Multi-Threading

what is Multi-tasking?

the concept of doing or executeing multiple tasks parllely is known as a Multi-tasking.

the Multi-tasking can be categorized into two types,they are

1).Process Based Multi-Tasking

ex: Multi-processing

2).Thread Based Multi-Tasking

ex: Multi-threading

what is multi-threading?

the concept of doing or executeing multiple threads parallely,is known as a multi-threading.

what is thread?

a thread is a light-weight process,which is used to represent the business logics to perform a single operation/action.

python supporting multi-threading concept,in python by default every module act as a one thread,that thread is called main-thread.

in python,we can implement the multi-threading concept by using threading module.

how to create a thread?

we can define any logic as a thread by overriding run() of Thread class of threading module.

ex:

```
import threading
print("welcome")
class test(threading.Thread):
    def run(self):
        for i in range(5):
            print("hai")
t1=test()
t1.run()
for j in range(5):
    print("hello")
```

output:
------
welcome
hai
hai
hai
hai
hai
hello
hello
hello
hello
hello

note:
----
in the above example we are creating a thread but it is not activated.

whenever we are calling run() logic directly,in that that run() logic dont act as a thread,it act as a normal method.

if we want to activete our run() logic as a thread by calling run() logic through the start() of Thread class threading module.

ex2:

```
import threading
print("welcome")
class test(threading.Thread):
    def run(self):
        for i in range(5):
            print("hai")
t1=test()
t1.start()
for j in range(5):
    print("hello")
```

output1:
------
welcome
haihello

haihello

haihello

haihello

haihello

output2:
----
welcome
hellohai

hellohai

hellohai

hellohai

hellohai

ex3:

```
import threading
print("welcome")
class x(threading.Thread):
    def run(self):
        for i in range(5):
            print("hai")
class y(threading.Thread):
    def run(self):
        for i in range(5):
            print("siva")
t1=x()
t1.start()
t2=y()
t2.start()
for j in range(5):
    print("hello")
```

output1:
------
welcome
haihellosiva

haihellosiva

haihellosiva

haihellosiva

haihellosiva

outpu2:

```
------
welcome
haisivahello


haisivahello


haisivahello


haisivahello


haisivahello
```

thread life cycle?

---

every thread haveing 4-states,they are

- Born/create state
- active satte
- suspend/blocked/waiting state
- dead state

whenever we are creating a thread,our thread in 'born' state.

whenever we are calling our thread through the start() of a Thread class of threading module then only our thread is going to active state other wise our thred is not activated.

whenever our thread is going to active state,that thread is executed successfully,after executeing the thread our thread is going to Dead state.

if we want to suspend the one thread execution by using sleep() of time module.

whenever time is over then immediately our thread is moveing to suspended/blocked state to active state.

note:

---

the thread's execution is not under the programmer controller.

the thread's execution is under the control of job schedular.

what is job schedular?
---
a job-schedular is a predefined program,which is used to scduling the job's based on scheduling algorithm's like

sjfs --> shortest job first serve

fcfs --> first come first serve

round robbin

priority queues

scheduling the job's means to assign which job/thread is executed first,which jon/thread is executed second,....,which job/thread is executed last.

how to suspend one thread execution temporary some time?
---
we can suspend the one thread execution temporary some time by calling sleep() of time module.

the sleep() takes the time in seconds format

ex: 4

```
import threading
import time
print("welcome")
class x(threading.Thread):
    def run(self):
        for i in range(5):
            print("hai")
class y(threading.Thread):
    def run(self):
        time.sleep(10)
        for i in range(5):
            print("siva")
t1=x()
t1.start()
t2=y()
t2.start()
for j in range(5):
    print("hello")
```

output:
---
welcome

```
haihello

haihello

haihello

haihello

haihello

siva
siva
siva
siva
siva
```

whenever the multiple threads to access the same functionality or logic at a time,in that case our thread's are going to be critical section.

whenever threads are going to be critical section in that case dead lock is occured.

ex:

```
import threading
def wishes(a):
    print("hello",a,"good morning")
class x(threading.Thread):
    def run(self):
        wishes("siva")
class y(threading.Thread):
    def run(self):
        wishes("krishna")
t1=x()
t1.start()
t2=y()
t2.start()
```

output:

```
hellohello  sivakrishna  good morninggood morning
```

note:
----
in the above example deadlock is occured,it is a problem to overcome that problem by using synchronization concept.

thread synchronization?
----
the concept of avoide the multiple threads to access the same functionality

or logice at a time,is known as a thread synchronization.

we can implement the thread synchronization by using following ways,

Locks
Semaphore
Events
Conditions

we can implement the synchronization by using Locks,first we need to create a lock object.

we can create a lock object by calling Lock() threading module.

```
lock_obj=threading.Lock()
```

we can acquire the lock on any logic by calling acquire() of lock_obj.

```
lock_obj.acquire()
```

we can release the lock on any logic by calling release() of lock_obj.

```
lock_obj.release()
```

if any one thread acquire the lock on any logic in that case that the remaing threads are going to be waiting state until that thread release the lock on that logic.

ex:

```
---
import threading
def wishes(a):
    print("hello",a,"good morning")
lock_obj=threading.Lock()
class x(threading.Thread):
    def run(self):
        lock_obj.acquire()
        wishes("siva")
        lock_obj.release()
class y(threading.Thread):
    def run(self):
        lock_obj.acquire()
        wishes("krishna")
        lock_obj.release()
t1=x()
t1.start()
t2=y()
t2.start()
```

output:

```
------
hello siva good morning
hello krishna good morning

ex:
---
join():
        Wait until the thread terminates.


import threading
print("welcome")
def wishes(a):
    print("hello",a,"good morning")
lock_obj=threading.Lock()
class x(threading.Thread):
    def run(self):
        lock_obj.acquire()
        wishes("siva")
        lock_obj.release()
class y(threading.Thread):
    def run(self):
        lock_obj.acquire()
        wishes("krishna")
        lock_obj.release()
t1=x()
t1.start()
t2=y()
t2.start()
t1.join()
t2.join()
print("bye")

output:
------
welcome
hello siva good morning
hello krishna good morning
bye

start():
        Start the thread's activity.

run():
      Method representing the thread's activity.

active_count()
        Return the number of Thread objects currently alive.
```

```
current_thread()
        Return the current Thread object

enumerate()
        Return a list of all Thread objects currently alive.

is_alive()
        Return wheather the thread is alive or not

ex:
---
import threading
print("welcome")
class x(threading.Thread):
    def run(self):
        print("hai")
class y(threading.Thread):
    def run(self):
        print("hello")
t1=x()
t1.start()
t2=y()
t2.start()
print(threading.active_count())

print("bye")

output:
------
welcome
hai4hello
bye

ex2:
---
import threading
print("welcome")
class x(threading.Thread):
    def run(self):
        print("hai")
class y(threading.Thread):
    def run(self):
        print("hello")
t1=x()
t1.start()
t2=y()
t2.start()
print(threading.enumerate())

print("bye")
```

output:
------
welcome
haihello[<_MainThread(MainThread, started 20024)>, <Thread(SockThread, started
daemon 11112)>, <x(Thread-1, started 13992)>, <y(Thread-2, started 15336)>]
bye

ex3:

```
import threading
print("welcome")
class x(threading.Thread):
    def run(self):
        print("hai")
class y(threading.Thread):
    def run(self):
        print("hello")
t1=x()
t1.start()
t2=y()
t2.start()
print(threading.current_thread())
print("bye")
```

output:
------
welcome
hai<_MainThread(MainThread, started 15064)>hello
bye

ex4:

```
import threading
print("welcome")
class x(threading.Thread):
    def run(self):
        print("hai")
class y(threading.Thread):
    def run(self):
        print("hello")
t1=x()
t1.start()
t2=y()
t2.start()
print(t1.name)
t1.name="x-class thread"
print(t1.name)
print(t1.is_alive())
t1.join()
```

```
print(t1.is_alive())
print("bye")


output:
------
welcome
haiThread-1hello


x-class thread
False
False
bye

note:
----
start():
        Start the thread's activity.

run():
      Method representing the thread's activity.

active_count()
        Return the number of Thread objects currently alive.


current_thread()
        Return the current Thread object

enumerate()
        Return a list of all Thread objects currently alive.

is_alive()
        Return wheather the thread is alive or not

join():
        Wait until the thread terminates.

name:
        to set the name to thread and to get the name of the thread.

ex:
---
to create a threads by using functions

import threading
print("welcome")
def msg(x):
    print("hai",x,"good morning")
```

```
t1=threading.Thread(target=msg,name="Thread-1",args=("siva",))
t1.start()
t1.join()
t2=threading.Thread(target=msg,name="Thread-2",args=("rama",))
t2.start()
t2.join()
print("bye")

output:
------
welcome
hai siva good morning
hai rama good morning
bye

what is GIL?
-------------
GIL(Global Interpreter Lock),it allows internally only one thread execution at a
time.
```