COMPSCI 105 S1 2017
Principles of Computer Science

15 Queue(1)

Agenda & Readings

- Agenda
  - Introduction
  - Queue Abstract Data Type (ADT)
  - Implementing a queue using a list
- Reference:
  - Textbook: Problem Solving with Algorithms and Data Structures
    - Chapter 3: Basic Data Structures

Can you think of other examples of queues?

What is a Queue?

- Queues are appropriate for many real-world situations
  - Example: A line to buy a movie ticket
  - Computer applications, e.g. a request to print a document
- A queue is an ordered collection of items where the addition of new items happens at **one end** (the rear or back of the queue) and the removal of existing items always takes place at the **other end** (the front of the queue)
  - New items enter at the **back**, or rear, of the queue
  - Items leave from the **front** of the queue
  - **First-in, first-out (FIFO)** property:
    - The first item inserted into a queue is the first item to leave

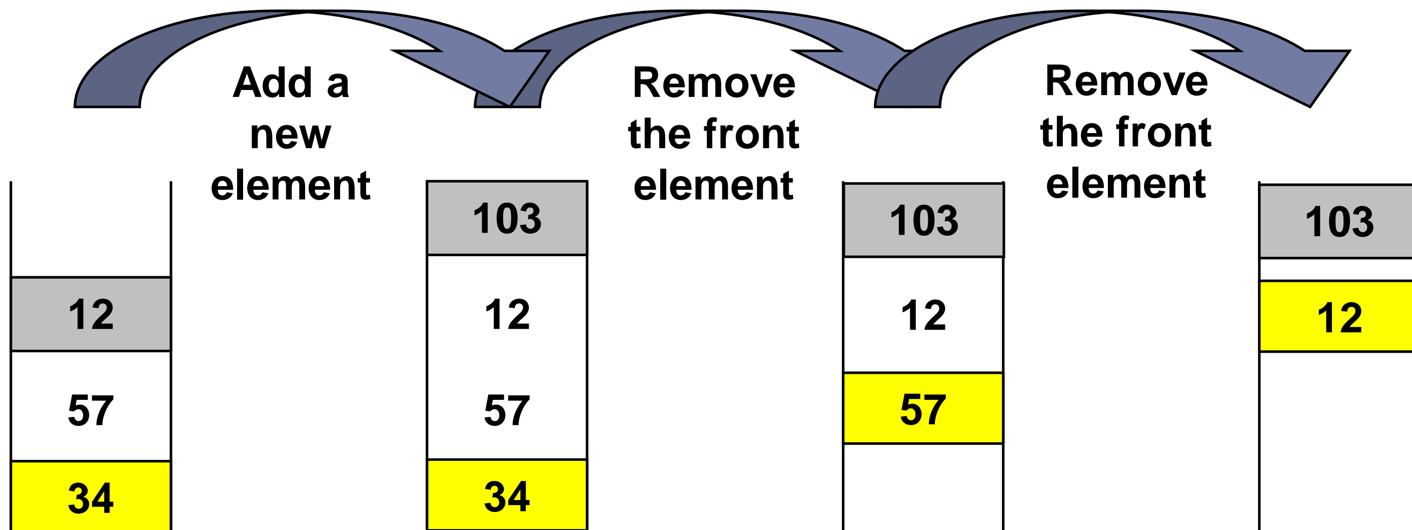
What is a Queue?

- Queues implement the FIFO (first-in first-out) policy:
  - For example: the printer / job queue!

Queue Example

- Add only to the **rear** of a Queue
- Remove only from the **front** of the Queue
  - Note: The last item placed on the queue will be the last item removed

First In - First Out (FIFO)



ADT Queue Operations

- What are the operations which can be used with a Queue Abstract Data?
  - Create an empty queue:
  - Determine whether a queue is empty:
  - Add a new item to the queue:
    - enqueue
  - Remove from the queue the item that was added earliest:
    - dequeue
  - Retrieve from the queue the item that was added earliest:
    - peek

ADT Queue Operations

- What are the operations which can be used with a Queue Abstract Data?

The Queue Abstract Data Type

- Queue() creates a new queue that is empty
  - It needs no parameters and returns an empty queue
- enqueue(item) adds a new item to the rear of the queue
  - It needs the item and returns nothing
  - The queue is modified
- dequeue() removes the front item from the queue
  - It needs no parameters and returns the item
  - The queue is modified

Queue(), enqueue(item) and dequeue() are critical operations in order to manipulate the elements of the queue

The Queue Abstract Data Type

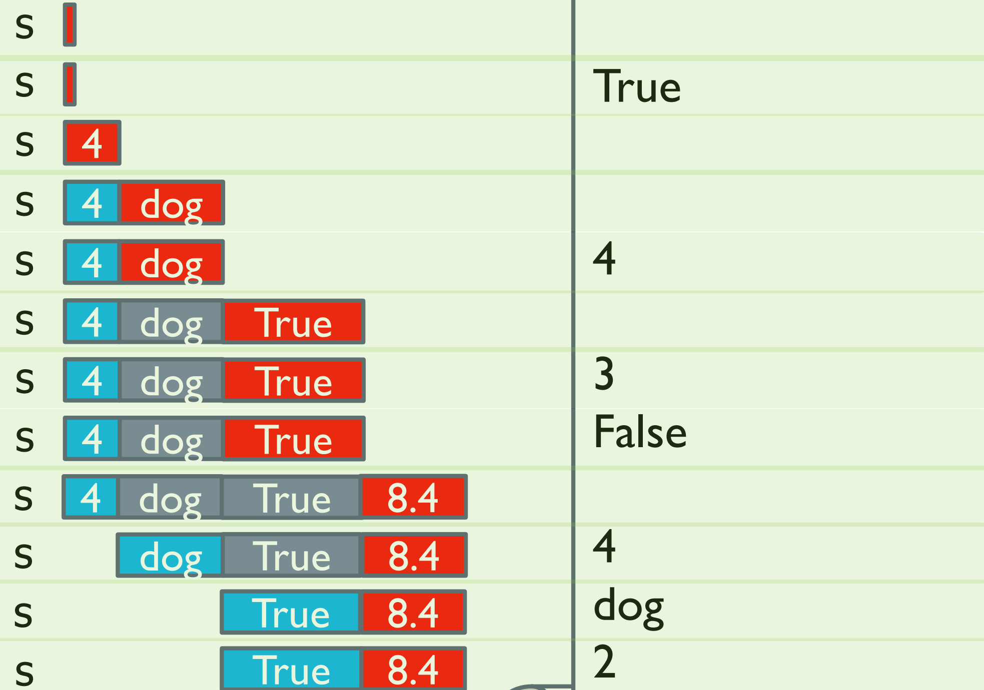
- peek() returns the earliest item from the queue but does not remove it
  - It needs no parameters
  - The queue is not modified
- is_empty() tests to see whether the queue is empty
  - It needs no parameters and returns a boolean value
  - The queue is not modified
- size() returns the number of items in the queue
  - It needs no parameters and returns an integer
  - The queue is not modified

peek(), is_empty() and size() are useful to allow the users to retrieve the properties of the queue but they are not necessary

Code Example - Application

- Code:

```
s = Queue()
print(s.is_empty())
s.enqueue(4)
s.enqueue('dog')
print(s.peek())
s.enqueue(True)
print(s.size())
print(s.is_empty())
s.enqueue(8.4)
s.dequeue()
s.dequeue()
print(s.size())
```



Code Example - Application

- Code:

```
s = Queue()
print(s.is_empty())
s.enqueue(4)
s.enqueue('dog')
print(s.peek())
s.enqueue(True)
print(s.size())
print(s.is_empty())
s.enqueue(8.4)
s.dequeue()
s.dequeue()
print(s.size())
```

```
s = []
s = []
s = [4]
s = [4, 'dog']
s = [4, 'dog']
s = [4, 'dog', True]
s = [4, 'dog', True]
s = [4, 'dog', True]
s = [4, 'dog', True, 8.4]
s = ['dog', True, 8.4]
s = [True , 8.4]
s = [True , 8.4]
```

```
True
4
3
False
4
dog
2
```

Exercise 1

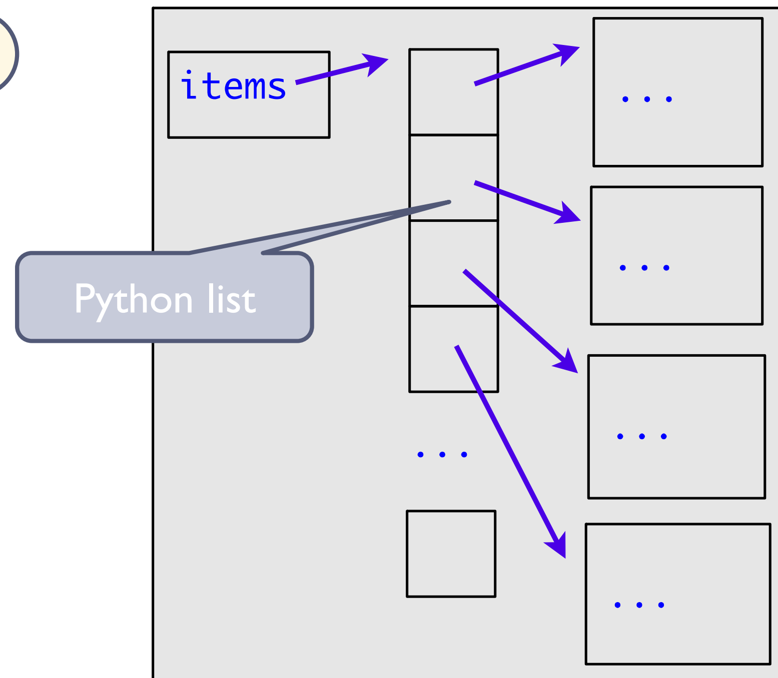
- What is the output of the following code fragment?

```
s = Queue()
print(s.is_empty())
s.enqueue(4)
s.enqueue('dog')
print(s.peek())
print(s.size())
print(s.is_empty())
s.dequeue()
s.enqueue(3)
s.dequeue()
print(s.size())
```

Code Example - Application

- We use a python List data structure to implement the queue

```
class Queue:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def size(self):
        return len(self.items)
……
```

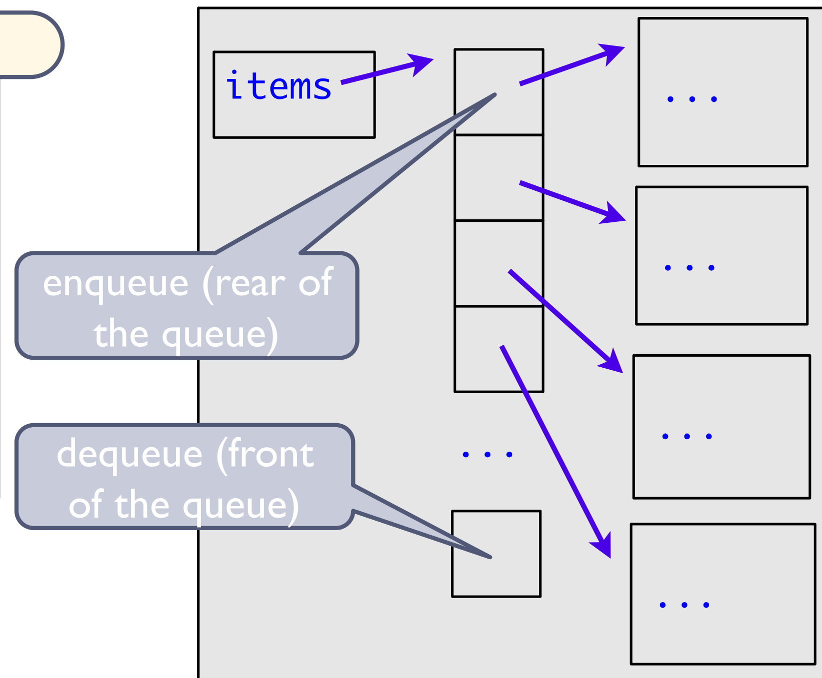


Code Example - Application

- We use a python List data structure to implement the queue
  - Version 1
    - The addition of new items takes place at the beginning of the list
    - The removal of existing items takes place at the end of the list

```
class Queue:
 ……
    def enqueue(self, item):
        self.items.insert(0,item)
    def dequeue(self):
        return self.items.pop()
```

Big-O?
enqueue()/search(): O(n)
dequeue()/peek(): O(1)

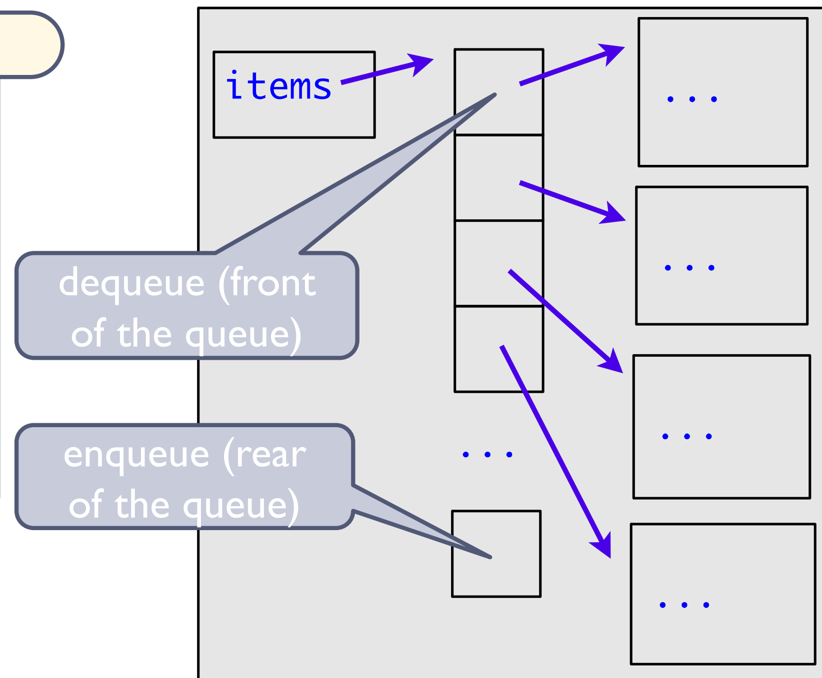


Code Example - Application

- We use a python List data structure to implement the queue
  - Version 2
    - The addition of new items takes place at the beginning of the list
    - The removal of existing items takes place at the end of the list

```
class Queue:
 ......
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        return self.items.pop(0)
```

Big-O?
enqueue(): O(1)
dequeue()/peek()/search(): O(n)



Exercise 2

- What is the output of the following code fragment?

```
from Queue import Queue
try:
    q = Queue()
    q.enqueue(2)
    q.enqueue(4)
    q.enqueue(6)
    while not q.is_empty():
        print(q.dequeue())
except IndexError:
    print ('empty queue')
```

Comparisons between Queue & Stack

- Behaviour:
  - The behaviour of a stack is like a Last-In-First-Out (LIFO) system
  - The behaviour of a queue is like a First-In-First-Out (FIFO) system
- Implementation with Python list:
  - The list methods make it very easy to use a list as a stack
    - To add an item to the top of the stack, using append()
    - To retrieve an item from the top of the stack, using pop() without an explicit index
  - It is not efficient to use a list as a queue
    - To add or remove an item from the end of list are fast, using append() and pop()
    - To add or remove an item at the beginning of list are slow (because all of the other elements have to be shifted by one)

Comparisons between Queue & Stack

- Big O:
  - Stack
    - push(): O(1)
    - pop(): O(1)
    - peek(): O(1)
    - search(): O(n)
  - Queue (best scenario)
    - enqueue(): O(n)
    - dequeue(): O(1)
    - peek(): O(1)
    - search(): O(n)

Summary

- The definition of the queue operations gives the ADT queue first-in, first-out (FIFO) behavior
- Python lists support simple implementations of queues