



Lab-Report

Report No: 02

Course code: ICT-3208

Course title: Computer Network Lab

Date of Performance: 20 / 1 / 2021

Date of Submission: 26 / 1 / 2021

Submitted by

Name: Golam Kibria Tuhin

ID: IT-18015

3th year 2nd semester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Lab Report No. : 02

Lab Report Name: Programming with python.

Theory:

Python functions:

Functions are reusable pieces of programs. They allow you to give a name to a block of statements, allowing you to run that block using the specified name anywhere in the program and any number of times. This is known as calling the function.

Local Variables:

Variables declared inside a function definition are not related in any way to other variables with the same names used outside the function (variable names are local to the function). This is called the scope of the variable. All variables have the scope of the block they are declared in starting from the point of definition of the name.

The global statement:

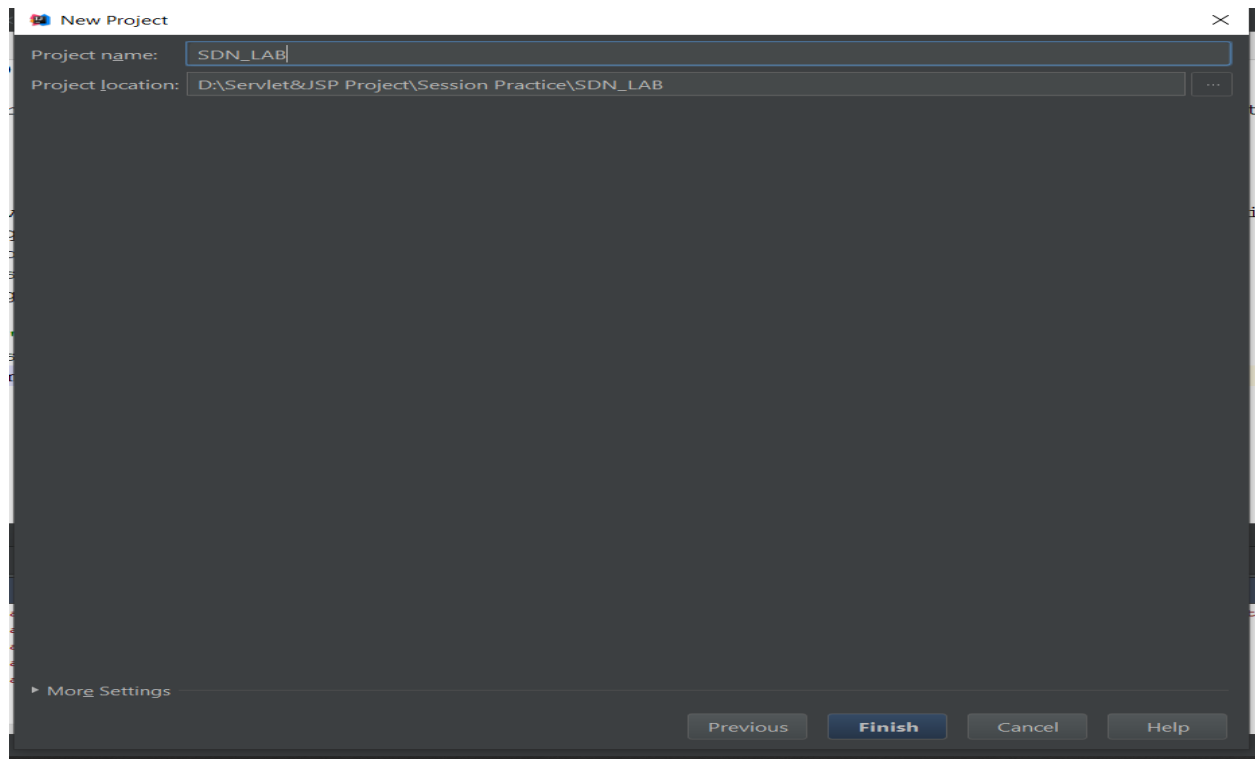
Variables defined at the top level of the program are intended global. Global variables are intended to be used in any functions or classes). Global statement allows defining global variables inside functions as well.

Modules:

Modules allow reusing a number of functions in other programs.

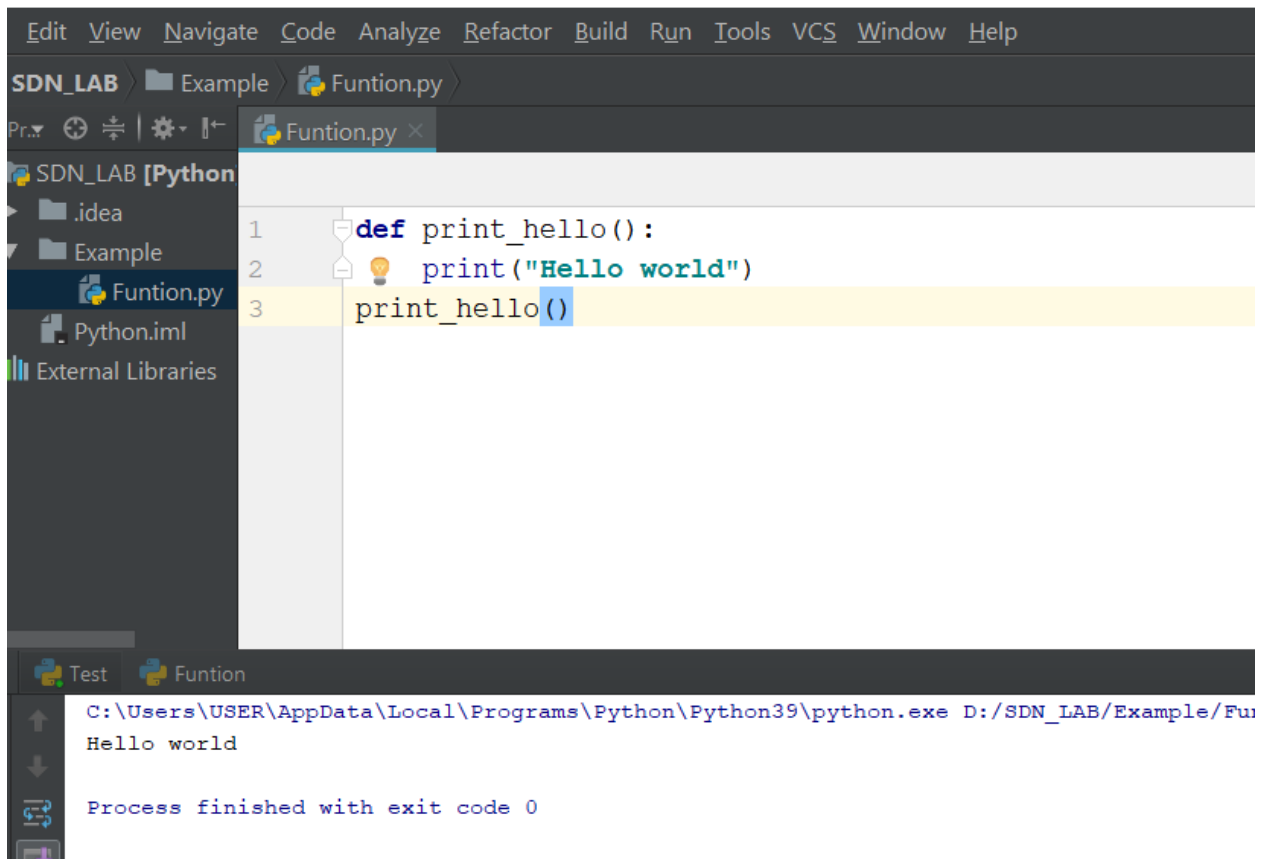
Exercises:

Exercise 4.1.1: Create a Python project with SDN-LAB.



Exercise 4.1.2: Python function (save as function.py)

SDN_LAB - [D:\SDN_LAB] - [Python] - ...\Example\Funtion.py - IntelliJ IDEA 2017.1.2



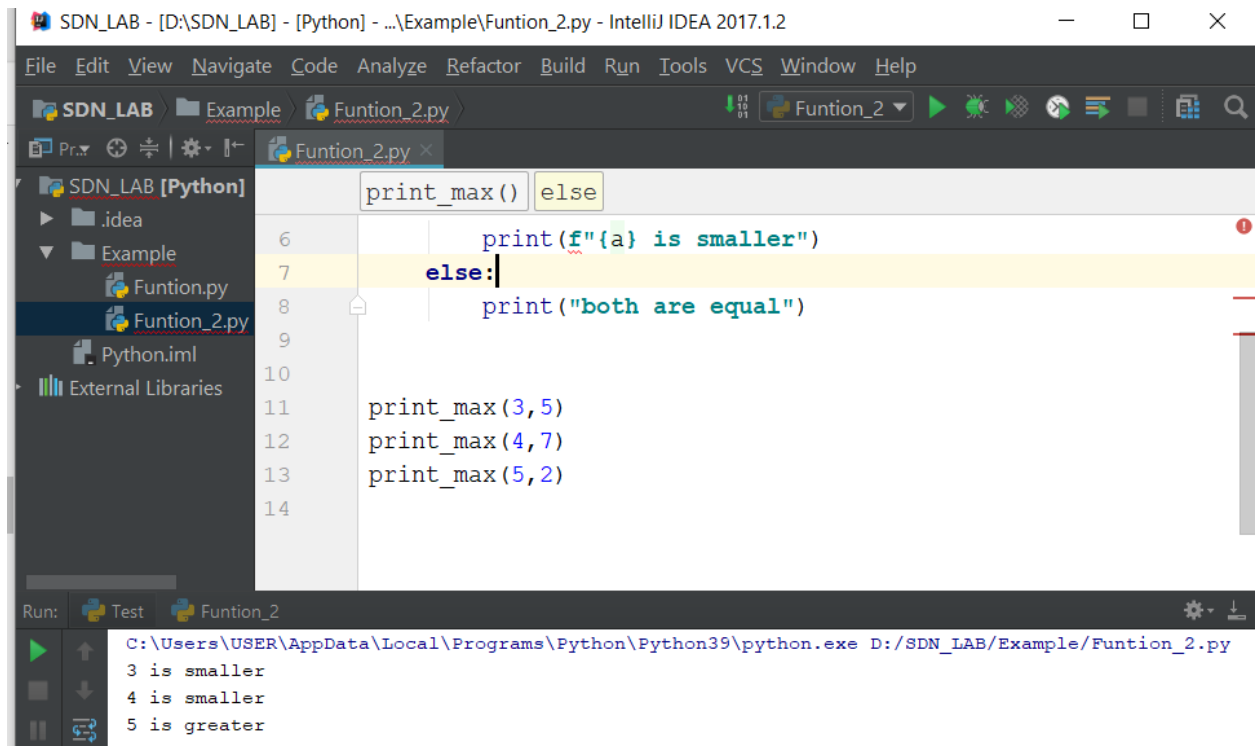
```
1 def print_hello():
2     print("Hello world")
3     print_hello()
```

Test Funtion

C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Fun
Hello world

Process finished with exit code 0

Exercise 4.1.3: Python function (save as function_2.py)



```
1 def print_max(a, b):
2     if a < b:
3         print(f"{a} is smaller")
4     elif a > b:
5         print(f"{b} is smaller")
6     else:
7         print("both are equal")
8
9
10
11 print_max(3, 5)
12 print_max(4, 7)
13 print_max(5, 2)
14
```

Run: Test Funtion_2

C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Funtion_2.py

3 is smaller
4 is smaller
5 is greater

Exercise 4.1.4: Local variable

The screenshot shows the IntelliJ IDEA 2017.1.2 interface. The main editor displays a Python file named `Local_Variable.py` with the following code:

```
1 def func(x):  
2     print("The value of x is {x}")  
3     x = 2  
4     print("change the value of x to local {x}")  
5  
6 func(200);  
7  
8  
9  
10
```

The file explorer on the left shows the project structure: `SDN_LAB [Python]` with subfolders `.idea` and `Example`. The `Example` folder contains `Function.py`, `Function_2.py`, and `Local_Variable.py`. The `Local_Variable.py` file is selected.

The Run tool window at the bottom shows the execution output for the `Local_Variable` configuration:

```
C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Local_Variable  
The value of x is 200  
change the value of x to local 2  
Process finished with exit code 0
```

Exercise 4.1.5: Global variable :

SDN_LAB - [D:\SDN_LAB] - [Python] - ...\Example\Global_Variable.py - IntelliJ IDEA 2017.1.2

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

SDN_LAB > Example > Global_Variable.py

Global_Variable.py

```
1 global_variable = 100
2 def local_variable(x):
3     x=5
4     print(f"The value of X as local is {x}")
5
6 local_variable(global_variable)
7 print(f"The value of x as global variable is {global_variable}")
8
9
```

Run: Test Global_Variable

C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Global_Variabl
The value of X as local is 5
The value of x as global variable is 100
Process finished with exit code 0

Exercise 4.1.6: python modules

Global_Variable.py × Module_Demo.py × Module.py ×

```
1 def welcome():
2     print("hello, This is our module")
3     __version__='1.0'
4
5
```

SDN_LAB - [D:\SDN_LAB] - [Python] - ...\Example\Module_Demo.py - IntelliJ IDEA 2017.1.2

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
```

SDN_LAB Example Module_Demo.py

Global_Variable.py x Module_Demo.py x Module.py x

SDN_LAB [Python] D:\SDN_LAB

```
1
2 import Module
3 Module.welcome()
4 print(Module.__version__)
5
6
```

Run: Test Module_Demo

```
C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Module_Demo.py
hello, This is our module
1.0
```

Exercise 4.2.1: Printing your machine's name and IPv4 address

SDN_LAB - [D:\SDN_LAB] - [Python] - ...\Example\Local_Machine.py - IntelliJ IDEA 2017.1.2

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
```

SDN_LAB Example Local_Machine.py

Local_Machine.py x

SDN_LAB [Python] D:\SDN_LAB

```
1 import socket
2 Host name is {socket.gethostname()}"
3 'Ip address is {socket.gethostbyname(socket.gethostname())}"
```

Run: Test Local_Machine

```
C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Local_Machine.py
Host name is DESKTOP-3QPCIE5
Ip address is 192.168.0.106
Process finished with exit code 0
```

Exercise 4.2.2: Retrieving a remote machine's IP address

The screenshot shows an IDE window with the file `Remote_Machine.py` open. The code is as follows:

```
try
1  import socket
2  try:
3      print("Remote host name is: www.python.org")
4      print(f"Ip address if: {socket.gethostbyname('www.python.org')}")
5  except socket.error as err_msg:
6      print(f"Error accessing www.python.org and detail {err_msg}")
7
8
9
10
```

The Run window at the bottom shows the execution output:

```
C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/Remote_Machine.py
Remote host name is: www.python.org
Ip address if: 151.101.156.223
```

Exercise 4.2.3: Converting an IPv4 address to different formats.

The screenshot shows an IDE window with the file `IpConvert.py` open. The code is as follows:

```
1  import socket
2  from binascii import hexlify
3
4  def convert_ipv4_address():
5      for ip_addr in ['127.0.0.1', '192.168.0.1']:
6          packed_ip_addr = socket.inet_aton(ip_addr)
7          unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
8          print(f"IP address: {ip_addr} => packed: {hexlify(packed_ip_addr)} Unpacked: {unpacked_ip_addr}")
9
10 convert_ipv4_address()
```

The Run window at the bottom shows the execution output:

```
C:\Users\USER\AppData\Local\Programs\Python\Python39\python.exe D:/SDN_LAB/Example/IpConvert.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
```

Exercise 4.2.4: Finding a service name, given the port and protocol

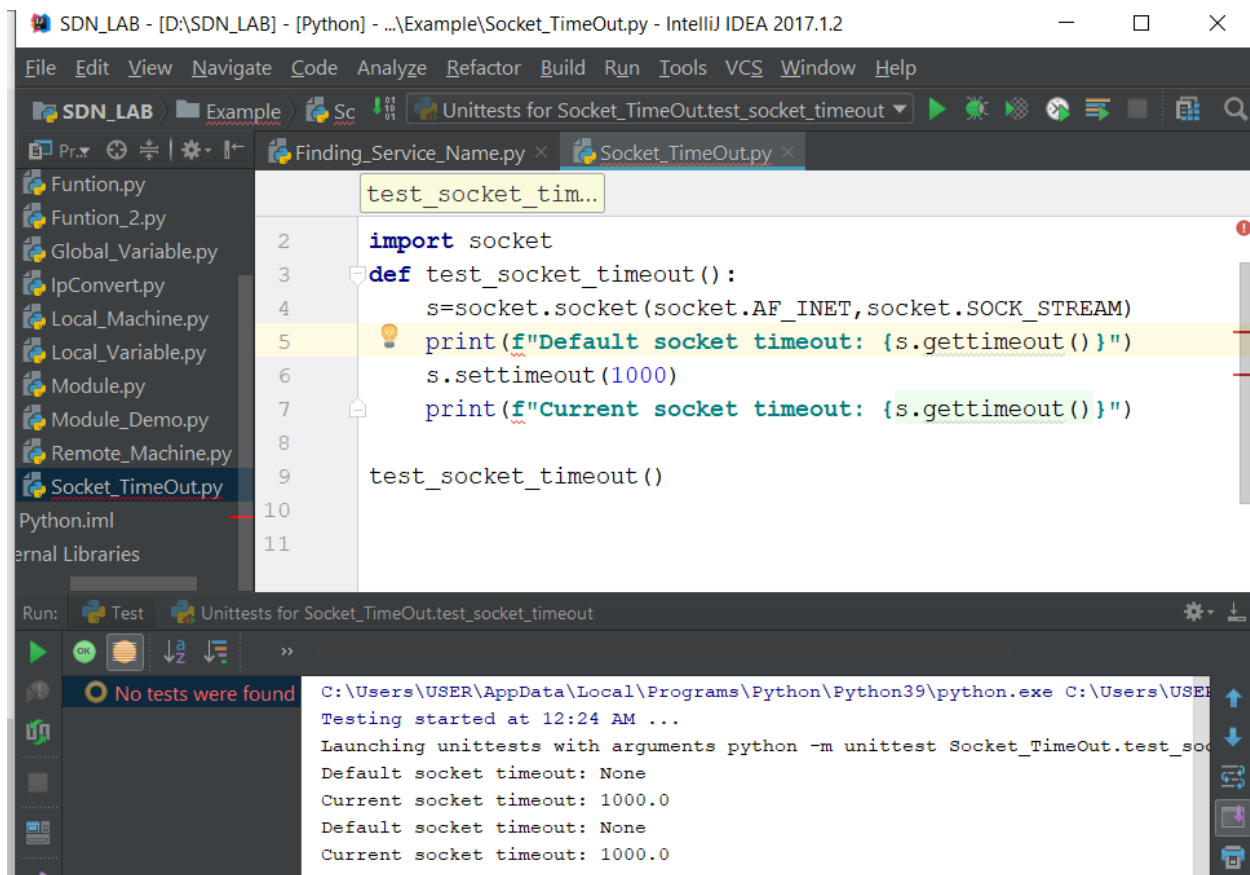
The screenshot shows a Python IDE with a file named `Finding_Service_Name.py`. The code defines a function `find_service_name()` that iterates over a list of ports `[80, 25]`. For each port, it prints the port number and the corresponding service name using `socket.getservbyport()`. The function is then called. The output in the console shows the service names for ports 80, 53, and 25.

```
1 import socket
2 def find_service_name():
3     for port in [80, 25]:
4         print(f"Port: {port} => service name: {socket.getservbyport(port)}")
5     print(f"Port: {53} => service name: {socket.getservbyport(53)}")
6
7 find_service_name()
8
9
10
```

Output:

```
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

Exercise 4.2.5: Setting and getting the default socket timeout.



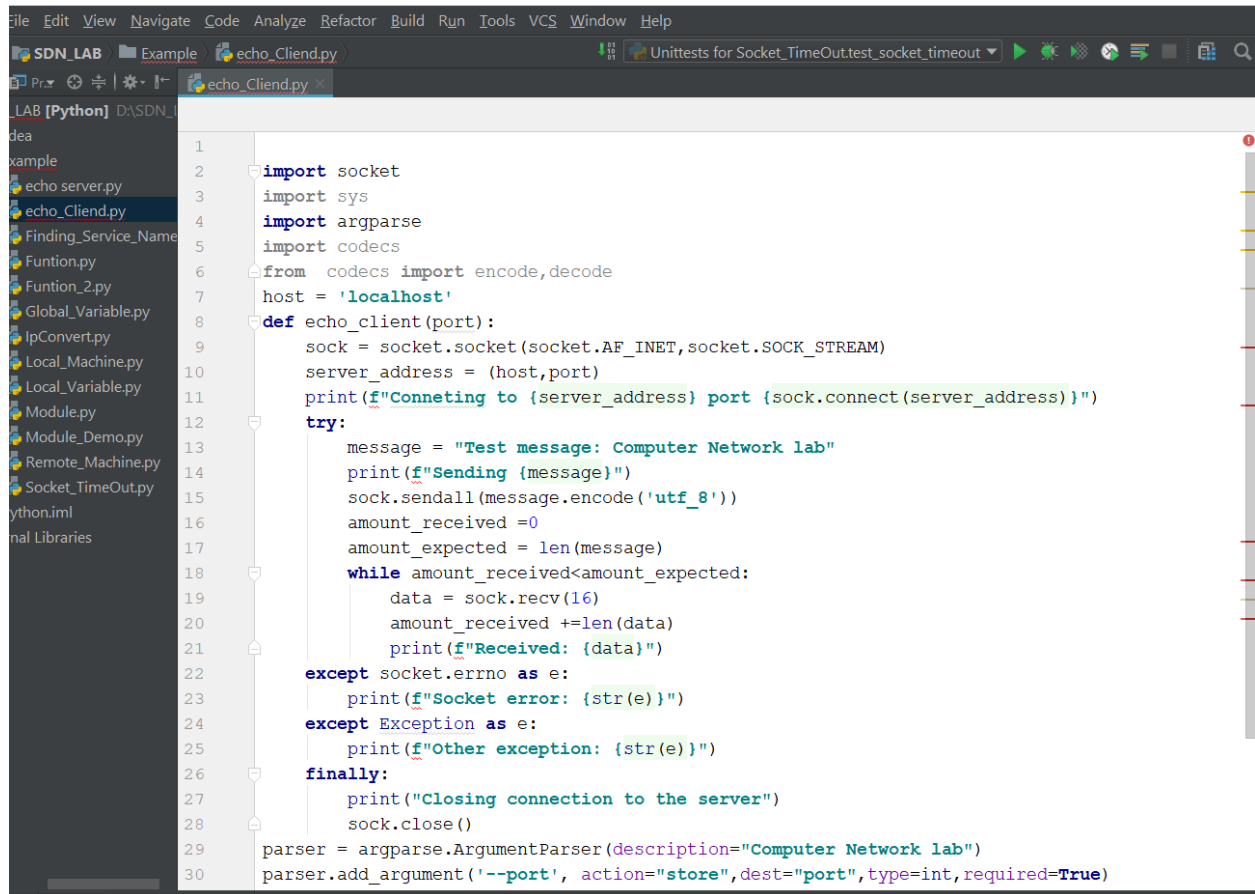
Exercise 4.2.6: Writing a simple echo client/server application (Tip: Use port 9900)

Server code:

```
SDN_LAB - [D:\SDN_LAB] - [Python] - ...example\echo server.py - IntelliJ IDEA 2017.1.2
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SDN_LAB Example echo server.py Unittests for Socket_TimeOut.test_socket_timeout
echo server.py
SDN_LAB [Python] D:\SDN_LAB
idea
Example
echo server.py
Finding_Service_Name
Funtion.py
Funtion_2.py
Global_Variable.py
IpConvert.py
Local_Machine.py
Local_Variable.py
Module.py
Module_Demo.py
Remote_Machine.py
Socket_TimeOut.py
Python.iml
ernal Libraries

4 import argparse
5 import codecs
6 from codecs import encode, decode
7 host='localhost'
8 data_payload=2048
9 backlog =5
10 def echo_server(port):
11     sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
12     sock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
13     server_address = (host,port)
14     print(f"Starting up echo server on {server_address} port {server_address}")
15     sock.bind(server_address)
16     sock.listen(backlog)
17     while True:
18         print("Willing to receive message from client")
19         client,address = sock.accept()
20         data = client.recv(data_payload)
21         if data:
22             print(f"Data: {data}")
23             client.send(data)
24             print(f"sent {data} bytes back to {address}")
25             client.close()
26
27 parser = argparse.ArgumentParser(description="Computer Network lab")
28 parser.add_argument('--port', action="store", dest="port", type=int, required=True)
29 given_args = parser.parse_args()
30 port = given_args.port
31 echo_server(port)
32
33
```

Client code:

A screenshot of a code editor window. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. The title bar shows 'SDN_LAB' and 'Example'. The file explorer on the left lists files like 'echo_server.py', 'echo_Client.py', 'Finding_Service_Name.py', 'Funtion.py', 'Funtion_2.py', 'Global_Variable.py', 'IpConvert.py', 'Local_Machine.py', 'Local_Variable.py', 'Module.py', 'Module_Demo.py', 'Remote_Machine.py', and 'Socket_TimeOut.py'. The main editor area shows the code for 'echo_Client.py'. The code imports 'socket', 'sys', 'argparse', and 'codecs'. It defines a function 'echo_client(port)' that connects to 'localhost' on a specified port, sends a test message, receives a response, and prints it. It also includes a 'finally' block to close the connection and an 'argparse' section to handle command-line arguments for the port.

```
1 import socket
2 import sys
3 import argparse
4 import codecs
5
6 from codecs import encode, decode
7 host = 'localhost'
8
9 def echo_client(port):
10     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11     server_address = (host, port)
12     print(f"Connecting to {server_address} port {sock.connect(server_address)}")
13     try:
14         message = "Test message: Computer Network lab"
15         print(f"Sending {message}")
16         sock.sendall(message.encode('utf_8'))
17         amount_received = 0
18         amount_expected = len(message)
19         while amount_received < amount_expected:
20             data = sock.recv(16)
21             amount_received += len(data)
22             print(f"Received: {data}")
23     except socket.error as e:
24         print(f"Socket error: {str(e)}")
25     except Exception as e:
26         print(f"Other exception: {str(e)}")
27     finally:
28         print("Closing connection to the server")
29         sock.close()
30
31 parser = argparse.ArgumentParser(description="Computer Network lab")
32 parser.add_argument('--port', action="store", dest="port", type=int, required=True)
```

What you need to do for running the program?

Answer:

To understand python socket programming, we need to know about three interesting topics – **Socket Server**, **Socket Client** and **Socket**.

So, what is a server? Well, a server is a software that waits for client requests and serves or processes them accordingly.

On the other hand, a client is requester of this service. A client program request for some resources to the server and server responds to that request.

Which program you need to run first client of server?

Answer:

To use python socket connection, we need to import socket module. Then, sequentially we need to perform some task to establish connection between server and client. We can obtain host address by using `socket.gethostname()` function.

Explain how the program works?

Answer:

We have said earlier that a socket client requests for some resources to the socket server and the server responds to that request.

So we will design both server and client model so that each can communicate with them. The steps can be considered like this.

- i. Python socket server program executes at first and wait for any request
- ii. Python socket client program will initiate the conversation at first. iii. Then server program will response accordingly to client requests. iv. Client program will terminate if user enters "bye" message. Server program will also terminate when client program terminates, this is optional and we can keep server program running indefinitely or terminate with some specific command in client request.

What you need to do for communicating with another server in the classroom?

Answer:

A network socket is an endpoint of a two-way communication link between two programs or processes - client and server in our case - which are running on the network. This can be on the same computer as well as on different systems which are connected via the network.

Both parties communicate with each other by writing to or reading from the network socket. The technical equivalent in reality is a telephone communication between two participants. The network socket represents the corresponding number of the telephone line, or a contract in case of cell phones

• Question 5.1: Explain in your own words which are the difference between functions and modules?

Answer:

Python Function:

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Python gives you many built-in functions like `print()`, etc. but you can also create

your own functions. These functions are called user-defined functions **Python Module:**

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code

or

In programming, function refers to a segment that groups code to perform a specific task.

A module is a software component or part of a program that contains one or more routines.

That means, functions are groups of code, and modules are groups of classes and functions

- Question 5.2: Explain in your own words when to use local and global variables?

Answer:

Local Variable: Local variable is defined as a type of variable declared within programming block or subroutines. It can only be used inside the subroutine or code block in which it is declared. The local variable exists until the block of the function is under execution. After that, it will be destroyed automatically.

Here, 'a' and 'b' are local variables

Global Variable: A global variable in the program is a variable defined outside the subroutine or function. It has a global scope means it holds its value throughout the lifetime of the program. Hence, it can be accessed throughout the program by any function defined within the program, unless it is shadowed.

Example:

```
int a =4;
int b=5;
public int add(){
    return a+b;
}
```

Here, 'a' and 'b' are global variables.

- Question 5.3: Which is the role of sockets in computing networking? Are the sockets defined random or there is a rule?

Answer:

A socket uniquely identifies the endpoint of a communication link between two application ports.

A port represents an application process on a TCP/IP host, but the port number itself does not indicate the protocol being used: TCP, UDP, or IP. The application process might use the same port number for TCP or UDP protocols. To uniquely identify the destination of an IP packet arriving over the network, you have to extend the port principle with information about the protocol used and the IP address of the network interface; this information is called a socket. A socket has three parts: protocol, local-address, local-port

- Question 5.4: Why is relevant to have the IPv4 address of remote server?

Explain what is Domain Name System (DNS)?

Answer:

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources. Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6)

Conclusion:

Python plays an essential role in network programming. The standard library of Python has full support for network protocols, encoding, and decoding of data and other networking concepts, and it is simpler to write network programs in Python than that of C++. There are two levels of network service access in Python.

These are: Low-Level Access High-Level Access

In the first case, programmers can use and access the basic socket support for the operating system using Python's libraries, and programmers can implement both connection-less and connection-oriented protocols for programming.

Application-level network protocols can also be accessed using high-level access provided by Python libraries. These protocols are HTTP, FTP, etc.

A socket is the end-point in a flow of communication between two programs or communication channels operating over a network. They are created using a set of programming requests called socket API (Application Programming Interface). Python's socket library offers classes for handling common transports as a generic interface.

Sockets use protocols for determining the connection type for port-to-port communication between client and server machines. The protocols are used for: Domain Name Servers (DNS), IP addressing, E-mail, FTP (File Transfer Protocol) etc.