Using open()

- handle = open(filename, mode)

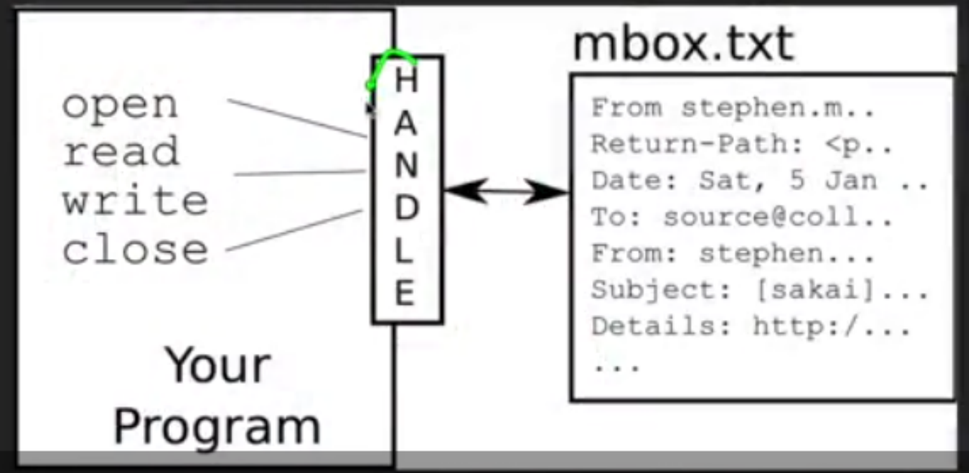
```
fhand = open('mbox.txt', 'r')
```

- returns a handle use to manipulate the file
- filename is a string
- mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file

It's just making the file available to the code that we're going to write.

What is a Handle?

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='UTF-8'>
```



So, this handle is
something that's sort of

File Handle as a Sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

variable, cheese in this case,
is going to take on the successive lines.

Reading the *Whole* File

We can read the whole file (newlines and all) into a single string

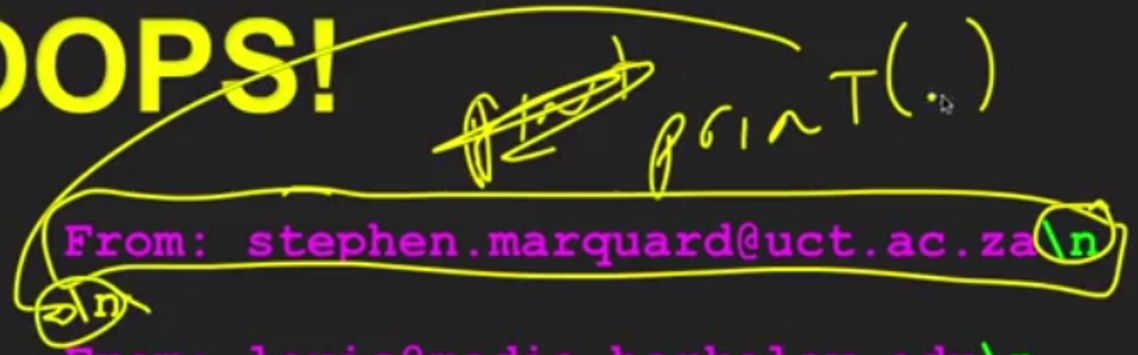
```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

And so this time, we will read the whole thing in with .read.

OOPS!

What are all these blank lines doing here?

- Each line from the file has a newline at the end
- The print statement adds a newline to each line



this is the newline that was added by the print statement.

Searching Through a File (fixed)

- We can strip the whitespace from the right-hand side of the string using rstrip() from the string library

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

- The newline is considered "white space" and is stripped

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....
```

So you are going to write these lines of code over and over and over again.

```
>>> line = 'A lot                    of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>
```

- When you do not specify a delimiter, multiple spaces are treated like one delimiter
- You can specify what delimiter character to use in the splitting

And we just got a list and
that list had one string in it and

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

Sat
Fri
Fri
Fri
...

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

the first space, then in another line of
code you'd search for the second space.

Dictionaries

- Lists index their entries based on the position in the list
- Dictionaries are like bags - no order
- So we index the things we put in the dictionary with a "lookup tag"

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

Money is the label, the index,
the tag, whatever you want to call it.

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]

>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

if there were more of these,
it would be 0, 1, and 2.

Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of key : value pairs
- You can make an empty dictionary using empty curly braces

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print(ooo)
{}
>>>
```

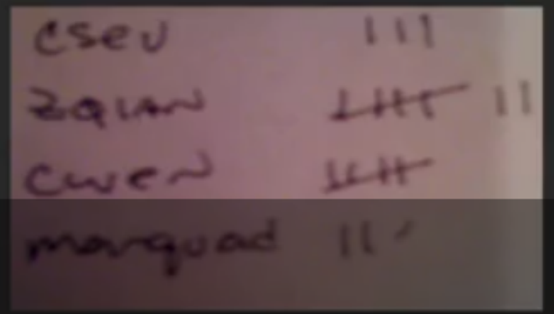
then we have key, value,
key, value, key, value.

When We See a New Name

When we encounter a new name, we need to add a new entry in the dictionary and if this the second or later time we have seen the name, we simply add one to the count in the dictionary under that name

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print(counts)
```

{'csev': 2, 'zqian': 1, 'cwen': 2}



And we're going to go cruising through there.

The get Method for Dictionaries

The pattern of checking to see if a key is already in a dictionary and assuming a default value if the key is not there is so common that there is a method called get() that does this for us

```
if name in counts:
    x = counts[name]
else :
    x = 0
```

```
x = counts.get(name, 0)
```

Default value if key does not exist (and no Traceback).

{'csev': 2, 'zqian': 1, 'cwen': 2}

So what counts.get is,
it says go look up in counts,

Simplified Counting with get()

We can use get() and provide a default value of zero when the key is not yet in the dictionary - and then just add one

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

Default

{'csev': 2, 'zqian': 1, 'cwen': 2}

you say oh, this is our little histogram trick. So if we look at it in slow motion.

Retrieving lists of Keys and Values

You can get a list of keys, values, or items (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

What is a “tuple”? - coming soon...

actually tell Python what we're doing right here is we're

Bonus: Two Iteration Variables!

- We loop through the key-value pairs in a dictionary using *two* iteration variables

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

```
jan 100
chuck 1
fred 42
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

- Each iteration, the first variable is the key and the second variable is the corresponding value for the key

I usually name this k and v or

Sorting Lists of Tuples

- We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary
- First we sort the dictionary by the key using the items() method and sorted() function

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> d.items()
dict_items([('a', 10), ('c', 22), ('b', 1)])
>>> sorted(d.items())
[('a', 10), ('b', 1), ('c', 22)]
```

But it sorts it based on the first thing.

Sort by Values Instead of Key

- If we could construct a list of tuples of the form (value, key) we could sort by value
- We do this with a for loop that creates a list of tuples

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...       tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

So if we think of these,
we can get key-value tuples and

```
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1

lst = list()
for key, val in counts.items():
    newtup = (val, key)
    lst.append(newtup)

lst = sorted(lst, reverse=True)

for val, key in lst[:10] :
    print(key, val)
```

The top 10 most common words

We open a file, we make a dictionary.

Even Shorter Version

```
>>> c = {'a':10, 'b':1, 'c':22}

>>> print( sorted( [ (v,k) for k,v in c.items() ] ) )

[(1, 'b'), (10, 'a'), (22, 'c')]
```

List comprehension creates a dynamic list. In this case, we

- make a list of reversed tuples and then sort it.

http://wiki.python.org/moin/HowTo/Sorting

And so the tricky bit here is
this syntax in the middle.

Regular Expression Quick Guide

^	Matches the beginning of a line
$	Matches the end of the line
.	Matches any character
\s	Matches whitespace
\S	Matches any non-whitespace character
*	Repeats a character zero or more times
*?	Repeats a character zero or more times (non-greedy)
+	Repeats a character one or more times
+?	Repeats a character one or more times (non-greedy)
[aeiou]	Matches a single character in the listed set
[^XYZ]	Matches a single character not in the listed set
[a-z0-9]	The set of characters can include a range
(	Indicates where string extraction is to start
)	Indicates where string extraction is to end

https://www.py4e.com/lectures3/Pythonlearn-11-Regex-Handout.txt

Dot means any character.

The Regular Expression Module

- Before you can use regular expressions in your program, you must import the library using "import re"
- You can use re.search() to see if a string matches a regular expression, similar to using the find() method for strings
- You can use re.findall() to extract portions of a string that match your regular expression, similar to a combination of find() and slicing: var[5:10]

And then findall is like an extraction.

Using re.search() Like find()

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print(line)
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print(line)
```

And so this is just showing you how
you use the search capability from

Using re.search() Like startswith()

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print(line)
```

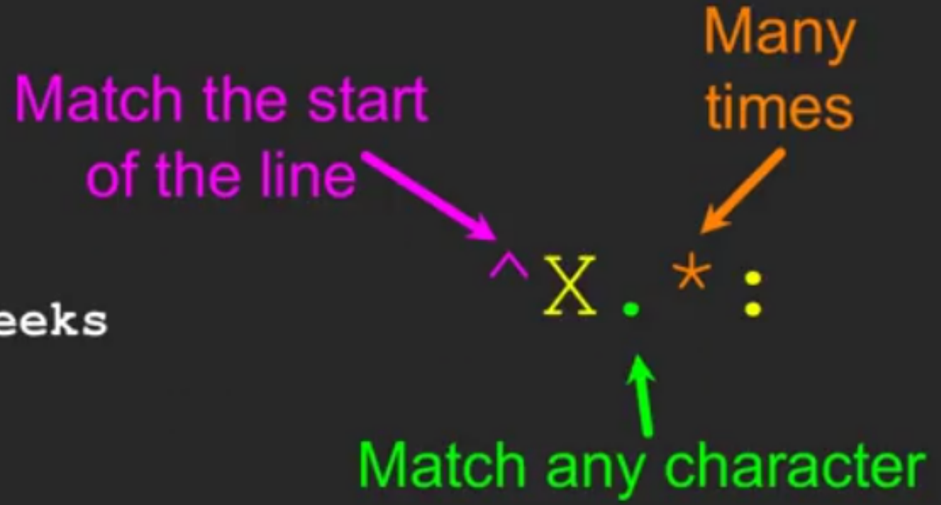
We fine-tune what is matched by adding special characters to the string

And now we're going to use startswith.

Fine-Tuning Your Match

Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks
```



the dot is any character, and
star means as many times as you like.

Matching and Extracting Data

When we use re.findall(), it returns a list of zero or more sub-strings that match the regular expression

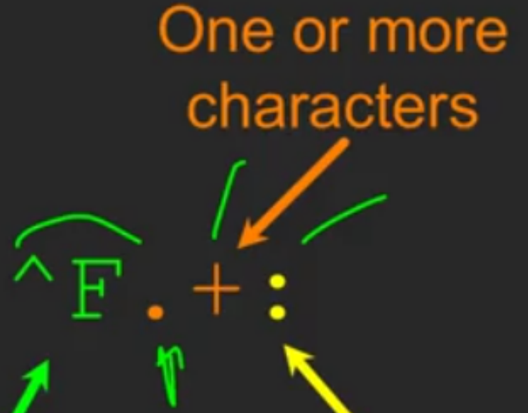
```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print(y)
['2', '19', '42']
>>> y = re.findall('[AEIOU]+',x)
>>> print(y)
[]
```

A E I O U, all uppercase.

Warning: Greedy Matching

The repeat characters (* and +) push outward in both directions (greedy) to match the largest possible string

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+:', x)
>>> print(y)
['From: Using the :']
```



Why not 'From:' ?

First character in the match is an F

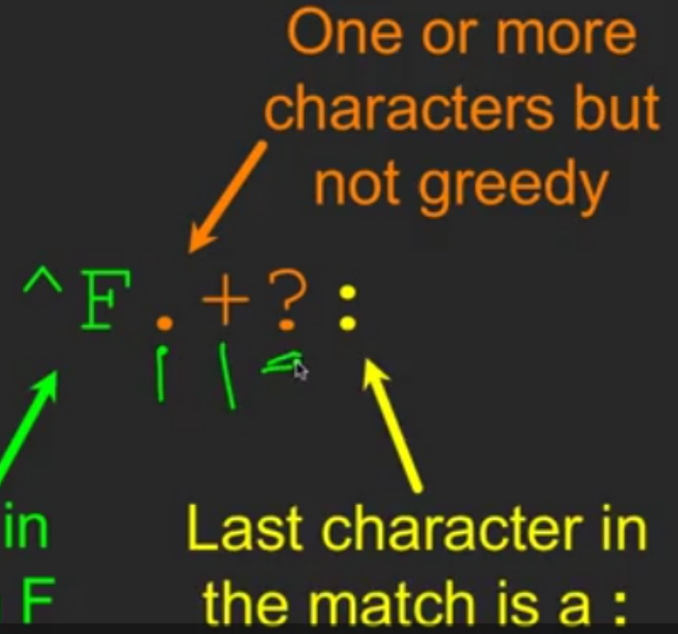
Last character in the match is a :

yeah there's the beginning F, and there's a characters and there's a colon, were done."

Non-Greedy Matching

Not all regular expression repeat codes are greedy!
If you add a ? character, the + and * chill out a bit...

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+?:', x)
>>> print(y)
['From:']
```



but don't be greedy.

Fine-Tuning String Extraction

You can refine the match for re.findall() and separately determine which portion of the match is to be extracted by using parentheses

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> y = re.findall('\S+@\S+',x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```

\S+@\S+

At least one non-whitespace character

So, that's a yes match,

Fine-Tuning String Extraction

Parentheses are not part of the match - but they tell where to start and stop what string to extract

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> y = re.findall('\S+@\S+',x)
>>> print(y)
['stephen.marquard@uct.ac.za']
>>> y = re.findall('^From (\S+@\S+)',x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```

```
^From (\S+@\S+)
```

So, it's a from, followed by a space,

The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```

we split it into words with spaces,

The Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print(y)

['uct.ac.za']
```

'@([^]*)'

Look through the string until you find an at sign

So, that says "buf," I've got what I want.

The Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print(y)
```

```
['uct.ac.za']
```

this is a single character but if the first letter

Even Cooler Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print(y)

['uct.ac.za']
```

'^From .*@([^]*)'

Starting at the beginning of the line, look for the string 'From '

We can fine tune this by saying I want to start with from in the line,

Spam Confidence

```
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1 :  continue
    num = float(stuff[0])
    numlist.append(num)
print('Maximum:', max(numlist))
```

```
X-DSPAM-Confidence: 0.8475
```

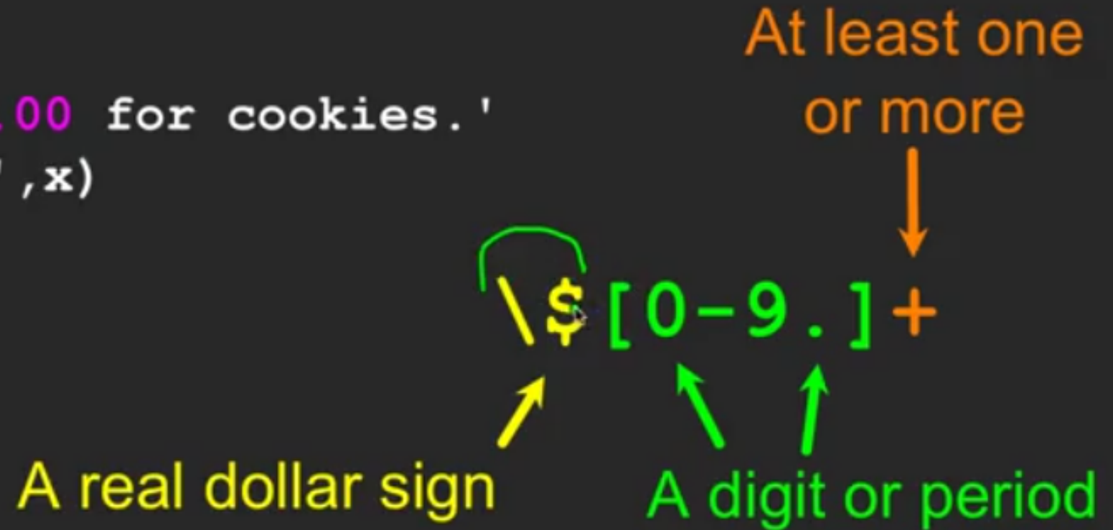
```
python ds.py
Maximum: 0.9907
```

So, this is similar to one of

Escape Character

If you want a special regular expression character to just behave normally (most of the time) you prefix it with '\'

```
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall('\$[0-9.]+',x)
>>> print(y)
['$10.00']
```



and the backslash just can be prefixed otherwise active character,

Common TCP Ports

- Telnet (23) - Login
- SSH (22) - Secure Login
- HTTP (80)
- HTTPS (443) - Secure
- SMTP (25) (Mail)
- IMAP (143/220/993) - Mail Retrieval
- POP (109/110) - Mail Retrieval
- DNS (53) - Domain Name
- FTP (21) - File Transfer

http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

port 80 is the one that we're going to play with the most it's HTTP.

Sockets in Python

Python has built-in support for TCP Sockets

```
import socket
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect( ('data.pr4e.org', 80) )
```

Host

Port

http://docs.python.org/library/socket.html

So, we're going to import the socket library to say,