Lesson 4

Numpy Math and Arrays

Now that you've made your own module full of functions, you can understand that other people will have done the same. The most useful for our purposes is called "numpy", which allows mathematics to be applied to arrays of numbers. Because we will be using several of these numpy functions, we will use the module.function method to import and use functions.

The following can either be typed into a script or run line by line at the python command line. Below I list them as a script.

---
array_tests.py

```
#library
import numpy as np

# Commands
# The Difference between Lists and Arrays
testlist = [3,4,6,5]
print( testlist + [2] )
testlist = testlist + [2]
testlist.append(5)
print( testlist*2 ) # duplicate the list

testarray = np.array(testlist)
print( testarray + [2] ) #same as print( testarray + 2 )
print( testarray*2 )

# Multi-Dimensional Arrays and Indices
a = np.array( [ [1,2,3] , [4,5,6] ] )
print( a )
print( a[1,2] ) #[row,column]
print( a[0:2,1] )
print( np.size(a))
print(np.shape(a) )
dims = np.shape(a)
print(dims)
print(dims[0]) # number of rows
print(dims[1]) # number of columns

b = np.zeros( [3,4], dtype=int)
b[1,2] = 9
print( b )
```

```
b[1,0:3] = [1,2,3]
print( b )

# Array Searches
non0_indices = (b > 0)
print( non0_indices )
print( np.count_nonzero(non0_indices) )
print( b[non0_indices] )
b[non0_indices] = 9
print( b )
```

Dictionary

The purpose of a dictionary in Python is to efficiently store and retrieve data by associating values with unique keys. By using key-value pairs, dictionaries provide a flexible and organized approach to managing data.

Example:

Create a dict_script.py

```
data = {
   "name": "Said",
   "age": 24,
   "city": "New York"
}
# where “name”, “age”, ”city” are the keys
# and “Said”, 26, “New York”
print(‘Name’, data[‘name])

# let’s modify the age
data["age"] = 26

# Adding a new key-value pair
data["occupation"] = "Research Assistant"

print(data)

# if you want to add another person, you can do this
data["name2"] = "Ana"
```

Files: Reading and Writing

Data is stored in files, and the details of how you read from or write to a file depends on the type of file. There are two basic file types: ASCII/text files that are readable by both humans and computer, and binary files that can only be read by computer, but are extremely fast for computers to read so are used for large files. We will only be looking at text files in this class.

We will be going through the file pyreadtest.txt, line by line. Open it in textwrangler to follow along. The data in it is for demonstration purposes only. Each line from this file will appear in quotes, the python commands used to read them will appear with the familiar '>>' prompt, though you may want to put all the commands in a module. You will want to open it using notepad++ separately to look at as we go through it, but if just reading through this guide, here it what is in it for future reference:

pyreadtest.txt

---

python test read file
3 4
3 rows 4 cols
18/06/2016
1 2 3 4
5 6 7 8
9 10 11 12
done

You'll see that some lines are pure text; some are pure numbers. Some represent useful data; some are just descriptive information. Our task is to pull out the useful data, put it in variables, and discard the descriptive information.

Here we go:

First, we are going to create a new python script called 'read_textfile.py'

Now, we have to open the file. Your file path may be different from this one.

#Library

```
import os
os.chdir(r'file_location')
```

#Commands

```
>> f = open('pyreadtest.txt' , 'r')
```

So we have opened the file and given it the file ID 'f' – and we use file IDs because who would want to spell out that whole file name each time? We have also opened it using the 'r' option to show that we want to read it only. This keeps us from accidentally writing to it.

Line 1: "python test read file"

This is a typical 'header line' to tell any humans what the file is about. We probably could not use it for any useful calculations, but we must read past it to get to the data. Since it's a character string we will read it as a line of text, which we will put into the variable we've decided to call 'textline'.

```
>> textline = f.readline()
```

If you want to know if textline includes that first line, just print it to see. Note that the readline function is attached to the file f, so if we have more than one file, we tell it which one to read from. The () tells us this function may have other options, but we will not need them.

Line 2: "3 4"

This will actually be the dimensions of an array further down. We will need these numbers and so will do the following set of commands to read the line as a character string then split it up into numbers.

```
>>  dimstr = f.readline()          # read a line of text
>> dimlist = dimstr.split()       # split at whitespace
>> rows = int(dimlist[0])         # convert each item to a number
>> cols = int(dimlist[1])
```

You can print out the rows and columns to see that we got it right. But what do we do if we see a line like this?

Line 3: " rows 3 cols 4"

This is meant to be helpful to humans, but a computer has to throw away the extra words to get the numbers. So we do almost the same thing as above.

```
>> dimstr = f.readline()          # read a line of text
>> dimlist = dimstr.split()       # split at white space
>> rows = int(dimlist[0])
>> cols = int(dimlist[2])
```

A slightly more complicated situation is found in the common way to write a date:

Line 4: "06/18/2016"

We will want to split this into day, month and year.

```
>>  datestr = f.readline()              # read a line of text
>> datelist= datestr.split('/')        # split at each slash
>> day = datelist[0]
>> month = datelist[1]
>> year = datelist[2]
```

So you could use this to pull out all files of a given month, for example.

Now that we have the dimensions and date for our array of data, let's read it!

Lines 5-8:
"1 2 3 4"
"5 6 7 8"
"9 10 11 12"

This is an array of numbers, so if you haven't done it yet,

```
>> import numpy as np
```

Now there are two ways to proceed: define an array of the right size first, and feed data in line by line, or read all the data as a vector, then reform it into an array. We will take the first approach, as it is actually more flexible, though slower on big files.

```
>> data_array = np.zeros([rows,cols],dtype=int)
>> for R in range(0,rows):
…       textline = f.readline()
…       data_array[R, : ] = np.fromstring(textline, dtype='int', sep='  ')
```

Try printing the data_array to see if it did it right. After this is only one line left:

Line 9: "done"

We don't need this, we can either read it, or just close the file

```
>> f.close()
```

It's important to close your files or else you can't use the variable 'f' again – you'd have to use different symbols. Also, when you write to a file, you can't see your file until closed.

The second approach only works if the data you are reading goes all the way to the end of the file. The word 'done' at the end will crash it, so you have to delete that last line and save the file again. Reopen the file and read to the point before the array. Then

```
>> vectorstr = f.read()
>> vector = np.fromstring(vectorstr, dtype=int, sep=' ')
```

```
>> data_array = vector.reshape(rows,cols)
```

This would be a faster and easier to read a data file. But it only works if the data array goes clear to the end of the file, and if you have more than two dimensions the reshape function will sometimes do strange things.

Writing to a Text File

Python will write character strings to a file line by line. This means any numbers must be converted to strings. So if we wanted to write the dimensions of the above data_array and the array itself to a file, we'd do it like so:

```
f = open('filename.txt', 'w')              # open the file to write
f.write('This file will storage an array \n')          # writing a phrase
f.write(str(data_array)) # converting the array into str
f.close()                                              # close
```

You can open the file in the text editor to see if it came out as expected.

Note that you can open files to read only: 'r' or to append to end of file: 'a'