CES 22 – Aula 7

Exceções, Threads

Objetivos

- Asserções e Exceções
- Threads

Tratamento de erros

- Python possui dois mecanismos para tratamento de erros e depuração de programas.
  - Asserções
  - Exceções

Asserções

- Asserções são verificadores de sanidade (sanity-check) que podem ser desligados após a execução de testes.
- São utilizados para verificar se o resultado de uma expressão ou função é correta ou se argumentos de entrada são válidos.

Sintaxe

Assert Expression[, Arguments]

```
def KelvinToFahrenheit(Temperature):
    assert (Temperature >= 0),"Colder than absolute zero!"
    return ((Temperature-273)*1.8)+32
print (KelvinToFahrenheit(273))
print (int(KelvinToFahrenheit(505.78)))
print (KelvinToFahrenheit(-5))
```

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in <module>
print KelvinToFahrenheit(-5) File "test.py", line 4, in KelvinToFahrenheit
assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

Exceção

- Uma exceção é um evento que ocorre durante a execução de um programa e que interrompe o fluxo normal das instruções. Geralmente ocorre quando é encontrado uma situação em que o interpretador não sabe como lidar.
- A exceção deve ser tratada imediatamente ou então o programa é terminado.

Aplicações

- Tratamento de erros.
- Notificação de eventos.
- Tratamento de casos especiais.
- Operações de finalização.
- Fluxos incomuns.

Exceção é uma subclasse de BaseException

- De BaseException são derivadas as classes
  - Exception – classe base para exceções definidas pelo usuário.
  - ArithmeticError – classe base para erros aritméticos.
  - BufferError – para erros relacionados a operações com buffers.
  - LookupError – para erros em operações com índices ou chaves.
- Exceções concretas:
  - AssertionError, AttributeError, EOFError, ImportError, IndexError, MemoryError, OSError, ZeroDivisionError, ...

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
.............
```

Tratando exceções

- Situações que podem causar exceções são protegidos por blocos **try:**

```
try:
    You do your operations here
     ......................
except ExceptionI:
    If there is ExceptionI, then execute this
block.
except ExceptionII:
    If there is ExceptionII, then execute this
block.  ......................
else:
    If there is no exception then execute this
block.
```

Exemplo

```
try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception
   handling!!")
except  IOError:
    print ("Error: can\'t find file or read data")
else:
    print ("Written content in the file successfully")
    fh.close()
```

try - except

try:

You do your operations here

.....

except:

If there is any exception, then execute this block.

.....

else:

If there is no exception then execute this block.

try-finally

try:

    You do your operations here;

    ……..

    Due to any exception, this may be skipped.

finally: This would always be executed. ……..

```
try:
    block-1 ...
except Exception1:
    handler-1 ...
except Exception2:
    handler-2 ...
except:
    handler-3 ...
else:
    else-block
finally:
    final-block
```

Argumento para exceções

```
# Define a function here.
def temp_convert(var):
    try:
        return int(var)
    except ValueError as Argument:
        print ("The argument does not contain
   numbers\n", Argument)
# Call above function here.
temp_convert("xyz")
```

```
The argument does not contain numbers
 invalid literal for int() with base 10: 'xyz'
```

Ativando exceções

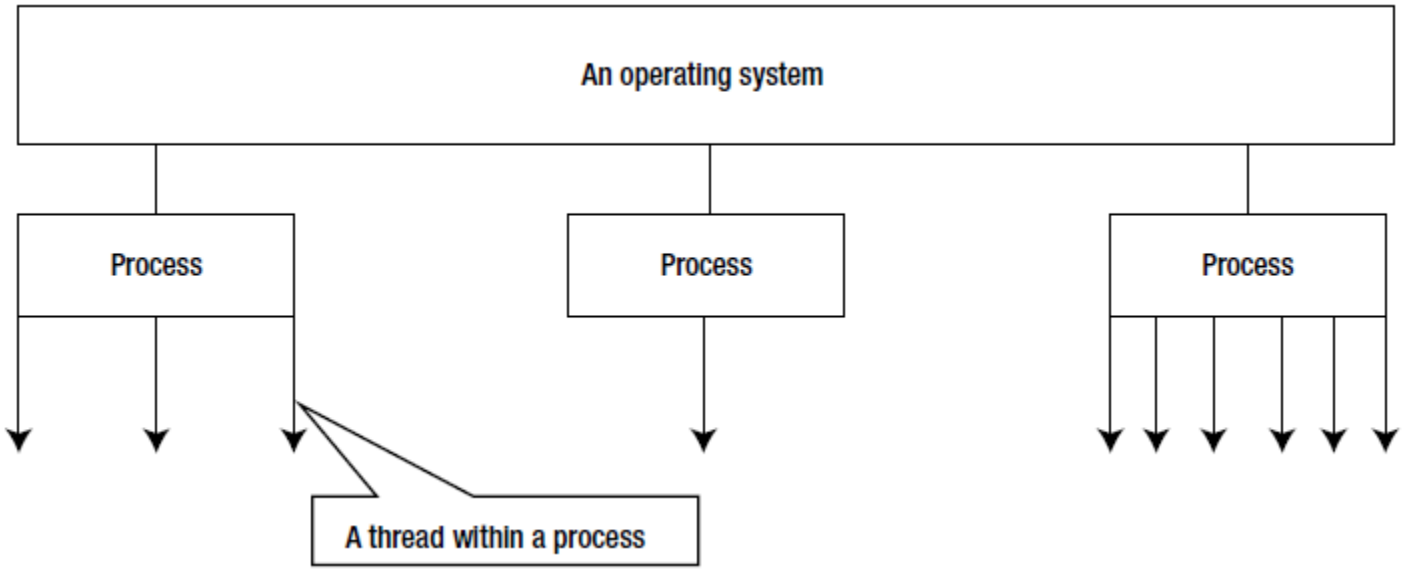
```
def functionName( level ):
    if level <1:
        raise Exception(level)
        # The code below to this would not be executed
        # if we raise the exception
    return level
try:
     l = functionName(-10)
     print ("level = ",l)
except Exception as e:
     print ("error in level argument",e.args[0])

========== RESTART: ==========
error in level argument -10
```

Exceções definidas pelo programador

```
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg


try:
    raise Networkerror("Bad hostname")
except Networkerror,e:
    print e.args
```

Threads

Threads


An operating system
Process
Process
Process
A thread within a process

Threads em Python

- Na implementação Cpython as threads são implementadas pela máquina virtual do Python e são desvinculadas do Sistema Operacional.
- As threads são fluxos de byte codes Python.
- Controle GLI (Global Interpreter Lock) impede que duas threads estejam ativas simultaneamente.

2 módulos para threads

- _thread (módulo básico)
- threading (recomendado)

```
#!/usr/bin/python3

import _thread
import time

# Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print ("%s: %s" % ( threadName,
time.ctime(time.time()) ))

# Create two threads as follows
try:
   _thread.start_new_thread( print_time, ("Thread-1", 2, ) )
   _thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
   print ("Error: unable to start thread")

while 1:
   pass
```

Criação de Threads usando o módulo threading

- Definir uma subclasse para a classe Thread.
- Sobrepor o método __init__ acrescentando argumentos adicionais.
- Sobrepor o método run com as instruções a serem executadas pela thread.

```
import threading
import time
exitFlag = 0
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, self.counter, 5)
      print ("Exiting " + self.name)
def print_time(threadName, delay, counter):
   while counter:
      if exitFlag:
         threadName.exit()
      time.sleep(delay)
      print ("%s: %s" % (threadName, time.ctime(time.time())))
      counter -= 1
```

```
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print ("Exiting Main Thread")
```

```
Starting Thread-1
Starting Thread-2
Thread-1: Fri Feb 19 10:00:21 2016
Thread-2: Fri Feb 19 10:00:22 2016
Thread-1: Fri Feb 19 10:00:22 2016
Thread-1: Fri Feb 19 10:00:23 2016
Thread-2: Fri Feb 19 10:00:24 2016
Thread-1: Fri Feb 19 10:00:24 2016
Thread-1: Fri Feb 19 10:00:25 2016
Exiting Thread-1
Thread-2: Fri Feb 19 10:00:26 2016
Thread-2: Fri Feb 19 10:00:28 2016
Thread-2: Fri Feb 19 10:00:30 2016
Exiting Thread-2
Exiting Main Thread
```

Sincronização

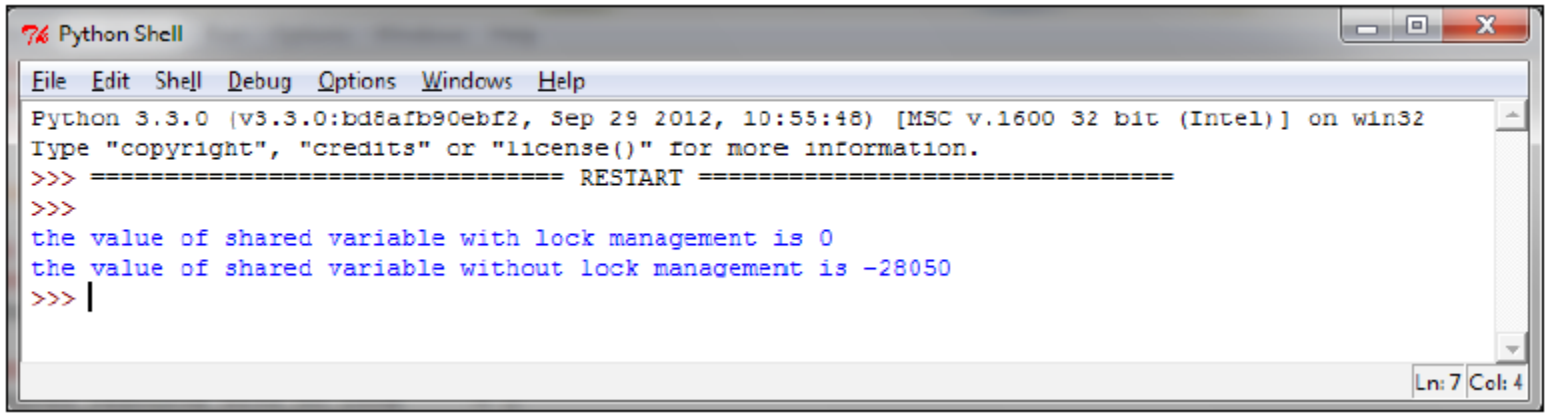
- Com recursos compartilhados por várias threads, aparece o problema de disputa por recursos (race condition).
- Por exemplo, acessar um contador requer verificar o valor e depois incrementar. Se o contador for compartilhado, uma thread pode acessar um valor que ainda não foi incrementado.

Solução

- Objeto lock
  - Operações acquire e release.
  - Acquire: permite acesso ou bloqueia a thread.
  - Release: libera acesso e desbloqueia thread.

```
import threading

shared_resource_with_lock     = 0
shared_resource_with_no_lock     = 0
COUNT = 100000
shared_resource_lock = threading.Lock()


####LOCK MANAGEMENT##
def increment_with_lock():
    global shared_resource_with_lock
    for i in range(COUNT):

        shared_resource_lock.acquire()
        shared_resource_with_lock += 1
        shared_resource_lock.release()

def decrement_with_lock():
    global shared_resource_with_lock
    for i in range(COUNT):
        shared_resource_lock.acquire()
        shared_resource_with_lock -= 1
        shared_resource_lock.release()

####NO LOCK MANAGEMENT ##
def increment_without_lock():
    global shared_resource_with_no_lock
    for i in range(COUNT):
        shared_resource_with_no_lock += 1

def decrement_without_lock():
    global shared_resource_with_no_lock
    for i in range(COUNT):
        shared_resource_with_no_lock -= 1
```

```
####the Main program
if __name__ == "__main__":
    t1 = threading.Thread(target = increment_with_lock)
    t2 = threading.Thread(target = decrement_with_lock)
    t3 = threading.Thread(target = increment_without_lock)
    t4 = threading.Thread(target = decrement_without_lock)
    t1.start()
    t2.start()
    t3.start()
    t4.start()
    t1.join()
    t2.join()
    t3.join()
    t4.join()
    print ("the value of shared variable with lock management is %s"\
           %shared_resource_with_lock)
    print ("the value of shared variable with race condition is %s"\
           %shared_resource_with_no_lock)
```

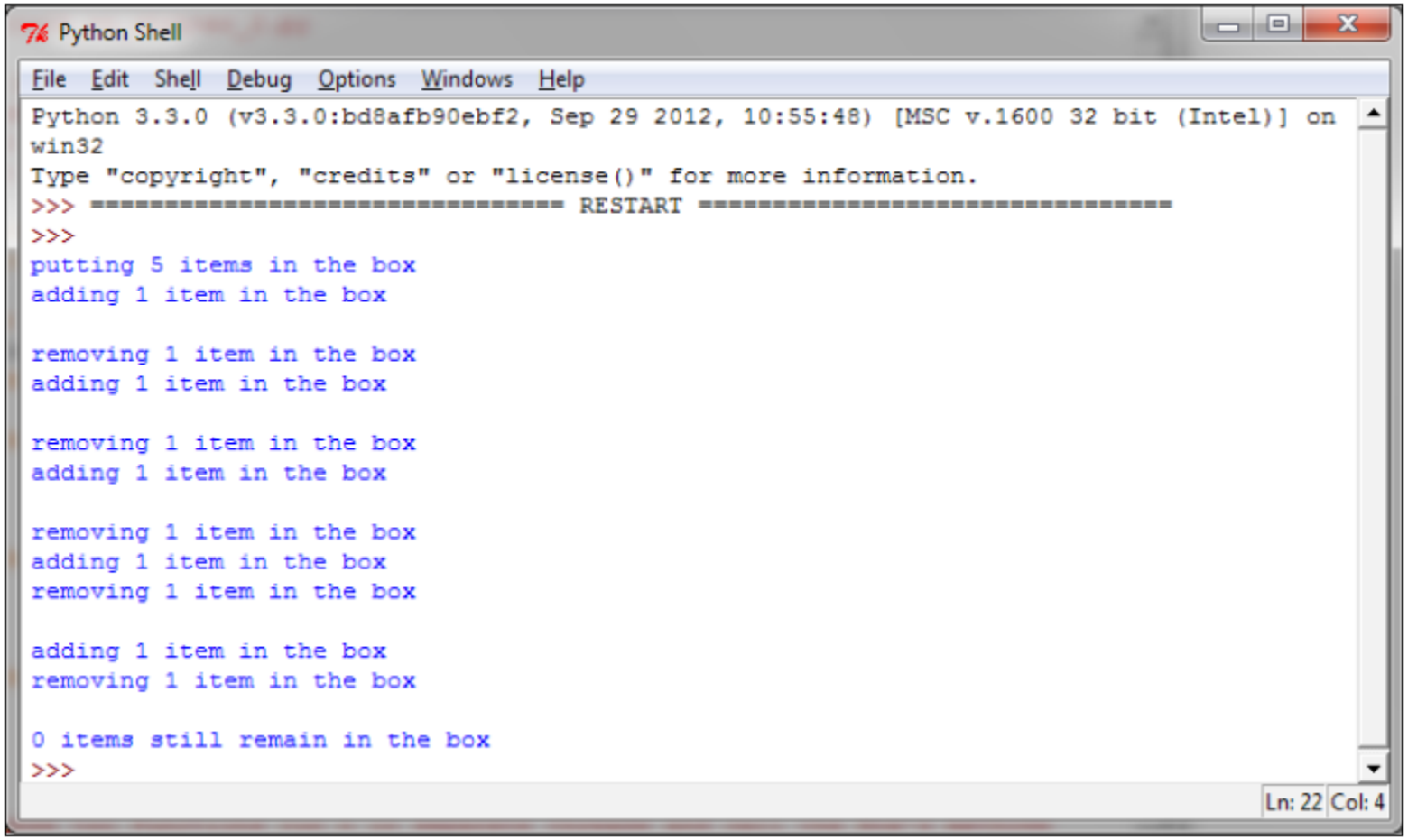

Python Shell
File Edit Shell Debug Options Windows Help
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:55:48) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
the value of shared variable with lock management is 0
the value of shared variable without lock management is -28050
>>>
Ln: 7 Col: 4

Sincronização com Rlock (recursive Lock)

```
import threading
import time

class Box(object):
    lock = threading.RLock()
    def __init__(self):
        self.total_items = 0
    def execute(self,n):
        Box.lock.acquire()
        self.total_items += n
        Box.lock.release()
    def add(self):
        Box.lock.acquire()
        self.execute(1)
        Box.lock.release()
    def remove(self):
        Box.lock.acquire()
        self.execute(-1)
        Box.lock.release()
```

```
## These two functions run n in separate
## threads and call the Box's methods

def adder(box,items):
    while items > 0:
        print ("adding 1 item in the box\n")
        box.add()
        time.sleep(5)
        items -= 1

def remover(box,items):
    while items > 0:
        print ("removing 1 item in the box")
        box.remove()
        time.sleep(5)
        items -= 1
```

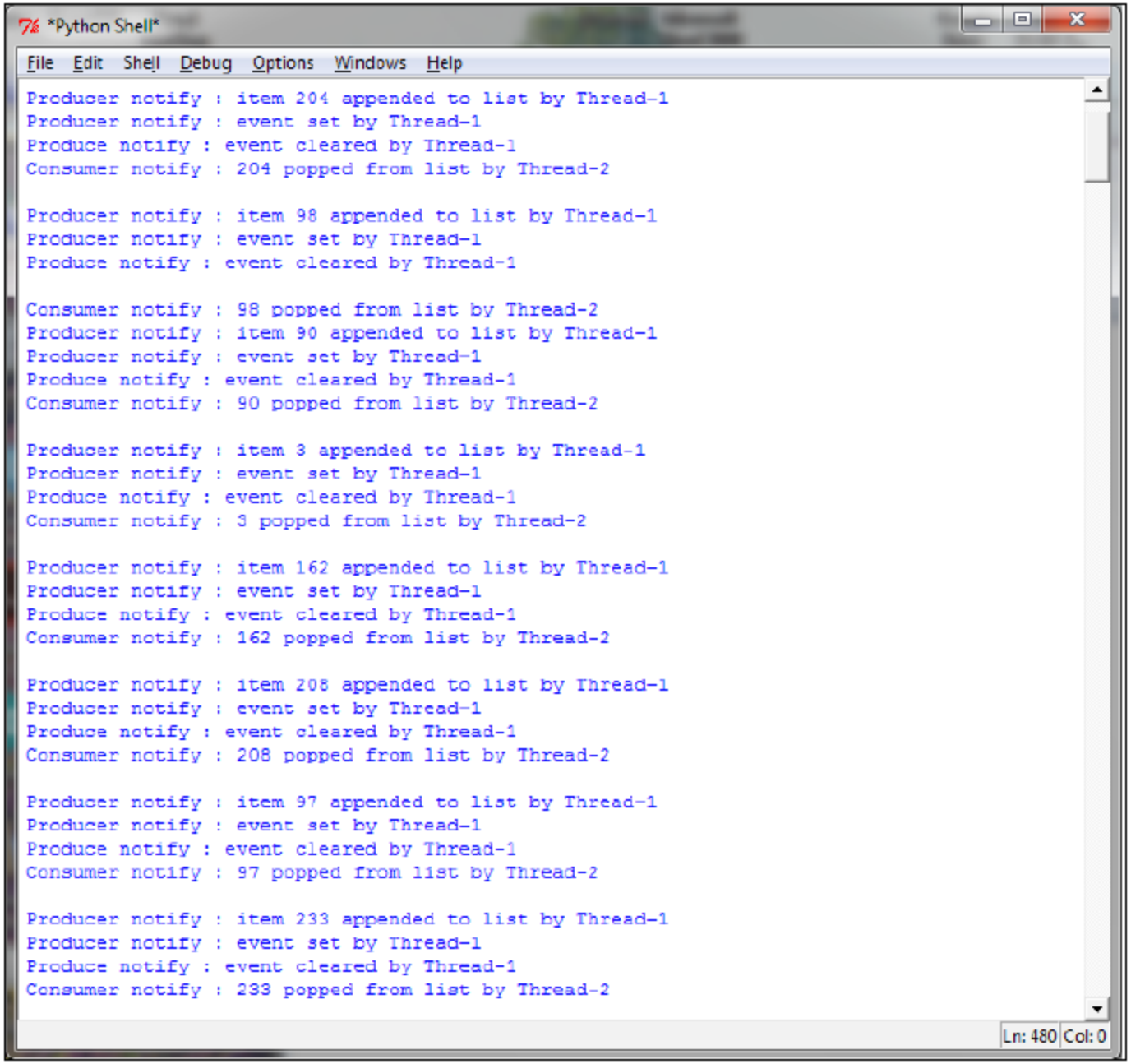
```
## the main program build some
## threads and make sure it works
if __name__ == "__main__":
    items = 5
    print ("putting %s items in the box " % items)
    box = Box()
    t1 = threading.Thread(target=adder,args=(box,items))
    t2 = threading.Thread(target=remover,args=(box,items))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
    print ("%s items still remain in the box " % box.total_items)
```

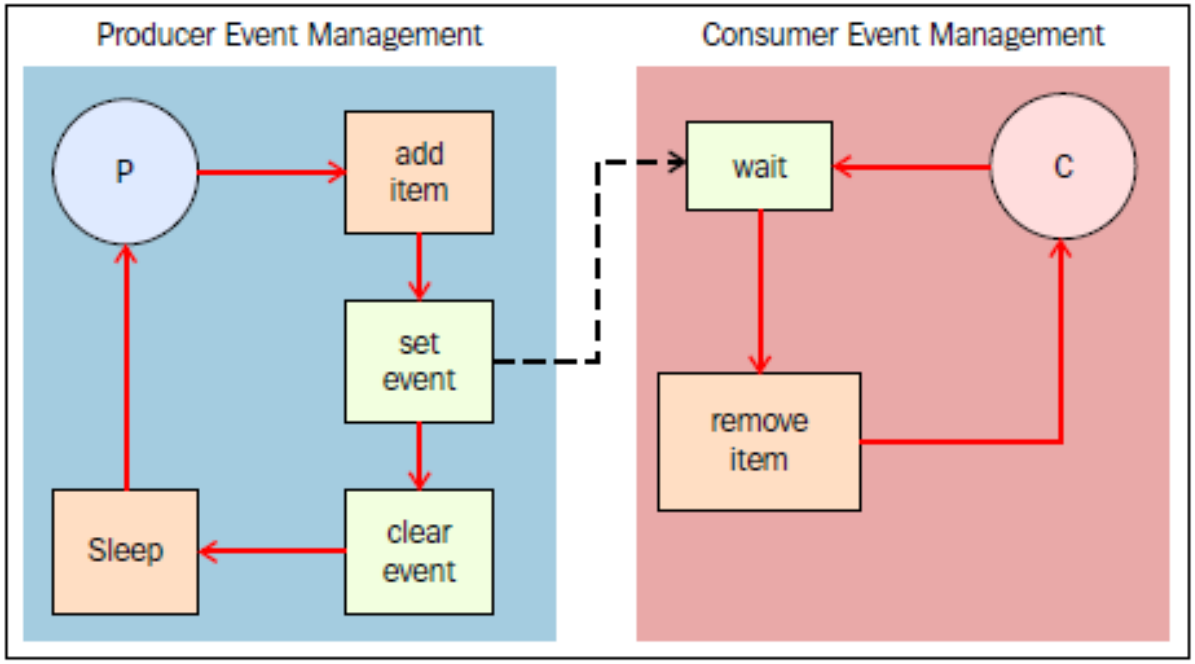

Python Shell
File Edit Shell Debug Options Windows Help
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:55:48) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
putting 5 items in the box
adding 1 item in the box
removing 1 item in the box
adding 1 item in the box
removing 1 item in the box
adding 1 item in the box
removing 1 item in the box
adding 1 item in the box
removing 1 item in the box
adding 1 item in the box
removing 1 item in the box
0 items still remain in the box
>>>
Ln: 22 Col: 4

Sincronização com Eventos

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

items = []
event = Event()

class consumer(Thread):
    def __init__(self, items, event):
        Thread.__init__(self)
        self.items = items
        self.event = event

    def run(self):
        while True:
            time.sleep(2)
            self.event.wait()
            item = self.items.pop()
            print ('Consumer notify : %d popped from list by %s'\
                   %(item, self.name))
```

```
class producer(Thread):
    def __init__(self, integers, event):
        Thread.__init__(self)
        self.items = items
        self.event = event

    def run(self):
        global item
        for i in range(100):
            time.sleep(2)
            item = random.randint(0, 256)
            self.items.append(item)
            print ('Producer notify : item N° %d appended \
                   to list by %s'\
                   % (item, self.name))
            print ('Producer notify : event set by %s'\
                   % self.name)
           self.event.set()
           print ('Produce notify : event cleared by %s \n'\
                  % self.name)
           self.event.clear()


if __name__ == '__main__':
    t1 = producer(items, event)
    t2 = consumer(items, event)
    t1.start()
    t2.start()
    t1.join()
        t2.join()
```


Python Shell
File Edit Shell Debug Options Windows Help
Producer notify : item 204 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 204 popped from list by Thread-2
Producer notify : item 98 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 98 popped from list by Thread-2
Producer notify : item 90 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 90 popped from list by Thread-2
Producer notify : item 3 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 3 popped from list by Thread-2
Producer notify : item 162 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 162 popped from list by Thread-2
Producer notify : item 208 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 208 popped from list by Thread-2
Producer notify : item 97 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 97 popped from list by Thread-2
Producer notify : item 233 appended to list by Thread-1
Producer notify : event set by Thread-1
Produce notify : event cleared by Thread-1
Consumer notify : 233 popped from list by Thread-2
Ln: 480 Col: 0

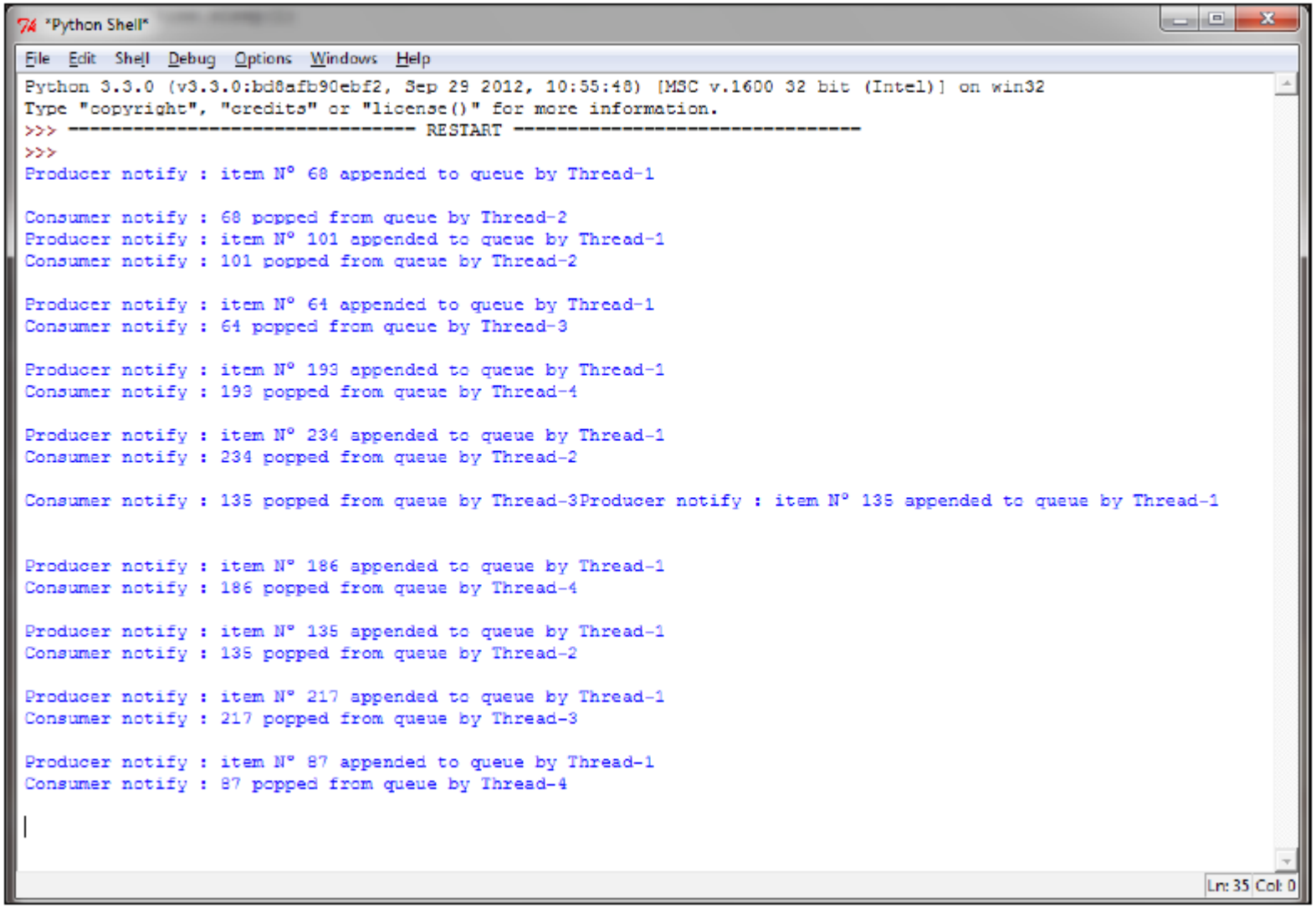


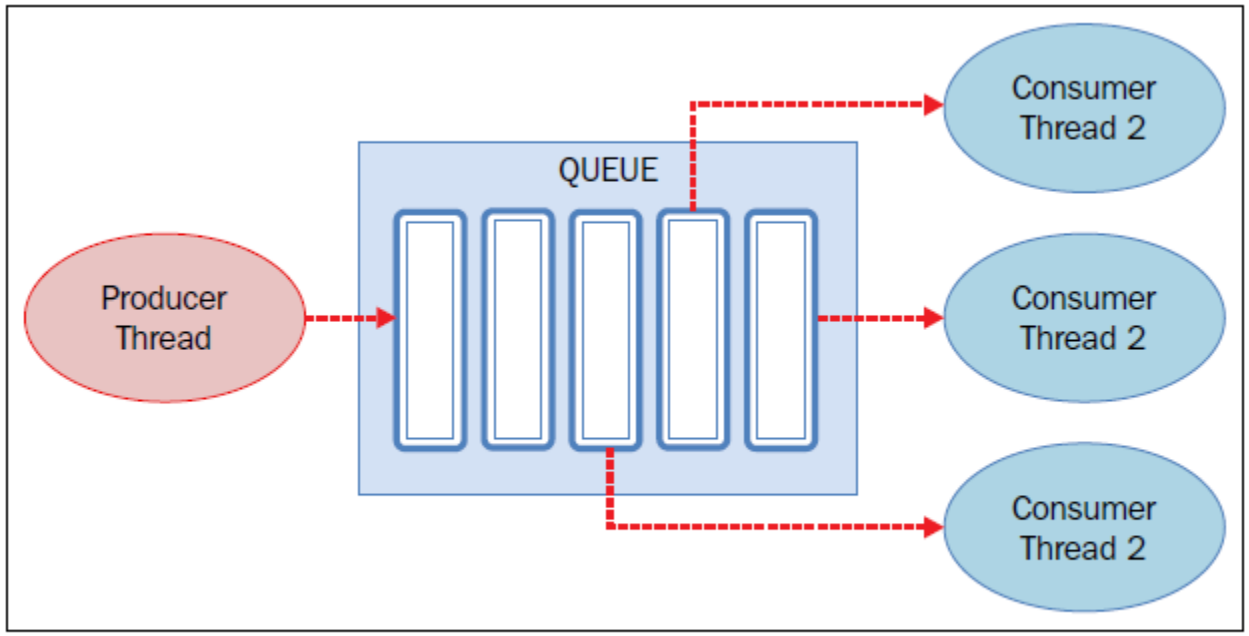
Thread synchronization with event objects

Comunicação usando uma fila (queue)

```
from threading import Thread, Event
from queue import Queue
import time
import random


class producer(Thread):
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue

    def run(self) :
        for i in range(10):
            item = random.randint(0, 256)
            self.queue.put(item)
            print ('Producer notify: item N°%d appended to queue by %s
                   \n'\
                   % (item, self.name))
            time.sleep(1)
```

```
class consumer(Thread):
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue

    def run(self):
        while True:
            item = self.queue.get()
            print ('Consumer notify : %d popped from queue by %s'\
                   % (item, self.name))
            self.queue.task_done()


if __name__ == '__main__':
        queue = Queue()
        t1 = producer(queue)
        t2 = consumer(queue)
        t3 = consumer(queue)
        t4 = consumer(queue)
        t1.start()
        t2.start()
        t3.start()
        t4.start()
        t1.join()
        t2.join()
        t3.join()
        t4.join()
```


Python Shell
File Edit Shell Debug Options Windows Help
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:55:48) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Producer notify : item N° 68 appended to queue by Thread-1
Consumer notify : 68 popped from queue by Thread-2
Producer notify : item N° 101 appended to queue by Thread-1
Consumer notify : 101 popped from queue by Thread-2
Producer notify : item N° 64 appended to queue by Thread-1
Consumer notify : 64 popped from queue by Thread-3
Producer notify : item N° 193 appended to queue by Thread-1
Consumer notify : 193 popped from queue by Thread-4
Producer notify : item N° 234 appended to queue by Thread-1
Consumer notify : 234 popped from queue by Thread-2
Consumer notify : 135 popped from queue by Thread-3Producer notify : item N° 135 appended to queue by Thread-1
Producer notify : item N° 186 appended to queue by Thread-1
Consumer notify : 186 popped from queue by Thread-4
Producer notify : item N° 135 appended to queue by Thread-1
Consumer notify : 135 popped from queue by Thread-2
Producer notify : item N° 217 appended to queue by Thread-1
Consumer notify : 217 popped from queue by Thread-3
Producer notify : item N° 87 appended to queue by Thread-1
Consumer notify : 87 popped from queue by Thread-4
Ln: 35 Col: 0



Thread synchronization with the queue module

Exercício

- Implemente o problema do produtor consumidor usando Rlock ao invés de uma fila.