**Queues**

A queue is a simple data structure best thought of as a line of customers in the supermarket. It is sometimes described as a Fast In First Out or FIFO structure. It comes equipped with only two operations:

push(object) (sometimes enqueue(object))will put the given data object at the end of the current queue. pop() (sometimes dequeue())will remove the object at the front of the queue and return it for use in the main program.

Some queue structures also provide a third operation called peek(). This will return the value at the front of the queue without removing it.

A simple representation is:



2

Push

1



3

Push

2 1



4

Push

3 2 1



3

2

1

Pop

4



2

Pop

4 3



3

Pop

4

Queues are often used as *buffers* in programs. They store data and objects in the order it arrives and then the program processes the objects in order. An example is in a web browser constructing a full web page to display. This will usually consist of many objects; some text in html that contains tags giving details about where objects are to be placed on screen, some graphics files such as backgrounds, pictures etc., there may be other items like music files or videos or java apps etc. The browser will process the, usually small, html file immediately and store the other items in a queue as they arrive. They are then removed from the front of the queue and added to the webpage to complete it. With a slow internet connection or a particularly complicated webpage you can often see this happening. Another example might be when copying a large file from a hard disc drive to a usb stick. The data will be read from the hard drive (possibly in chunks), stored temporarily in a queue in RAM and then written to the usb drive. It cannot be written directly because the read speed of the hard disc and the write speed of the usb will not be the same.

# Exercise:

1. A queue called q initially contains "Brian" at the end, then "Sue and "Bill" at the front. Draw pictures as a row of boxes showing the state of the queue after each sequence of operations. (One diagram for each part)
   1. q.push("Fred"); q.push("Alice"); q.pop()
   2. (continued from part a)) q.pop(); q.push("Brian"); q.pop(); q.push("Brian")
2. A queue called q initially contains "a" at the end, then "b" and "c" at the front. What is the output of each of the following sequences of operations?
   1. print(q.pop()); q.pop(); print(q.pop())
   2. q.pop(); print(q.pop()); print(q.pop())

# (turn over)

1. A program uses two queues. q1 is the same as q in question 2, q2 is initially empty.
   1. What is the output of the following sequence of operations?

q2.push(q1.pop()); print(q1.pop()); print(q2.pop()); print(q1.pop())

* 1. Using the same two stacks as at the start, and a combination of the four operations q1.push(q2.pop()), q2.push(q1.pop()), print(q1.pop()), print(q2.pop()) **and no others**, can you write instructions to print out a then b then c?
  2. Parts a) and b) have printed out two of the six arrangements of the three letters abc. Write sets of instructions to print out as many of the other arrangements as you can, each time starting with the original two queues. Are any arrangements impossible?

1. A queue is modelled by using the middle of a Python list called q, together with two integer variables called end and front that keep track of the location of the end and front of the queue. Initially we have q=[0,0,0,0,0,0,12,7,14,42,0,0,0], end=6 and front=9. The operation push simply sets end=end-1 and then overwrites the entry at location end. The operation pop will read the value at

location front and then set front=front-1 (no need to waste time deleting). Write out the values of q, end and front after each of the following sequences of operations (only one set of three answers per part).

* 1. q.push(5); q.push(3); q.pop(); q.pop()
  2. (continued from a)) q.push(7); q.push(8); q.push(3); q.pop(); q.push(4)
  3. What happens if you now try q.push(8)? Can you suggest a way to modify the push operation to deal with this?