

```
In [1]: 1 import time
        2 from pynq.overlays.base import Base0verlay
        3 import socket
        4
        5 base = Base0verlay("base.bit")
        6 btns = base.btns_gpio
        7 leds = base.leds
```

Sockets

This notebook has both a client and a server functionality. One PYNQ board in the group will be the client and SENDS the message. Another PYNQ board will be the server and RECEIVES the message.

Server

Here, we'll build the server code to LISTEN for a message from a specific PYNQ board.

When we send/receive messages, we need to pieces of information which will tell us where to send the information. First, we need the IP address of our friend. Second, we need to chose a port to listen on. For an analogy, Alice expects her friend, Bob, to deliver a package to our back door. With this information, ALICE (server ip address) can wait at the BACK DOOR (port) for BOB (client ip address) to deliver the package.

Format of the information ipv4 address: ###.###.###.### (no need for leading zeros if the number is less than three digits) port: ##### (it could be 4 or 5 digits long, but must be >1024)

Use the socket documentation (Section 18.1.3) to find the appropriate functions <https://python.readthedocs.io/en/latest/library/socket.html>

```
In [3]: 1 #this will be the PYNQ
        2
        3
        4 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        5 # TODO:
```

```
In [3]: 1 #THIS WILL BE THE PYNQ
2
3
4 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 # TODO:
6 # 1: Bind the socket to the pynq board <CLIENT-IP> at port <LISTENING-PORT>
7 sock.bind(('192.168.0.189', 12345))
8
9 sock.listen(1)
10 print("listening\n..")
11 # 2: Accept connections
12 conn, addre = sock.accept()
13 print("connected\n")
14 # 3: Receive bytes from the connection
15 data = conn.recv(1024)
16 # 4: Print the received message
17 print(f"recieved: {data.decode()}")
18
19 conn.close()
20 sock.close()
```

listening

..

connected

recieved: Hello world!

Client

Now, we can implement the CLIENT code.

Back to the analogy, now we're interested in delivering a package to our friend's back door. This means BOB (client ip address) is delivering a package to ALICE (server ip address) at her BACK DOOR (port)

Remember to start the server before running the client code

ALICE (server ip address) at her BACK DOOR (port)

Remember to start the server before running the client code

```
In [*]: 1 #this will be my MAC
        2
        3
        4 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        5 # TODO:
        6 # 1: Connect the socket (sock) to the <SERVER-IP> and choosen port <LISTENING-PORT>
        7 #sock.connect(('192.168.0.189', 12345))
        8 sock.connect(('192.168.0.116', 12345))
        9 # 2: Send the message "Hello world!\n"
       10 message = "Hello world!\n"
       11 sock.send(message.encode())
       12 # 3: Close the socket
       13 print("socket closed")
       14 sock.close()
```

On your server, you should see the message and then the server will shutdown! When we close a socket, both the client and the server are disconnected from the port.

Instead, change the function above to send 5 messages before closing.

The pseudocode looks like this

- connect the socket
- for i in range(5)
 - msg = input("Message to send: ")
 - send the message (msg)
- close the socket