

Name: Md.Abdullah

ID : IT-17015

Lab report no :06

lab report name : Programing with Python

objectives:

- Understand how python function works
- Understand the use of global and local variables
- Understand how python modules works
- Learning the basis of networking programing 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.

Methodology:

Defining functions:

Functions are defined using the def keyword. After this keyword comes an identifier name for the function, followed by a pair of parentheses which may enclose some names of variables, and by the final colon that ends the line.

```
def XX_YY(variable1, variable2):  
    # block belonging to the function  
    # End of function
```

Defining local and global variables:

Local and global variables can be defined using:

```
x = 50 #Local  
global x
```

Defining modules:

There are various methods of writing modules, but the simplest way is to create a file with a .py extension that contains functions and variables.

```
def xx_yy():  
    aa
```

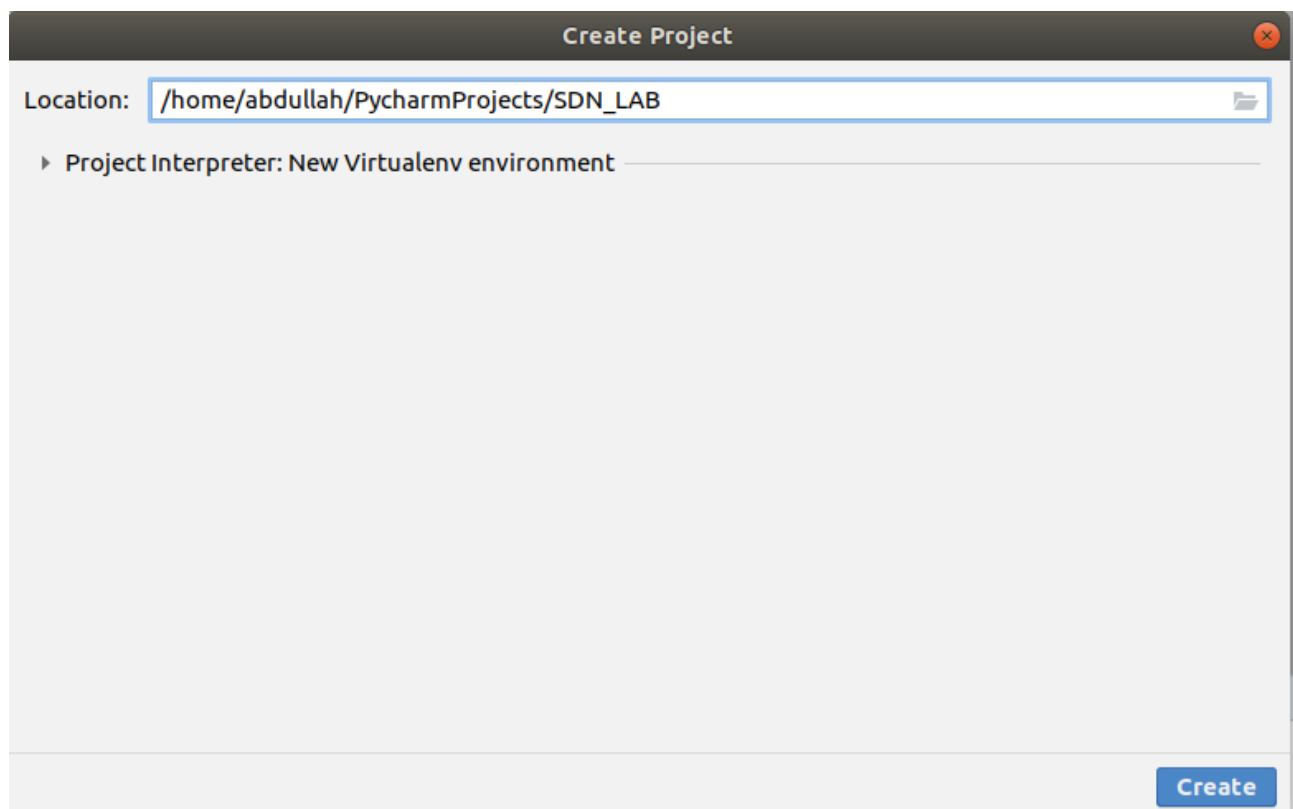
Using modules:

A module can be imported by another program to make use of its functionality. This is how we can use the Python standard library as well.

```
import xx_yy
```

Exercise 4.1.1: Create a python project using with SDN_LAB**Ans:**

1. Click on file menu
2. Create new project
3. Give the project name SDN_LAB



Exercise 4.1.2: Python function (save as function.py).Create python scrip using the syntax provided below.


```
def say_hello():  
    # block belonging to the function  
    print('hello world')  
    # End of function  
  
if __name__ == '__main__':  
    say_hello() # call the function
```

Which is the output of this function? Does the function need any parameter?

Ans: No this function does not need any parameter.

```
def say_hello():  
    print('Hello')  
if __name__ == '__main__':  
    say_hello()
```

Output:



Run: function x

/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/function.py"

Hello

Process finished with exit code 0

Exercise 4.1.3: Python function (save as function_2.py).Create python scrip using the syntax provided below.

```
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)
```

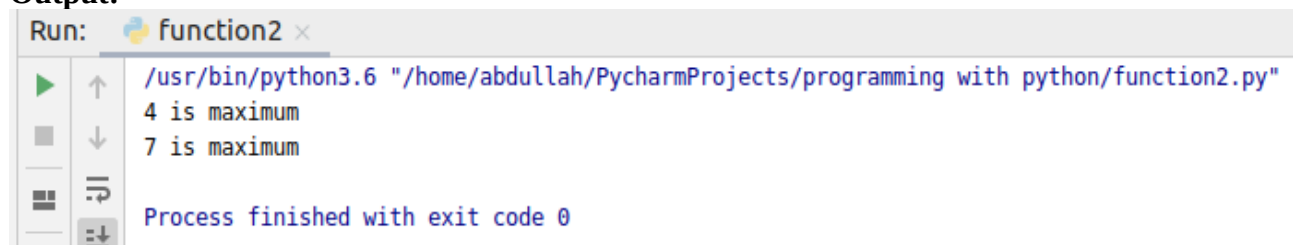
Which is the output of this function? Does the function need any parameter?

Ans:

No this function does not require any parameter.

```
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)
    x=5
    y=7
    print_max(x,y)
```

Output:



```
Run: function2 x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/function2.py"
4 is maximum
7 is maximum
Process finished with exit code 0
```

Exercise 4.1.4: Local variable (save as function_local.py). Create python scrip using the syntax provided below.

```
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)
```

Which is the final value of variable x? Why variable x does not change to 2?

Ans:

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)
```

Output:

Variable x does not change to 2 because x is a local variable with in a function.

```
Run: function_local x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/function_local.py"
x is 50
Changed local x to 2
x is still 50
Process finished with exit code 0
```

Exercise 4.1.5: Global variable (save as function_global.py).Create python scrip using the syntax provided below.

```
x = 50

def func():
    global x
    print('x is', x)
    x = 2
    print('Changed global x to', x)

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

Which is the final value of variable x? Why variable x change this time?

Ans:

x=50

```
def func():
    global x
    print('x is', x)
    x = 2
    print('Changed local x to', x)
if __name__ == '__main__':
    func()
    print('x is still', x)
```

Output: Becuase x is a global variable.

```
Run: function_global x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/function_global.py"
x is 50
Changed local x to 2
x is still 2
Process finished with exit code 0
```

Exercise 4.1.6: Python modules

Create python scrip using the syntax provided below (save as mymodule.py).

```
def say_hi():  
    print('Hi, this is mymodule speaking.')
```



```
__version__ = '0.1'
```

Create python scrip using the syntax provided below (save as module_demo.py).

```
import mymodule
```



```
if __name__ == '__main__':  
    mymodule.say_hi()  
    print('Version', mymodule.__version__)
```

Run the script, which is the role of import?

Ans:

mymodule.py

```
def say_hi():  
    print('Hi this is my module speaking')
```



```
__version__='0.1'
```

module_demo.py

```
import mymodule
```



```
if __name__=='main':  
    mymodule.say_hi()  
    print('Version :',mymodule.__version__)
```

Role of import:

An import means importing all the things that the mymodule.py has such as function,variable etc.

Create python scrip using the syntax provided below (save as module_demo2.py).

```
from mymodule import say_hi, __version__
```



```
if __name__ == '__main__':  
    say_hi()  
    print('Version', __version__)
```

Run the script, which is the role of from, import?

Ans:

module_demo2.py

```
from mymodule import say_hi,__version__
```



```
if __name__=='main':  
    say_hi()  
    print('Version ',__version__)
```

Role of from,import:

from keyword is used when we want a specific methods or functions.
Import keyword is used to import all the functions ,variables in it.

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

Create python scrip using the syntax provided below (save as local_machine_info.py):

```
import socket

def print_machine_info():
    host_name = socket.gethostname()
    ip_address = socket.gethostbyname(host_name)
    print ("    Host name: %s" % host_name)
    print ("    IP address: %s" % ip_address)
if __name__ == '__main__':
    print_machine_info()
```

Run the script, which module the program uses? Provide two additional functions of socket.

Ans:

The program used the socket module.

import socket

def print_machine_info():

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

host_name=socket.gethostname()

ip_address=socket.gethostbyname(host_name)

s.bind((host_name,1024))

print("Host name: % s" % host_name)

print("IP address: % s " % ip_address)

if __name__=='__main__':

print_machine_info()

Output:

Exercise 4.2.2: Retrieving a remote machine's IP address

Create python scrip using the syntax provided below (save as remote_machine_info.py):

```
import socket

def get_remote_machine_info():
    remote_host = 'www.python.org'
    try:
        print ("    Remote host name: %s" % remote_host)
        print ("    IP address: %s" %socket.gethostbyname(remote_host))
    except socket.error as err_msg:
        print ("Error accesing %s: error number and detail %s"
%(remote_host, err_msg))

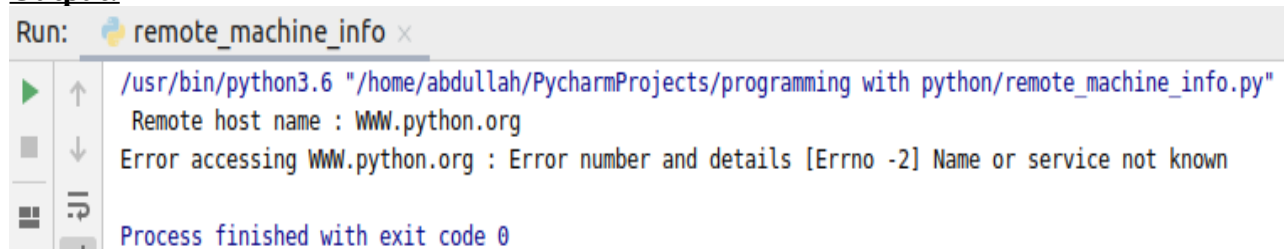
if __name__ == '__main__':
    get_remote_machine_info()
```

Run the script, which is the output? Modify the code for getting the RMIT website info.

Ans:

```
import socket
def remote_machine_info():
    remote_host='WWW.python.org'
    try:
        print(' Remote host name :',remote_host)
        print('IP address is :',socket.gethostbyname(remote_host))
    except socket.error as err_msg:
        print('Error accessing %s : Error number and details %s'%(remote_host,err_msg))
if __name__=='__main__':
    remote_machine_info()
```

Output:



```
Run: remote_machine_info x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/remote_machine_info.py"
Remote host name : WWW.python.org
Error accessing WWW.python.org : Error number and details [Errno -2] Name or service not known
Process finished with exit code 0
```


Exercise 4.2.3: Converting an IPv4 address to different formats. Create python scrip using the syntax below (save as `ip4_address_conversion.py`):

```
import socket
from binascii import hexlify

def convert_ip4_address():
    for ip_addr in ['127.0.0.1', '192.168.0.1']:
        packed_ip_addr = socket.inet_aton(ip_addr)
        unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
        print ("    IP Address: %s => Packed: %s, Unpacked: %s"
              %(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr))

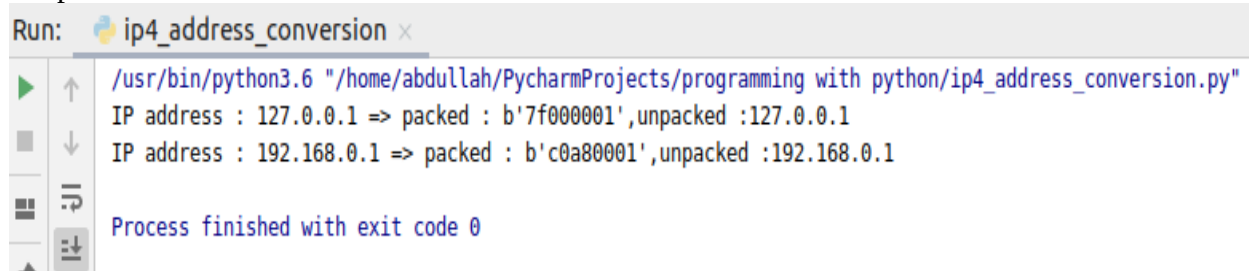
if __name__ == '__main__':
    convert_ip4_address()
```

Run the script, which is the output? How binascii works?

Ans:

```
import socket
from binascii import hexlify
def convert_ip():
    for ip_addr in ['127.0.0.1', '192.168.0.1']:
        packed_ip_addr=socket.inet_aton(ip_addr)
        unpacked_ip_addr=socket.inet_ntoa(packed_ip_addr)
        print('IP address : %s => packed : %s,unpacked :%s'%
              (ip_addr,hexlify(packed_ip_addr),unpacked_ip_addr))
if __name__=='__main__':
    convert_ip()
```

Output:



```
Run: ip4_address_conversion x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/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
```

The [binascii](#) module contains a number of methods to convert between binary and various ASCII-encoded binary representations. Normally, you will not use these functions directly but use wrapper modules like [uu](#), [base64](#), or [binhex](#) instead. The [binascii](#) module contains low-level functions written in C for greater speed that are used by the higher-level modules.

Exercise 4.2.4: Finding a service name, given the port and protocol. Create python scrip using the syntax below (save as finding_service_name.py):

```
import socket
def find_service_name():
    protocolname = 'tcp'
    for port in [80, 25]:
        print ("Port: %s => service name: %s" %(port,
socket.getservbyport(port, protocolname)))
    print ("Port: %s => service name: %s" %(53,
socket.getservbyport(53, 'udp')))

if __name__ == '__main__':
    find_service_name()
```

Run the script, which is the output? Modify the code for getting complete the table:

Port	Protocol Name
21	
22	
110	

Ans:

```
import socket
def find_service_name():
    protocolname = 'tcp'
    for port in [80, 25]:
        print ("Port: %s => service name: %s" %(port,socket.getservbyport(port,
protocolname)))

    print ("Port: %s => service name: %s" %(53,socket.getservbyport(53, 'udp'))))

if __name__ == '__main__':
    find_service_name()
```

Output:

```
Run: finding_service_name x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/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
```

For port number 21,22,100:

```
import socket
def find_service_name():
    protocolname = 'tcp'
    for port in [21,22,100]:
```

```

        print ("Port: %s => service name: %s" %(port,socket.getservbyport(port,
protocolname)))
        print ("Port: %s => service name: %s" %(53,socket.getservbyport(53, 'udp'))))
if __name__ == '__main__':
    find_service_name()

```

Output:

```

Run: find_service_name2 x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/find_service_name2.py"
Port: 21 => service name: ftp
Port: 53 => service name: domain
Port: 22 => service name: ssh
Port: 53 => service name: domain
Traceback (most recent call last):
  File "/home/abdullah/PycharmProjects/programming with python/find_service_name2.py", line 9, in <module>
    find_service_name()
  File "/home/abdullah/PycharmProjects/programming with python/find_service_name2.py", line 5, in find_service_name
    print ("Port: %s => service name: %s" %(port,socket.getservbyport(port, protocolname)))
OSError: port/proto not found

Process finished with exit code 1

```

Exercise 4.2.5: Setting and getting the default socket timeout.Create python scrip using the syntax below (save as socket_timeout.py):

```

import socket

def test_socket_timeout():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print ("Default socket timeout: %s" %s.gettimeout())
    s.settimeout(100)
    print ("Current socket timeout: %s" %s.gettimeout())

if __name__ == '__main__':
    test_socket_timeout()

```

Run the script, which is the role of socket timeout in real applications?

Ans:

```

import socket
def test_socket_timeout():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print ("Default socket timeout: %s" %s.gettimeout())
    s.settimeout(100)
    print ("Current socket timeout: %s" %s.gettimeout())
if __name__ == '__main__':
    test_socket_timeout()

```

Output:

```

Run: socket_timeout x
/usr/bin/python3.6 "/home/abdullah/PycharmProjects/programming with python/socket_timeout.py"
Default socket timeout: None
Current socket timeout: 100.0

Process finished with exit code 0

```

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

Ans:

Server:

```
import socket
import sys
import argparse
import codecs
from codecs import encode, decode
host='localhost'
data_payload=4096
backlog = 5
def echo_server(port):
    sock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    server_address=(host,port)
    print('Starting up echo server on %s port %s'%server_address)
    sock.listen(backlog)
    while True:
        print('Waiting to receive message from client')
        client,address=sock.accept()
        data=client.recv(data_payload)
        if data:
            print('Data : %s'%data)
            client.send(data)
            print('Sent %s bytes back to %s'%(data,address))
            client.close()
if __name__=='__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,required=True)
    given_args = parser.parse_args()
    echo_server(1024)
```

Client:

```
import socket
import sys
import argparse
import codecs
from codecs import encode, decode
host='localhost'
def echo_client(port):
    sock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server_address=(host,port)
    print('Connecting to %s port %s'%server_address)
    sock.connect(server_address)
    try:
        message = "Test message: SDN course examples"
        print("Sending %s" % message)
```

```

sock.sendall(message.encode('utf_8'))
amount_received = 0
amount_expected = len(message)
while amount_received < amount_expected:
    data = sock.recv(16)
    amount_received += len(data)
    print("Received: %s" % data)
except socket.errno as e:
    print("Socket error: %s" % str(e))
except Exception as e:
    print("Other exception: %s" % str(e))
finally:
    print("Closing connection to the server")
    sock.close()
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(1024)

```

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

Ans:

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

A module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

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

Ans:

Local variable is defined within a function and it can be only accessed inside the function

Global variable is defined outside a function and it can be called anywhere of that script

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

Ans:

Role of sockets in computing networking:

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.

Question 5.4: Why is relevant to have the IPv4 address of remote server? Explain what is Domain Name System (DNS)?

Ans:

DNS:

DNS means domain name system. Domain Name System helps to resolve the host name to an address. It uses a hierarchical naming scheme and distributed database of IP addresses and associated names.

Discussion:

After completing the exercise I know functions, modules, local and global variables in python. I also know about socket programming in python. Also know about server and clients.