Threading by defining function

In [ ]:
```
from threading import Thread
from time import sleep

def myfunction():
    for x in range(10):
        print(x**3)
        sleep(1)

t1=Thread(target=myfunction)
t1.start()
t1.join()
```

```
0
1
8
27
64
125
```

In [ ]:
```
def myfunction():
    for x in range(10):
        print(x**2)
        sleep(1)
def myfunction2():
    for x in range(10,20):
        print(x**2)
        sleep(1)

t1=Thread(target=myfunction)
t2=Thread(target=myfunction2)
t2.start()

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

Threading by class

In [ ]:
```
import threading

class MyThread(threading.Thread):
    def __init__(self):
        super(MyThread,self).__init__()

    def run(self):
        print("running")

thread_list=[]
for i in range(10):
    thread =MyThread()
    thread_list.append(thread)
    thread.start()
```

Creating a thread without using a class

In [ ]:
```
from threading import *

def display():
    print('Hello there')

for i in range(5):
    t=Thread(target=display)
    t.start()
    t.join()
```

In [ ]:
```
def display(msg):
    print(msg**2)

for i in range(10):
    t=Thread(target=display,args=(i,))
    t.start()
```

In [ ]:
```
class MyThread(Thread):
    # constructor that calls Thread class constructor
    def __init__(self,str,n):
        Thread.__init__(self)
        self.str=str
        self.n=n
    #override the run() method of Thread class

    def run(self):
        print(self.str,self.n)
#creat an instance of MyTHread class and pass the string

t1=MyThread('Hello',8)
t1.start()
t1.join()
```

In [ ]:
```
class MyThread:
    def __init__(self,str):
        self.str=str
    def display(self,x,y):
        print(self.str)
        print('The Product=',x*y)
# create an instance to our class and store 'Hello' string

obj =MyThread('Hello')
#create a thread to run display method of obj

t1=Thread(target=obj.display, args=(50,2))
t1.start()
#run the thread
#it can also be written as

t1=Thread(target=obj.display(50,2))
t1.start()
```

Thread Class Methods

t.start(): starts thread

t.join([timeout]): wait till thread terminate of timeout occours

t.is_alive()

t.setName(name)

t.getName()

t.name: same as t.Daemon

t.setDaemon(flag): makes a daemon thread if flag is true
Daemon flags are functions keeps executing the thread all the time

t.isDaemon()

t.daemon: Used to set True or False for Daemon flag, Used as t.daemon=True

Single tasking using Thread

In [ ]:
```
#exaple 1 without thread
import time

def sqr(n):
    for x in n:
        print('remainder after dividing by 2',x%2)
        time.sleep(1)

def cube(n):
    for x in n:
        print('remainder after dividing by 3',x%3)
        time.sleep(1)

n=[1,2,3,4,5,6,7,8]
start=time.time()
cube(n)
sqr(n)
end=time.time()
print(end-start)
```

In [ ]:
```
#exaple 2 with thread
from threading import *
import time

def sqr(n):
    for x in n:
        print('1remainder after dividing by 2',x%2)
        time.sleep(1)

def cube(n):
    for x in n:
        print('2remainder after dividing by 3',x%3)
        time.sleep(1)

n=[1,2,3,4,5,6,7,8]
start=time.time()
t1=Thread(target=sqr, args=(n,))
t2=Thread(target=cube, args=(n,))
t1.start()
t2.start()
t2.join()
t1.join()
end=time.time()
print(end-start)
```

in example 2 we see that when the tread t1 is sleeping it switches to t2 and vice versa so it takes less time and runs the program when one of the thread is sleeping while in example 1 we see that when one part of program is sleeoing whole program is sleeping

In [ ]: