

## Introduction to Python

- is more powerful than Tcl
  - applicable to larger systems development
- has cleaner syntax than Perl
  - easier to maintain
- does not compete head on with Java
  - python is a scripting language
  - whereas Java is a systems language similar to C++

## Python can be simple

```
#!/usr/bin/python
print "hello world"
```

## Python Modules allow for problem decomposition

- similar to Modula-2

```
#!/usr/bin/python
title = "hello world"
```

- above is called `myfile.py`

```
#!/usr/bin/python
import myfile
print myfile.title
```

- whereas this file is called `hello2.py`

- when run prints hello world

## Alternative import

```
#!/usr/bin/python
from myfile import title
print title
```

- note that all python modules need to be saved as *name.py*
  - so in our example the module `myfile` was saved into a file called `myfile.py`

## Python builtin types

- python contains many builtin types
  - use them..
- builtin objects make simple programs easy to understand
  - lists, dictionaries, exist, don't reinvent the wheel
- built in objects are more efficient than custom data types

## Builtin objects

- numbers 3.14159, 1234
- strings 'spam', "fred's"
- lists [1, [2, 'three'], 4]
- dictionaries {'food': 'spam', 'taste': 'yum'}
- tuples (1, 'spam', 4, 'U')
- files text=open('/etc/passwd', 'r').read()

## Expression operators

- or, and, not
  - <, <=, >, >=, ==, <>, !=
  - x | y
  - z & y
  - x << y
  - x >> y
  - x[i]
  - x[i:y]
  - x.y
  - x(y)
- logical operators (short circuit)
  - comparison operators
  - bitwise or
  - bitwise and
  - shift left by y bits
  - shift right by y bits
  - indexing
  - slicing
  - qualifying (imports)
  - function calls

## Strings

- concatenation via +
  - repeated via \*

```
#!/usr/bin/python
print "hi " * 4
```

- yields

```
hi hi hi hi
```

## Slicing

- given a string in Python
- ```
s = "hello world"
```
- you can obtain portion of string via: `s[2:5]`
  - yields: `llo`

## Slicing

- first character has index 0
  - and also -11
  - last character index is 10 in this example
  - last character index is also -1
- negative values start at right and move to the left
- strings can be sliced using positive and negative values

## Statements

- assignment, calls, print, if/else/elif, for, while, break/continue
- try, except, raise,
- def, return
  - function definitions and returning values

## Statements

- class
- assert
- exec
- del
- global

## Example 8 times table

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

## Constructing a simple client and server

- the following should be called `client.py`

```
#!/usr/bin/python
import sys
from socket import *
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
```

## 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:
            print "received data", data
            connection.send('echo -> ' + data)
        else:
            break
    connection.close()
```

## Run the server and client from the command terminal

- start the server first

```
$ python server.py
```

- now run the client in a *different* terminal:

```
$ python client.py
echo -> Hello world
```

## Python has many support modules

- see the global module index in the [python online docs](http://floppsie.comp.glam.ac.uk/python/html/index.html) (<http://floppsie.comp.glam.ac.uk/python/html/index.html>)
- take note of the `struct` module
- this module provides a list of method which performs conversions between Python values C objects
- it can be very usefully employed in constructing protocols
- it uses format strings as compact descriptions of the layout data
  - similar to `printf` and friends in C

## Python has many support modules

- read the [documentation](http://docs.python.org/library/struct.html) (<http://docs.python.org/library/struct.html>) and write a small program to print your username in with 20 characters
- your program should output

```
01234567890123456789
<fred                >
```

## Python has many support modules

```
#!/usr/bin/python

import struct
import getpass

print " 01234567890123456789"
print "<",
username = getpass.getuser()
print username,
print ">",
# answer
print "<",
print struct.pack("20s", username),
print ">",
```

- the program above pads your username with character 0's to make up a 20 character string

```
$ python teststruct.py | od -x
0000000 3020 3231 3433 3635 3837 3039 3231 3433
0000020 3635 3837 0a39 203c 6167 7569 2073 203e
0000040 203c 7266 6465 0000 0000 0000 0000 0000
0000060 0000 0000 0000 3e20 000a
0000071
```

## Write a Python program to output your UID as two bytes in network order

```
#!/usr/bin/python

import struct
import os

print os.getuid()
# answer
print struct.pack("!H", os.getuid())
```

## TFTP client implementation coursework support

- useful references
  - [rfc1350](http://www.faqs.org/rfcs/rfc1350.html) (<http://www.faqs.org/rfcs/rfc1350.html>)
  - [python struct module](http://docs.python.org/library/struct.html) (<http://docs.python.org/library/struct.html>)
- you will need to use the `select` function in Python to wait for a `tftp` packet to discover a timeout

## select

- look for the line of code in `tftp-skel.py` which matches
- ```
r, w, e = select.select([sock], [], [], 5.0)
```
- it follows the behaviour found in [select](http://docs.python.org/library/select.html) (<http://docs.python.org/library/select.html>)
- when your code works change this line to
- ```
r, w, e = glamnetsim.simselect([sock], [], [], 5.0)
```
- as this function behaves in exactly the same way but it introduces occasional errors

## Skeleton coursework code

- [tftp-skel.py](#) (`tftp-skel.py`)
- [glamnetsim.py](#) (`glamnetsim.py`)

## TFTP client implementation coursework support

- useful references
  - [rfc1350](http://www.faqs.org/rfcs/rfc1350.html) (<http://www.faqs.org/rfcs/rfc1350.html>)
  - [python struct module](http://docs.python.org/library/struct.html) (<http://docs.python.org/library/struct.html>)
- you will need to use the `select` function in Python to wait for a `tftp` packet to discover a timeout

**select**

- use this function to obtain a packet or wait for a timeout
- ```
r, w, e = select.select([sock], [], [], 5.0)
```
- it follows the behaviour found in [select](http://docs.python.org/library/select.html) (<http://docs.python.org/library/select.html>)
- when your code works change this line to
- ```
r, w, e = glamnetsim.simselect([sock], [], [], 5.0)
```
- as this function behaves in exactly the same way but it introduces the occasional timeout event

**select**

- when this function returns the values of `w` and `e` will always be `[]` but the value of `r` will either be `[]` or `[sock]`
  - the value `[]` means a timeout occurred and the value `[sock]` means this socket has some data which is ready to be read

**select**

- to read the packet from the socket you can:
- ```
(packet, (address, port)) = sock.recvfrom(512+4)
```
- which reads up to 516 bytes of raw data into `packet` and assigns the IP address and port value of the tftp server