# Python Networking Reference Guide

Girls' Programming Network

# Girls' Programming Network - Canberra

This reference guide was written by GPN Canberra to supplement the GPN Australia Sassy Security Chatbots Networking project.

**Writers**
Catherine Murdoch
Matt Brindley

# Table of Contents

The Internet is an important part of the set of connections which we come across in our day to day lives. It can be basically described as a series of computer systems, all speaking to one another and passing messages along between them. When we talk about the connections between these computer systems, we're describing **computer networking**.

There are many key concepts to learn in the field of computer networking and in this reference document we'll cover the **Internet Protocol**, **sockets** and how to do **network programming in Python**.

## The Internet Protocol

The Internet Protocol (or **IP**) is a set of rules which tell computers how to speak to one another when they are connected (perhaps by WiFi or an Ethernet cable). The rules say that in order for a computer system to join in the networking fun, it must have a unique address which other systems can use to send messages to it - this is the **IP address.** You can think of an IP address a bit like a phone number, where everyone has a number they can be called with, and just like a person can have more than one phone number, a computer can have more than one IP address.

### How do I read an IP address?

IP addresses look like 4 sets of up to 3 numbers, separated by dots. Each number in the IP address can have a value from **0** up to **255**. For example:

192.168.1.14

In this address, we can see that each of the numbers is between **0** and **255**. The address is split up into these four pieces for reasons we won't get into here but you might notice that when you talk to other students or your tutors at the GPN events, their computers may have IP addresses that look very similar to yours.

### Different IP addresses for different uses

There are some special IP addresses and the most important one for any network programmer to know is:

127.0.0.1

This is a special IP address that every computer can use to talk to itself. It is also known as 'localhost' because it is 'local' to a single 'host' (computer). If you wanted to run two Python programs that talk to each other on the same computer, 127.0.0.1 is the IP address to use.

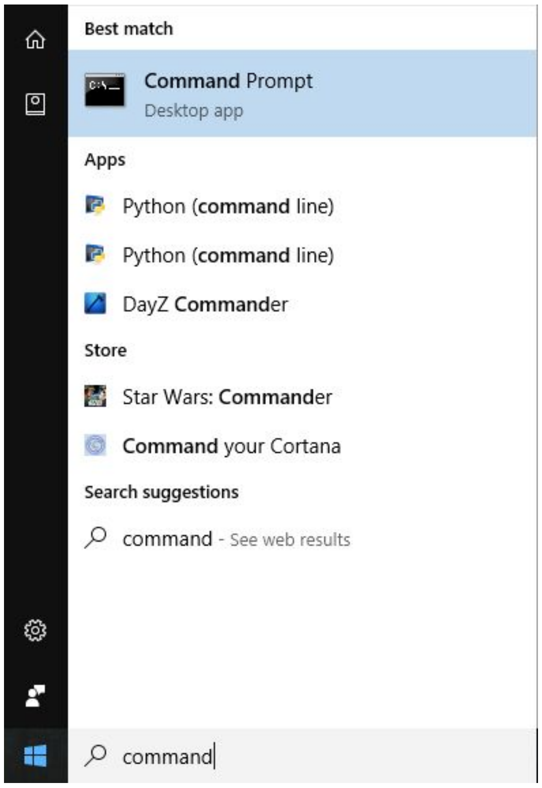
## How do I find my IP address on a local network?

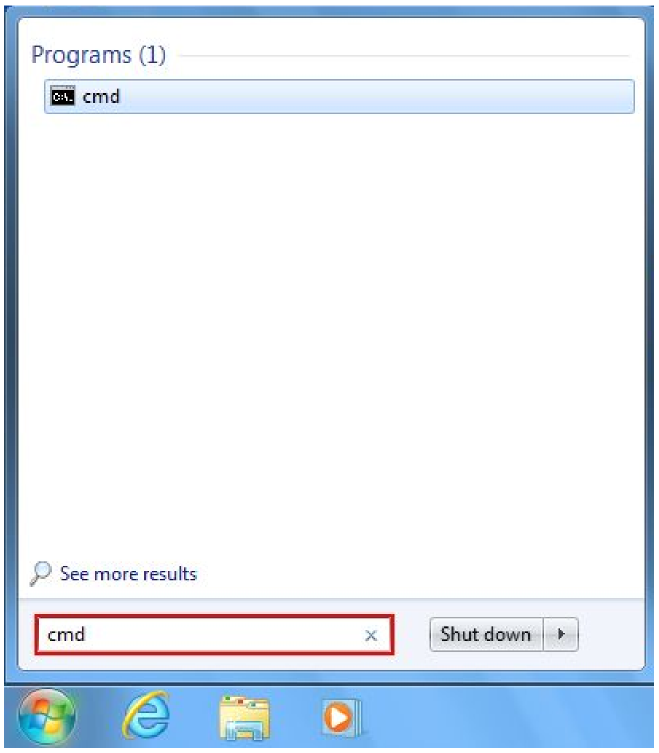
If you want to talk to another computer on the same local network, for example during a Girls Programming Network event you want your Python program to talk to another student's' Python program, follow these steps to find your local network IP address.

### Windows Computers

1. Open a command prompt (this is also sometimes called **cmd**)

For Windows 10 (left) and 7 (right):

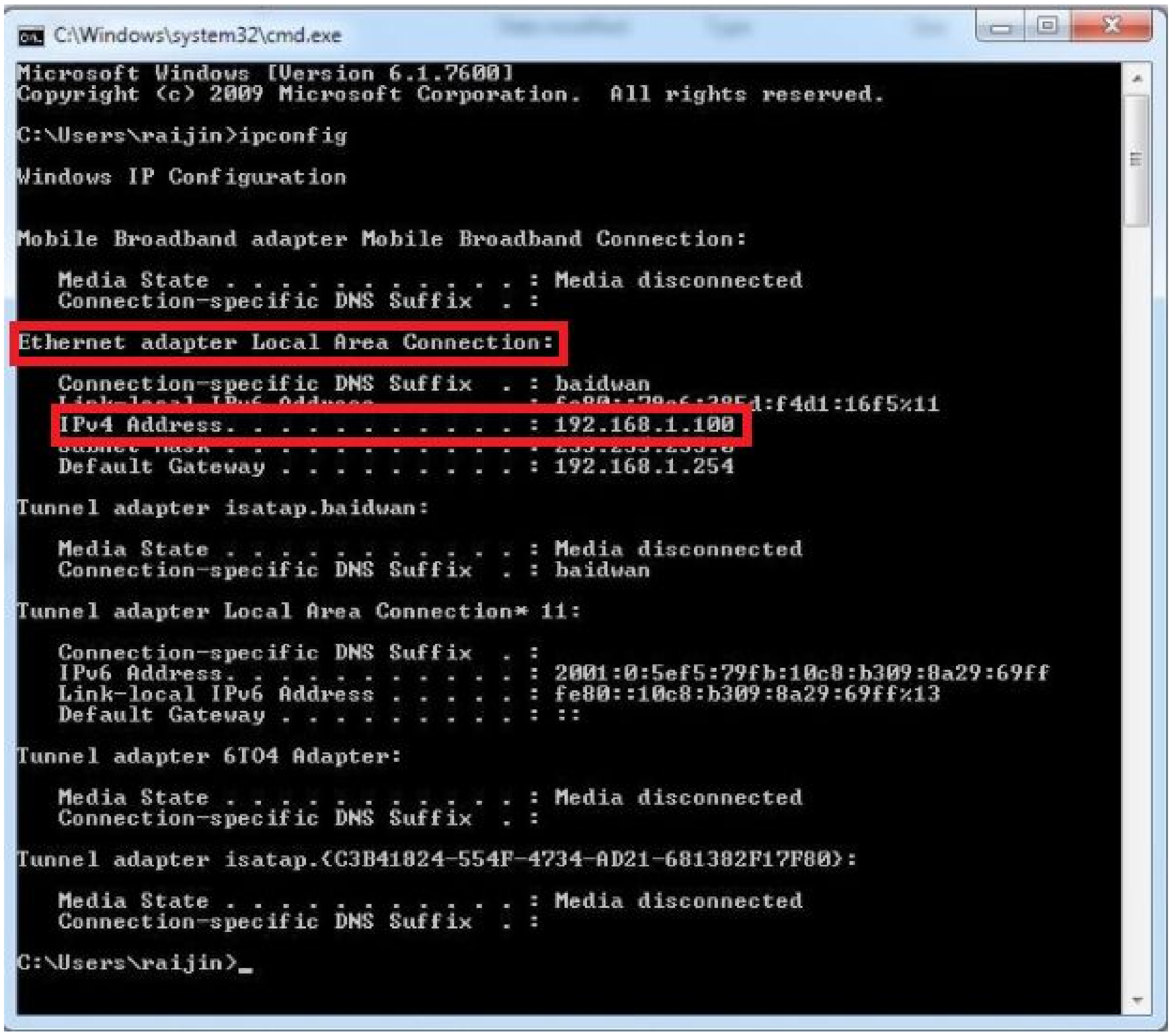




2. Then type **ipconfig** in the window which pops up and look at the text which is printed into the window
3. Identify the correct block of text for your computer's network connection
   a. For an **Ethernet connection** (a cable plugged into your computer), look for the block out text which starts with **Ethernet Adapter**.
   b. For a **Wireless connection** (WiFi), look for **Wireless Adapter**.
4. Look for **IPv4 Address** in the block of text you identified in step 2. That will be your local IP address.

In the example below, the IP address is:

192.168.1.100



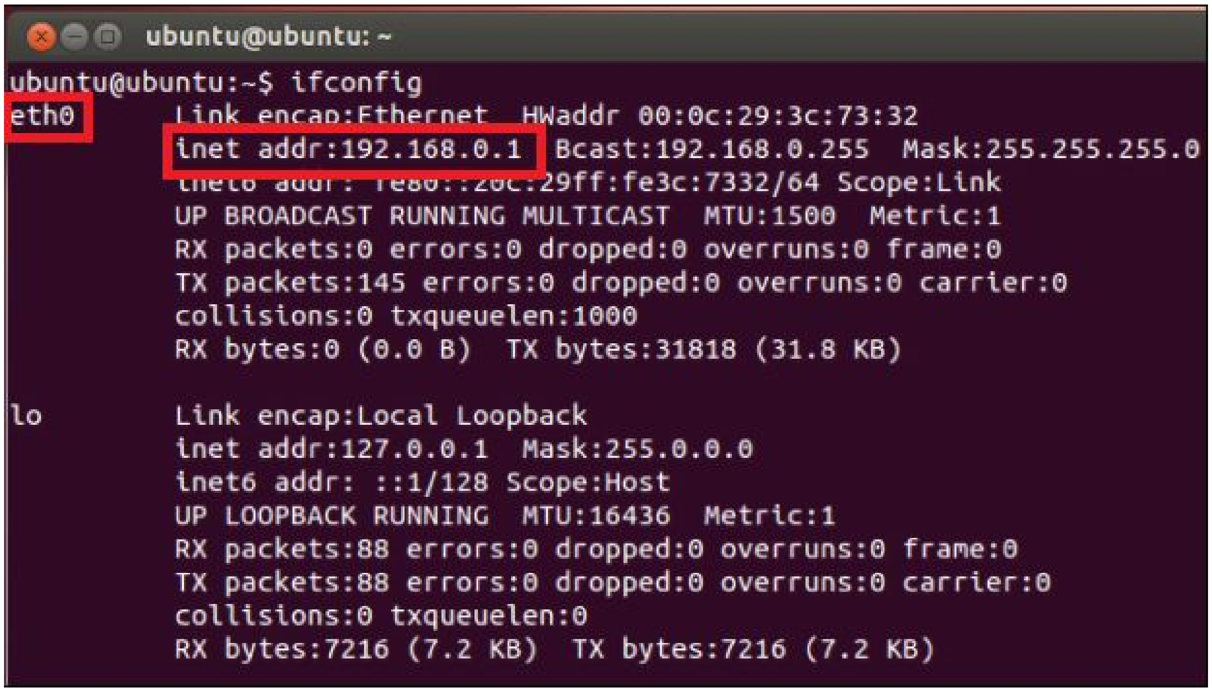
```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\raijin>ipconfig

Windows IP Configuration


Mobile Broadband adapter Mobile Broadband Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : baidwan
   Link-local IPv6 Address . . . . . : fe80::79a6:385d:f4d1:16f5%11
   IPv4 Address. . . . . . . . . . . : 192.168.1.100
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.254

Tunnel adapter isatap.baidwan:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : baidwan

Tunnel adapter Local Area Connection* 11:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:0:5ef5:79fb:10c8:b309:8a29:69ff
   Link-local IPv6 Address . . . . . : fe80::10c8:b309:8a29:69ff%13
   Default Gateway . . . . . . . . . : ::

Tunnel adapter 6TO4 Adapter:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter isatap.{C3B41824-554F-4734-AD21-681382F17F80}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\raijin>_
```

## Linux Computers

1. Open the terminal program by finding it in the start menu.
    a. You may be able to type the word **terminal** and have the computer search for the program for you but if you are having trouble, ask a tutor for help.
    b. Cheat sheet: Press **Alt + F2**, then type **gnome-terminal** in the popup window
2. Identify the correct block of text for your computer's network connection
    a. For an **Ethernet connection** (a cable plugged into your computer), look for the block of text which starts with **eth0** or something similar
    b. For a **Wireless connection** (WiFi), look for **wlan0** or something similar
3. Look for **inet addr:** in the block of text you identified in step 2. That will be your local IP address.

In the example below, the IP address is:

192.168.0.1



```
ubuntu@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:3c:73:32
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe3c:7332/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:145 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:31818 (31.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:88 errors:0 dropped:0 overruns:0 frame:0
          TX packets:88 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7216 (7.2 KB)  TX bytes:7216 (7.2 KB)
```

We can also see the localhost address, **127.0.0.1**, in the output text as well.

# Sockets

## What are Sockets?

When two computers want to talk to one another over a network, they each create an object called a **socket**. We use sockets to represent the connection between these computers while they are communicating. Sockets come in a few different varieties and you'll need to know which one to use depending on what you want to accomplish at the time.

- Sockets can **listen** for communications from other computers
- Sockets can **connect** to other computers which are listening

In the initial exercises you'll complete at GPN events, you will only need to **connect** to other computers which the tutors have set up for you.

You could think of connecting two computers together using sockets like making a phone call to a friend. You type a phone number into your mobile phone and click 'dial' or 'call'. Then your phone tries to make a connection to your friends phone. But your friend has to accept the phone call first, before you can communicate with them.

## Ports

Now imagine that one phone is shared by a whole family of people. The people in this family represent all the different programs that might be running on a computer that want to talk over a network. When you call your friend on their shared phone, when someone answers you then have to say the name of the person you want to talk to. This is kind of how **ports** are used.

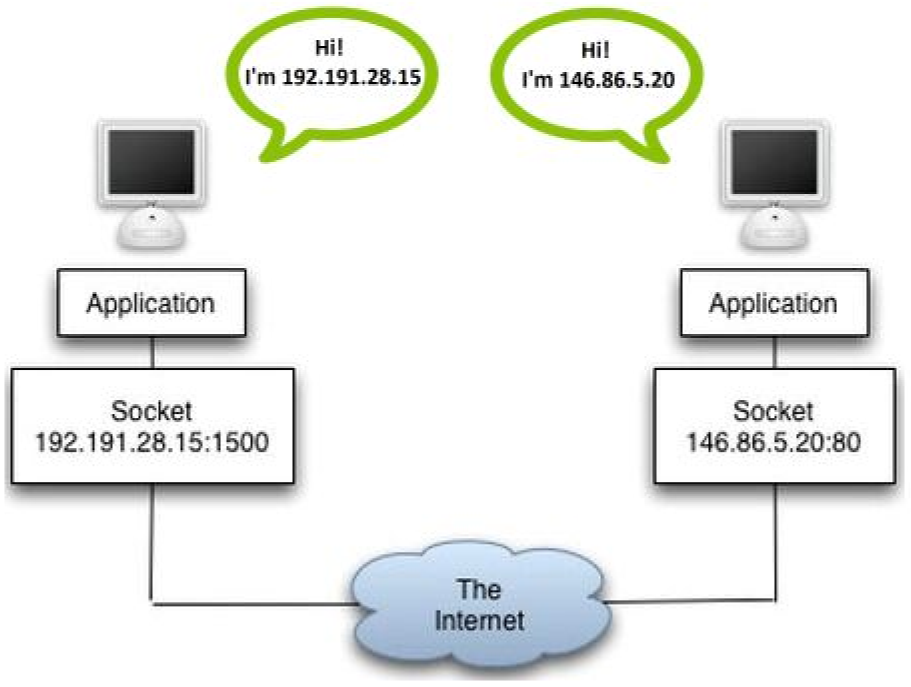
If each computer was only able to have one socket open at a time, then only one program would be able to talk over a network at a time. But we like to do lots of things all at the same time on our computers. We have programs for browsing the internet, streaming music, playing online games, chatting with friends, all talking over a network at the same time. Each of these different programs will be running on the **same computer** and have the **same IP address**, but they will be using **different ports**, and using different ports allows us to separate all the conversations all the programs are having so they don't get in each other's way. Each program only cares about the port that they have been assigned.

## Tying it all together

A socket needs two pieces of information in order to make a connection with someone else: an IP address and a port.
In the example above, your mobile phone represents one side of the connection and is like a socket, and your friends mobile phone is the other side of the connection and is like the other socket. But in order to

connect your phone (or socket) to your friends, you have to supply the address of the person you are calling and your friend has to pick up the call. The computers in the diagram below are communicating using sockets over the internet and must have completed this set of steps to begin sending message to each other.

In the diagram below, there is an application running on the computer with IP address 192.191.28.15, which wants to make a connection to a program running on the computer with IP address 146.86.5.20. The application running on 146.86.5.20 only cares about connections on its port 80, and will ignore any incoming calls for any ports other than 80.



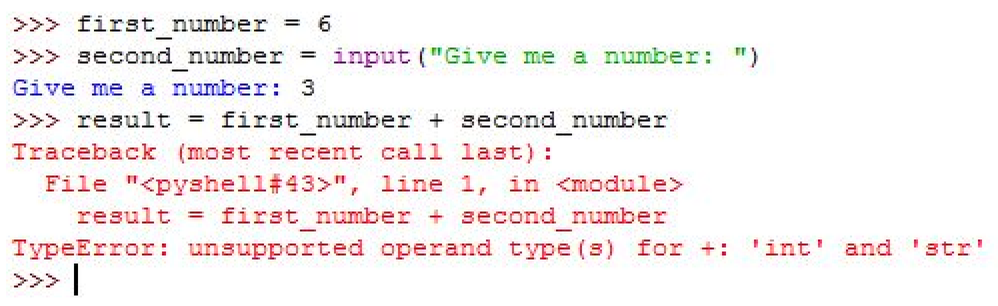
## Common Errors and Bugs

You might come across some error messages or bugs during your time writing networking programs with Python. In this section you will find some common errors and bugs and an explanation of why each might have happened.

### The difference between Errors and Bugs

Errors and bugs are not the same thing.

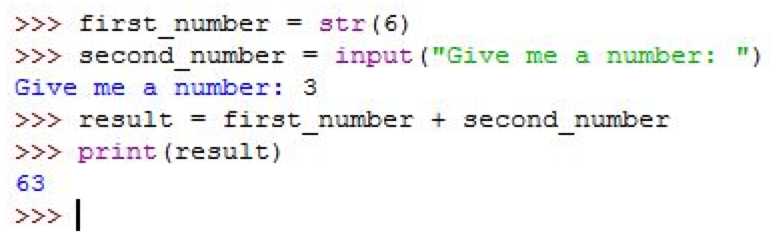
An **error** happens when there’s something wrong with the code that causes Python to not understand what you’re trying to make it do, or there is something stopping Python from being able to follow your instructions. Errors are simple to identify and fix because Python tells you what the problem is.
Here is an example of an error:



```
>>> first_number = 6
>>> second_number = input("Give me a number: ")
Give me a number: 3
>>> result = first_number + second_number
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    result = first_number + second_number
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 
```

Python is telling us exactly where the error is and what the problem is. In the expression “result = first_number + second_number”, two two operands (first_number and second_number) cannot be added together. Python even tells us that first_number is of type ‘int’ or Integer, and second_number is of type ‘str’ or String. We can easily fix this by ensuring first_number and second_number are both the same type, either by using str(first_number) to both to be strings, or int(second_number) to force both to be integers.

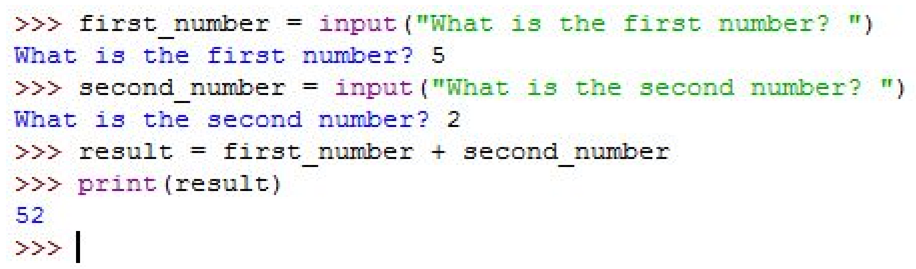
Here is the fixed code:



```
>>> first_number = str(6)
>>> second_number = input("Give me a number: ")
Give me a number: 3
>>> result = first_number + second_number
>>> print(result)
63
>>> 
```

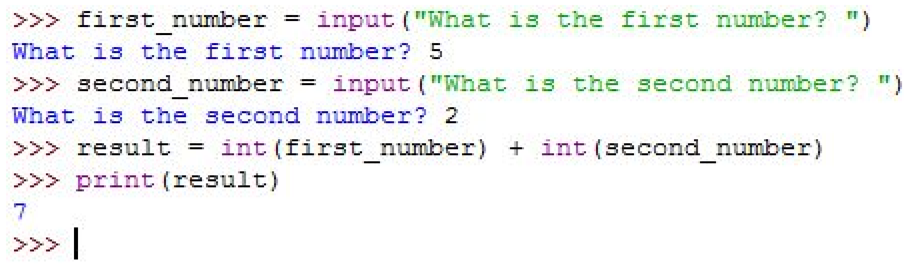
A **bug** is when the code is technically correct and runs, but it’s doing something other than what you expected or meant it to do. Bugs are sometimes a little harder to find and fix, because translating from english to Python code isn’t always straightforward and sometimes things get lost in translation.

Here is an example of a bug:



```
>>> first_number = input("What is the first number? ")
What is the first number? 5
>>> second_number = input("What is the second number? ")
What is the second number? 2
>>> result = first_number + second_number
>>> print(result)
52
>>> 
```

In this example, you ask the user to supply two number, then print the result of the first number plus the second number, 5 plus 2 equals 7. Except when you print the result, it prints both numbers next to each other. The code is correct and runs, but it doesn't do what you expect, which is to print 7. The way to fix this bug is to recognise that input() returns a String, which needs to be converted to an Integer before they can be treated as numbers.
Here is the fixed code:



```
>>> first_number = input("What is the first number? ")
What is the first number? 5
>>> second_number = input("What is the second number? ")
What is the second number? 2
>>> result = int(first_number) + int(second_number)
>>> print(result)
7
>>> 
```

If you have an error or a bug that you're having trouble with, let a tutor know so we can help you fix it!

## Connection Refused

```
socket.error: [Errno 111] Connection refused
socket.error: [Errno 61] Connection refused
```

These errors happen when there is nothing on the other side listening for connections. This could be because you are using the wrong IP address, port, or both, or that the program you are trying to connect to isn't running. Make sure you're using the correct IP address and port, and if you're trying to connect to a python Program, make sure it's running on the other computer.

## Permission Denied

```
socket.error: [Errno 13] Permission Denied
```

This error happens when you try to bind to a port lower than 1024. Either use a higher port (try 8888) or run python as an administrator.

## Broken Pipe

```
socket.error: [Errno 32] Broken pipe
```

This error happens when one side of a connection tried to write to the socket, but the other side has already closed the connection. Check the python program on the other side of the connection is still running and hasn't ended/terminated early.

## Address Already In Use

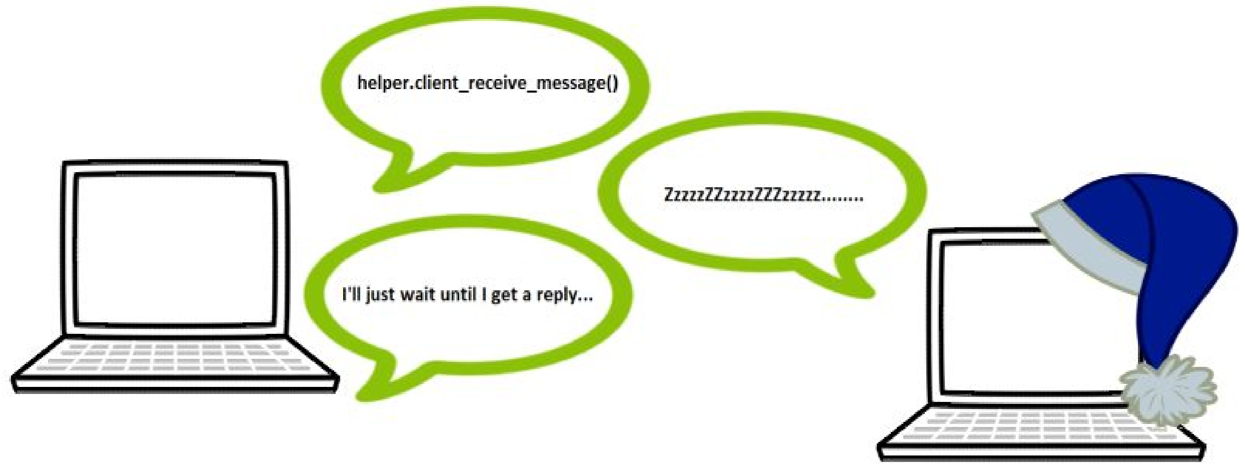
```
socket.error [Errno 98] Address already in use
```

This error happens when your python program crashes or quits without closing the socket properly. In this case the operating system is still reserving the port that was previously in use, so it can't be used again until the operating system stops reserving it. Either wait a few minutes and try again until it works, or use a different port. The best way to stop this happening is to make sure you do 'socket.close()' on your socket before your python program terminates.

## My program is freezing when it should be sending/receiving messages

This is a common bug when writing networking code with Python. When a program is trying to receive messages over the network, this is known as a **blocking** call. This means the rest of the program is blocked from running until something happens, in this case, data is received.

If your program is trying to communicate with another program over a network but freezes, try to find out which Python instruction is being blocked, and if it's one where you should be receiving data, check that the program you're trying to communicate with is actually sending data to you.


helper.client_receive_message()
ZzzzzZZzzzzZZZzzzzz........
I'll just wait until I get a reply...

## Glossary

| | |
|---|---|
| **Command Prompt** | The Windows command line interface where you can run commands like ipconfig. |
| **Computer Network** | A collection of computer systems connected together so they may communicate. |
| **Connect (socket)** | To attempt to begin communicating with another computer system using sockets. |
| **Ethernet** | A computer network connection which uses a cable which is plugged into the computer and something else. |
| **ifconfig** | The Linux command to run to find your IP address. |
| **ipconfig** | The Windows command to run to find your IP address. |
| **IP Address** | The unique address of a computer with a network connection. |
| **Listen (socket)** | To wait for another computer to begin communicating using sockets. |
| **Network Programming** | The art of writing a computer program which can communicate with other programs across a computer network. |
| **Ping** | To send a message to a computer which it should reply to if it is online. |
| **Port** | An object unique to each program running on a single computer that communicates over a network. |
| **Socket** | An object in a computer program which represents a connection to another computer system. |
| **Terminal** | The Linux command line interface where you can run commands like Ifconfig. |
| **Timeout** | The error which occurs when a computer takes too long to respond to network communications. |