Producer Consumer Problem:

The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time.

```
from threading import Thread

class ProducerThread(Thread):
    def run(self):
        pass

class ConsumerThread(Thread):
    def run(self):
        pass
```

The problem describes two processes, the producer and the consumer, who share a common buffer.

So we keep one variable which will be global and will be modified by both Producer and Consumer threads. Producer produces data and adds it to the buffer. Consumer consumes data from the buffer i.e removes it from the buffer.

```
from threading import Thread, Lock
import time
import random

buffer = []
lock = Lock()

class ProducerThread(Thread):
    def run(self):
        nums = range(5) #Will create the list [0, 1, 2, 3, 4]
        global buffer
        while True:
            num = random.choice(nums) #Selects a random number from list [0, 1,
2, 3, 4]
            lock.acquire()
            buffer.append(num)
            print("Produced", num)
            lock.release()
            time.sleep(random.random())


class ConsumerThread(Thread):
    def run(self):
        global buffer
        while True:
            lock.acquire()
            if not buffer:
                print ("Nothing in buffer, but consumer will try to consume")
            num = buffer.pop(0)
            print ("Consumed", num)
            lock.release()
            time.sleep(random.random())


ProducerThread().start()
ConsumerThread().start()
```

Explanation:

- We started one producer thread(hereafter referred as producer) and one consumer thread(hereafter referred as consumer).
- Producer keeps on adding to the buffer and consumer keeps on removing from the buffer.
- Since buffer is a shared variable, we keep it inside lock to avoid race condition.
- At some point, consumer has consumed everything and producer is still sleeping. Consumer tries to consume more but since buffer is empty, an **IndexError** is raised.
- But on every execution, before IndexError is raised you will see the print statement telling "Nothing in buffer, but consumer will try to consume", which explains why you are getting the error.

We found this implementaion as the wrong behaviour.

####What is the correct behaviour?

When there was nothing in the buffer, consumer should have stopped running and waited instead of trying to consume from the buffer. And once producer adds something to the buffer, there should be a way for it to notify the consumer telling it has added something to buffer. So, consumer can again consume from the buffer. And thus IndexError will never be raised.

About Condition

- Condition object allows one or more threads to wait until notified by another thread. Taken from here.

And this is exactly what we want. We want consumer to wait when the buffer is empty and resume only when it gets notified by the producer. Producer should notify only after it adds something to the buffer. So after notification from producer, we can be sure that buffer is not empty and hence no error can crop if consumer consumes.

- Condition is always associated with a lock.
- A condition has acquire() and release() methods that call the corresponding methods of the associated lock.

Condition provides acquire() and release() which calls lock's acquire() and release() internally, and so we can replace lock instances with condition instances and our lock behaviour will keep working properly.

Consumer needs to wait using a condition instance and producer needs to notify the consumer using the condition instance too. So, they must use the same condition instance for the wait and notify functionality to work properly.

```
from threading import Condition,Thread,Lock
import random
import time
buffer = []
lock = Lock()
condition = Condition()

class ConsumerThread(Thread):
    def run(self):
```

```
        global buffer
        while True:
            condition.acquire()
            if not buffer:
                print ("Nothing in buffer, consumer is waiting")
                condition.wait()
                print ("Producer added something to buffer and notified the
consumer")
            num = buffer.pop(0)
            print ("Consumed", num)
            condition.release()
            time.sleep(random.random())

class ProducerThread(Thread):
    def run(self):
        nums = range(5)
        global buffer
        while True:
            condition.acquire()
            num = random.choice(nums)
            buffer.append(num)
            print ("Produced", num)
            condition.notify()
            condition.release()
            time.sleep(random.random())

ProducerThread().start()
ConsumerThread().start()
```

Explanation:

- For consumer, we check if the buffer is empty before consuming.
- If yes then call **wait()** on condition instance.
- wait() blocks the consumer and also releases the lock associated with the condition. This lock was held by consumer, so basically consumer loses hold of the lock.
- Now unless consumer is notified, it will not run.
- Producer can acquire the lock because lock was released by consumer.
- Producer puts data in buffer and calls notify() on the condition instance.
- Once notify() call is made on condition, consumer wakes up. But waking up doesn’t mean it starts executing.
- notify() does not release the lock. Even after notify(), lock is still held by producer.
- Producer explicitly releases the lock by using condition.release().
- And consumer starts running again. Now it will find data in buffer and no IndexError will be raised.