Lesson-16: Threads

Threads:

What is a thread?
What is Multithreading?
What is the difference between a ‘process’ and a ‘thread’?
What are the uses of Threads?

IMPORTANT NOTE:

The Thread class represents an activity that is run in a separate thread of control. There are two ways to specify the activity: **by passing a callable object to the constructor**, or **by overriding the run( ) method in a subclass**. No other methods (except for the constructor) should be overridden in a subclass. In other words, *only* override the __init__( ) and run( ) methods of this class.

Program 1:

```
#thread creation method - 1

import threading

print(threading.current_thread( ))
print(threading.main_thread( ))
print(threading.current_thread( ).getName( ))

def thread_fun(str):
    for i in range(5):
        print(str)

    print(threading.current_thread( ))
    print(threading.current_thread( ).getName())
    threading.current_thread( ).setName("My First Thread")
    print(threading.current_thread().getName( ))

t = threading.Thread(target=thread_fun, args = ("Hello Subhash",))

t.start( )      #internally calls Thread.run( )
```

Output:

```
<_MainThread(MainThread, started
140735275934464)>
<_MainThread(MainThread, started
140735275934464)>
MainThread
Hello Subhash
Hello Subhash
Hello Subhash
Hello Subhash
Hello Subhash
<Thread(Thread-1, started
4447309824)>
Thread-1
My First Thread
```

Program 2:

```
#thread creation method - 2

import threading

class MyThreadClass(threading.Thread):
    def __init__(self,str):
        super( ).__init__( )
        self.str = str

    def run(self):
        print("Entering into:", end=' ')
        threading.current_thread( ).setName("My Second Thread")
        print(threading.current_thread().getName( ))
        self.my_thread( )

    def my_thread(self):
        for i in range(5):
            print(self.str)

print("Here is my ", threading.current_thread( ).getName( ))

thread_one = MyThreadClass("Hello Subhash")
thread_one.start( )
```

Output:

```
Here is my MainThread
Entering into: My Second Thread
Hello Subhash
Hello Subhash
Hello Subhash
Hello Subhash
Hello Subhash
```

Program 3:

```
#thread creation method - 3

import threading

#thread creation method - 3

class MyThreadClass:
    def __init__(self,str):
        self.str = str


    def my_thread(self, str):
        print("Entering into:", end=' ')
        threading.current_thread().setName("My Second Thread")
        print(threading.current_thread().getName())
        for i in range(5):
            print(self.str, ", ", str)

print("Here is my ", threading.current_thread().getName())
```

Output:

```
Here is my MainThread
Entering into: My Second Thread
Hello Subhash ,  How Are You?
Hello Subhash ,  How Are You?
Hello Subhash ,  How Are You?
Hello Subhash ,  How Are You?
Hello Subhash ,  How Are You?
```

```
thread_obj = MyThreadClass("Hello Subhash")

thread = threading.Thread(target=thread_obj.my_thread, args=("How Are You?",))
thread.start( )
```

Program 4:

```
import threading
#Creating multiple threads

class MyThreadClass:

    def my_thread(self, str):
        print("Entering into:", end=' ')
        threading.current_thread().setName(str[1])
        print(threading.current_thread().getName())
        for i in range(5):
            print(str[0])

print("Here is my ", threading.current_thread().getName())

thread_obj_one = MyThreadClass()
thread_obj_two = MyThreadClass()

thread_one = threading.Thread(target=thread_obj_one.my_thread, args=(["Hello Subhash, How Are You?","Second Thread"],))
thread_two = threading.Thread(target=thread_obj_two.my_thread, args=(["I Am Awesome, Thank You","Third Thread"],))

thread_one.start( )
thread_two.start( )
```

Output:

```
Here is my  MainThread
Entering into: Second Thread
Hello Subhash, How Are You?
Hello Subhash, How Are You?
Hello Subhash, How Are You?
Hello Subhash, How Are You?
Hello Subhash, How Are You?
Entering into: Third Thread
I Am Awesome, Thank You
I Am Awesome, Thank You
I Am Awesome, Thank You
I Am Awesome, Thank You
I Am Awesome, Thank You
```

Program 5:

```
import threading
import time
#Problem With Multithreading

class TakeMyPieceOfCake:

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

    def take_my_cake(self, piece_needed):

        threading.current_thread().setName(piece_needed[1])
        if self.piece_available >= piece_needed[0]:
            print( "{0} piece given to {1}".format(piece_needed[0],
threading.current_thread().getName()))
            time.sleep(3)
            self.piece_available = self.piece_available - piece_needed[0]
        else:
            print("No more cake pieces available")


print("Here is my ", threading.current_thread().getName())

give_to_person = TakeMyPieceOfCake(1)

thread_one = threading.Thread(target=give_to_person.take_my_cake, args=([1,"Second Thread"],))
thread_two = threading.Thread(target=give_to_person.take_my_cake, args=([1,"Third Thread"],))

thread_one.start( )
thread_two.start( )
```

Output:

```
Here is my  MainThread
1 piece given to Second Thread
1 piece given to Third Thread
```

<u>Program 6:</u>

```
import threading
import time
#Solving Multithreading With 'Lock'

class TakeMyPieceOfCake:

    def __init__(self,piece_available):
        self.piece_available = piece_available
        self.lock_it = threading.Lock( )

    def take_my_cake(self, piece_needed):
        self.lock_it.acquire( )
        threading.current_thread( ).setName(piece_needed[1])
        if self.piece_available >= piece_needed[0]:
            print( "{0} piece given to {1}".format(piece_needed[0], threading.current_thread(
).getName( )))
            time.sleep(3)
            self.piece_available = self.piece_available - piece_needed[0]
        else:
            print("No more cake pieces available")

        self.lock_it.release( )

print("Here is my ", threading.current_thread().getName())

give_to_person = TakeMyPieceOfCake(1)

thread_one = threading.Thread(target=give_to_person.take_my_cake, args=([1,"Second Thread"],))
thread_two = threading.Thread(target=give_to_person.take_my_cake, args=([1,"Third Thread"],))

thread_one.start( )
thread_two.start( )
```

Output:

```
Here is my  MainThread
1 piece given to Second Thread
No more cake pieces available
```

Program 7:

```
# Demonstrating ‘Deadlock’ problem

import threading

#Let us take two locks

lock_one = threading.Lock( )
lock_two = threading.Lock( )

def TakeBook( ):
    lock_one.acquire( )
    print("Locked Library Database")
    print("Checking for book availability")
    lock_two.acquire()
    print("Checking for number of books available")
    lock_two.release()
    lock_one.release()
    print("Book Issued")

def ReturnBook( ):
    lock_two.acquire()
    print("Locked Books Counter")
    print("Updating number of books available")
    lock_one.acquire()
    print("Checking for book availability")
    lock_one.release()
    lock_two.release()
    print("Book Returned")

#Creating two threads

person_taking_book = threading.Thread(target=TakeBook)
person_returning_book = threading.Thread(target=ReturnBook)
person_taking_book.start( )
person_returning_book.start( )
```

Output:

```
Locked Library Database
Checking for book availability
Locked Books Counter
Updating number of books
available

(Continues to wait - Deadlock)
```

Program 8:

```
# Demonstrating possible solution for ‘Deadlock’ problem

import threading

#Let us take two locks

lock_one = threading.Lock()
lock_two = threading.Lock()

def TakeBook():
    lock_one.acquire()
    print("Locked Library Database")
    print("Checking for book availability")
    lock_two.acquire()
    print("Checking for number of books available")
    lock_two.release()
    lock_one.release()
    print("Book Issued")

def ReturnBook():
    lock_one.acquire()
    print("Locked Books Counter")
    print("Updating number of books available")
    lock_two.acquire()
    print("Checking for book availability")
    lock_two.release()
    lock_one.release()
    print("Book Returned")

#Creating two threads

person_taking_book = threading.Thread(target=TakeBook)
person_returning_book = threading.Thread(target=ReturnBook)
person_taking_book.start()
person_returning_book.start()
```

Output:

```
Locked Library Database
Checking for book availability
Checking for number of books
available
Book Issued
Locked Books Counter
Updating number of books
available
Checking for book availability
Book Returned
```

Program 9:

Output:

```
#Producer Consumer - Bad Method

import threading
import time
class Producer:

    def __init__(self):
        self.l = [ ]
        self.production_done = False
        self.i = 0

    def produce(self):
        while True:
            if self.production_done == False:

             for self.i in range(100):
                    self.l.append(self.i)
                    print("Another item produced...")
                    self.production_done = True
             if( self.i == 100 ):
                 break;

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

    def consume(self):
        while True:
            if self.producer_handle.production_done == True:
                print("Item Consumed")
                print(self.producer_handle.l)
                self.producer_handle.production_done = False;
            if(self.producer_handle.i == 100):
                break;

producer_obj = Producer()
consumer_obj = Consumer(producer_obj)

tone = threading.Thread(target=producer_obj.produce)
ttwo = threading.Thread(target=consumer_obj.consume)

tone.start( )
ttwo.start( )
```

```
Another item produced...
Item Consumed
[0]
Another item produced...
Item Consumed
[0, 54]
Another item produced...
Item Consumed
[0, 54, 48]
Another item produced...
Item Consumed
[0, 54, 48, 54]
Another item produced...
Item Consumed
[0, 54, 48, 54, 5]
Another item produced...
Item Consumed
[0, 54, 48, 54, 5, 85]
Another item produced...
Item Consumed
[0, 54, 48, 54, 5, 85, 66]
Another item produced...
Item Consumed
[0, 54, 48, 54, 5, 85, 66, 71]
```

<u>Program 10:</u>

#Producer Consumer - Better Than Bad Method - Yet, Bad Method

```
import threading
import time
class Producer:

    def __init__(self):
        self.l = [ ]
        self.production_done = False
        self.i = 0

    def produce(self):
        for self.i in range(11):
            time.sleep(1)
            if self.production_done == False:
                self.l.append(self.i)
                print("Another item produced...")
                self.production_done = True
            if(self.i == 10):
                time.sleep(1)
                self.i = 11

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

    def consume(self):
        while self.producer_handle.i <= 10:
            if self.producer_handle.production_done == True:
                print("Item Consumed")
                print(self.producer_handle.l)
                self.producer_handle.production_done = False;
                time.sleep(1)

producer_obj = Producer()
consumer_obj = Consumer(producer_obj)

tone = threading.Thread(target=producer_obj.produce)
ttwo = threading.Thread(target=consumer_obj.consume)

tone.start()
ttwo.start()
```

Output:

```
Another item produced...
Item Consumed
[0]
Another item produced...
Item Consumed
[0, 1]
Another item produced...
Item Consumed
[0, 1, 2]
Another item produced...
Item Consumed
[0, 1, 2, 3]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7, 8]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

<u>Program 11:</u>

```
#Producer Consumer - Good Method, But, Not Great Method

import threading
import time
class Producer:

    def __init__(self):
        self.l = [ ]
        self.condition_variable = threading.Condition( )

    def produce(self):

        for i in range(11):
            self.condition_variable.acquire( )
            self.l.append(i)
            print("Another item produced...")
            self.condition_variable.notify()
            self.condition_variable.release()
            time.sleep(2)


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

    def consume(self):

        for i in range(11):
                time.sleep(1)
                self.producer_handle.condition_variable.acquire( )
                self.producer_handle.condition_variable.wait(timeout=0)
                print("Item Consumed")
                print(self.producer_handle.l)
                self.producer_handle.condition_variable.release( )
                time.sleep(1)

producer_obj = Producer()
consumer_obj = Consumer(producer_obj)

tone = threading.Thread(target=producer_obj.produce)
ttwo = threading.Thread(target=consumer_obj.consume)

tone.start()
ttwo.start()
```

Output:

```
Another item produced...
Item Consumed
[0]
Another item produced...
Item Consumed
[0, 1]
Another item produced...
Item Consumed
[0, 1, 2]
Another item produced...
Item Consumed
[0, 1, 2, 3]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7, 8]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Another item produced...
Item Consumed
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Program 12:

Output:

```
#Producer Consumer - Great Method, Can Be Improved
import threading
import time
import queue

class Producer:

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

    def produce(self):

        for i in range(11):
            self.q.put(i)
            print("Another item produced...")
            time.sleep(1)


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

    def consume(self):

        for i in range(11):
            print("Item Consumed: ", self.producer_handle.q.get(i))

producer_obj = Producer( )
consumer_obj = Consumer(producer_obj)

tone = threading.Thread(target=producer_obj.produce)
ttwo = threading.Thread(target=consumer_obj.consume)

tone.start( )
ttwo.start( )
```

```
Another item produced...
Item Consumed:  0
Another item produced...
Item Consumed:  1
Another item produced...
Item Consumed:  2
Another item produced...
Item Consumed:  3
Another item produced...
Item Consumed:  4
Another item produced...
Item Consumed:  5
Another item produced...
Item Consumed:  6
Another item produced...
Item Consumed:  7
Another item produced...
Item Consumed:  8
Another item produced...
Item Consumed:  9
Another item produced...
Item Consumed:  10
```