

# Lab-Report

Report No:02

Course code: ICT-3207

Course title: Computer Networks

Date of Performance:16/01/2021

Date of Submission:

## Submitted by

Name: Afrin Zaman & Mahfuza  
Talukdar

ID:IT-18008 & IT-18009

3<sup>rd</sup> year 2<sup>nd</sup> semester

Session: 2017-2018

Dept. of ICT

## Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

## **Theory:**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

**Modules:** Modules allow reusing a number of functions in other programs.

## **Exercises:**

**Create a python project using with SDN LAB**

Project name:

Project contents:  
☒ Use default

Directory:

Project type  
 Choose the project type  
☒ Python ☐ Jython ☐ IronPython

Grammar Version

Interpreter  
[Please configure an interpreter before proceeding.](#)

Additional syntax validation: <no additional grammars selected>

☒ Add project directory to the PYTHONPATH  
☐ Create 'src' folder and add it to the PYTHONPATH  
☐ Create links to existing sources (select them on the next page)  
☐ Don't configure PYTHONPATH (to be done manually later on)

Working sets  
☐ Add project to working sets

Working sets:

### Python function (save as function.py)

The screenshot shows an IDE with a project named 'Hello World'. The file explorer on the left shows a folder 'Hello World' containing several files: 'For.py', 'Fourth.py', 'function.py' (selected), 'Hello.py', 'Hello World', 'IF.py', 'Second.py', and 'Third.py'. The editor window shows the code for 'function.py':

```

1 def say_hello():
2     # block belonging to the function
3     print('hello world')
4     # End of function
5 if __name__ == '__main__':
6     say_hello() # call the function
7

```

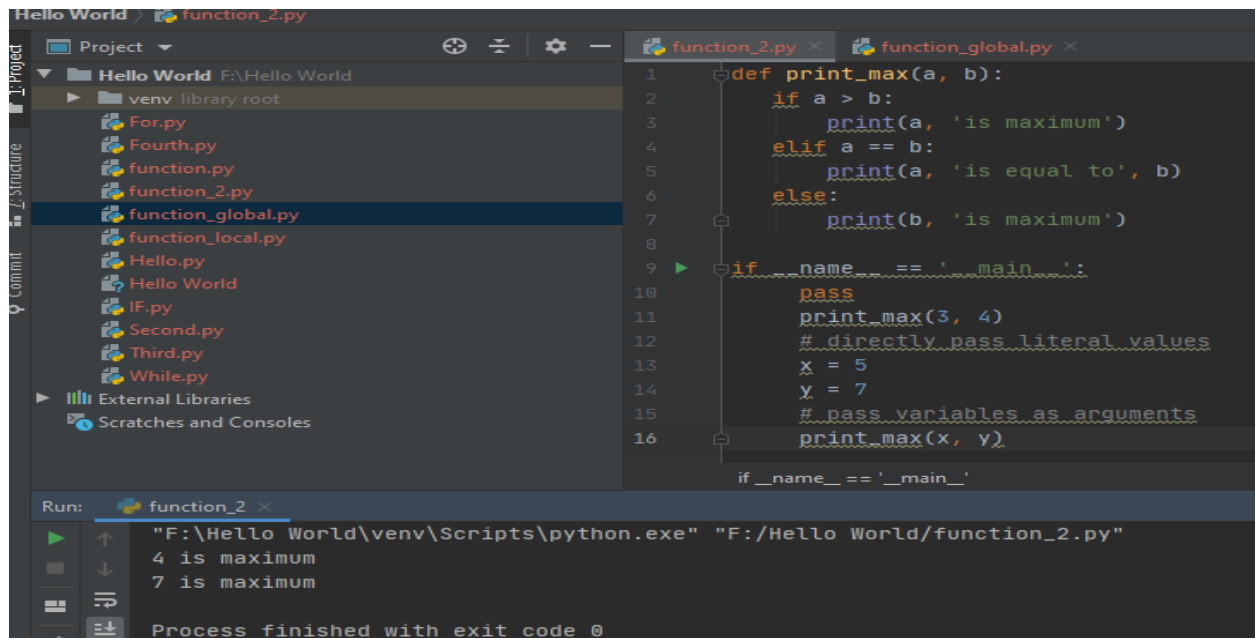
Below the editor, the 'Run' console shows the execution of the function:

```

Run: function
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function.py"
hello world
Process finished with exit code 0

```

### Python function (save as function 2.py)



```
def print_max(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')

if __name__ == '__main__':
    pass
    print_max(3, 4)
    # directly pass literal values
    x = 5
    y = 7
    # pass variables as arguments
    print_max(x, y)
```

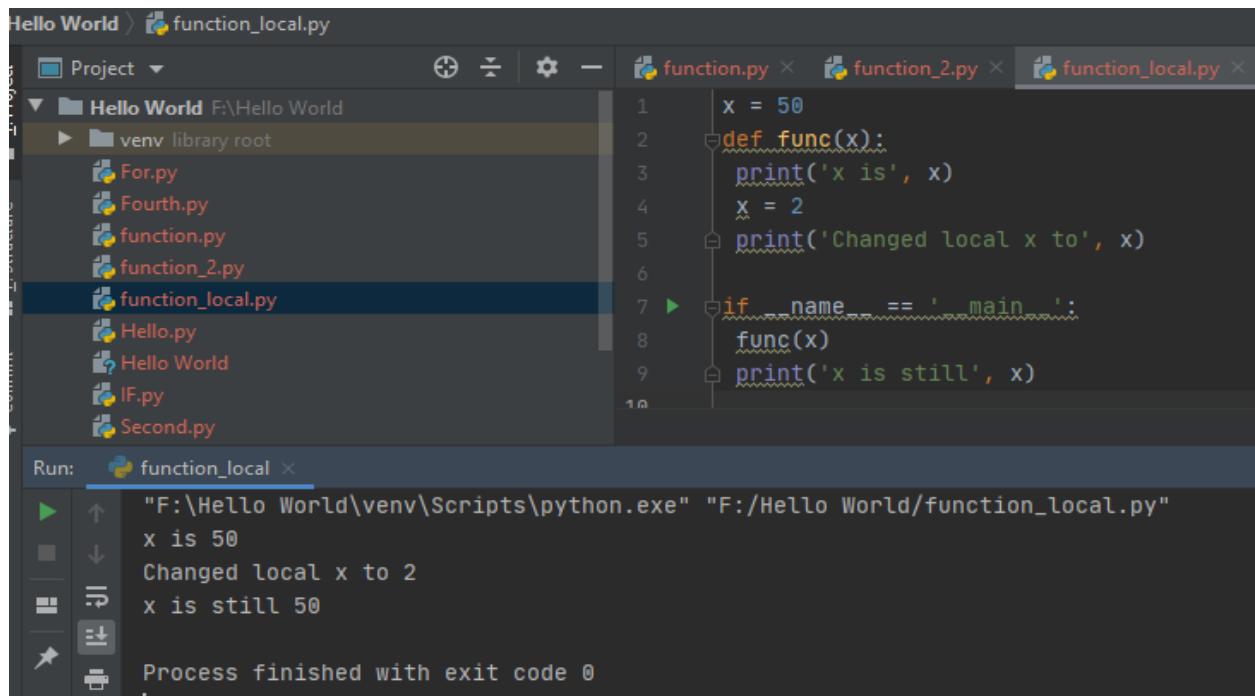
Run: function\_2

"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function\_2.py"

4 is maximum  
7 is maximum

Process finished with exit code 0

### Local variable (save as function\_local.py)



```
x = 50
def func(x):
    print('x is', x)
    x = 2
    print('Changed local x to', x)

if __name__ == '__main__':
    func(x)
    print('x is still', x)
```

Run: function\_local

"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function\_local.py"

x is 50  
Changed local x to 2  
x is still 50

Process finished with exit code 0

### Global variable (save as function\_global.py)

The screenshot shows an IDE with a project named 'Hello World'. The file explorer on the left lists several files, with 'function\_global.py' selected. The main editor displays the code for 'function\_global.py':

```
1 x = 50
2 def func():
3     global x
4     print('x is', x)
5     x = 2
6     print('Changed global x to', x)
7 if __name__ == '__main__':
8     func()
9     print('Value of x is', x)
```

Below the editor, the 'Run' panel shows the command executed: `"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function_global.py"`. The output of the program is displayed as follows:

```
x is 50
Changed global x to 2
Value of x is 2
```

The panel also indicates that the process finished with exit code 0.

## Python modules

The screenshot shows an IDE with three tabs open: 'function\_2.py', 'mymodule.py', and 'function\_global.py'. The 'mymodule.py' tab is active, showing the following code:

```
1 def say_hi():
2     print('Hi, this is mymodule speaking.')
3     __version__ = '0.1'
```

```
Project
└─ Hello World
   └─ venv
      └─ library root
         └─ For.py
            └─ Fourth.py
               └─ function.py
                  └─ function_2.py
                     └─ function_global.py
                        └─ function_local.py
                           └─ Hello.py
                              └─ Hello World
                                 └─ IF.py
                                    └─ module_demo.py
                                       └─ mymodule.py
```

```
1 import mymodule
2 if __name__ == '__main__':
3     mymodule.say_hi()
4     print('Version', mymodule.__version__)
5
6 from mymodule import say_hi, __version__
7
8 if __name__ == '__main__':
9     say_hi()
10    print('Version', __version__)
```

```
Run: module_demo
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/module_demo.py"
Hi, this is mymodule speaking.
Version 0.1
Hi, this is mymodule speaking.
Version 0.1
Process finished with exit code 0
```

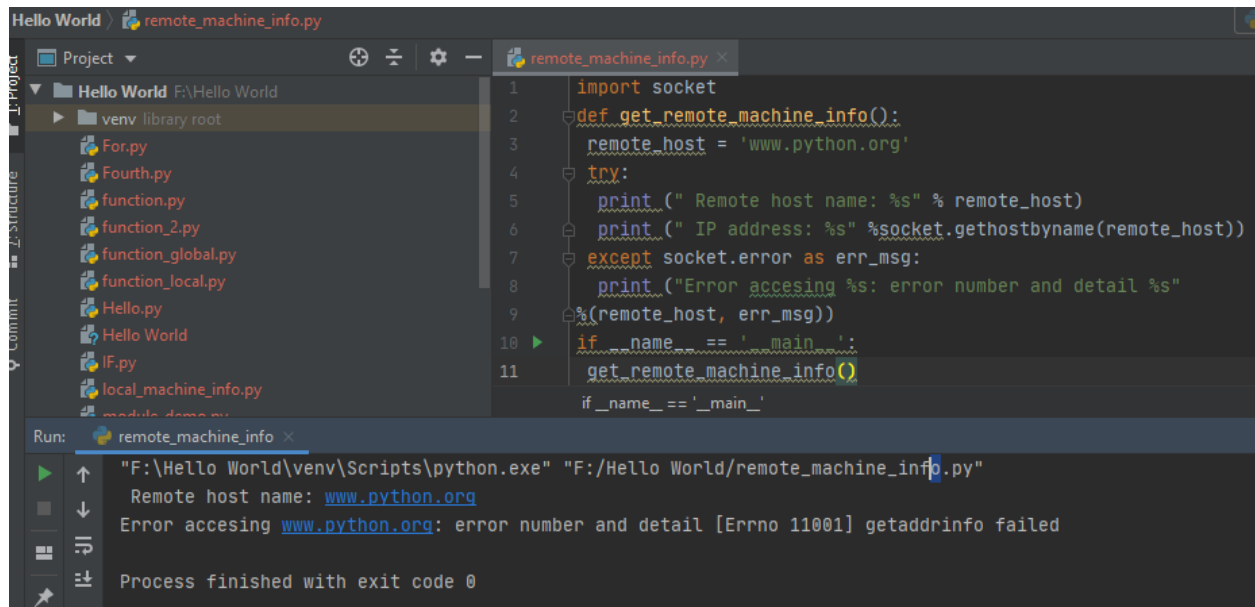
## Printing your machine's name and IPv4 address

```
Project
└─ Hello World
   └─ venv
      └─ library root
         └─ For.py
            └─ Fourth.py
               └─ function.py
                  └─ function_2.py
                     └─ function_global.py
                        └─ function_local.py
                           └─ Hello.py
                              └─ Hello World
                                 └─ IF.py
                                    └─ local_machine_info.py
```

```
1 import socket
2 def print_machine_info():
3     host_name = socket.gethostname()
4     ip_address = socket.gethostbyname(host_name)
5     print(" Host name: %s" % host_name)
6     print(" IP address: %s" % ip_address)
7 if __name__ == '__main__':
8     print_machine_info()
```

```
Run: local_machine_info
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/local_machine_info.py"
Host name: DESKTOP-FB5S452
IP address: 127.0.0.1
Process finished with exit code 0
```

## Retrieving a remote machine's IP address



```
1 import socket
2 def get_remote_machine_info():
3     remote_host = 'www.python.org'
4     try:
5         print(" Remote host name: %s" % remote_host)
6         print(" IP address: %s" % socket.gethostbyname(remote_host))
7     except socket.error as err_msg:
8         print("Error accessing %s: error number and detail %s"
9               %(remote_host, err_msg))
10 if __name__ == '__main__':
11     get_remote_machine_info()
12 if __name__ == '__main__':
```

Run: remote\_machine\_info x

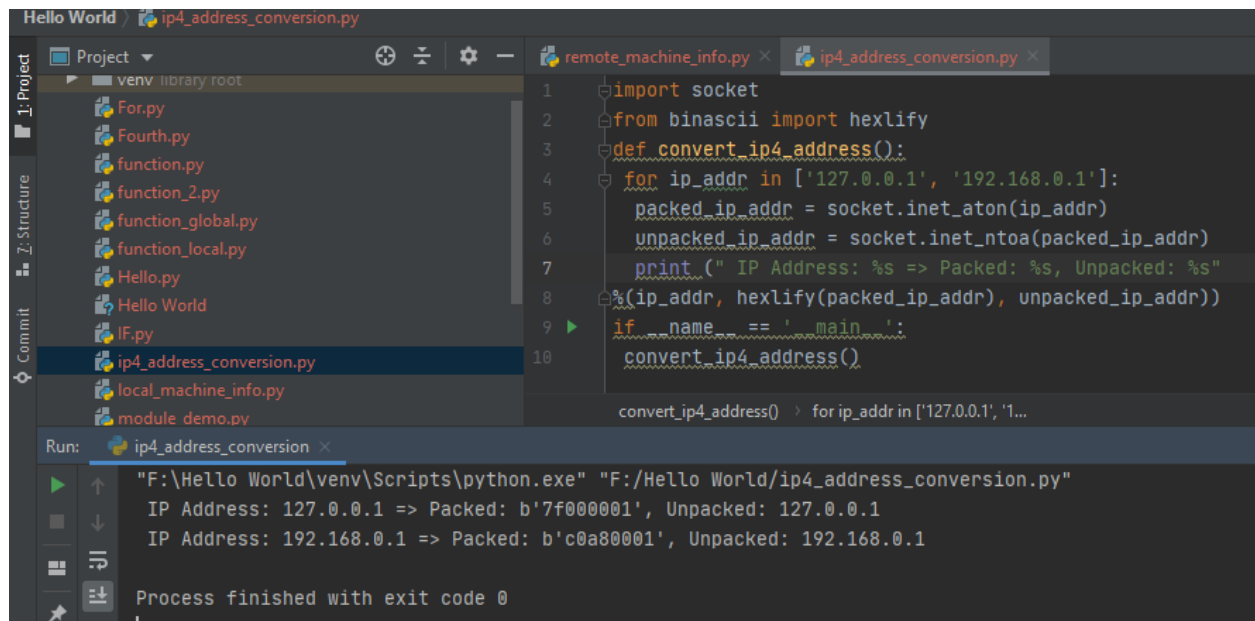
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/remote\_machine\_info.py"

Remote host name: [www.python.org](http://www.python.org)

Error accessing [www.python.org](http://www.python.org): error number and detail [Errno 11001] getaddrinfo failed

Process finished with exit code 0

## Converting an IPv4 address to different formats



```
1 import socket
2 from binascii import hexlify
3 def convert_ip4_address():
4     for ip_addr in ['127.0.0.1', '192.168.0.1']:
5         packed_ip_addr = socket.inet_aton(ip_addr)
6         unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
7         print(" IP Address: %s => Packed: %s, Unpacked: %s"
8               %(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr))
9 if __name__ == '__main__':
10     convert_ip4_address()
11 convert_ip4_address() > for ip_addr in ['127.0.0.1', '1...
```

Run: ip4\_address\_conversion x

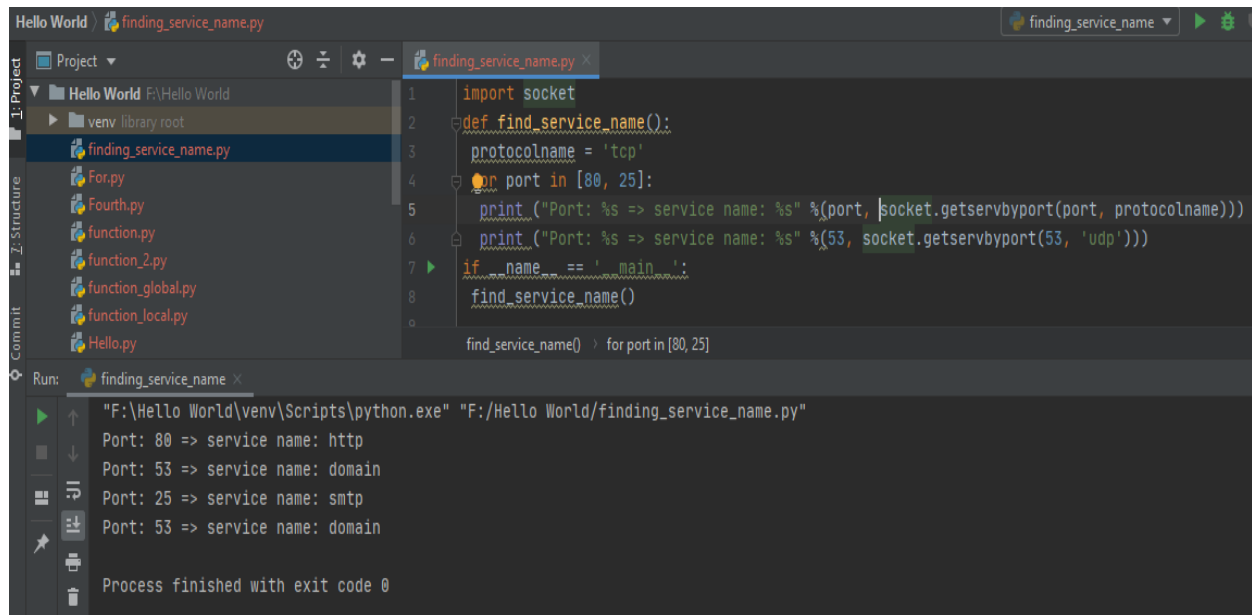
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/ip4\_address\_conversion.py"

IP Address: 127.0.0.1 => Packed: b'7f000001', Unpacked: 127.0.0.1

IP Address: 192.168.0.1 => Packed: b'c0a80001', Unpacked: 192.168.0.1

Process finished with exit code 0

## Finding a service name, given the port and protocol

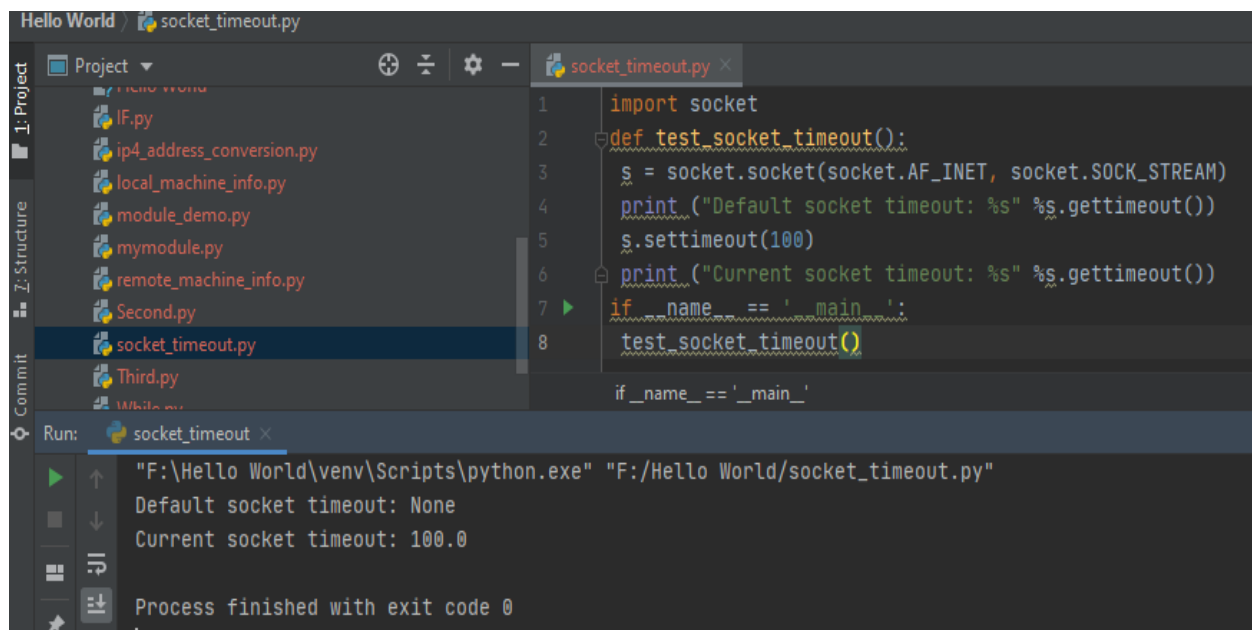


```
1 import socket
2 def find_service_name():
3     protocolname = 'tcp'
4     for port in [80, 25]:
5         print("Port: %s => service name: %s" %(port, socket.getservbyport(port, protocolname)))
6     print("Port: %s => service name: %s" %(53, socket.getservbyport(53, 'udp')))
7 if __name__ == '__main__':
8     find_service_name()
9
10 find_service_name() > for port in [80, 25]
```

Run: finding\_service\_name ×

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/finding_service_name.py"
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
Process finished with exit code 0
```

## Setting and getting the default socket timeout



```
1 import socket
2 def test_socket_timeout():
3     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4     print("Default socket timeout: %s" %s.gettimeout())
5     s.settimeout(100)
6     print("Current socket timeout: %s" %s.gettimeout())
7 if __name__ == '__main__':
8     test_socket_timeout()
9
10 if __name__ == '__main__'
```

Run: socket\_timeout ×

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/socket_timeout.py"
Default socket timeout: None
Current socket timeout: 100.0
Process finished with exit code 0
```

## Writing a simple echo client/server application (Tip: Use port 9900)

### Server Code:



```
echo_server.py x
1 import socket
2 import sys
3 import argparse
4 import codecs
5 from codecs import encode, decode
6 host = 'localhost'
7 data_payload = 4096
8 backlog = 5
9 def echo_server(port):
10     """ A simple echo server """
11     # Create a TCP socket
12     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     # Enable reuse address/port
14     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
15     # Bind the socket to the port
16     server_address = (host, port)
17     print ("Starting up echo server on %s port %s" %server_address)
18     sock.bind(server_address)
19     # Listen to clients, backlog argument specifies the max no. of queued connections
20     sock.listen(backlog)
21
22     while True:
23         print ("Waiting to receive message from client")
24         client, address = sock.accept()
25         data = client.recv(data_payload)
26         if data:
27             print ("Data: %s" %data)
28             client.send(data)
29             print ("sent %s bytes back to %s" % (data, address))
30             # end connection
31             client.close()
32
33 if __name__ == '__main__':
34     parser = argparse.ArgumentParser(description='Socket Server Example')
35     parser.add_argument('--port', action="store", dest="port", type=int,
36                         required=True)
37     given_args = parser.parse_args()
38     port = given_args.port
39     echo_server(port)
40
41 echo_server()
```

Client Code:

```
echo_server.py x echo_client.py x
1  #!/usr/bin/env python
2  import socket
3  import sys
4  import argparse
5  import codecs
6  from codecs import encode, decode
7  host = 'localhost'
8  def echo_client(port):
9      """ A simple echo client """
10     # Create a TCP/IP socket
11     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12     # Connect the socket to the server
13     server_address = (host, port)
14     print("Connecting to %s port %s" % server_address)
15     sock.connect(server_address)
16     # Send data
17     try:
18         #Send data
19         message = "Test message: SDN course examples"
20         print("Sending %s" % message)
21         sock.sendall(message.encode('utf_8'))
22         # Look for the response
23         amount_received = 0
```

```
echo_server.py x echo_client.py x
22     # Look for the response
23     amount_received = 0
24     amount_expected = len(message)
25     while amount_received < amount_expected:
26         data = sock.recv(16)
27         amount_received += len(data)
28         print("Received: %s" % data)
29     except socket.errno as e:
30         print("Socket error: %s" %str(e))
31     except Exception as e:
32         print("Other exception: %s" %str(e))
33     finally:
34         print("Closing connection to the server")
35     sock.close()
36     if __name__ == '__main__':
37         parser = argparse.ArgumentParser(description='Socket Server Example')
38         parser.add_argument('--port', action="store", dest="port", type=int,
39                             required=True)
40         given_args = parser.parse_args()
41         port = given_args.port
42         echo_client(port)
```

## **Discussion:**

Compared to many languages, Python is easy to learn and to use. Its functions can be carried out with simpler commands and less text than most competing languages. And this might explain why it's soaring in popularity, with developers, coding students and tech companies.

It's not an exaggeration to say that Python plays a small part of all of our lives. It's one of those invisible forces with a presence in our mobile devices, web searches and gaming (and beyond). So it was an obvious choice for inclusion in our full stack coding bootcamp. Here's an introduction to the language itself, and some of the everyday but profound, things that Python is used for.