Threading

In [1]:
```
import time

def counting():
    for i in range(1, 10):
        time.sleep(1)
        print(i)
```

In [2]:
```
def alphabets():
    for i in range(ord('A'), ord('Z')+1):
        time.sleep(0.5)
        print(chr(i))
```

In [ ]:
```
# function call
counting()
alphabets()
```

```
1
2
3
4
5
6
7
8
9
A
B
C
```

In [ ]:
```
import threading
def counting():
    for i in range(1, 11):
        time.sleep(2)
        print('\t', i)
def alphabets():
    for i in range(ord('A'), ord('Z')+1):
        time.sleep(1)
        print(chr(i))
```

create new threads

In [ ]:
```
t1 = threading.Thread(target=counting)
t2 = threading.Thread(target=alphabets)
```

start the thread

In [ ]:
```
t1.start()
t2.start()
```

using a derived class of thread

In [ ]:
```
class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print ("Starting " + self.name)
        print_time(self.name, 5, self.counter)
        print ("Exiting " + self.name)

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print ("{}: {}".format(threadName, time.ctime(time.time())))
        counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 3)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print ("Exiting Main Thread")
```

In [ ]:
```
class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print ("Starting " + self.name)
        print_time(self.name, 5, self.counter)
        print ("Exiting " + self.name)

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print ("{}: {}".format(threadName, time.ctime(time.time())))
        counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 3)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()
print ("Exiting Main Thread")
```

Synchronization

In [1]:
```
import threading
import time
```

In [2]:
```
balance = 200

print('initial balance value : ', balance)

class myThread(threading.Thread):
    def __init__(self, name, target ):
        threading.Thread.__init__(self)
        self.name = name
        self.target = target

    def run(self):
        print('\nStarting Thread', self.name)
        # function call
        self.target()

def foo1():
    print("\n foo 1 called \n")
    time.sleep(3)
    final_balance = balance * 2
    print('\nFinal Balance', final_balance)

def foo2():
    print('\n foo 2 called \n')
    global balance
    balance /= 2
    print('\nvalue of balance updated ', balance)

thread1 = myThread(name = 1, target= foo1)
thread2 = myThread(name = 2, target= foo2)

thread1.start()
thread2.start()
```

```
initial balance value :  200

Starting Thread 1

 foo 1 called


Starting Thread 2

 foo 2 called


value of balance updated  100.0

Final Balance 200.0
```

In [4]:
```
balance = 200

print('initial balance value : ', balance)

class myThread(threading.Thread):
    def __init__(self, name, target ):
        threading.Thread.__init__(self)
        self.name = name
        self.target = target

    def run(self):
        print('\n\nStarting Thread', self.name)
        # acquire
        threadLock.acquire()
        print('\nLock acquired for thread :', self.name)

        # function call
        self.target()

        # free lock
        threadLock.release()
        print('\nLock released for thread :', self.name)


def foo1():
    print("\n foo 1 called \n")
    time.sleep(3)
    final_balance = balance * 2
    print('\nFinal Balance', final_balance)

def foo2():
    print("\n foo 2 called \n")
    global balance
    balance /= 2
    print('\nvalue of balance updated ', balance)

# creating a lock object
threadLock = threading.Lock()


thread1 = myThread(name = 1, target= foo1)
thread2 = myThread(name = 2, target= foo2)

thread1.start()
thread2.start()
thread1.join()
thread2.join()
```

```
initial balance value :  200


Starting Thread

Starting Thread  1

Lock acquired for thread : 1

 foo 1 called

2

Final Balance 400

Lock released for thread : 1

Lock acquired for thread : 2

 foo 2 called


value of balance updated  100.0

Lock released for thread : 2
```

In [ ]:
```

```