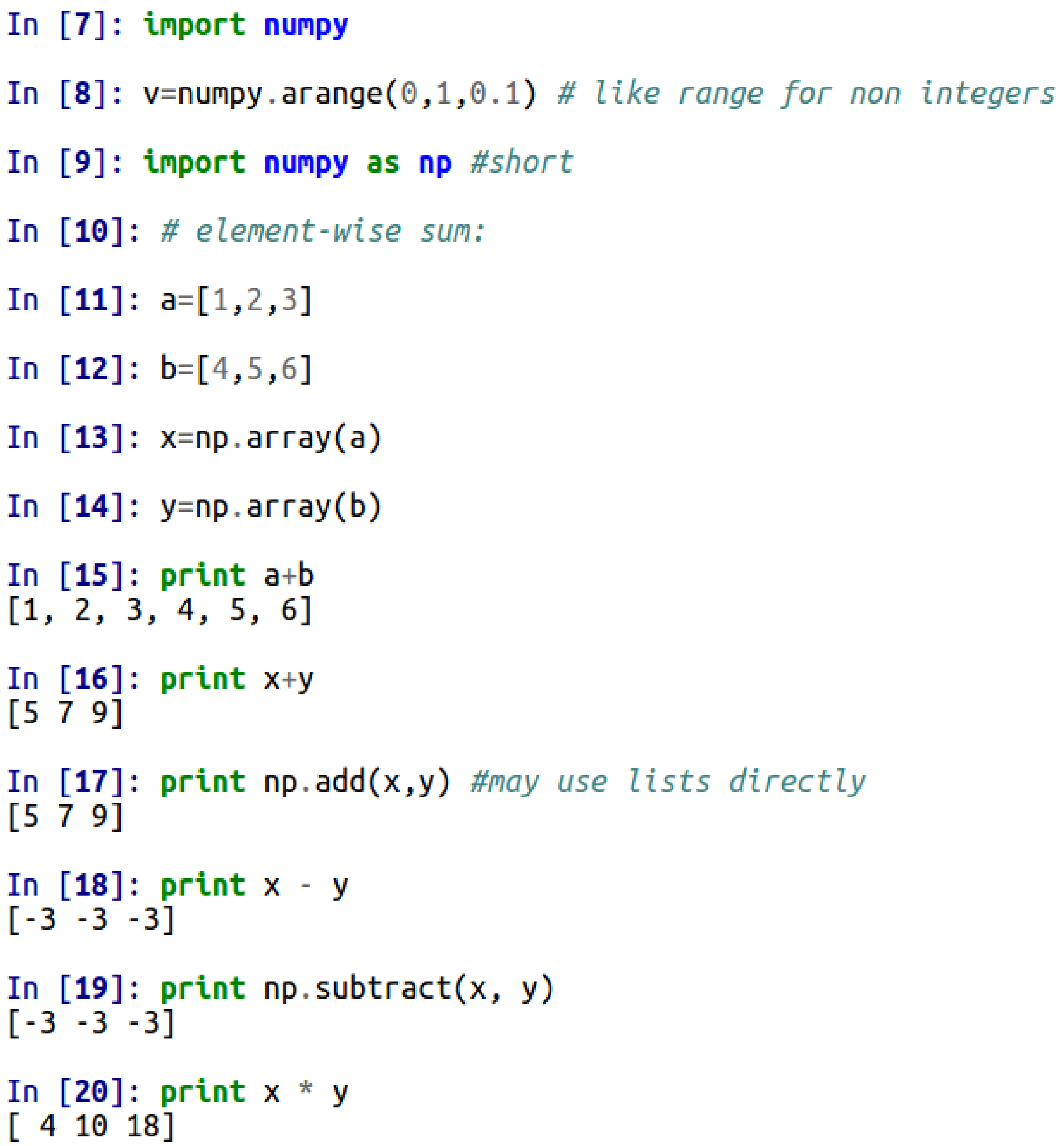
File with python code that contain variables, objects, methods, and functions that you can import and use
via dot notation

import module

from module import somethings

from module import something as nickname

numpy
matplotlib
neuron



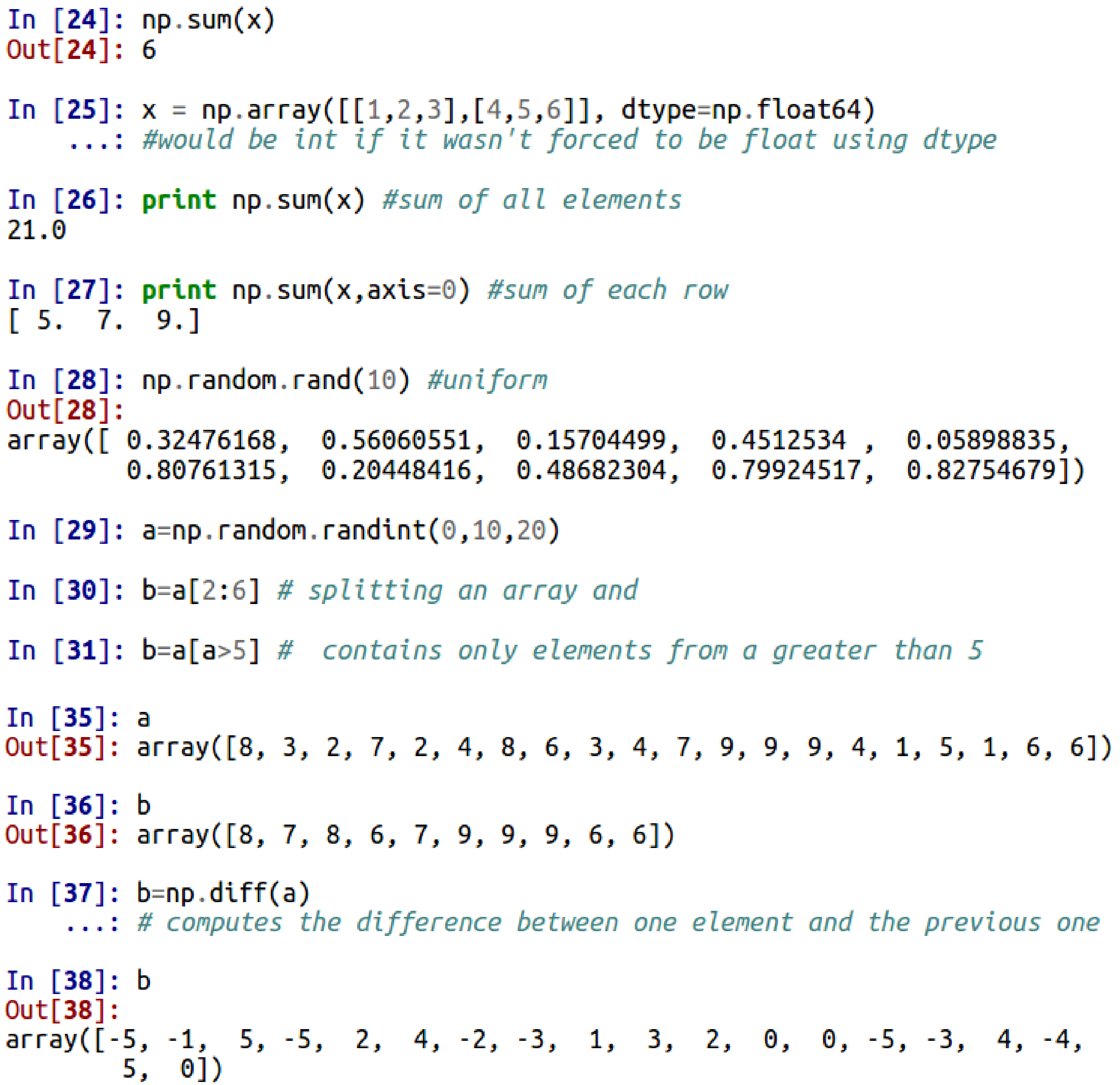
```
In [7]: import numpy

In [8]: v=numpy.arange(0,1,0.1) # like range for non integers

In [9]: import numpy as np #short

In [10]: # element-wise sum:

In [11]: a=[1,2,3]

In [12]: b=[4,5,6]

In [13]: x=np.array(a)

In [14]: y=np.array(b)

In [15]: print a+b
[1, 2, 3, 4, 5, 6]

In [16]: print x+y
[5 7 9]

In [17]: print np.add(x,y) #may use lists directly
[5 7 9]

In [18]: print x - y
[-3 -3 -3]

In [19]: print np.subtract(x, y)
[-3 -3 -3]

In [20]: print x * y
[ 4 10 18]
```

Modules

Numpy

Elementwise opperations
List -> np.array

Pseudo-random generators



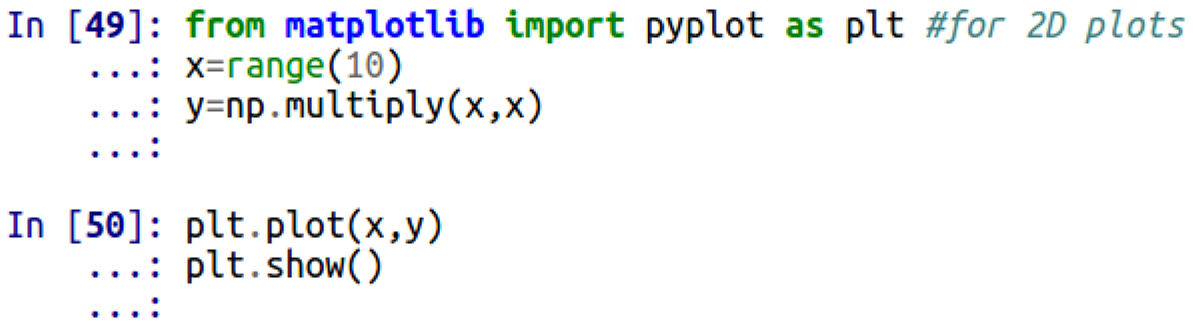
```
In [24]: np.sum(x)
Out[24]: 6

In [25]: x = np.array([[1,2,3],[4,5,6]], dtype=np.float64)
    ...: #would be int if it wasn't forced to be float using dtype

In [26]: print np.sum(x) #sum of all elements
21.0

In [27]: print np.sum(x,axis=0) #sum of each row
[ 5.  7.  9.]

In [28]: np.random.rand(10) #uniform
Out[28]:
array([ 0.32476168,  0.56060551,  0.15704499,  0.4512534 ,  0.05898835,
        0.80761315,  0.20448416,  0.48682304,  0.79924517,  0.82754679])

In [29]: a=np.random.randint(0,10,20)

In [30]: b=a[2:6] # splitting an array and

In [31]: b=a[a>5] #  contains only elements from a greater than 5

In [35]: a
Out[35]: array([8, 3, 2, 7, 2, 4, 8, 6, 3, 4, 7, 9, 9, 9, 4, 1, 5, 1, 6, 6])

In [36]: b
Out[36]: array([8, 7, 8, 6, 7, 9, 9, 9, 6, 6])

In [37]: b=np.diff(a)
    ...: # computes the difference between one element and the previous one

In [38]: b
Out[38]:
array([-5, -1,  5, -5,  2,  4, -2, -3,  1,  3,  2,  0,  0, -5, -3,  4, -4,
        5,  0])
```

Modules

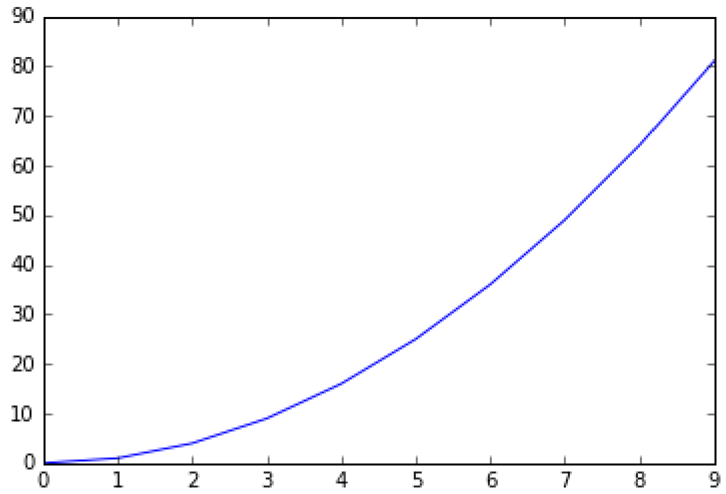
Matplotlib

Nice 2D plots

SIMPLE PLOT



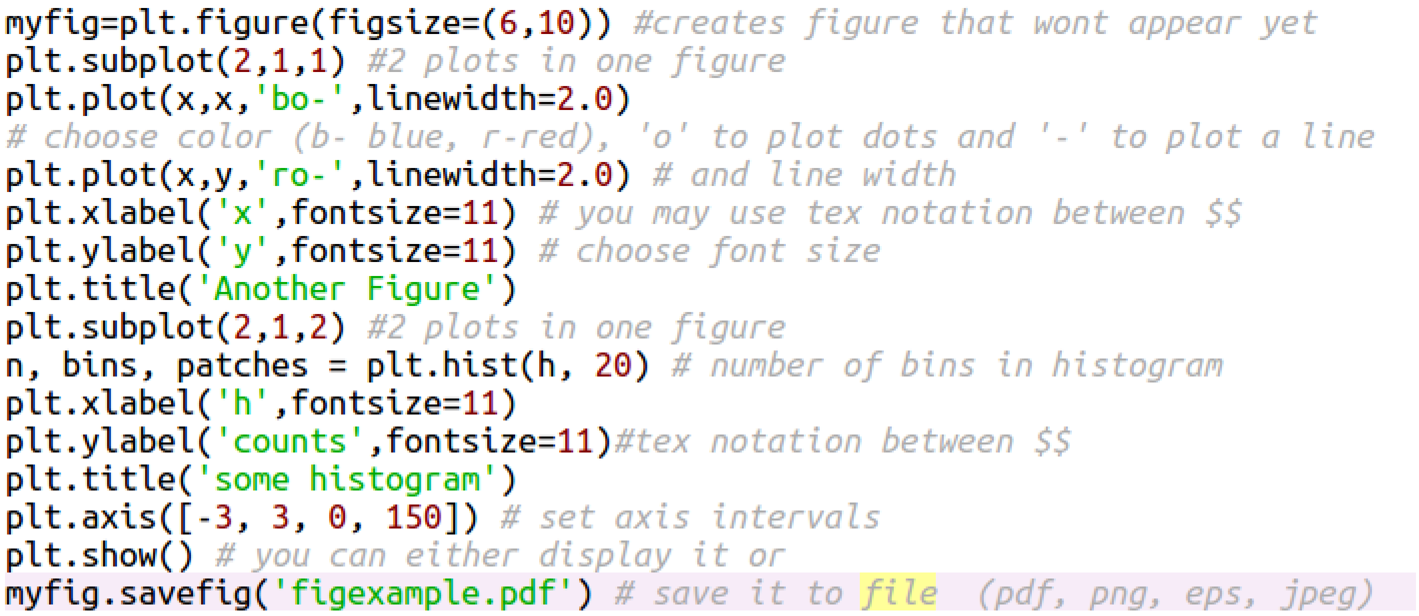
```
In [49]: from matplotlib import pyplot as plt #for 2D plots
    ...: x=range(10)
    ...: y=np.multiply(x,x)
    ...:

In [50]: plt.plot(x,y)
    ...: plt.show()
    ...:
```



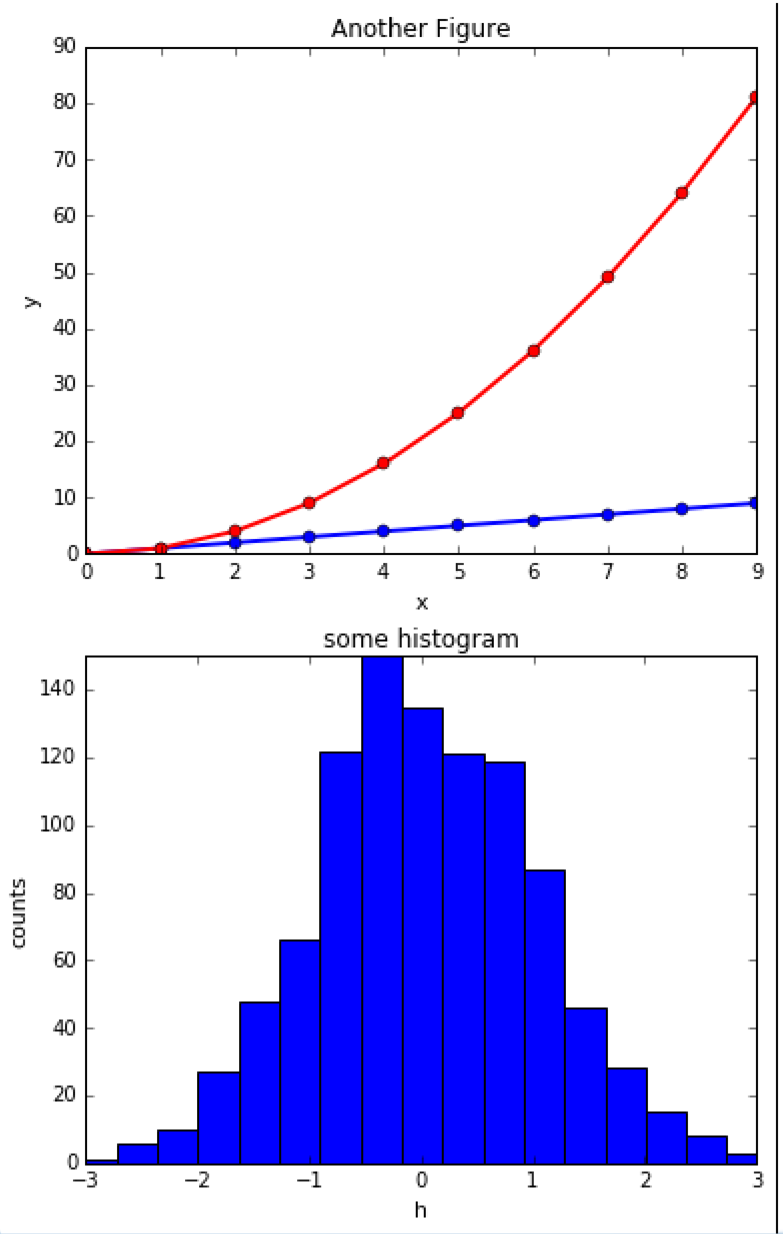
Modules

Matplotlib

Nice 2D plots



```
myfig=plt.figure(figsize=(6,10)) #creates figure that wont appear yet
plt.subplot(2,1,1) #2 plots in one figure
plt.plot(x,x,'bo-',linewidth=2.0)
# choose color (b- blue, r-red), 'o' to plot dots and '-' to plot a line
plt.plot(x,y,'ro-',linewidth=2.0) # and line width
plt.xlabel('x',fontsize=11) # you may use tex notation between $$
plt.ylabel('y',fontsize=11) # choose font size
plt.title('Another Figure')
plt.subplot(2,1,2) #2 plots in one figure
n, bins, patches = plt.hist(h, 20) # number of bins in histogram
plt.xlabel('h',fontsize=11)
plt.ylabel('counts',fontsize=11)#tex notation between $$
plt.title('some histogram')
plt.axis([-3, 3, 0, 150]) # set axis intervals
plt.show() # you can either display it or
myfig.savefig('figexample.pdf') # save it to file  (pdf, png, eps, jpeg)
```



NEURON-python tutorial
http://neuron.yale.edu/neuron/static/docs/neuronpython/index.html

numpy for matlab users
https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

matplotlib
http://matplotlib.org/users/pyplot_tutorial.html

Reference: Hines, Davison and Muller “NEURON and Python”, Frontiers in neuroinformatics, 2009

Try running and plotting Fig.3 example -> HDM2009_ex2.py