Introduction to Python

Scripting in Python

25th November 2016

Opening Files

```
f = open('foo.txt','r') # read
f.close()
```

Opening Files

```
f = open('foo.txt','r') # read
f.close()
```

```
f = open('foo.txt','w') # write
f.close()
```

Opening Files

```
f = open('foo.txt','r') # read
f.close()
```

```
f = open('foo.txt','w') # write
f.close()
```

```
f = open('foo.txt','a') # append
f.close()
```

Opening Byte Files

If your file is not a simple text file you might have to treat it as a byte file. e.g. PDF, pickle files etc

```
f = open('foo.txt','rb') # read
f.close()
```

```
f = open('foo.txt','wb') # write
f.close()
```

```
f = open('foo.txt','ab') # append
f.close()
```

Better Opening Files

- You should always close a file that you open.
- The OS *should* take care of it if your program closes.

Better Opening Files

- You should always close a file that you open.
- The OS *should* take care of it if your program closes.
- The best way to handle this is with *context handlers*:

```
with open('foo.txt','r') as f:
    # code
```

Reading Files

- The file handle has an internal state - position in the file.

Reading Files

- The file handle has an internal state - position in the file.
- Several of the "read" functions move that state forwards

Reading Files

- The file handle has an internal state - position in the file.
- Several of the "read" functions move that state forwards
- Use seek() to move around this state

Reading Files

- The file handle has an internal state - position in the file.
- Several of the "read" functions move that state forwards
- Use seek() to move around this state
- tell() returns the current line number

```
with open('foo.txt','r') as f:
    lines = f.readlines() # list of lines
    print f.readlines() # prints []
    f.seek(0)
    f.readlines() == lines # True
```

Reading Files

- readlines() returns an array of the lines
- read() returns a string of the whole file

Reading Files

- readlines() returns an array of the lines
- read() returns a string of the whole file
- Both of these require you to store whole file in memory, bad for big files

Reading Files

- readlines() returns an array of the lines
- read() returns a string of the whole file
- Both of these require you to store whole file in memory, bad for big files
- Iteration to the rescue!

```
with open('foo.txt','r') as f:
    for line in f:
        #analyse line
    # f is now at end of file, seek to restart
```

Writing Files

```
with open('foo.txt','w') as f:
    f.write("Hello␣there\n") # note the \n
```

Writing Files

```
with open('foo.txt','w') as f:
    f.write("Hello there\n") # note the \n
```

```
    x = "foo" + str(100) + "bar \n"
    f.write(x)
```

Demo

Demo Time!

Serialisation

- Sometimes you need to save your data (or send it).

Serialisation

- Sometimes you need to save your data (or send it).
- We have seen how to write to a file - great for numbers, text etc.
- What about more complicated objects?

Serialisation

- Sometimes you need to save your data (or send it).
- We have seen how to write to a file - great for numbers, text etc.
- What about more complicated objects?
- Python's pickle library lets you store your objects to disk

Pickle example

```
x = some_huge_data_structure()
# Lets store it on disk
pickle.dump(x, open('mySaveFile.pb','wb'))
# ... some other script
y = pickle.load(open('mySaveFile.pb', 'rb))
```

Not enough time

- PyPDF2 is a comprehensive PDF python library
- Concatenate PDF files, create new ones etc.

Not enough time

- PyPDF2 is a comprehensive PDF python library
- Concatenate PDF files, create new ones etc.
- Demo time - splitter.py

OS

- The os library lets you interact with the file system.
- e.g. view the files in the current directory, sort by last time modified

OS

- The os library lets you interact with the file system.
- e.g. view the files in the current directory, sort by last time modified
- As always, the docs are the place to go - good detailed info on all the useful functions.

os - important methods

```
os.chdir(path) #change current directory
os.listdir(path) # list files in path
os.rename(src, dest) # rename file
os.execv(path, args) # execute program at path
os.tmpfile() # a temporary file
```

re

- Regular expressions (Regex) are one of the most powerful single tools in programming.

re

- Regular expressions (Regex) are one of the most powerful single tools in programming.
- Spend a few hours learning it, its worth it!
- Plenty of resources online, especially http://www.regular-expressions.info/tutorial.html
- Python library is called re

Command Line Arguments

- When you launch a program the OS hands it the arguments it was launched with.

Command Line Arguments

- When you launch a program the OS hands it the arguments it was launched with.
- e.g.

```
python test.py
```

Command Line Arguments

- When you launch a program the OS hands it the arguments it was launched with.
- e.g.

```
python test.py
```

- Access these using the sys module in Python

```
import sys
sys.argv # list of arguments
sys.argv[0] # name of current program
```

The argparse Library

- For robust argument handling use the argparse library.

The argparse Library

- For robust argument handling use the argparse library.
- e.g.

```
python test.py -f 20 --input-file demoFile.dat
```

The argparse Library

- For robust argument handling use the argparse library.
- e.g.

```
python test.py -f 20 --input-file demoFile.dat
```

- Arguments have long and short versions, can be optional, be entered with different orders etc.

Demo

Demo Time - todo generator