CYBER SECURITY AND NETWORK

Lochana koralage

How to Hack with Python

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory

```
#!/usr/bin/python

counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

Python Strings

```
str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

Standard Data Types :

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters

- ✓ Numbers
- ✓ String
- ✓ List
- ✓ Tuple
- ✓ Dictionary

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas

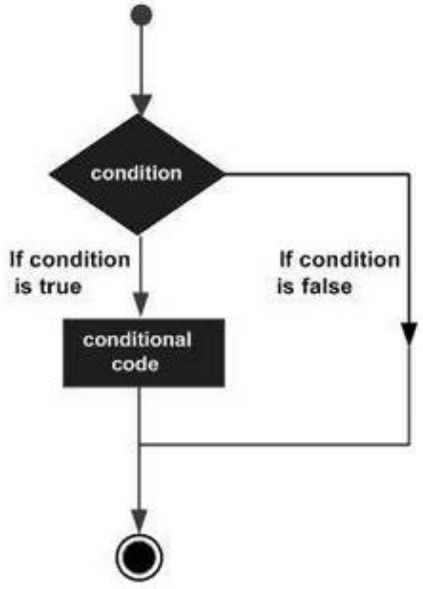
```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple              # Prints the complete tuple
print tuple[0]           # Prints first element of the tuple
print tuple[1:3]         # Prints elements of the tuple starting from 2nd till 3rd
print tuple[2:]          # Prints elements of the tuple starting from 3rd element
print tinytuple * 2      # Prints the contents of the tuple twice
print tuple + tinytuple  # Prints concatenated tuples
```

Functions The Python language comes with preinstalled functions**.**

What is function ? In Python, a function is a group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks

- ✓ int() – Use this function to truncate numeric data. It simply gives the integer part of
- ✓ the argument.
- ✓ len() – This function counts the items in a list.
- ✓ exit() – This function lets you exit a program.
- ✓ max() – With this function, you can determine the highest value of a list.
- ✓ type() – Use this function to identify the data type of a Python object.
- ✓ float() – This function converts its argument into a floating-point numeral.
- ✓ sorted() – Use this function to sort the entries of a list.
- ✓ range() – This function gives a list of numbers between two specific values. You
- ✓ need to set the said values as the function’s arguments.

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions**.**

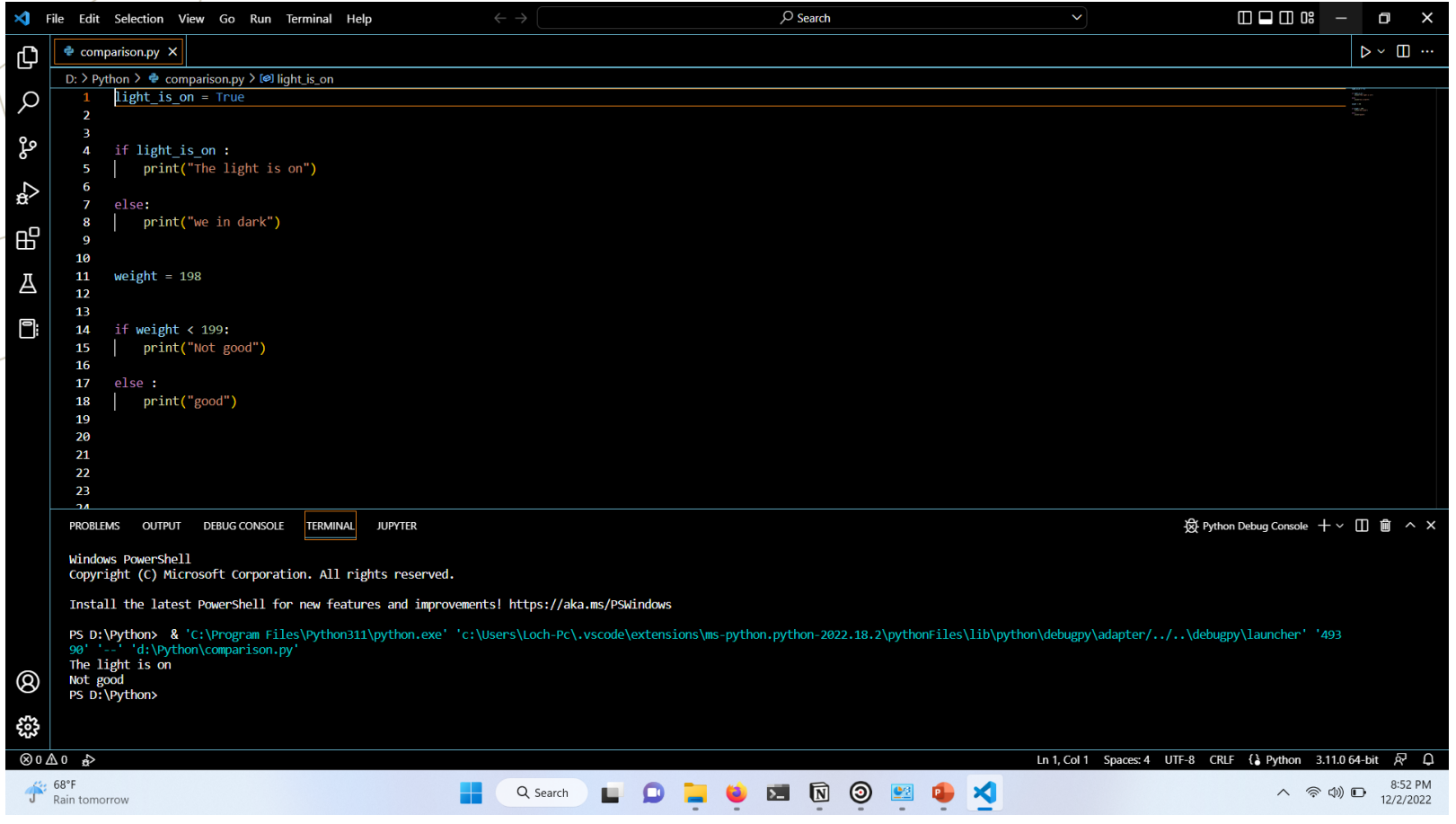


#!/usr/bin/python var = 100 if (var == 100) : print "Value of expression is 100" print "Good bye!"

How to Network with the Python Language

Python has a module called “socket”. This module allows you to build network connections using the Python language. Let’s see how this module works. For this example, you’ll use “socket” to build a TCP (Transmission Control Protocol) connection

Comparison and else



Functions

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

The steps that you need to take are:

1. Import the right module.
2. Create a variable that belongs to a class called “socket”. Set “practice” as the variable’s name.
3. Use the method named “connect()” to establish a connection to a port. The actual process ends here. The remaining steps will show you some of the things you can do after establishing a connection.
4. Use “recv” to acquire 1024 data bytes from the current socket.
5. Save the information in a new variable called “sample”.
6. Print the information inside the “sample” variable.
7. Terminate the connection.
8. Save the code as “samplesocket” and issue “chmod”.

Packets

Many protocols, including the principal protocols in the Internet protocol suite, employ a technique called packetization to help manage data while it's being transmitted across a network.

```
#!usr/bin/env python
import socket
practice = socket.socket()
practice.connect((“192.168.1.107”, 22))
sample = practice.recv(1024)
print sample
practice.close
```

```
#a socket object is created for communication
clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# now connect to the web server on port 80
# - the normal http port
clientsocket.connect(("www.google.com", 80)
```

```
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#bind the socket to a public host and a well-known port
serversocket.bind((socket.gethostname(), 80))

#become a server socket and listen for connections
serversocket.listen(5)
```

Python Sockets - SOCK_STREAM and SOCK_DGRAM

SOCK_STREAM	SOCK_DGRAM
For TCP protocols	For UDP protocols
Reliable delivery	Unrelible delivery
Guaranteed correct ordering of packets	No order guaranteed
Connection-oriented	No notion of connection(UDP)
Bidirectional	Not Bidirectional

Socket Module in Python

socket_family: This is either AF_UNIX or AF_INET. We are only going to talk about INET sockets in this tutorial, as they account for at least 99% of the sockets in use.

socket_type: This is either SOCK_STREAM or SOCK_DGRAM.

Protocol: This is usually left out, defaulting to

Client Socket Methods

connect( )

To connect to a remote socket at an address. An address format(host, port) pair is used for AF_INET address family.

Server Socket Methods

Following are some server socket methods:

bind( )

This method binds the socket to an address. The format of address depends on socket family mentioned above(AF_INET).

listen(backlog)

This method listens for the connection made to the socket. The backlog is the maximum number of queued connections that must be listened before rejecting the connection.

accept( )

This method is used to accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair(conn, address) where conn is a new socket object which can be used to send and receive data on that connection, and address is the address bound to the socket on the other end of the connection.

Working with TCP Sockets

```
#!/usr/bin/python

#This is tcp_server.py script

import socket                                   #line 1: Import
socket module

s = socket.socket()                             #line 2: create a
socket object
host = socket.gethostname()             #line 3: Get current machine
name
port = 9999                                     #line 4: Get
port number for connection

s.bind((host,port))                             #line 5: bind
with the address

print "Waiting for connection..."
s.listen(5)                                     #line 6:
listen for connections

while True:
        conn,addr = s.accept()          #line 7: connect
and accept from client
        print 'Got Connection from', addr
        conn.send('Server Saying Hi')
```

```
!/usr/bin/python

#This is tcp_client.py script

import socket

s = socket.socket()
host = socket.gethostname()     # Get current machine name
port = 9999                                     # Client wants
to connect to server's

# port number 9999
s.connect((host,port))
print s.recv(1024)                              # 1024 is bufsize
or max amount

# of data to be received at once
s.close()
```

Requests with urllib

```
from urllib.request import urlopen
response = urlopen('http://www.debian.org')
Response
```

Response objects

```
response.url
 response = urlopen('http://www.debian.org’)
 response.read(50)
response.status
```

```
import urllib.error
from urllib.request import urlopen
try:
    urlopen('http://www.ietf.org/rfc/rfc0.txt')
except urllib.error.HTTPError as e:
        print('status', e.code)
        print('reason', e.reason)
        print('url', e.url)
```

```
def download_file(bucket, s3_name, local_path):
url = 'http://{}.{}/{}'.format(bucket, endpoint, s3_name)
r = requests.get(url, auth=auth)
if r.ok:
open(local_path, 'wb').write(r.content)
print('Downloaded {} OK'.format(s3_name))
else:
xml_pprint(r.text)
```