Lab 3

Testing:
- put client & server in different folders
- use different types of files for testing
  - text file that is < 1024 bytes
  - larger image file (~200Kb)
- use scripts to "reset" and execute various tests.

How do I know what command the client is sending?

CLIENT: "GET file.txt" → UTF-8 encoded string

GET fixed - can't be anything else

DON'T DO THIS:

client:

```
s.send("GET".encode())
s.send(filename.encode())
# s.recv() to get file size
```

server →

```
cmd = s.recv(1024).decode()
filename = s.recv(1024).decode()
# s.send() filesize
```

```
print("Sending: GET")
s.send(----)          print("sent GET")
print("Sending:", filename)
s.send(....)          print("sent", filename)
print("Waiting for file size")
filesize = s.recv(...)
print("Received:", filesize)
```

```
print("Receiving command")
cmd = s.recv(....)
print("Received", cmd)
print("Receiving filename")
filename = s.recv(()--)
print("Received:", filename)
print("Sending file size")
s.send(...)
print("Sent:", x)
```

Using "verbose" flags

python3 client.py -v GET file1.txt

↑ verbose flag → print some debug statements

```
verbose = "-v" in sys.argv
:
if verbose: print("Sending GET")
s.send(----)
if verbose: print("Sent GET")
:
```

-vv → extra info

-vvv → even more

-vvvv → crazy!

```
verbose = 0
if "-v" in sys.argv: verbose = 1
if "-vv" in sys.argv: verbose = 2
:
```

```
⋮
if verbose >= 1 : print ("Connected to client at...")
⋮
if verbose >= 2 : print ("Sending GET")
s.send(...)
if verbose >= 2 : print ("Sent GET")
⋮
```

Lab 3 consequences:

```
s.send("READY".encode("utf-8"))
- - - - - - - - - - -
ready = s.recv(1024).decode("utf-8")
while ready != "READY":
    ready = ready + s.recv(1024).decode("utf-8")
```

- works when size (or content) of the expected data is known
  - but, when it's not (eg the command text in Lab 3), the protocol

should give you a way of knowing that the data is complete.

1) send expected # bytes previously (eg as a 4-byte unsigned integer)

2) if text data, use a "sentinel" at the end → special character or set of characters that would not appear in actual data
   → most common is \n
   → "Line-based protocol"

```
nl = ord("\n")    # integer value of newline
data = s.recv(1024)
while data[-1] != nl:
    data = data + s.recv(1024)
cmd = data.decode("utf-8")
```

— also worry about receiving file data:

```
while #not done:
    data = s.recv(1024)
    f.write(data)
```