Python lab 4: Input and Output

Dr Ben Dudson

Department of Physics, University of York

18^{th} February 2011

http://www-users.york.ac.uk/~bd512/teaching.shtml

From last time...

- Last two labs we've started using NumPy and Matplotlib to create arrays and plot data in 1 and 2 dimensions
- By now you should have a reasonably good idea how to calculate mathematical functions in Python and plot the results in a variety of ways
- Those of you who have done the extra tasks will also have seen how to make simple (but pretty!) animations

This time, we'll look at input and output

Saving to file

Now we've done all these calculations, we need a way to save the results to a file.

- As a record of what was done, so can refer to later
- To transfer to a different program for further analysis
- If the calculation took a long time to do, then we can save the results and then re-use later

Writing to files

In Python, instead of printing results to screen using **print**, we can write our results to a file. A simple example is:

```
f = open('output.txt', 'w')
f.write('Hello')
f.close()
```

Open the file "output.txt", and you'll see it contains

```
Hello
```

Writing to files

In Python, instead of printing results to screen using **print**, we can write our results to a file. A simple example is:

```
f = open('output.txt', 'w')
f.write('Hello')
f.close()
```

Open the file "output.txt", and you'll see it contains

```
Hello
```

The main steps are:

```
f = open('output.txt', 'w')   # Opens a file for writing
f.write('Hello')              # Writes "Hello" to the file
f.close()                     # Close the file again
```

Writing to files

- Unfortunately, the write function can only write text, not other variables like numbers or arrays.
- To print other variables to a file, we first need to convert them to text. To do this, Python has the str() function:

```
x = random.random(5)
f = open('output.txt', 'w')
f.write(str(x))
f.close()
```

Here str(x) converts x from an array into text (a **string**). write() then takes this text and writes it to the file.

Converting to text

- str() is the simplest way to convert values to text
- If we want to write out a line of text with two values x and y, we could use

```
f.write(str(x))
f.write(" ")
f.write(str(y))
f.write("\n")
```

Converting to text

- str() is the simplest way to convert values to text
- If we want to write out a line of text with two values x and y, we could use

```
f.write(str(x))
f.write(" ")
f.write(str(y))
f.write("\n")
```

This writes some space between x and y, then a **new line** \n to start a new line. A neater way is to join the text together first before writing:

```
f.write(str(x) + " " + str(y) + "\n")
```

Converting to text

- str() is the simplest way to convert values to text
- If we want to write out a line of text with two values x and y, we could use

```
f.write(str(x))
f.write(" ")
f.write(str(y))
f.write("\n")
```

  This writes some space between x and y, then a **new line** \n to start a new line. A neater way is to join the text together first before writing:

```
f.write(str(x) + " " + str(y) + "\n")
```

- A final way which allows more control over your text is to use

```
f.write("%f %f\n" % (x, y))
```

Reading from files

- Having written all this data to file, at some point we will probably want to read it back into Python
- This is usually slightly more tricky because our program has to understand text and turn it back into numbers, arrays etc.

Reading from files

- Having written all this data to file, at some point we will probably want to read it back into Python
- This is usually slightly more tricky because our program has to understand text and turn it back into numbers, arrays etc.
- This is a big subject, so here we'll just study a common situation you'll encounter: a table of numbers separated with spaces or tabs

```
0.0  0.972448511667
0.5  0.370089336609
1.0  0.472616491322
1.5  0.985337885974
...
```

Reading from files

- We need to read the file bit by bit from start to end
- First step is to read in the file line by line

```
f = open('output.txt', 'r')
line = f.readline()
print line
f.close()
```

This opens the file, reads in one line, prints out the text and closes the file

Reading from files

- We need to read the file bit by bit from start to end
- First step is to read in the file line by line

```
f = open('output.txt', 'r')
while True:
    line = f.readline()
    if len(line) == 0:
        break
    print line
f.close()
```

This goes around in a loop loop, reading in one line at a time until it reaches an empty line (end of file)

Reading from files

- We need to read the file bit by bit from start to end
- First step is to read in the file line by line

```
f = open('output.txt', 'r')
while True:
    line = f.readline()
    if len(line) == 0:
        break
    print line
f.close()
```

This goes around in a loop loop, reading in one line at a time until it reaches an empty line (end of file)

```
0.0  0.972448511667
```

Reading from files

- We need to read the file bit by bit from start to end
- First step is to read in the file line by line

```
f = open('output.txt', 'r')
while True:
    line = f.readline()
    if len(line) == 0:
        break
    print line
f.close()
```

This goes around in a loop loop, reading in one line at a time until it reaches an empty line (end of file)

```
0.0  0.972448511667

0.5  0.370089336609
```

Reading from files

- We need to read the file bit by bit from start to end
- First step is to read in the file line by line

```
f = open('output.txt', 'r')
while True:
    line = f.readline()
    if len(line) == 0:
        break
    print line
f.close()
```

This goes around in a loop loop, reading in one line at a time until it reaches an empty line (end of file)

```
0.0  0.972448511667

0.5  0.370089336609

1.0  0.472616491322
```

Reading from files

Now we have split the file into lines so can deal with one at a time

```
0.0   0.972448511667
```

Reading from files

Now we have split the file into lines so can deal with one at a time

```
0.0   0.972448511667
```

Next step is to split this line into individual numbers. Luckely Python comes with a way to split up text.

```
print line                    -> "0.0   0.972448511667"
```

Reading from files

Now we have split the file into lines so can deal with one at a time

```
0.0   0.972448511667
```

Next step is to split this line into individual numbers. Luckely Python comes with a way to split up text.

```
print line                  -> "0.0   0.972448511667"

print line.split()          -> ['0.0', '0.972448511667']
```

This produces a list of strings, one for each number. We can then print one at a time

Reading from files

Now we have split the file into lines so can deal with one at a time

```
0.0   0.972448511667
```

Next step is to split this line into individual numbers. Luckely Python comes with a way to split up text.

```
print line                  -> "0.0   0.972448511667"

print line.split()          -> ['0.0', '0.972448511667']
```

This produces a list of strings, one for each number. We can then print one at a time

```
columns =  line.split()
print columns[0]            -> "0.0"
print columns[1]            -> "0.972448511667"
```

Reading from files

- We now have the strings for the individual numbers. Note that this is not the same as having the numbers: we need to convert the text into numbers using float ()

Reading from files

- We now have the strings for the individual numbers. Note that this is not the same as having the numbers: we need to convert the text into numbers using float()
- A program to read this file into Python line by line, split into columns, and convert the first two columns into numbers is:

```
f = open('output.txt', 'r')
while True:
    line = f.readline()
    if len(line) == 0:
        break
    columns = line.split()
    x = float(columns[0])
    y = float(columns[1])
    print x, y
f.close()
```

Pickling

Because reading and writing to file is so common, Python has a convenient way to store values in a file and retrieve them later. Say we want to save an array to a file

```
from numpy import *
import pickle

x = linspace(0,10,5)

output = open('test.txt', 'wb')
pickle.dump(x, output)
output.close()
```

Pickling

Later we can read this file using

```
from numpy import *
import pickle

file = open('test.txt', 'rb')
x = pickle.load(file)
file.close()
print x
```

```
[  0.    2.5   5.    7.5  10. ]
```

Saving multiple variables

To save several variables into a file, just use pickle.dump() once for each variable

```
x = linspace(0,10,5)
t = "Some text"
output = open('test.txt', 'wb')
pickle.dump(x, output)
pickle.dump(t, output)
output.close()
```

Saving multiple variables

To save several variables into a file, just use pickle.dump() once for each variable

```
x = linspace(0,10,5)
t = "Some text"
output = open('test.txt', 'wb')
pickle.dump(x, output)
pickle.dump(t, output)
output.close()
```

and then read them in the **same order**

```
input = open('test.txt', 'wb')
x = pickle.load(input)
t = pickle.load(input)
input.close()
```

Pickle files

What do these pickle files look like?

```
cnumpy.core.multiarray
_reconstruct
p0
(cnumpy
ndarray
p1
(I0
tp2
S'b'
p3
tp4
...
```

Pickle files

What do these pickle files look like?

```
cnumpy.core.multiarray
_reconstruct
p0
(cnumpy
ndarray
p1
(I0
tp2
S'b'
p3
tp4
...
```

- Can recognise some things in there like "numpy"
- Files which are easy to read tend not to be efficient
- This file format is specific to Python so can't be transferred between programs easily
- This isn't a problem as long as we only want to use it from Python

Some alternatives

There are many alternatives to pickling which are widely used but may be less convenient

- For Comma Separated Variables (CSV) files, there is a Python module called csv
- JavaScript Object Notation (JSON) is widely used, particularly in web applications. It's text-based so not suitable for large arrays of data. There's a built-in Python module (in version 2.6 and higher) called json for this format
- NetCDF, HDF5 are common formats for storing large arrays of data. Modules are available for Python
- Protocol buffers. This is a binary format developed at Google which also has modules for lots of languages. Modules are available for Python

See `http://docs.python.org/library/` for module documentation

Opening and closing files

- Files are another type of variable (an object), which represent open files
- These are created using the open function

```
file = open("name", mode)
```

- mode tells Python what you want to do with the file

'w'	write
'wb'	write (binary)
'r'	read
'rb'	read (binary)
'a'	append
'ab'	append (binary)

- When you're done with the file, close the file using

```
file.close()
```

Summary

- Values can be converted to text using `str()`
- Text strings can be joined together using '+'
- Text can be written to file by opening the file, writing text, then closing the file
- There are lots of functions to help read text files, including `readline()` and `split()` You can use these to read in many simple file layouts
- If you just want to use files with Python, then the `pickle` module is a good option
- Many other options available for storing data and interchanging with other programs

`http://www-users.york.ac.uk/~bd512/teaching.shtml`