Multithreading

Types of applications

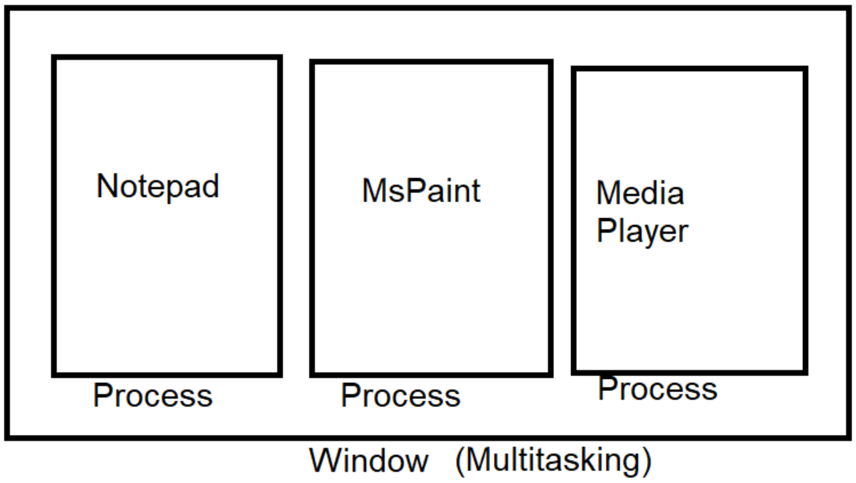
1. Single tasking applications
2. Multitasking applications

Single tasking application
An application which performs one operation or task is called single tasking application.
Task is nothing but an operation performed by an application.

Multitasking applications
An application which allows to perform more than one operation simultaneously or concurrently is called multitasking application.

1. Process based multitasking
2. Thread based multitasking

Process based multitasking
Process is nothing instance of program or program under execution.
Simultaneous execution of more than one program is called process based multitasking.



Process based multitasking is useful for developing operating systems.

Thread based multitasking

Thread based multitasking is useful at application level. Default flow of execution of program is sequential. One operation depends on another operation.

```
def even():
    for num in range(1,21):
        if num%2==0:
            print(f'Even:{num}')

def odd():
    for num in range(1,21):
        if num%2!=0:
            print(f'Odd:{num}')

even()
odd()
```

Output:

```
Even:2
Even:4
Even:6
Even:8
Even:10
Even:12
Even:14
Even:16
Even:18
Even:20
Odd:1
Odd:3
Odd:5
Odd:7
Odd:9
Odd:11
Odd:13
Odd:15
Odd:17
Odd:19
```

Thread performs operation independent of other operations. Simultaneous execution of more than one thread is called threading based multitasking.
Thread is instance of a process.
Thread is independent path of execution within program.

Advantage of multitasking

1. Utilizing CPU idle time
2. Sharing Resources

For developing thread based applications python provides “threading” module.

How to create thread?
Threading module provides Thread class (Data type). Using Thread class or data type user threads are created.

1. Callable object
2. Using inheritance

Callable object
In this approach thread is created by giving input as a function to thread object.
A function is injected to thread object.

Syntax: Thread(target=None)

Target is a function object which is used by thread to perform operation.

Thread scheduling is done by PVM (Python Virtual Machine)
There are different scheduling algorithms; one of it is time slicing. In time slicing every thread is given predefined time.

Example

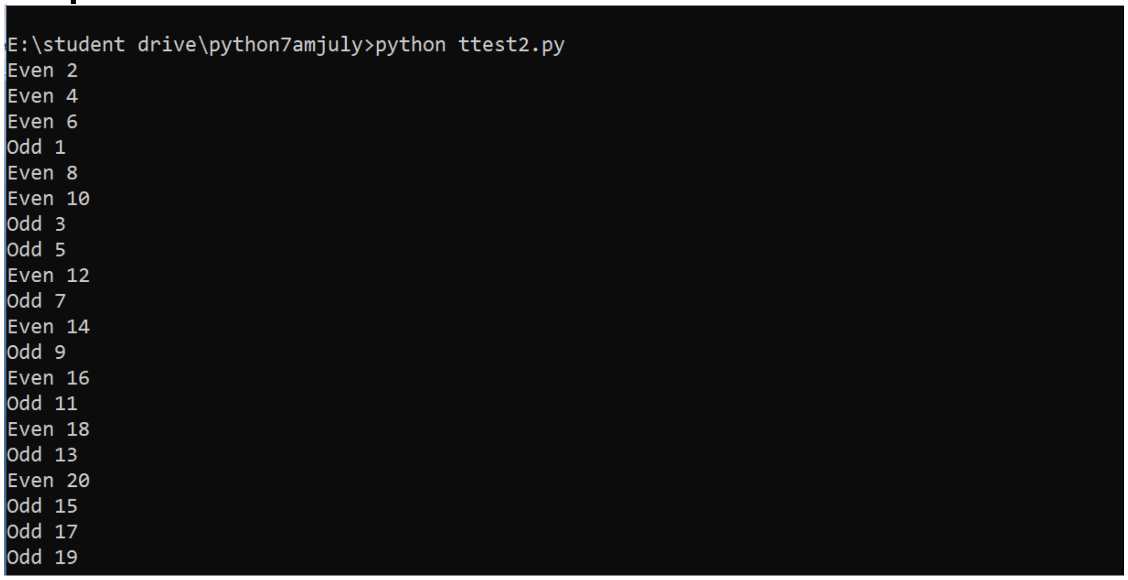
```
import threading
def even():
   for n in range(1,21):
      if n%2==0:
         print(f'Even {n}')

def odd():
    for n in range(1,21):
        if n%2!=0:
            print(f'Odd {n}')



t1=threading.Thread(target=even)
t2=threading.Thread(target=odd)
t1.start()
t2.start()
```

Output:



```
E:\student drive\python7amjuly>python ttest2.py
Even 2
Even 4
Even 6
Odd 1
Even 8
Even 10
Odd 3
Odd 5
Even 12
Odd 7
Even 14
Odd 9
Even 16
Odd 11
Even 18
Odd 13
Even 20
Odd 15
Odd 17
Odd 19
```

Execution of thread is done by invoking start() method.

Example:

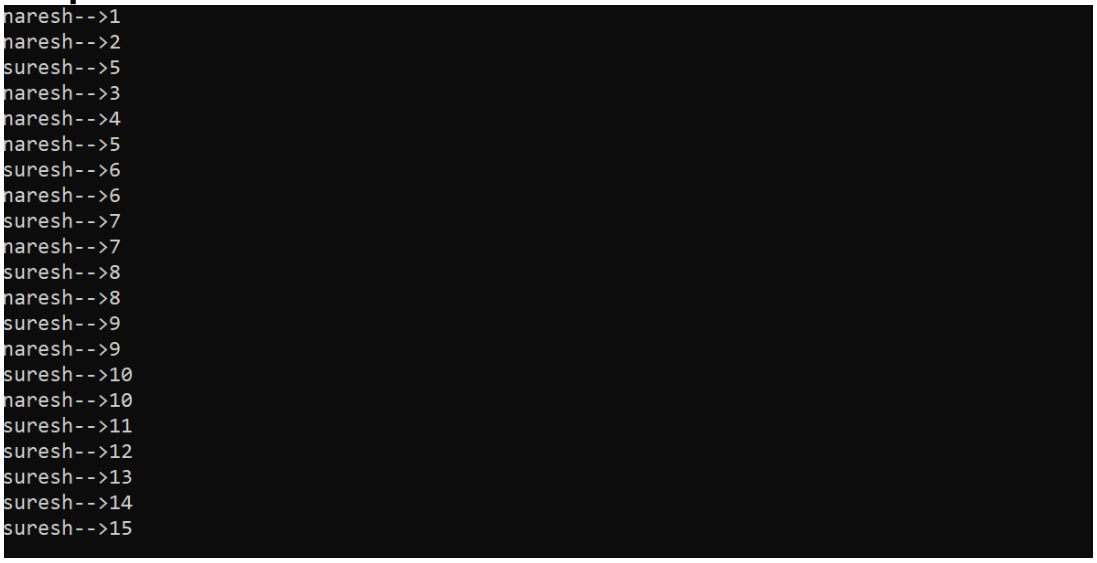
```
import threading

def printNum(name,a,b):
    for n in range(a,b):
        print(f'{name}-->{n}')


t1=threading.Thread(target=printNum,args=("naresh",1,11))
```

```
t2=threading.Thread(target=printNum,args=("suresh",5,16))
t1.start()
t2.start()
```

Output:



```
naresh-->1
naresh-->2
suresh-->5
naresh-->3
naresh-->4
naresh-->5
suresh-->6
naresh-->6
suresh-->7
naresh-->7
suresh-->8
naresh-->8
suresh-->9
naresh-->9
suresh-->10
naresh-->10
suresh-->11
suresh-->12
suresh-->13
suresh-->14
suresh-->15
```

Creating thread by inheriting Thread class
A thread can be created by inheriting Thread class.

Syntax:

```
class <user-thread-class-name>(threading.Thread):
         def __init__(self):
             super().__init__()
         def run(self):
              operation of thread
```

Operation of thread is written inside run() method.

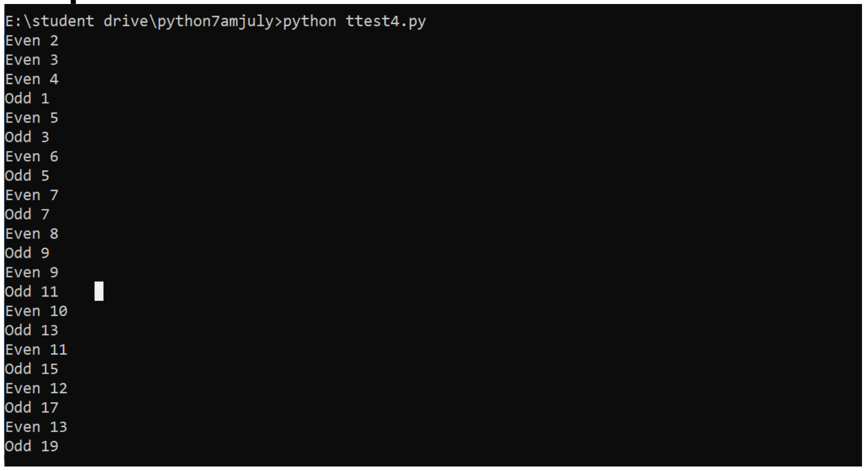
Example:

```
import threading
class EvenThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self): # Overriding method
        for n in range(2,20):
            print(f'Even {n}')
```

```
class OddThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self):
        for n in range(1,20,2):
            print(f'Odd {n}')


t1=EvenThread()
t2=OddThread()
t1.start()
t2.start()
```

Output:



```
E:\student drive\python7amjuly>python ttest4.py
Even 2
Even 3
Even 4
Odd 1
Even 5
Odd 3
Even 6
Odd 5
Even 7
Odd 7
Even 8
Odd 9
Even 9
Odd 11
Even 10
Odd 13
Even 11
Odd 15
Even 12
Odd 17
Even 13
Odd 19
```

Life Cycle of a thread