The Queue module

This module provides a thread-safe queue implementation. It provides a convenient way of moving Python objects between different threads.

Example: Using the Queue module

```
# File: queue-example-1.py

import threading
import Queue
import time, random

WORKERS = 2

class Worker(threading.Thread):

    def __init__(self, queue):
        self.__queue = queue
        threading.Thread.__init__(self)

    def run(self):
        while 1:
            item = self.__queue.get()
            if item is None:
                break # reached end of queue

            # pretend we're doing something that takes 10-100 ms
            time.sleep(random.randint(10, 100) / 1000.0)

            print "task", item, "finished"

#
# try it

queue = Queue.Queue(0)

for i in range(WORKERS):
    Worker(queue).start() # start a worker

for i in range(10):
    queue.put(i)

for i in range(WORKERS):
    queue.put(None) # add end-of-queue markers


task 1 finished
task 0 finished
task 3 finished
task 2 finished
task 4 finished
task 5 finished
task 7 finished
task 6 finished
task 9 finished
task 8 finished
```

You can limit the size of the queue. If the producer threads fill the queue, they will block until items are popped off the queue.

Example: Using the Queue module with a maximum size

```
# File: queue-example-2.py

import threading
import Queue
```

```
import time, random

WORKERS = 2

class Worker(threading.Thread):

    def __init__(self, queue):
        self.__queue = queue
        threading.Thread.__init__(self)

    def run(self):
        while 1:
            item = self.__queue.get()
            if item is None:
                break # reached end of queue

            # pretend we're doing something that takes 10-100 ms
            time.sleep(random.randint(10, 100) / 1000.0)

            print "task", item, "finished"

#
# run with limited queue

queue = Queue.Queue(3)

for i in range(WORKERS):
    Worker(queue).start() # start a worker

for item in range(10):
    print "push", item
    queue.put(item)

for i in range(WORKERS):
    queue.put(None) # add end-of-queue markers
```

```
push 0
push 1
push 2
push 3
push 4
push 5
task 0 finished
push 6
task 1 finished
push 7
task 2 finished
push 8
task 3 finished
push 9
task 4 finished
task 6 finished
task 5 finished
task 7 finished
task 9 finished
task 8 finished
```

You can modify the behavior through subclassing. The following class provides a simple priority queue. It expects all items added to the queue to be tuples, where the first member contains the priority (lower value means higher priority):

Note: Python 2.4 does not use a mutable sequence for the internal queue, so the following example no longer works. An updated version will be posted at a later time.

Example: Using the Queue module to implement a priority queue

```
# File: queue-example-3.py

import Queue
import bisect
```

```
Empty = Queue.Empty

class PriorityQueue(Queue.Queue):
    "Thread-safe priority queue"

    def _put(self, item):
        # insert in order
        bisect.insort(self.queue, item)

#
# try it

queue = PriorityQueue(0)

# add items out of order
queue.put((20, "second"))
queue.put((10, "first"))
queue.put((30, "third"))

# print queue contents
try:
    while 1:
        print queue.get_nowait()
except Empty:
    pass
```

```
third
second
first
```

And here's a simple stack implementation (last-in first-out, instead of first-in, first-out):

Example: Using the Queue module to implement a stack

```
# File: queue-example-4.py

import Queue

Empty = Queue.Empty

class Stack(Queue.Queue):
    "Thread-safe stack"

    def _put(self, item):
        # insert at the beginning of queue, not at the end
        self.queue.insert(0, item)

    # method aliases
    push = Queue.Queue.put
    pop = Queue.Queue.get
    pop_nowait = Queue.Queue.get_nowait

#
# try it

stack = Stack(0)

# push items on stack
stack.push("first")
stack.push("second")
stack.push("third")

# print stack contents
try:
    while 1:
        print stack.pop_nowait()
except Empty:
    pass
```

```
third
second
```

```
first
```