Socket Programming

Sockets

- API for inter process communication
- An integer, a thing called socket and methods for the same
- Different machines/processes
- Berkely
- In python as well

Server

1. create a socket
2. bind the socket to an address and port
3. listen for incoming connections
4. wait for clients
5. accept a client
6. send and receive data

```

import socket

host = ''
port = 50000
backlog = 5
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind((host,port))
s.listen(backlog)
while 1:
    client, address = s.accept()
    data = client.recv(size)
    if data:
        client.send(data)
    client.close()
```

Client

1. create a socket
2. connect to the server
3. send and receive data

```
import socket

host = 'localhost'
port = 50000
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.connect((host,port))
s.send('Hello, world')
data = s.recv(size)
s.close()
print('Received:', data )
——————————
(sadanand@lxmayr10 \@ ˜)python    client.py
Received: Hello, world
(sadanand@lxmayr10 \@ ˜)
```

To Note

- In `recv`, one might not get all the data from the server in a single go. In such a case, a loop until data received in None is advised.
- If the server dies, then the client will hang (almost) (as good as)

A word about sockets

- Blocking Sockets: The socket is blocked until the request is satisfied. When the remote system writes on to it, the operation is completed and execution resumes.
- Non Blocking Sockets: Error conditions are to be handled properly. Doesn't wait for the remote system. It will be informed.

Sockets Programming and Pickling

In python, objects can be send from sockets to sockets with the help of the Pickle Module.
The code snippet in the next slide explains this.

```
Client Side:

pickledStuff = pickle.dumps (PickleableObject)
self.channel.send (pickledStuff)


Server Side:
x = pickle.loads(client.recv(1024))
```

Threads and Processes

- Threads exist as subsets of a process (not independent)
- Multiple threads within a process share state as well as memory and other resources
- Threads share their address space
- No IPC needed.
- Context switching is typically faster

CAN SHARE GLOBAL VARIABLES

```
import threading
class MyThread ( threading.Thread ):
    def run ( self ):
        print('Insert some thread stuff here.')
        print('It\'ll be executed...yeah....')
        print('There\'s not much to it.')

MyThread().start()
____________________________________________

Insert some thread stuff here.
It'll be executed...yeah....
There's not much to it.
```

```
theVar = 1
class MyThread2 ( threading.Thread ):
    def run ( self ):
        global theVar
        print('This is thread ' + str ( theVar )
        print('Hello and good bye.')
        theVar = theVar + 1
for x in xrange (4):
    MyThread2().start()
----------------------------------------------------------------
This is thread 1 speaking.
Hello and good bye.
This is thread 2 speaking.
Hello and good bye.
This is thread 3 speaking.
Hello and good bye.
This is thread 4 speaking.
```

```
Hello and good bye.
```

Locks and Threads

- Multiple threads can communicate using a global variable
- But when two threads access the same variable at the same time?
- There are locks available

```
import threading
import time
from random import randint
class MyThread2 ( threading.Thread ):
    lock = threading.Lock()
    tcnt = 0

    def __init__(self, gname):
        threading.Thread.__init__(self)
        self.name = gname

    def run ( self ):
        time.sleep(randint(1, 5))
        print('This is thread ' + str(self.name)
                    + ' speaking. (call order)'
        MyThread2.lock.acquire()
        MyThread2.tcnt += 1
        MyThread2.lock.release()
        print('Hello and good bye from thread )
                     reached', MyThread2.tcnt

for x in xrange (4):
    MyThread2(x).start()
```

```
This is thread 1 speaking. (call order)
Hello and good bye from thread reached 1
This is thread 0 speaking. (call order)
Hello and good bye from thread reached 2
This is thread 3 speaking. (call order)
Hello and good bye from thread reached 3
This is thread 2 speaking. (call order)
Hello and good bye from thread reached 4
```

Speed-up Lists

- `array` : Homogenious entries. Limited space than 16 bytes for every item
- `deque` : More efficient in cases of append and left deletion/pop
- `bisect` : Keep it sorted. And do it while insertion.
- `heapq` : Maintain a heap

```
>>> from array import array
>>> a = array('H', [4000, 10, 700, 22222])
>>> sum(a)
26932
>>> a[1:3]
array('H', [10, 700])


>>> from collections import deque
>>> d = deque(["task1", "task2", "task3"])
>>> d.append("task4")
>>> print("Handling", d.popleft())
Handling task1
```

```
>>> import bisect
>>> scores = ((100, 'perl'), (200, 'tcl'),
              (400, 'lua'), (500, 'python'))
>>> bisect.insort(scores, (300, 'ruby'))
>>> scores
((100, 'perl'), (200, 'tcl'), (300, 'ruby'),
 (400, 'lua'), (500, 'python'))


>>> from heapq import heapify, heappop, heappush
>>> data = (1, 3, 5, 7, 9, 2, 4, 6, 8, 0)
>>> heapify(data)
>>> heappush(data, -5)
>>> (heappop(data) for i in range(3))
(-5, 0, 1)
```

Processes and Pipes

- When the client and server are running in the same system, we can use pipes.
- They can be used as files
- `os.popen(cmd, [mode, [bufsize]])` : Returns a pipe which is an `stdout` for cmd, from where the output can be read
- `os.popen2(cmd, [mode, [bufsize]])`: Similar, but an `stdin` too.

```
from contextlib import closing
import os
def ls(dir):
    with closing(os.popen("ls %s" % dir)) as pipe:
        for line in pipe:
            yield line


for filename in ls("/tmp"):
    print(filename)
```

Switch Case .. or Almost the Same

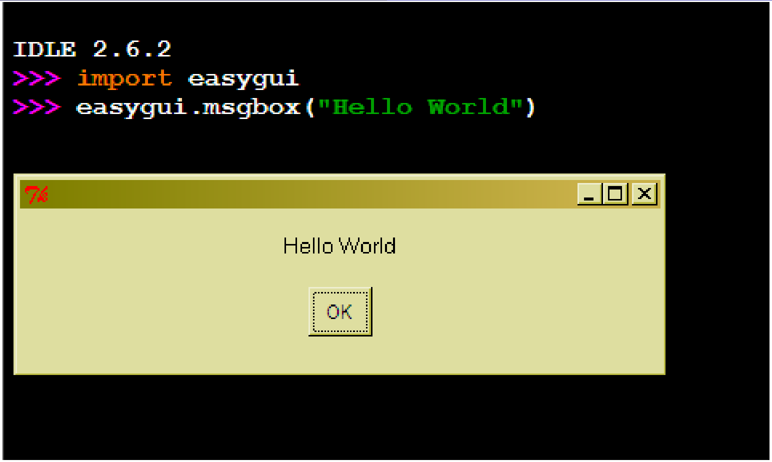
- Python doesn't provide switch case
- In many cases we can still make use of python constructs to bypass `if..elif..elif..`
- The key is function pointers

```
def key_1_pressed():
    print('Key 1 Pressed')

def key_2_pressed():
    print('Key 2 Pressed')

def key_3_pressed():
    print('Key 3 Pressed')

def unknown_press():
    print('Unknown Key Pressed')


def dealkey_trad(keycode):
    if    keycode == 1:
        key_1_pressed()
    elif keycode == 2:
        key_2_pressed()
    elif keycode == 3:
        key_3_pressed()
    else:
        unknown_key_pressed()

def dealkey_unusual(kc):
    functions = {1: key_1_pressed,
                 2: key_2_pressed,
                 3: key_3_pressed}
    functions.get(kc, unknown_press)()

dealkey_unusual(3) --- Prints Key 3 Pressed
dealkey_trad(4) --- Prints Unknown Key Pressed
```

Easy Gui

- Runs in Windows
- `http://easygui.sourceforge.net`

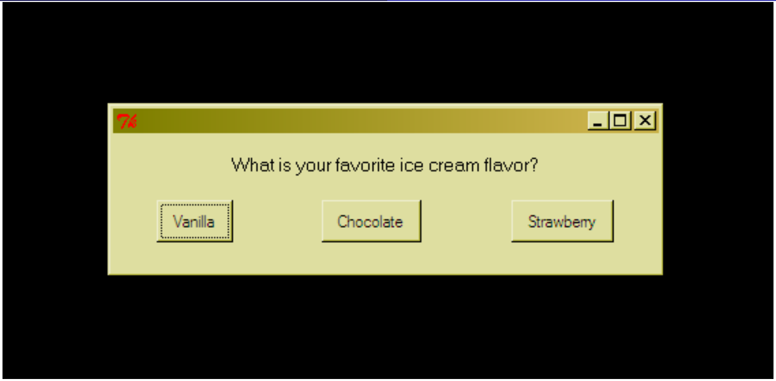
The Book : Hello World (All programs from there)

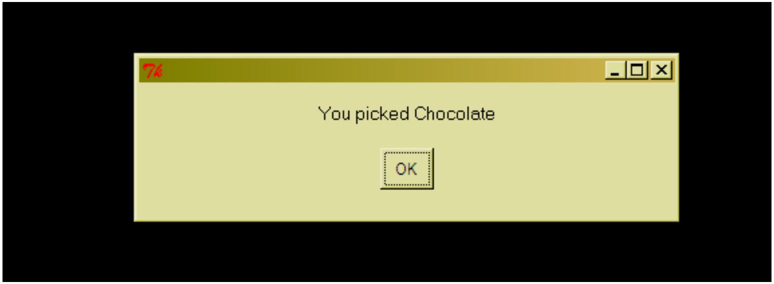


```
IDLE 2.6.2
>>> import easygui
>>> easygui.msgbox("Hello World")
```

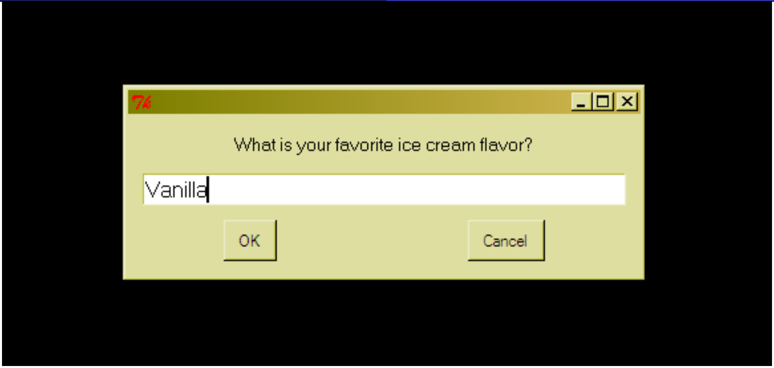
```
Hello World
```

```


import easygui
flavor = easygui.buttonbox("What is your favorite
         ice cream flavor?",
         choicess = ('Vanilla', 'Chocolate',
                     'Strawberry'))
easygui.msgbox ("You picked " + flavor)
```


What is your favorite ice cream flavor?
Vanilla
Chocolate
Strawberry


You picked Chocolate

```python


import easygui
flavor = easygui.enterbox("What is your favorite
                                                ice cream flavor?",
                                                 default = 'Vanilla')
easygui.msgbox ("You entered " + flavor)
```


Tk
What is your favorite ice cream flavor?
Vanilla
OK
Cancel

```
import random, easygui

secret = random.randint(1, 99)
guess = 0
tries = 0

easygui.msgbox("I have a secret!
              It is a number from 1 to 99.
                 I'll give you 6 tries.")

while guess != secret and tries < 6:
    guess = easygui.integerbox("guess?")
    if not guess: break
    if guess < secret:
        easygui.msgbox(str(guess) + " too low!")
    elif guess > secret:
        easygui.msgbox(str(guess) + " too high!")
    tries = tries + 1

if guess == secret:
    easygui.msgbox("got it!")
else:
    easygui.msgbox("No more guesses!")
```

Games with PyGame

- Use `pygame`
- `pygame.org`

Zero Knowledge Proofs

Graphs

- Hamiltonian Path/Cycle
- Graph Isomorphism
- Going in the Cave

Graph Isomorphism

To check whether two given graphs G and H are isomorphs, when we know the mapping f from G to H (w.l.g), All we need to do is confirm that the mapping is a bijection.
i.e, check for every node $g \in G$ that, $h = f(g) \in H$ is unique.
Also, one has to confirm that the set of edges too satisfy this property. i.e, $e_{iG} \in E_G$ has a unique $e_{iH} \in E_H$.

```
def isomorph(self, other, foo):
    gnodes = self.nodes.keys()
    hnodes = other.nodes.keys()
    if len(nodes) != len(hnodes): return False
    filtered = filter(lambda v:
                foo(v) not in set(hnodes), gnodes)
    if filtered: return False

    HEDGES = set((e for e in other.edges()))
    for (u, v) in self.edges():
        hedge = (foo(u), foo(v))
        if hedge not in hedges:
            return False
        hedges.remove(hedge)

    return False if hedges else True
```

What did we do?

- Basic Data Types, Operators
- Control Structures
- Collection Types
- Classes / Objects, Anonymous Classes
- Modules, Importing them
- Basic IO, Files
- Lambda Functions, Other Functional Proramming tools
- Regular Expressions
- URLs and HTTP, XML/HTML Parsing

What did we do?

- Shelves
- Iterators, Generators
- Socket Programming
- Pickles
- Threads, Pipes
- Decorators
- Static Variables/Functions
- GUI (a little bit)
- ZKP

Spell Checker

We saw that it wasn't too hard to check spelling. But develop it to the spellchecker tools in Unix.

Code Beautifier

No much details needed.
One could look into `indent(1)`

Encrypt / Decrypt

Implement cryptography algorithms starting from Caeser's code, upto RSA.
Have a measure of security and choose the kind of encryption prefferred.
(Sub: Primality testing algorithsm)

Handwriting to Image

Have images of characters handwritten, then convert the text (typed) into sequences of those images and finally to a single image.
(No idea, how hard it could be)

Hangman

The name explains it all.

- Text based
- Gui based (Tkinter / Pygame)

Could be Fuuuuuuuuun!

Tic Tac Toe

Implement Tic Tac Toe
(if you dare enough, implement Chess (text/gui))

Simply Algorithsm

Choose random algorithms and implement some (10-15) of them.
`http://en.wikipedia.org/wiki/List_of_algo`

Emacs Doctor?

If you know him/her, nice. If you don't, know him/her.
And have a duplicate in Python (instead of LISP)

Star Locator

Could be complicated.
Given the date of birth of a person, and the desired date and the location on earth locate the birthday star of that person.
i.e., Where to look in the sky (at what time) to see the star.

ASCII Art?

JPEG to ASCII??

ACM Problems

Choose ACM programming contest problems and code them.

Word Guesser

Guesses the word you are going to type and suggest that. (create a small text field; the recommended list of words appear on a panel on the right side, choose with ctrl-#)
TRIEs and Splay Trees - I think.

Shell Gui

- Shell - prompt and scrollable screen.
- Command parser (or direct OS output)
- Command history
- Background running
- Output directions
- Tab-Completion
- Pipes?

Some links

- http://www.norvig.com/21-days.html (learn programming in 10 years)
- http://www.pythonchallenge.com/
- http://freshmeat.net/articles/python-projects

Some Questions and Self Evaluation

I have some questions here. Please write down the answers and give it back.

The points for each question is 2^n for some n.

If the solution is perfect	You get all the marks.
If the solution is MOL fine	You get $2^n - 2^{n-2}$
If the solution is barely ok	You get $2^n - 2^{n-1}$
If you can look it up in 10 mins	You get $2^n - 2^{n-1} - 2^{n-2}$
Otherwise	You know math. ;)

The final Exam

- When?
- How? = How many?
- Assignments carry 30%
- New people can write, can submit assignments now, but with less weightage.

Problems

- Small client for HTTP
- Implement a graph and check-for-hamiltonian
- Server Client - Sockets, Threading, Sending data with Pickle Client sends some datatype, Server sends back the length of the object
- Server Client - Pipes