C8 Socket Programming

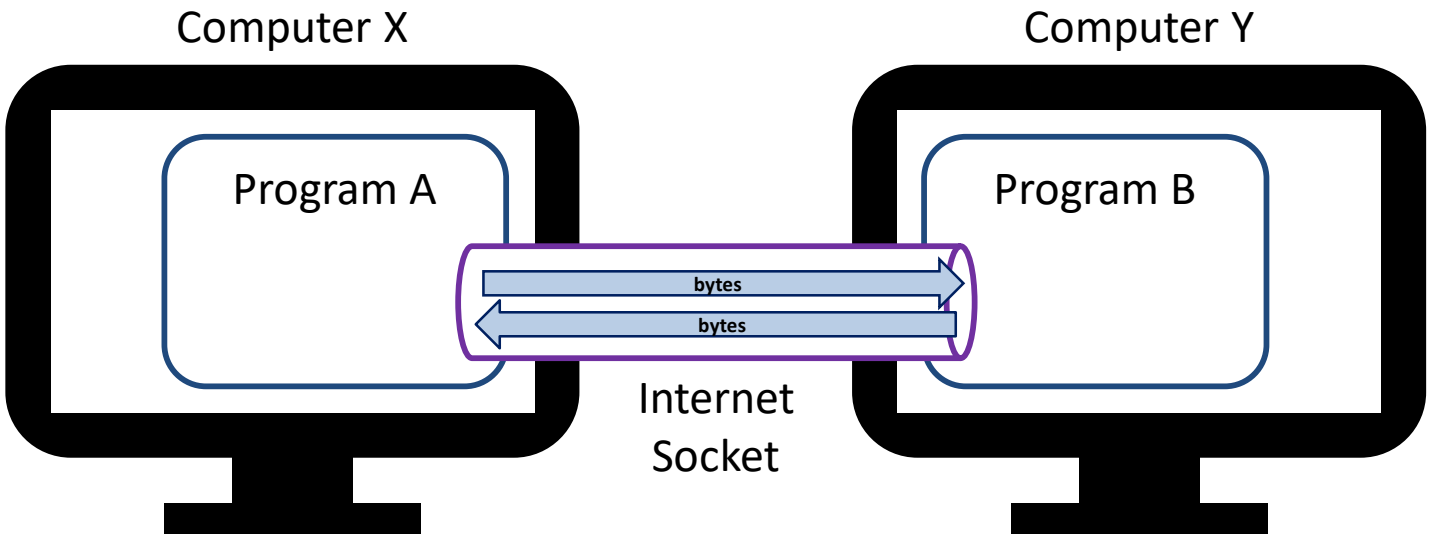
How programs communicate?

What is a Socket?

A socket connection is like a bi-directional communication pipe between two running programs.

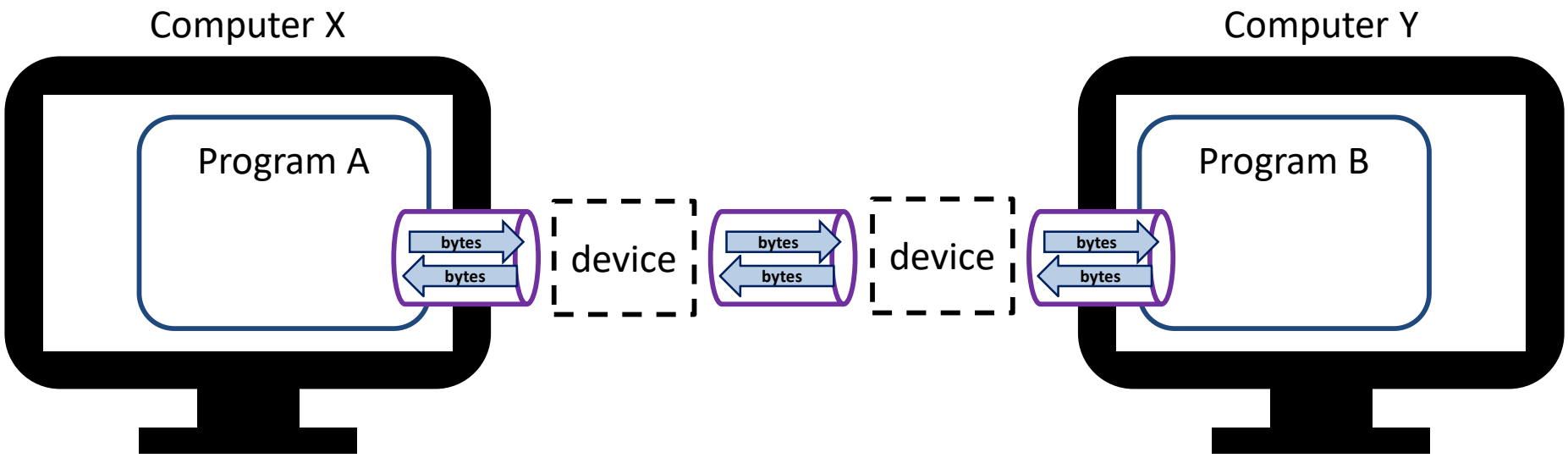
The Internet Socket, which is most common, uses the TCP/IP * protocol so that computers can access each other even over the network.

** Transmission Control Protocol and Internet Protocol*



For simplicity, we illustrate an Internet Socket as a pipe connecting the two computers.

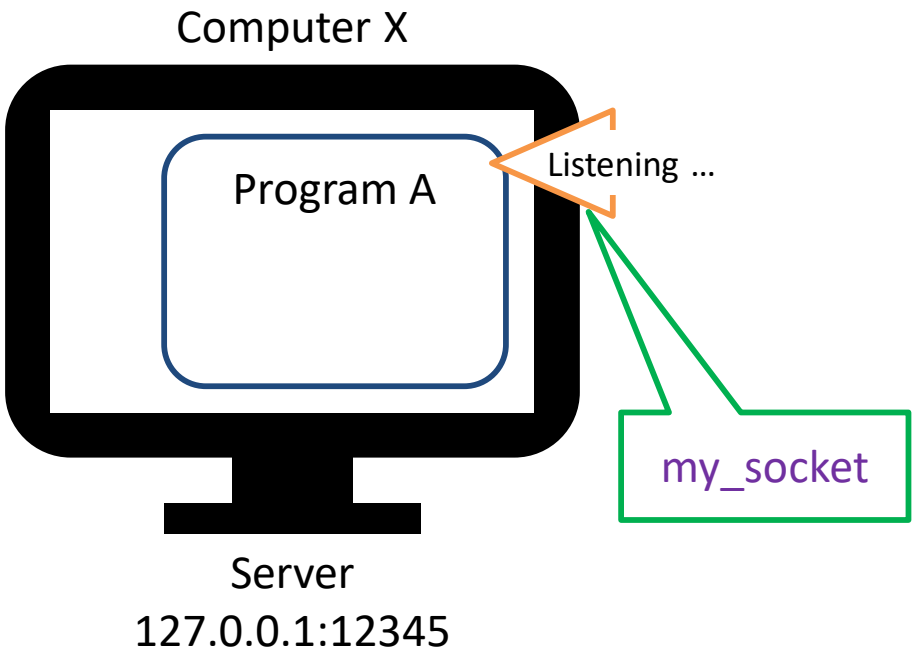
In reality, data communication between two computers passes through multiple devices before reaching its destination.



Create a server listening for client

server.py

```
from socket import *

my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()
```

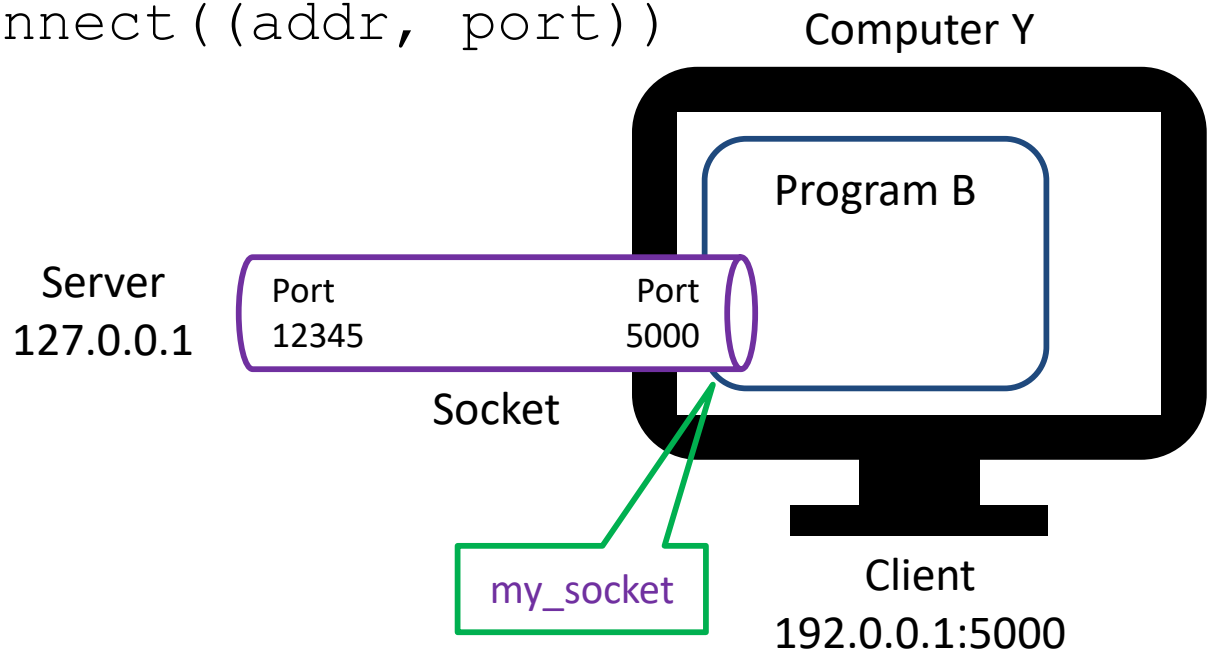


These 4 lines of code are essential to initialise my_socket()

Connecting a client to the server

client.py

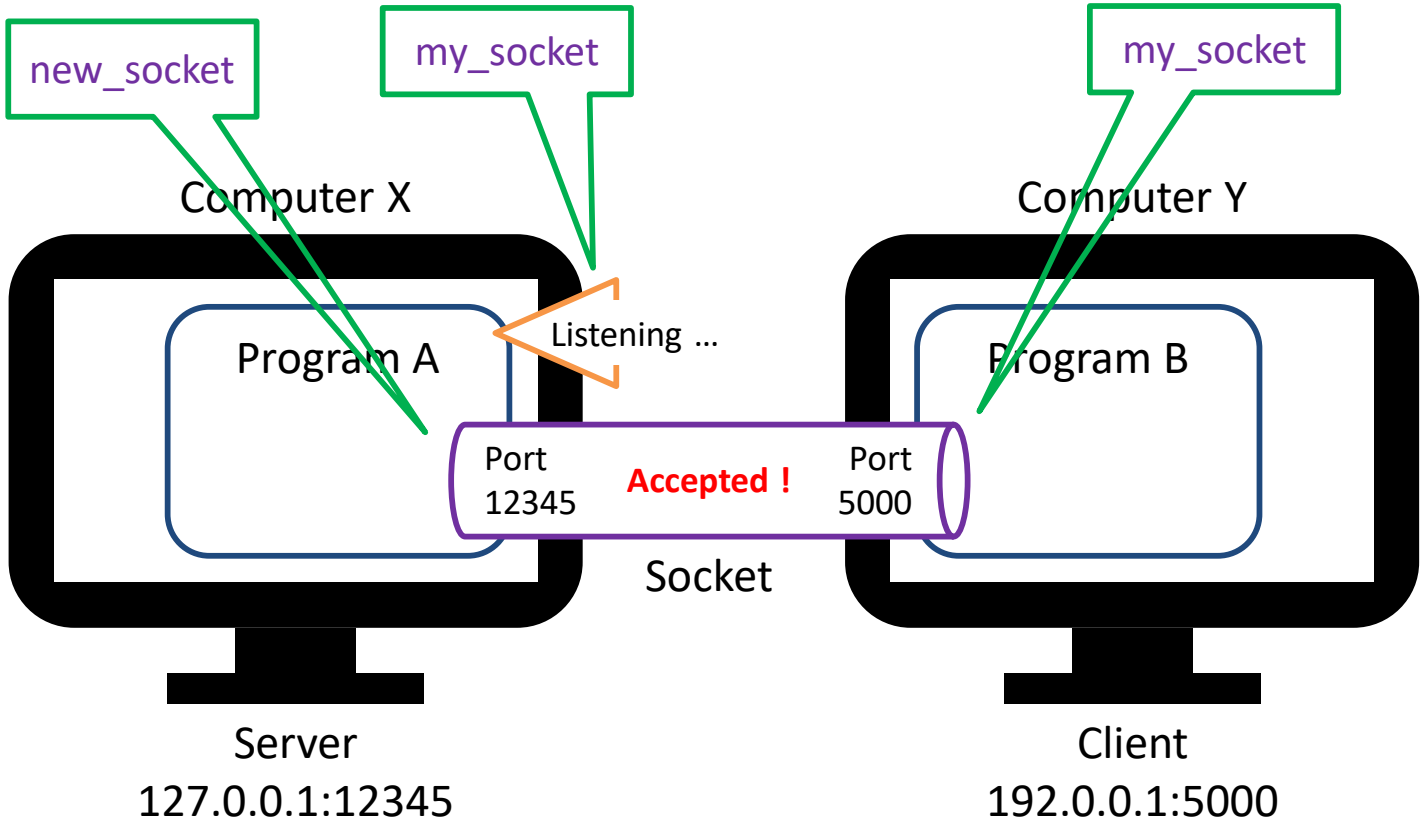
```
from socket import *

addr=input('Enter IPv4 address of server: ')
port=int(input('Enter port number of server: '))

my_socket.connect((addr, port))
```



Server accepts the client

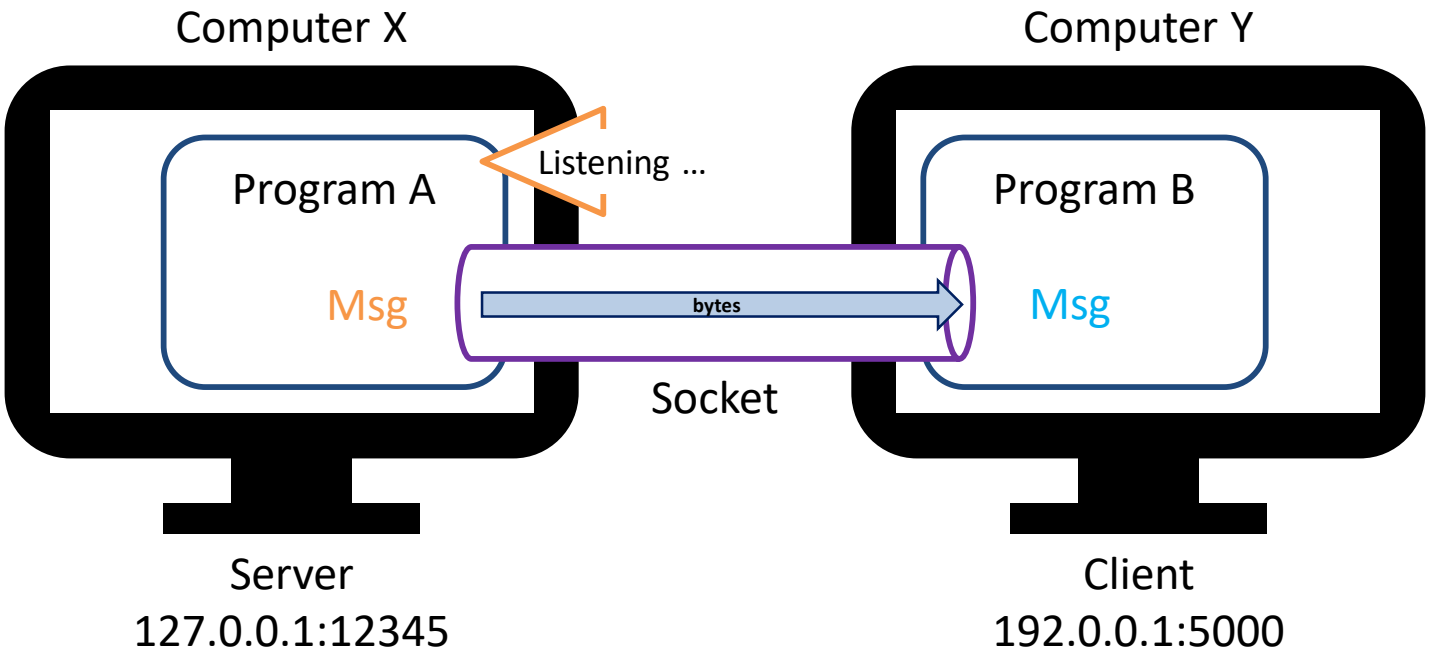
server.py

```
new_socket, addr = my_socket.accept()
Print('Connected to: ' + str(addr))
```



The `socket.accept()` method will **block** the program until a new client connects; then it will create a `new_socket` with the client's address.

Server sends a message



server.py

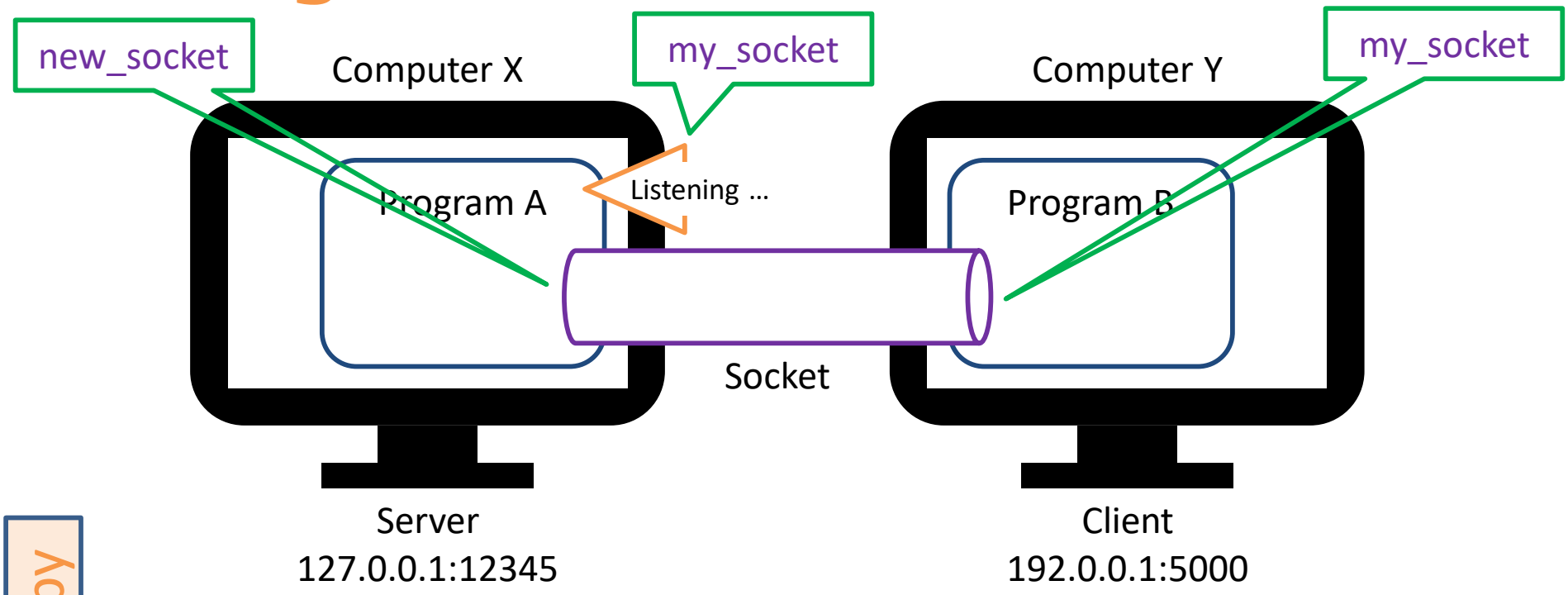
```
Msg = 'Hello from server'
new_socket.sendall(Msg.encode())
```

bufsize: max amount of data to be received at once. Should be small power of 2, eg 2^{10}

```
Msg = my_socket.recv(1024)
Print(Msg.decode())
```

client.py

Closing all the sockets



server.py

```
new_socket.close()
my_socket.close()
```

```
my_socket.close()
```

client.py

Unicode and Encodings

Python's Bytes Type

Sockets work at a very basic level, they can only send and receive data in the form of raw bytes.

ie. data is encoded into a sequence of 8-bit characters.

```
msg = 'string'.encode()
type(msg) → <class 'bytes'>
msg → b'string'
len(msg) → 6 which is same as len('string')

print(msg.decode()) → 'string'
```

Message need to be **encoded** before sending and be **decoded** after receiving.

Python's Bytes Type

The Chinese character 中 can be written as the `str` literal `'\u4e2d'` in Python. The use of the escape code `'\u'` to produce the character by specifying its **Unicode code point**.

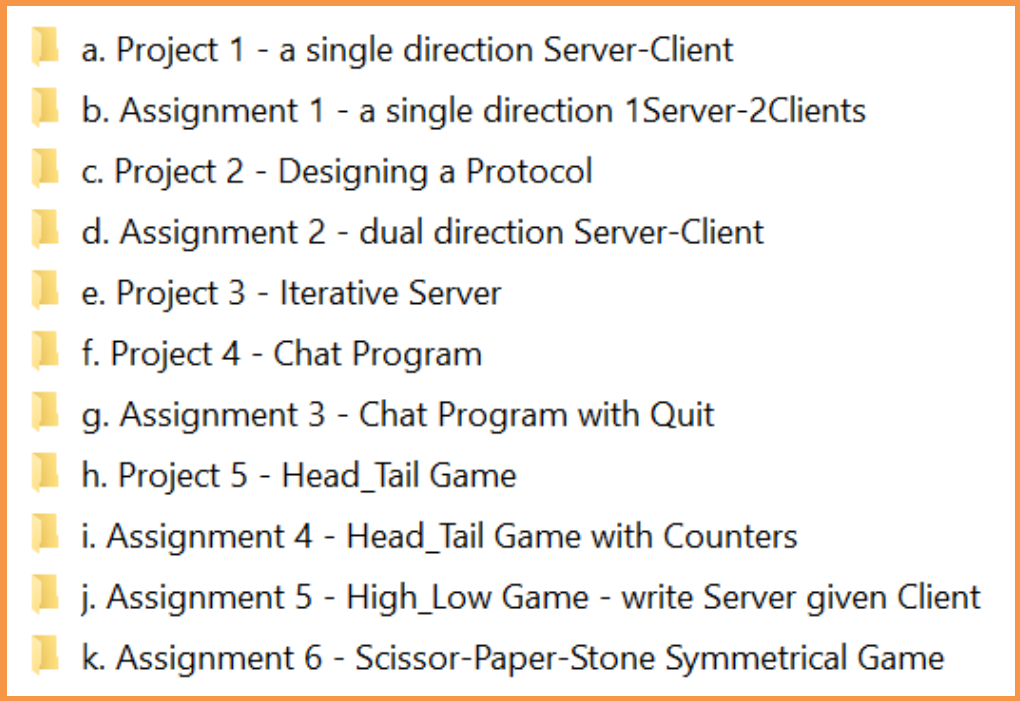
```
msg1 = '\u4e2d'
type(msg1) → <class 'str'>
len(msg1) → 1
print(msg1) → 中
```

Encode
before sending

```
msg2 = msg1.encode()
type(msg2) → <class 'bytes'>
len(msg2) → 3 in UTF-8
print(msg2) → b'\xe4\xb8\xad'
```

Decode
after receiving

```
msg3 = msg2.decode()
type(msg3) → <class 'str'>
len(msg3) → 1
print(msg3) → 中
```



Project 1

Single direction Server-Client communication

server.py

```
from socket import *

my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()

new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))
Msg = 'Hello from server\n'
new_socket.sendall(Msg.encode())

new_socket.close()
my_socket.close()
```

client.py

```
from socket import *

my_socket = socket()

addr=input('Enter IPv4 address of server: ')
port=int(input('Enter port number of server: '))
my_socket.connect((addr, port))

Msg = my_socket.recv(1024)
print(Msg.decode())

my_socket.close()
```

Assignment 1

Single direction communication from 1 Server to 2 Clients

Modify the `server.py` in Project 1 so that the server can communicate with two clients.

server.py

```
from socket import *

my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()

new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))
Msg = 'Hello from server\n'
new_socket.sendall(Msg.encode())

new_socket.close()
my_socket.close()
```

Same as `client.py` in Project 1:

client1.py

```
from socket import *

my_socket = socket()

addr=input('Enter IPv4 address of server: ')
port=int(input('Enter port number of server: '))

my_socket.connect((addr, port))
Msg = my_socket.recv(1024)
print(Msg.decode())

my_socket.close()
```

Same as `client.py` in Project 1:

client2.py

```
from socket import *

my_socket = socket()

addr=input('Enter IPv4 address of server: ')
port=int(input('Enter port number of server: '))

my_socket.connect((addr, port))
Msg = my_socket.recv(1024)
print(Msg.decode())

my_socket.close()
```

Project 2

Designing a Protocol

Why do we need a Protocol?

When a server send a long sequence of bytes, some of the data packet may be delayed during transportation through the network.

To understand this, we will break the data in to two sequences. After sending out the first piece of data, we call the `sleep()` method in the `time` module to simulate a delay in the busy network, then call `socket.sendall()` again to send the second piece of data.

server.py

```
from socket import *
from time import *

my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()
new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))

new_socket.sendall(b'Hello fr')
sleep(0.1)
new_socket.sendall(b'om server\n')

new_socket.close()
my_socket.close()
```

Run the `client.py` from Project 1 to receive the message from the server.

[Project 1] client.py

```
from socket import *
my_socket = socket()
my_socket.connect(('127.0.0.1', 12345))

print(my_socket.recv(1024).decode())

my_socket.close()
```

In this example, the client receives only the first part of the data and closes the socket. This will produce an error when the server tries to send the second piece of data.

In general, we should never assume that `socket.recv()` has received all the bytes that were sent.

We can agree on a protocol beforehand that any data we transmit will always end with a newline character `\n` and that the data itself will never contain the `\n` character.

The following `client.py` search for the `\n` character to detect the end of a transmission:

client.py

```
from socket import *
my_socket = socket()
my_socket.connect(('127.0.0.1', 12345))
data = b''
while b'\n' not in data:
    data += my_socket.recv(1024)
print(data.decode())

my_socket.close()
```

We will use this method whenever we need to receive any data.

Assignment 2

Bi-direction Server-Client communication

Assignment 2 :

Modify the `server.py` and `client.py` (with protocol), **given in the next two slides**, so that it will be a bi-directional communication between the server and the client.

server.py

```
from socket import *

my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()

new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))

new_socket.sendall(b'Hello from server\n')

new_socket.close()
my_socket.close()
```

client.py (with protocol)

```
from socket import *

my_socket = socket()
my_socket.connect(('127.0.0.1', 12345))

data = b''
while b'\n' not in data:
    data += my_socket.recv(1024)
print(data.decode())

my_socket.close()
```

Project 3

Iterative Server

Why we need a iterative server?

Currently, the server program exits immediately after it finishes working with a client. In reality, the server program should run continuously and is always listening for the clients' connection requests.

server.py

```
...
while True:
    print('Type Ctrl-F6 or close the shell
          to terminate the server.')
    new_socket, addr = my_socket.accept()
    print('Connected to: ' + str(addr))
    new_socket.sendall(b'Hello from server\n')
    new_socket.close()
my_socket.close()
```

Must include the 4 essential lines of code to initialise my_socket()

Iterative servers are easy to write, but they are limited and can only handle one client at a time.

Run the `client.py` from Project 2 to receive the message from the server.

client.py (with protocol)

```
from socket import *

my_socket = socket()
my_socket.connect(('127.0.0.1', 12345))

data = b''
while b'\n' not in data:
    data += my_socket.recv(1024)
print(data.decode())

my_socket.close()
```

Since the server is still running, you may execute the client program again, a few more times, after it closes `my_socket`.

Project 4

A Chat Program

server.py

```
...
print('Type Ctrl-F6 or close the shell to
      terminate the server.')
chat_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))
while True:
    data = input('Input message: ').encode()
    chat_socket.sendall(data + b'\n')
    print('WAITING FOR CLIENT ...')
    data = b''
    while b'\n' not in data:
        data += chat_socket.recv(1024)
    print('Client wrote: ' + data.decode()
chat_socket.close()
my_socket.close()
```

Must include the 4 essential lines of code to initialise my_socket()

Sending

Receiving

The client program is similar, except the order of sending and receiving is reversed.

client.py

```
from socket import *
chat_socket = socket()
chat_socket.connect(('127.0.0.1', 12345))

while True:
    print('WAITING FOR SERVER ...')
    data = b''
    while b'\n' not in data:
        data += chat_socket.recv(1024)
    print('Server wrote: ' + data.decode()
    data = input('Input message: ').encode()
    chat_socket.sendall(data + b'\n')

chat_socket.close()
```

Receiving

Sending

Assignment 3

A Chat Program with Quit

Assignment 3 : Chat program with Quit

Currently, there is no way to exit our chat programs other than using the Ctrl-F6 or closing the IDLE shell.

Modify the `chat_client.py` and `chat_server.py` so that both programs exit once the word **'BYE'** * is mentioned by either user. Remember to close all the sockets before exiting.

* The user may use both small or capital letters.

Project 5

A Turn-Based Game – "Guess Head or Tail"

Consider the following 'Guess Head or Tail' game:

head_tail.py

```
from random import *
def game(guess):
    if guess == 'H' or guess == 'T':
        if guess == choice(['H', 'T']):
            return 'You are right!'
        else:
            return 'Sorry, you are wrong.'
    elif guess == 'Q':
        return 'Thanks for playing the game. Bye!'
    else:
        return 'Please enter head (H), tail (T) \
               or Q to quit'
```

head_tail.py

```
def main():
    while True:
        guess = input('\n\nI will toss a coin, \
                       guess if it is a head (H) \
                       or tail (T) : ').upper()
        if guess == 'Q':
            break
        else:
            print(game(guess))
```

Copy the codes for `guess()` and `main()` and save them in the `head_tail.py` program. Run the program and play the game. Observe how the game is being implemented.

server.py

```
from socket import *
my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()

print('Type Ctrl-F6 or close the shell to \
       terminate the server.')
new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))

while True:

new_socket.close()
my_socket.close()
```

Must include the program code for game(guess)

Convert the code for `main()` and include it here

server.py

Observe how the following server program communicate with the client.

```
while True:
    new_socket.sendall(b'I will toss a coin, \
    guess if it is a head (H) or tail (T) : ')

    client_guess =  b''
    while b'\n' not in client_guess:
        client_guess += new_socket.recv(1024)
    client_guess = client_guess.decode()[-2]

    new_socket.sendall(game(client_guess)\
                       .encode() + b'\n')
    if client_guess == 'Q':
        break
```

client.py

```
from socket import *
my_socket = socket()
my_socket.connect(('127.0.0.1', 12345))

while True:
    data = my_socket.recv(1024)
    guess = input(data.decode()).upper()
    my_socket.sendall(guess.encode() + b'\n')
    data =  b''
    while b'\n' not in data:
        data += _socket.recv(1024)
    print(data.decode())

    if guess == 'Q':
        break

my_socket.close()
```

Send user's input to the server

Display message from the server

Assignment 4

A Turn-Based Game – "Guess Head or Tail" with Counters

Assignment 4 : Add Counters to the "Guess Head or Tail" game

N = Total number of guesses

C = Total number of correct guesses

Modify the server and client program to display the following when the user ended the game:

```
'You have guessed correctly C out of N times.'
```

Assignment 5

A Turn-Based Game – "Guess High or Low"

Assignment 5 : Write the server program

1. Study the "Guess High or Low" game.
2. Use, but do not modify, the provided client program.
3. Using the template for the server program, write the necessary code to communicate with the client program for the game.

Consider the following 'Guess High or Low' game:

high_low.py

```
from random import *

num = randint(1,100)
def game(guess):
    if guess.isdigit():
        if int(guess) == num:
            return 'You win!'
        elif int(guess) > num:
            return 'HIGH'
        else:
            return 'LOW'
    else:
        return 'Please enter number 1 to 100'
```

high_low.py

```
def main():
    counter = 5
    while counter > 0:
        guess = input('Guess a number 1 to 100: ')
        if game(guess) == 'You win!':
            print(game(guess))
            break
        else:
            print(game(guess))
        counter -= 1
    if counter == 0:
        print('You ran out of tries! Game over.')
```

Copy the codes for `guess()` and `main()` and save them in the `high_low.py` program. Run the program and play the game. Observe how the game is being implemented.

client.py

```
from socket import *
s = socket()
s.connect(('127.0.0.1', 12345))

while True:
      data = b''
      while b'\n' not in data:
            data += s.recv(1024)
      received = data.decode().strip()
      if received == 'LOW':
            print('Your guess is too low.')
      elif received == 'HIGH':
            print('Your guess is too high.')
      elif received == 'GUESS':
            guess = int(input('Enter guess (1-100): '))
            s.sendall(str(guess).encode() + b'\n')
      elif received == 'WIN':
            print('You win!')
            break
      elif received == 'GAMEOVER':
            print('You ran out of tries! Game over.')
            break

s.close()
```

Do not modify the client program.

(template) server.py

```
from socket import *
my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()

print('Type Ctrl-F6 or close the shell to \
       terminate the server.')
new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))

while True:


new_socket.close()
my_socket.close()
```

Must include the program code for game(guess) **and the random value** num

Write the code and include it here

Assignment 6

Symmetrical Game – "Scissor, Paper, Stone"

Assignment 6 :

1. The provided "Scissor-Paper-Stone" program code is for a player to play against the computer "Robot".
2. Modify the program code so that it becomes a symmetrical game where a human "Player1" plays against another human "Player2".
3. Write the server and client program codes for the symmetrical game.

Consider the following 'Scissor, Paper, Stone' game:

```
# player1 is a human     # player2 is a robot

from random import *
def game(opponent):
    player2 = choice(['Scissor', 'Paper',
'Stone'])
    print('Player2: ', player2)
    if player2 == opponent:
        return 'Draw'
    elif (player2 == 'Scissor' and opponent == \
'Paper') or (player2 == 'Paper' and opponent == \
'Stone') or (player2 == 'Stone' and opponent == \
'Scissor'):
        return 'Player2 wins!'
    else:
        return 'Player1 wins!'
```

```
def main():
    print('You are Player1 playing against Player2')
    counter = 3
    d = {'1':'Scissor', '2':'Paper', '3':'Stone'}
    while counter > 0:
        player1 = input('Player1: Enter (1)Scissor,\
                    (2)Paper or (3)Stone : ')
        if player1 in d:
            print('Player1: ', d[player1])
            print(game(d[player1]))
            counter -= 1
        else:
            print('Please enter only 1, 2 or 3.')
```

The End

Socket Programming