(d) Directory Tools

(1) Display list of files (or) sub directory from Directory using 'dir'

C:>> dir /B

↳ Name of its directory

Output:

Parts
Span.txt
temp.bin

(2) Display list of files (or) sub directory from ~~Directory~~ current working directory (or) Any directory Use 'ls' command

Example

C:> C:\Parts\bin\ ls

Output

b.txt
A.txt
Bin etc.

(3) Reading list of file from Directory and display in same line.

Page No.
Date : / /

Here we have to import os first.

Example:

```
>> import os
>>> os.popen('dir/b').readline()
```

Output

['Posts', 'post', 't.txt']

(4) Reading file from Directory using iterator

```
>>> for line in os.open('dir/b'):
        print(line)
```

(5) glob module for displaying file in specific pattern

Example

```
>> import glob
>> glob.glob('*')
```

↑ display all from current directory

Page No.
Date : / /

>> glob.glob('*.bin')

display list of file which contain extention ".bin".

(6) list the content of Directory using os.listdir call

>> import os

>> os.listdir(os.curdir)

It display list of file and subdirectory from current working directory

(7) Spliting and joining listing Result

>> dirname = r'C:\temp\parts'

Directory name

>> import glob
>> for file in glob.glob(dirname):
    head, tail = os.path.split(file)
    print(head, tail)

For Example:

parts n.txt
parts m.txt

Page No.
Date : / /

Join

```
>>> import os
>>> for file in os.listdir(name):
        Print (os.path.join(dirname, file))
```

output:

```
C:\temp\path\part.txt
C:\temp\part\m.txt
```

(9) Walking Directory Tree

os.walk is generator function - at each directory in its tree, it yields a three item tuple, containing the name of its current directory as well as list of both all the files and all its subdirectories in its current directory.

Example:

```
>> import os
>>> for(dirname, subdir, file) in os.walk('.'):
        Print (dirname)
        for fname in file:
            Print (os.path.join(dirname, fname))
```

Output:

```
.\m.txt
.\F1.txt
.\part\p1
.\part\p2 . .
```

Page No.
Date / /

(10) moving to next subdirectory

```
>> gen = os.walk('c:\\temp\\test')
>> [redacted]
>> gen.__next__()
```

Output:

```
[roots] , ['r.html, m.text ...]
```

↑ next subdirectory in test Folder

(11) list the file in directory using recursive

```
import sys, os

def mylister(currdir):
    Printf(currdir)
    for file in os.listdir(currdir):
        Path = os.path.join(currdir, file)
        if not os.path.isdir(Path):
            Print(path)
        else:
            mylister(path)
```

II PROGRAM EXITS

Page No.
Date : / /

(1) System Module exit

>> import sys

>>> try:
    sys.exit()
except SystemExit:
    print('ignore exit');

(2) OS module exit

def out():
    import os
    os._exit(99)

status code for exiting

(3) Shell script exit status code

>> echo $status

(4) exit status with os.system cmd
os.open

>> import os
>> pipe = os.open("Python test.py")
>> pipe.read()
Hi
>> stat = pipe.close()

return status code.

Page No.
Date : / /

THREADING

* Thread is a light weight Process that represents a seperate path of execution of a group of statement

* It Provides control to execute the Program.

Program (To Find the currently running thread)

```
import threading
Print( threading.currentthread().getName())
if threading.current-thread() == threading.main-thread():
    Print("main thread")
else:
    Print("User thread")
```

MULTI THREADING

* Here more then one threads created to control more then task and executes simultaneously.

Use of thread:

(1) Handing multiple client on a network

(2) Create games and animation.

Page No.
Date : / /

CREATING THREAD:

In Python thread can be created in three different ways.

(1) creating a thread without using class

(2) creating a thread by creating subclass to Thread class

(3) creating a thread ~~by~~ without creating subclass to Thread class.

WAY 1 (THREAD WITHOUT CLASS)

```
from threading import *    // import thread module

def display():
    print('SRM').          } create method to control by thread.

for i in range(5)
    t = Thread(target = display)    // method for thread
                                    // creating Thread Object without class
    t.start()
```

start thread object

Page No.
Date : / /

WAY 2 (CREATING THREAD WITH SUBCLASS)

Steps

1. Create subclass from Thread
2. ~~Start thread~~ Create Thread object for subclass
3. start thread using start() method

EXAMPLE

```
From threading import Thread

class mythread (Thread) :                    // Step (1)
    def run(self) :
        for i in range(1,6):                 } Override run method.
            print(i)

t1 = mythread()                              // Step (2)

t1.start()                                   // Step (3)
```

Page No.
Date : / /

WAY 3

(CREATING THREAD WITHOUT SUBCLASS)

steps

① create one class
② create thread to control method defined inside it class. like way ①
③ start thread.

Example:

```
for threading import *

class mythread :                                    // step ①
    def __init__(self, str)
        self.str = str
// constructor

    def display(self, x, y)
        print(self.str)
// method.     print(x, y)

obj = mythread("SRM")                               // step ②

t1 = Thread(target = obj.display, args =(1,2))      // step ②
t1.start()                                          // step ③
```

Page No.
Date : / /

Thread class methods

(Assume t is thread object)

t.start → starts the thread.

t.join([timeout]) – waits until the thread terminates (or) time out

t.is_alive() → check whether thread is alive or not

t.getName() => gives a name to thread.

t.setName(name) => returns name of the thread.

t.name => This a property that represents the thread name

t.setDaemon(flag)
↳ makes a thread daemon thread if the flag is true

t.isDaemon() → check whether thread is Daemon

Page No.
Date : / /

MULTI THREAD PROGRAMMING:

```
from threading import *
from time import *

class cse:
    def display(self):
        for i in range(1,6):
            print(i)

obj1 = cse()
obj2 = cse()
```

Object for class

```
// Two Thread created. one for obj1 and another one for obj2
t1 = Thread(target = obj1.display)
t2 = Thread(target = obj2.display)

// Start 2 thread t1 and t2
t1.start()
t2.start()
```

Now t1 and t2 start to execute simultaniously.

Page No.
Date : / /

THREAD SYNCHRONIZATION

* When a thread is already acting on an object, preventing any other thread from acting on same object is called thread Synchronization (or) thread safe.

Advantage

① Avoid thread interference
② Avoid data inconsistancy

} major problem in multithreading without Synchronization

* In Python Synchronization achieved by two Technique.

① Using locks
② Using semaphores.

Locks

* It is used for lock the object on which the thread is acting

Steps

① Create lock Object for lock() class.
② The Lock the object using acquire() method
③ Release the object using release() method.

Page No.
Date : / /

EXAMPLE

```
from threading import *
from time import *

class cse :
    def display (self) :

class cse :
    def __init__ (self) :
        self.l = Lock()          // step①

    def display (self) :
        self.l.acquire()         // step②
        for i range(5)
            print (i)
        self.l.release()         // step③

ob1 = cse()
ob2 = cse()

t1 = thread (target = ob1.display)
t2 = thread (target = ob2.display)

t1.start()
t2.start()
```

Page No.
Date : / /

Using Semaphore:

A semaphore is an object that provides synchronization based on a counter. A semaphore is created as an object of Semaphore class

l = Semaphore (countervalue)

* if 'countervalue' is not given its default value will be 1.

* when acquire method called, the counter gets decremented by 1. when release called it is incremented by 1.

l. acquire() # make countervalue is 0

l. release() # make countervalue is 1

DEADLOCK

Page No.
Date : / /

One problem in Synchronization is deadlock problem.

* Deadlock occurs only when we lock the resource using Synchronization.

EXAMPLE

(SHOW DEAD LOCK OF THREADS DUE TO LOCKS ON OBJECT)

```
for threading import *
    l1 = LOCK()
    l2 = LOCK()

def task1():
    l1.acquire()
    l2.acquire()
    l2.release()
    l1.release()

def task2():
    l2.acquire()
    l1.acquire()
    l1.release()
    l2.release()

t1 = Thread(target = task1())
t2 = Thread(target = task2())
t1.start()
t2.start()
```

Page No.
Date : / /

AVOIDING DEADLOCK

```
for threading import *
    l1 = Lock()
    l2 = Lock()

def task1():
    l1.acquire()
    l2.acquire()
    l2.release()
    l1.release()

def task2():
    l2.acquire()
    l2.acquire()
    l2.release()
    l1.release()

t1 = Thread( target = task1())
t2 = Thread( target = task2())

t1.start()
t2.start()
```

ITC

(Inter Thread communication)

Using sleep method

* It provids communication between thread

// Consumer - Producer Problem

```
from threading import *
from time import *

class Producer
    def __init__(self):
        self.lst = []
        self.dataProduced = False

    def produce(self):
        for i in range(1,11):
            self.lst.append(i)
            sleep(1)
            print("Item Produced")
        self.dataProduced = True

class Consumer:
    def __init__(self, prod):
        self.prod = prod

    def consume(self):
        while self.prod.dataProduced==false:
            sleep(0.1)
        print(self.prod.lst)
```

Page No.
Date: / /

P = Producer()
C = consumer(p) // IPC

t1 = Thread(target = P.Produce)
t2 = Thread(target = C.Consume)

t1.start()

t2.start()

In this example, once item produced no ~~it~~ information passed to consumer when consumer is sleeping ~~state~~ state. Here consumer know only after time elapsed. This is drawback of sleep method.

To improve IPC, Python Provides two important ways for this.
They are

1. Using notify() and wait method
2. Using queue.

IPC Using notify() and wait/notify()

```
class Producer :
    def __init__(self):
        self.lst = []
        self.cv = condition()   # Create condition object to call notify() and wait() methods

    def Producer(self):
        self.cv.acquire()
        for i in range(1,11):
            self.lst.append(i)
            sleep(1)
            print("Item Produced")
        self.cv.notify()
        self.cv.release()
```

```
class consumer :
    def __init__(self, Prod):
        self.Prod = Prod

    def consume(self):
        self.Prod.cv.acquire()
        self.Prod.cv.wait(timeout=0)
        self.
        self.Prod.cv.release()
        print(self.Prod.lst)
```

Page No.
Date : / /

```
P = Producer()

C = Consumer(P)

t1 = Thread(target = P.produce)
t2 = Thread(target = C.consume)


t1.start()
t2.start()
```

ITC using a queue

Page No.
Date : / /

```
from threading import *
from queue import *

class Producer:
    def __init__(self):
        self.q = Queue()

    def Produce(self):
        for i in range(1,11):
            print("Producing item:", i)
            self.q.put(i)
```

Create object from Queue class

Predefined use put() method to add item

```
class Consumer:
    def __init__(self, Prod):
        self.Prod = Prod

    def consume(self):
        for i in range(1,11):
            print(self.Prod.q.get())
```

use get() method to receive item

Page No.
Date : / /

```
p = Producer()
c = consumer(p)
t1 = Thread(target = p.produce)
t2 = Thread(target = c.consume)
t1.start()
t2.start()
```