Mawlana Bhashani Science and Technology University

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

3rd year 2nd semester

Session: 2017-2018

Dept. of ICT

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Theory:

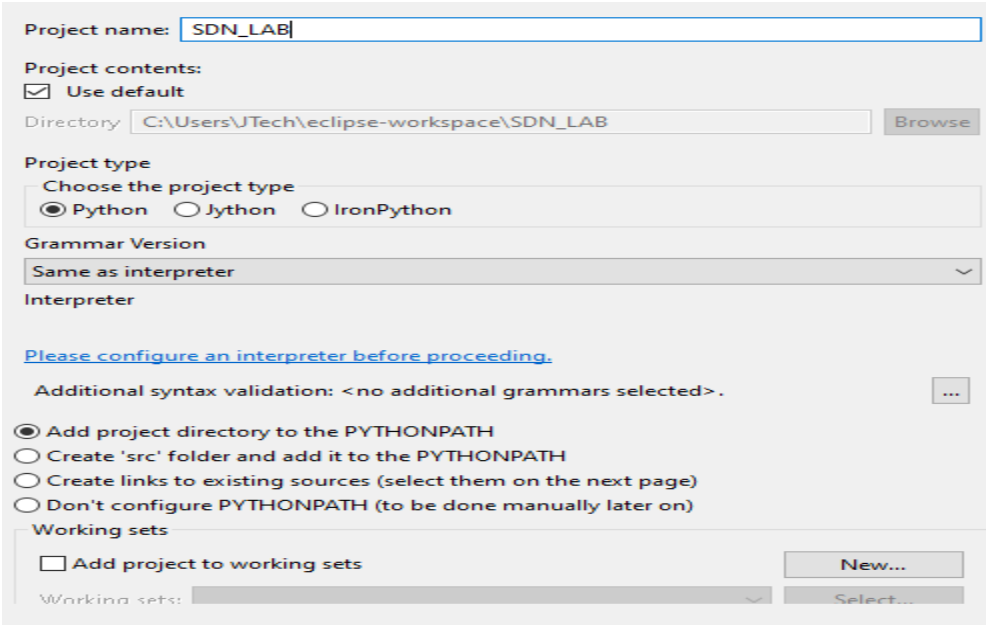
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to 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.

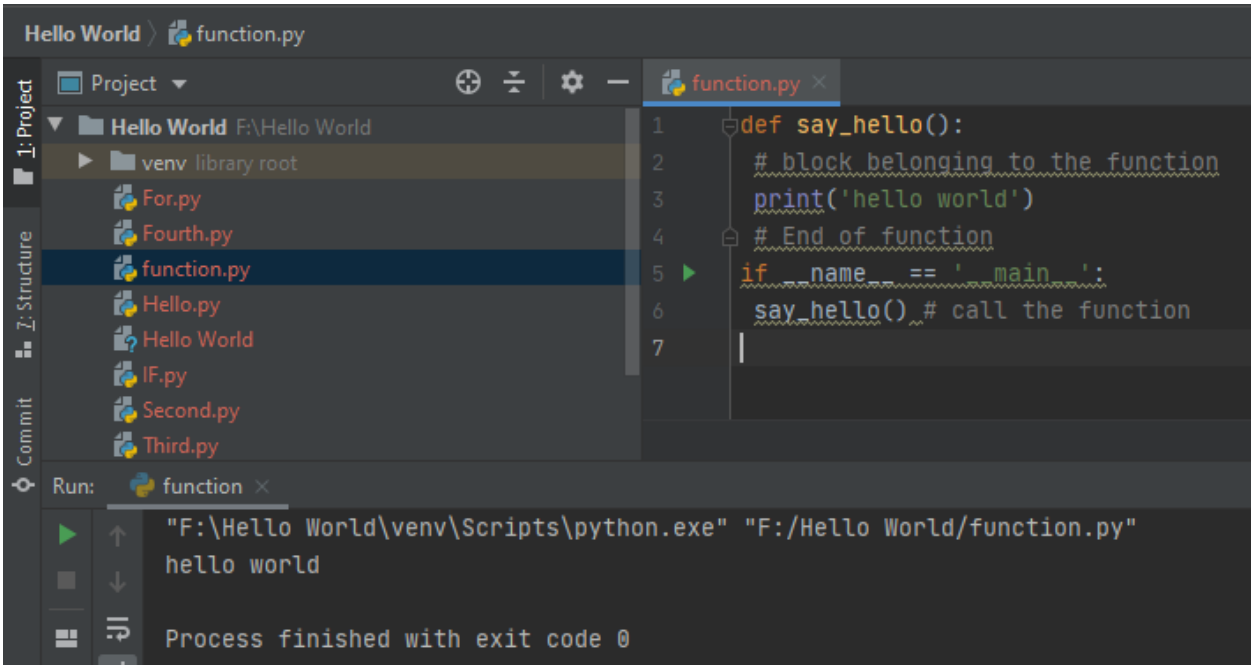
Modules: Modules allow reusing a number of functions in other programs.

Exercises:

Create a python project using with SDN_LAB



Python function (save as function.py)



Python function (save as function_2.py)

```python
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)
```

```
"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)

```python
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)
```

```
"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)

```python
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)
```

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function_global.py"
x is 50
Changed global x to 2
Value of x is 2

Process finished with exit code 0
```

Python modules

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

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

 from mymodule import say_hi, __version__

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

```
"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

```python
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()
```

```
"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

```python
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()
```

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/remote_machine_info.py"
 Remote host name: www.python.org
Error accesing www.python.org: error number and detail [Errno 11001] getaddrinfo failed

Process finished with exit code 0
```

Converting an IPv4 address to different formats

```python
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()
```

```
"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

```
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()
```

```
"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

```
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()
```

```
"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

```python
import socket
import sys
import argparse
import codecs
from codecs import encode, decode
host = 'localhost'
data_payload = 4096
backlog = 5
def echo_server(port):
 """ A simple echo server """
 # Create a TCP socket
 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 # Enable reuse address/port
 sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
 # Bind the socket to the port
 server_address = (host, port)
 print ("Starting up echo server on %s port %s" %server_address)
 sock.bind(server_address)
 # Listen to clients, backlog argument specifies the max no. of queued connections
 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))
 # end connection
 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()
 port = given_args.port
 echo_server(port)
```

echo_server()

Client Code:

echo_server.py echo_client.py

```python
#!/usr/bin/env python
import socket
import sys
import argparse
import codecs
from codecs import encode, decode
host = 'localhost'
def echo_client(port):
 """ A simple echo client """
 # Create a TCP/IP socket
 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 # Connect the socket to the server
 server_address = (host, port)
 print ("Connecting to %s port %s" % server_address)
 sock.connect(server_address)
 # Send data
 try:
     #Send data
     message = "Test message: SDN course examples"
     print ("Sending %s" % message)
     sock.sendall(message.encode('utf_8'))
     # Look for the response
     amount_received = 0
```

echo_server.py echo_client.py

```python
        # Look for the response
        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(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.