ANSWERS ASSIGNMENT – 14

TOPICS – THREADS IN PYTHON

1. **A Python Program to find the currently running thread in a Python.**

Solution:

```
from threading import *
print('Current Executing Thread:', current_thread().getName())
```

2. **A Python Program to create and use it to run a function.**

Solution:

```
from threading import *

# creation function
def display():
    print("Hello I am Running...")


# create thread and run the function 5 times
for i in range(5):
    # create the thread and specify the function as its target
    t = Thread(target=display)

    # run the thread
    t.start()
```

3. **A Python Program to pass arguments to a function and execute it using a thread.**

Solution:

```
from threading import *

# creation function
def display(str):
    print(str)

# create thread and run the function 5 times
for i in range(5):
    t = Thread(target=display, args=("Gauri P", ))
    t.start()
```

4. **A Python Program to create a thread by making our class as sub class to Thread.**

Solution:

```
# creating our own thread
from threading import *

# create a class as sub-class to Thread class
class MyThread(Thread):

    # Override the run() method of Thread class
    def run(self):
        for i in range(1, 6):
            print(i)

# create an instance of MyThread class
t = MyThread()

# start running the thread t
t.start()

# wait till the thread completes its job
t.join()
```

5. **A Python Program to create a thread that accesses the instance variables of a class.**

Solution:

```
# creating our own thread
from threading import *

# create a class as sub-class to Thread class
class MyThread(Thread):

    def __init__(self, str):
        Thread.__init__(self)
        self.str = str

    # Override the run() method of Thread class
    def run(self):
        print(self.str)


# create an instance of MyThread class and pass string
t = MyThread("Asma N")

# start running the thread t
t.start()

# wait till the thread completes its job
t.join()
```

6. **A Python Program to create a thread that acts on the object of a class that is not derived from the thread class.**

Solution:

```
# creating our own thread
from threading import *

# create a class as sub-class to Thread class
class MyThread:

    # A Constructor
    def __init__(self, str):
        self.str = str

    # Override the run() method of Thread class
    def display(self, x, y):
        print(self.str)
        print("The Args are: ", x, y)


# create an instance of MyThread class and pass string
t = MyThread("Hello")

# create a thread to run display method of object
t = Thread(target=t.display, args=(1, 2))

# start running the thread t
t.start()
```

7. **A Python Program to show single tasking using a thread that prepares tea.**

Solution:

```
# creating our own thread
from threading import *
from time import *

# create a class as sub-class to Thread class
class MyThread:

    # A method that perform 3 tasks one by one
    def prepareTea(self):
        self.task1()
        self.task2()
        self.task3()

    def task1(self):
        print("Boil Milk and tea powder fro 5 mins...", end=" ")
        sleep(5)
        print("Done..!!")

    def task2(self):
        print("Add sugar and boil for 3 mins...", end=" ")
        sleep(3)
        print("Done..!!")

    def task3(self):
        print("Filter and Serve...", end=" ")
        print("Done..!!")


# create an instance to our class
obj = MyThread()

# create a thread and run prepareTea method of object
t = Thread(target=obj.prepareTea)

# start running the thread t
t.start()
```

8. **A Python Program that performs two tasks using two threads simultaneously.**

Solution:

```
# creating our own thread
from threading import *
from time import *

# create a class as sub-class to Thread class
class Theatre:

    # constructor that accept a string
    def __init__(self, str1):
        self.str1 = str1

    # a methods that repeats for 5 tickets
    def movieshow(self):
        for i in range(1, 4):
            print(self.str1, ":", i)
            sleep(1)


# create two instances of Theatre class
t1 = Theatre("Cut Ticket")
t2 = Theatre("Show Chair")

# create two Threads to run movieshow()
t1 = Thread(target=t1.movieshow)
t2 = Thread(target=t2.movieshow)

# start running the thread t
t1.start()
t2.start()
```

9. **A Python Program where two threads are acting on the same method to allot a berth for the passenger.**

Solution:

```
from threading import *
from time import *

# creating our own thread
class Railway:

    # constructor that accept no of variables berths
    def __init__(self, available):
        self.available = available

    # a methods that reverse berth
    def reverse(self, wanted):

        # Display no of available berth
        print("Available no. of berths = ", self.available)

        # if available >= wanted, allot the berth
        if(self.available >= wanted):
            # find the thread name
            name = current_thread().getName()

            # display the berth is allocated for the person
            print("%d berth are allocated for %s" % (wanted, name))

            # make time delay so that ticket is printed
            sleep(1.5)

            # decrease the number of available berths
            self.available -= wanted

        else:
            # if available < wanted then say sorry
            print("Sorry, no berth to allot")
```

```
# create instances to Railway class
# Specify only one berth is available
obj = Railway(1)

# create two Threads to run movieshow()
t1 = Thread(target=obj.reverse, args=(1, ))
t2 = Thread(target=obj.reverse, args=(1, ))

# Give names to the threads
t1.setName("First Person")
t2.setName("Second Person")

# start running the threads
t1.start()
t2.start()
```

10. A Python Program displaying thread synchronization using locks.

Solution:

```
from threading import *
from time import *
# creating our own thread
class Railway:

    # constructor that accept no of variables berths
    def __init__(self, available):
        self.available = available

        # create a lock object
        self.l = Lock()

    # a methods that reverse berth
    def reverse(self, wanted):

        # lock the current object
        self.l.acquire()
```

```
        # Display no of available berth
        print("Available no. of berths = ", self.available)

        # if available >= wanted, allot the berth
        if(self.available >= wanted):
            # find the thread name
            name = current_thread().getName()

            # display the berth is allocated for the person
            print("%d berth are allocated for %s" % (wanted, name))

            # make time delay so that ticket is printed
            sleep(1.5)

            # decrease the number of available berths
            self.available -= wanted

        else:
            # if available < wanted then say sorry
            print("Sorry, no berth to allot")

        # task is completed, released the lock
        self.l.release()

# create instances to Railway class
# Specify only one berth is available
obj = Railway(1)

# create two Threads to run movieshow()
t1 = Thread(target=obj.reverse, args=(1, ))
t2 = Thread(target=obj.reverse, args=(1, ))

# Give names to the threads
t1.setName("First Person")
t2.setName("Second Person")

# start running the threads
t1.start()
t2.start()
```

11. A Python Program to show dead lock of threads due to locks on objects.

Solution:

```
# Dead lock of threads
from threading import *

# take 2 locks
l1 = Lock()
l2 = Lock()

# create function for booking a ticket
def bookticket():
    l1.acquire()
    print("Bookticket locked train")
    print("Bookticket wants to lock on compartment ")

    l2.acquire()
    print("Bookticket locked compartment ")

    l2.release()
    l1.release()

    print("Booking ticket done..")


# create function for cancelling a ticket
def cancleticket():

    l2.acquire()
    print("Cancelticket locked compartment ")
    print("Cancelticket wants to lock on compartment ")

    l1.acquire()
    print("Cancleticket locked compartment ")

    l1.release()
    l2.release()

    print("Cancellation of ticket done..")
# create two Threads
t1 = Thread(target=bookticket)
t2 = Thread(target=cancleticket)

# start running the threads
t1.start()
t2.start()
```

12. A Python Program with good logic to avoid deadlocks.

Solution:

```
# Dead lock of threads
from threading import *

# take 2 locks
l1 = Lock()
l2 = Lock()

# create function for booking a ticket
def bookticket():
    l1.acquire()
    print("Bookticket locked train")
    print("Bookticket wants to lock on compartment ")

    l2.acquire()
    print("Bookticket locked compartment ")

    l2.release()
    l1.release()

    print("Booking ticket done..")

# create function for cancelling a ticket
def cancleticket():

    l1.acquire()
    print("Cancelticket locked compartment ")
    print("Cancelticket wants to lock on compartment ")

    l2.acquire()
    print("Cancleticket locked compartment ")

    l2.release()
    l1.release()

    print("Cancellation of ticket done..")

# create two Threads
t1 = Thread(target=bookticket)
t2 = Thread(target=cancleticket)
# start running the threads
t1.start()
t2.start()
```

13. A Python Program where producer and consumer threads communicate with each other through a boolean type variable.

Solution:

```
from threading import *
from time import *


# create producer class
class Producer:

    def __init__(self):
        self.lst = []
        self.dataprodover = False

    def produce(self):
        # create item to add the list
        for i in range(1, 5):
            self.lst.append(i)
            sleep(2)
            print("Item Produce...")

        # inform the consumer that the data production is completed
        self.dataprodover = True


# create the consumer class
class Consumer:
    def __init__(self, prod):
        self.prod = prod

    def consume(self):
        # sleep for 100ms as long dataprodover is False
        while self.prod.dataprodover == False:
            sleep(0.1)

        # Display the content of the list when production is over
        print(self.prod.lst)


# create Producer object
p = Producer()

# create Consumer object and pass producer object
c = Consumer(p)
```

```
# create producer and consumer threads
t1 = Thread(target=p.produce)
t2 = Thread(target=c.consume)

# start running the threads
t1.start()
t2.start()
```

14. A Python Program where thread communication is done through notify() and wait() methods of condition object.

Solution:

```
from threading import *
from time import *


# create producer class
class Producer:

    def __init__(self):
        self.lst = []
        self.cv = Condition()

    def produce(self):

        # lock the conditional object
        self.cv.acquire()

        # create item to add the list
        for i in range(1, 5):
            self.lst.append(i)
            sleep(1)
            print("Item Produce...")

        # inform the consumer that the data production is completed
        self.cv.notify()

        # release the lock
        self.cv.release()
# create the consumer class
class Consumer:
    def __init__(self, prod):
        self.prod = prod
```

```
    def consume(self):

        # get lock on condition object
        self.prod.cv.acquire()

        # wait only for 1 sec after the production
        self.prod.cv.wait(timeout=1)

        # release the lock
        self.prod.cv.release()

        # display the content of the list
        print(self.prod.lst)


# create Producer object
p = Producer()

# create Consumer object and pass producer object
c = Consumer(p)

# create producer and consumer threads
t1 = Thread(target=p.produce)
t2 = Thread(target=c.consume)

# start running the threads
t1.start()
t2.start()
```

15. A Python Program that uses a queue in thread communication.

Solution:

```
from threading import *
from time import *
from queue import *


# create producer class
class Producer:

    def __init__(self):
        self.q = Queue()
```

```
    def produce(self):

        # create item to add the list
        for i in range(1, 5):
            print("Producing Item ...")
            self.q.put(i)
            sleep(1)


# create the consumer class
class Consumer:

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

    def consume(self):
        # receive items from the queue
        for i in range(1, 5):
            print("Receiving Item ...", self.prod.q.get(i))


# create Producer object
p = Producer()

# create Consumer object and pass producer object
c = Consumer(p)

# create producer and consumer threads
t1 = Thread(target=p.produce)
t2 = Thread(target=c.consume)

# start running the threads
t1.start()
t2.start()
```

16. A Python Program to understand the creation of daemon thread.

Solution:

```
from threading import *
from time import *


def display():
    for s in range(5):
        print("Normal thread :", end=" ")
        print(s+1)
        sleep(2)


def display_time():
    for s in range(5):
        print("Normal thread :", end=" ")
        print(s+1)
        sleep(2)


# create the normal thread and attach it to display() and run it
t = Thread(target=display)
t.start()

# create another thread and attach it to dispaly_time()
d = Thread(target=display_time)

# make the thread daemon
d.daemon = True

# run the daemon thread
d.start()
```