

Python While Loop

```
#!/usr/bin/python
n = 12
i = 1
while i<=12:
    print i,"x 8 =", i*8
    i=i+1
print "hello world"
```

Python Functions

```
#!/usr/bin/python
def mult8(i):
    return i*8
for i in range(1,13):
    print i,"x 8 =", mult8(i)
```

if statement and functions

```
#!/usr/bin/python
def mult8(i):
    print i,"x 8 =", i*8
    if i<12:
        mult8(i+1)
mult8(1)
```

Python Modules

- there are many Python modules available
- which cover many topics
 - networking modules
 - graphic modules, OpenGL, GUI, graphing
 - mail, http, telnet, pop3, imap modules
 - operating system modules
- html parsing modules
- examine the Python modules [python online docs](http://floppsie.comp.glam.ac.uk/python/html/index.html) (<http://floppsie.comp.glam.ac.uk/python/html/index.html>)

urllib

- used to download files from servers using
 - ftp, http and local file access

urllib example

```
#!/usr/bin/python

from urllib import urlretrieve

urlretrieve('http://floppsie.comp.glam.ac.uk/index.html',
            'temp.html')
```

urllib example

```
#!/usr/bin/python

import os
import urllib
Version = '1.5'
filename = 'python-%s.tar.gz' % Version
remoteaddr = 'ftp://ftp.python.org/pub/python/src/'
urllib.urlretrieve(remoteaddr + filename,
                  filename)
```

smtp module

- Simple Mail Transport Protocol is the most common protocol whereby email is transmitted across the Internet

```
#!/usr/bin/python

import smtplib, string, sys, time

mailserver = "localhost"

From = string.strip(raw_input('From: '))
To = string.strip(raw_input('To: '))
Subject = string.strip(raw_input('Subject: '))

Date = time.ctime(time.time())
Header = ('From: %s\nTo: %s\nDate: %s\nSubject: %s\n\n'
         % (From, To, Date, Subject))

Text = "my message"
server = smtplib.SMTP(mailserver)
failed = server.sendmail(From, To, Header + Text)
server.quit()
if failed:
    print 'failed to send mail'
else:
    print 'all done..'
```

Tutorial

- type in the urllib example given during the lecture and check that it works
 - now modify this example so that your script prompts the user for
 - the url
 - the local file name

Python Gotya's

- be careful to ensure that your code is indented correctly
- be very careful not to name your file to a name used by a library you are importing

Python Gotya's

- for example do **not** call this file `string.py`

- ```
#!/usr/bin/python
import string
words=string.split('hello world again')
print words
```

## Python Gotya's

- the python interpreter will read your file twice
  - one when you run the file
  - and again when it comes across the `import string!`
- name the file `teststring` and it will work fine
  - if you did call it `string.py` and run then you will need to remove `string.py` and also `string.pyc`

## Python and file handling

- file manipulation primitives are by default available
  - no need to import library to, read, write files

## Python and file handling

- creating a simple text file

```
#!/usr/bin/python
file = open('newfile.txt', 'w')
file.write('hello world in the new file\n')
file.write('and another line\n')
file.close()
```

## Python and file handling

```
#!/usr/bin/python
file = open('newfile.txt', 'w')
file.writelines(['hello world in the new file\n',
 'and another line\n'])
file.close()
```

## Python and file handling

```
#!/usr/bin/python
file = open('newfile.txt', 'r')
for line in file.readlines():
 print line
```

- many ways to read a file
  - `file.read()` returns a string containing all characters in the file
  - `file.read(N)` returns a string containing next N characters
  - `file.readline()` returns a string containing characters up to `\n`
  - `file.readlines()` returns the complete file as a list of strings each separated by `\n`

## Further Python Networking

- many python modules which give access to application layer networking services
  - ftp, http, telnet, etc

## Further Python Networking

- sometimes you may have to implement your own application layer protocol
- in which case you use `sockets` (a transport layer service)

### server.py

```
#!/usr/bin/python

from socket import *
myHost = ''
myPort = 2000

create a TCP socket
s = socket(AF_INET, SOCK_STREAM)
bind it to the server port number
s.bind((myHost, myPort))
allow 5 pending connections
s.listen(5)

while True:
 # wait for next client to connect
 connection, address = s.accept()
 while True:
 data = connection.recv(1024)
 if data:
 connection.send('echo -> ' + data)
 else:
 break
 connection.close()
```

### client.py

```
#!/usr/bin/python

import sys
from socket import *
serverHost = 'localhost'
serverHost = 'localhost'
serverPort = 2000

create a TCP socket
s = socket(AF_INET, SOCK_STREAM)

s.connect((serverHost, serverPort))
s.send('Hello world')
data = s.recv(1024)
print data
```

## To run the server client example

- open up another terminal and type this at the command line

- `$ python server.py`

- open up another terminal and type this:

- `$ python client.py`

## IMAP library

```
#!/usr/bin/python
import getpass, imaplib, string

m = imaplib.IMAP4_SSL('unimail.isd.glam.ac.uk')
m.login(getpass.getuser(), getpass.getpass())
m.select()
typ, data = m.search(None, 'ALL')
for num in string.split(data[0]):
 typ, data = m.fetch(num, '(RFC822)')
 print 'Message %s\n%s\n' % (num, data[0][1])
m.logout()
```

## Arguments in Python

- getopt, provides a useful method for handling arguments
  - in fact many languages have adopted getopt
  - C, C++, bash and python

## Autoftp arguments in python

```
#!/usr/bin/python
import sys, getopt

def Usage ():
 print "autoftp [-v][-p][-h]"
 sys.exit(0)

optlist, list = getopt.getopt(sys.argv[1:], ':vphf:')
print "optlist =", optlist
print "list =", list
for opt in optlist:
 print opt
 if opt[0] == '-h':
 Usage()
 if opt[0] == '-f':
 print "file found"
 if opt[0] == '-v':
 print "verbose found"
 if opt[0] == '-p':
 print "probeonly found"
```

## Autoftp arguments in python

- notice that the script fails if an unsupported option is issued

```

./autoftp2.py -x
...
getopt.GetoptError: option -x not recognised

```

## Better argument handling

- so we need a way to trap these errors
  - python uses an exception handler for this

```

#!/usr/bin/python

import sys, getopt

def Usage ():
 print "autoftp [-v][-p][-h]"
 sys.exit(0)

try:
 optlist, list = getopt.getopt(sys.argv[1:],
 ':vphf:')
except getopt.GetoptError:
 Usage()
 print "called exception"
 sys.exit(1)

for opt in optlist:
 print opt
 if opt[0] == '-h':
 Usage()
 if opt[0] == '-v':
 print "verbose found"
 if opt[0] == '-p':
 print "probeonly found"
 if opt[0] == '-f':
 print "file option found"

```

## Better argument handling

- when run it yields the following

```

./autoftp3.py -x
autoftp [-v][-p][-h]

```

## When is a module not a module?

- it is often useful to create a module
  - for yourself and others to use in the future
  - to subdivide the large problem set into a number of smaller modules
- sometimes a module might be able to operate as a stand alone program
  - consider autoftp could be organised as a module

## When is a module not a module?

- ```
if __name__ == "__main__":
    main()
```
- which means run the function `main` if this module is explicitly invoked by the user
 - note that it is not run if this module was imported

Example times module

- ```
#!/usr/bin/python
import sys

def multiplyby10(value):
 return value+'0'

if __name__ == "__main__":
 if len(sys.argv) == 2:
 print 'testing the times module'
 print multiplyby10(sys.argv[1])
```

## Example program

- ```
#!/usr/bin/python
import times, sys

if len(sys.argv) == 2:
    print 'importing the times module'
    print times.multiplyby10(sys.argv[1])
```

Example program

- note that the module `times` takes a string and adds a '0' to the left hand side
 - effectively multiply by 10
- note it also uses the `if __name__ ==` condition which only calls the `multiply` routine if this module was invoked as the main program by the user

Example program

- ```
./prog.py 12
importing the times module
120
```

- ```
./times.py 12
testing the times module
120
```

- exercise for the reader, add a function to perform divide and modulus of a numerical integer string

printf

- if any C programmer laments the lack of a printf function, you can roll your own:

mylibc.py

```
#!/usr/bin/python
#
# printf - keeps C programmers happy :-)
#
def printf (format, *args):
    print str(format) % args,
```

- please create this file (module) as it will be very useful when you start the coursework

printf

mytest.py

```
#!/usr/bin/python
from mylibc import printf
printf ("hello world\n")
printf ("an int: %d\n", 42)
printf ("a float: %f\n", 3.1415927)
```

- why does the output for a float differ from the constant value?