Python – like Perl meets Java

(n m)

```
def choose(n, m):
     r = 1
     j = n
     for i in range(1, m+1):
          r = r * j // i
          j = j - 1
     return r
```

GCD

```
def gcd(n, m):
     if n==m:
          return n
     elif n<m:
          return gcd(m, n)
     else:
          return gcd(n-m, m)
```

A class

```
     class fetchandadd:
         def __init__(self):
             self.value = 0
         def fa(self, n):
             r = self.value
             self.value = r + n
             return r
         def __str__(self):
             return str(self.value)
```

Using a class

```
     x = fetchandadd()
     x.fa(4)
     x.fa(7)
     x
     print x
```

Server (of sorts) using `nc`

```
nc -l 22222
```

client using `nc`

```
nc localhost 22222
```

Try the client in python

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 22222))
s.send('Come here watson\n')
s.send('I need you\n')
s.close()
```

Now try saving that into a script file graham.py
Run it by typing

```
python graham.py
```

Now add the following first line

```
#! /usr/bin/python
```

and make it executable

```
chmod a+x graham.ph
```

Try out the client

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 22222))
s.listen(1)
strm, addr = s.accept()
print addr
data1 = strm.recv(1000)
p data1
…
```

or after the accept (and stored in a file)

```
data = strm.recv(1000)
while(data):
     print data,
     data = strm.recv(1000)
strm.close()
s.close()
```

by adding a single `while(1)` make the program so that several sequential connections can be handled
you will have to do a little more indentation

Try out some variations on the send. First with one send

```
s.send("""Come here watson
I need you
""")
```

and another one

```
s.send("Come here Watson.  I need you.")
```

Problem: TCP doesn't know about lines! `recv` may not return a full line!

Java-link solution: Create separate streams (files) for each direction.

```
instrm  = strm.makefile('r',0)
outstrm = strm.makefile('w',0)
```

and do I/O something like:

```
data = instrm.readline()
outstrm.write(data)
```

Task:
Write something like a fetch-and-add where input comes from several sources.
Each connection sees the results or others.
Lines are "legal" only is they contain legal positive integers.

Start by just echoing lines to the client.