Basic Data Structures

Queues and Deques

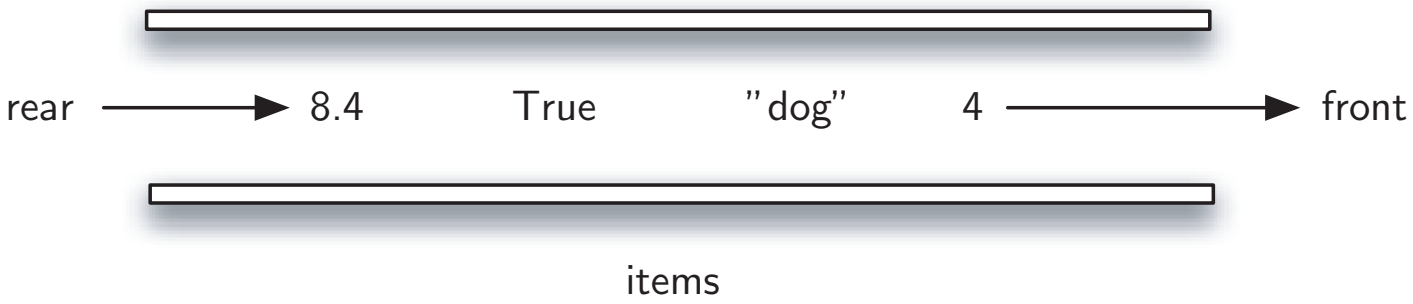
Brad Miller David Ranum

1/25/06

Outline

Outline

A Queue of Python Data Objects



Outline

- `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.
- `dequeue()` removes the front item from the queue. It needs no parameters and returns the item. The queue is modified.
- `isEmpty()` tests to see whether the queue is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the queue. It needs no parameters and returns an integer.

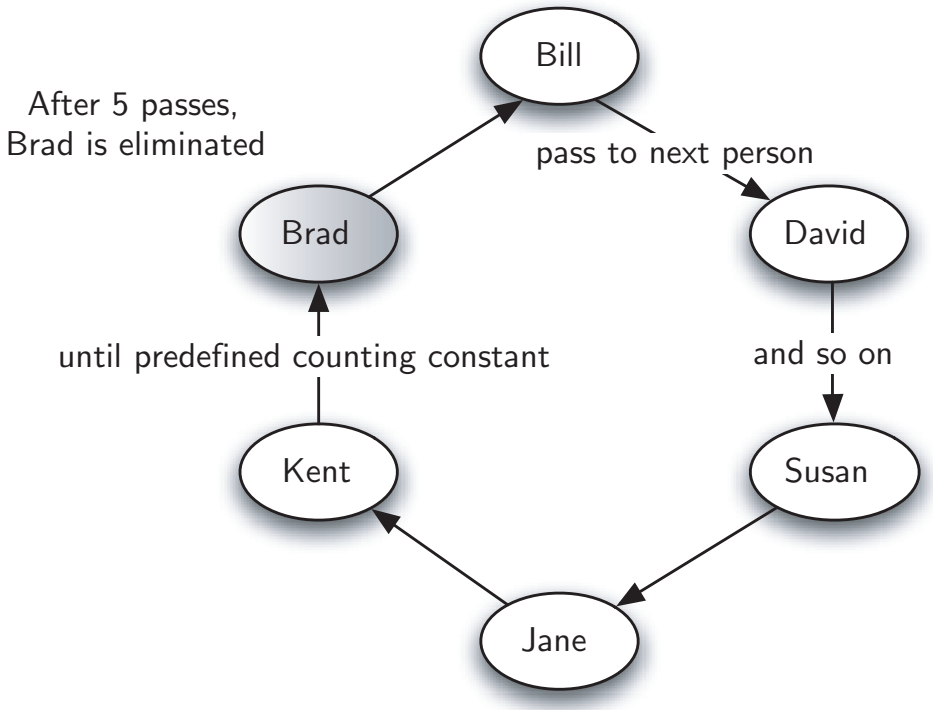
Outline

Queue Implementation in Python

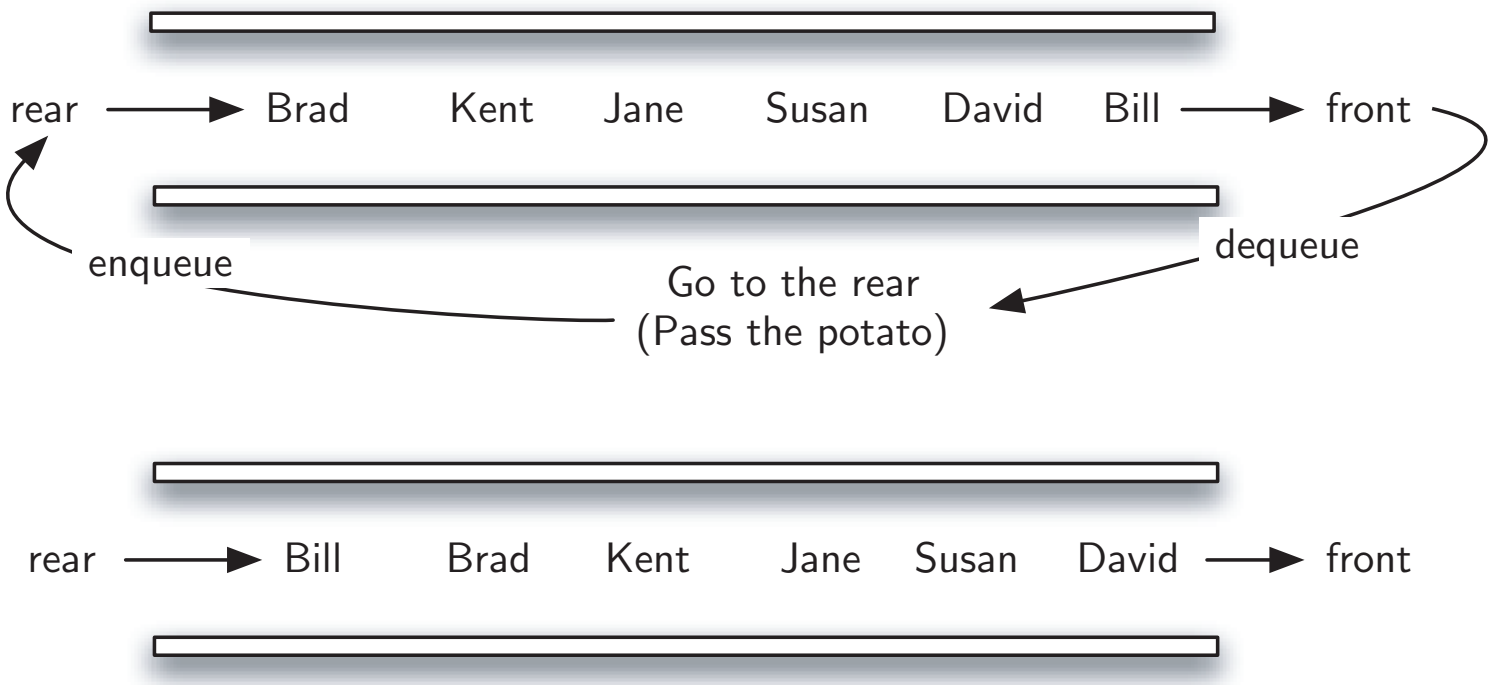
```
1  class Queue:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def enqueue(self, item):
9          self.items.insert(0,item)
10
11     def dequeue(self):
12         return self.items.pop()
13
14     def size(self):
15         return len(self.items)
```

Outline

A Six Person Game of Hot Potato



A Queue Implementation of Hot Potato

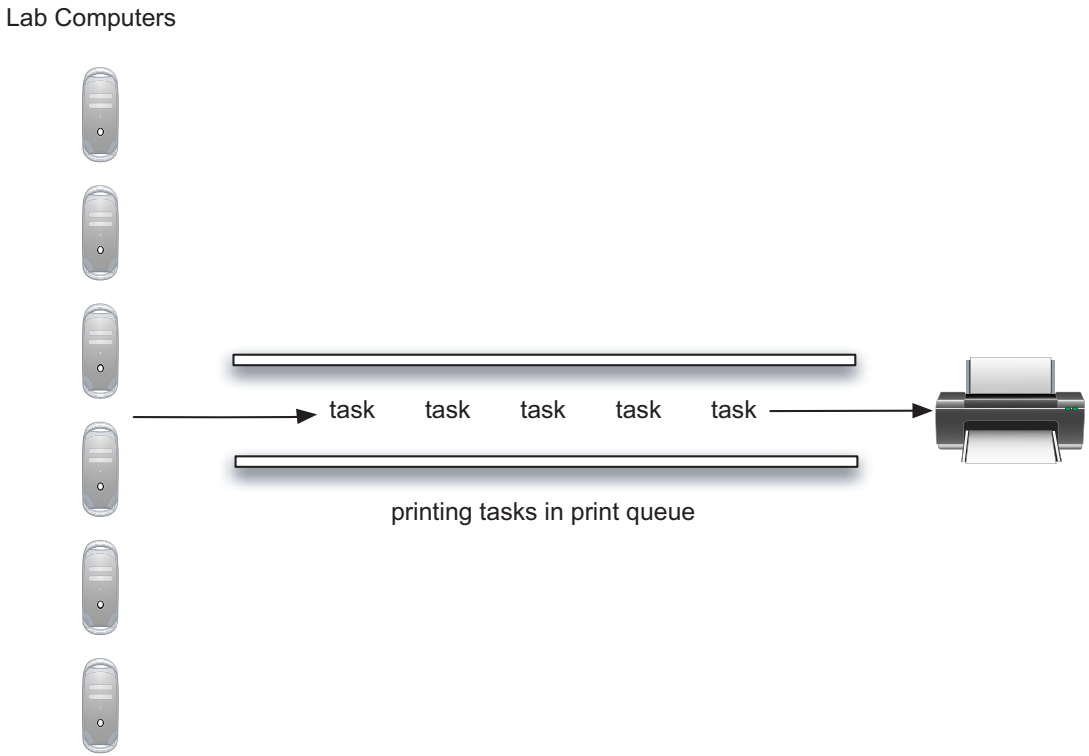


Hot Potato Simulation

```
1  def hotPotato(namelist, N):
2
3      simqueue = Queue()
4      for name in namelist:
5          simqueue.enqueue(name)
6
7      while simqueue.size() > 1:
8          for i in range(N):
9              simqueue.enqueue(simqueue.dequeue())
10
11         simqueue.dequeue()
12
13     return simqueue.dequeue()
```

Outline

Computer Science Laboratory Printing Queue



Printer Queue Simulation–The Printer Class I

```
1  class Printer:
2      def __init__(self, pages):
3          self.pagerate = pages
4          self.currentTask = None
5          self.timeRemaining = 0
6
7      def tick(self):
8          if self.currentTask != None:
9              self.timeRemaining = self.timeRemaining - 1
10             if self.timeRemaining == 0:
11                 self.currentTask = None
12
13
14
15
```

Printer Queue Simulation–The Printer Class II

```
16        def busy(self):
17            if self.currentTask != None:
18                return True
19            else:
20                return False
21
22        def startNext(self,newtask):
23            self.currentTask = newtask
24            self.timeRemaining = newtask.getPages() \
25                                 * 60/self.pagerate
```

Printer Queue Simulation–The Task Class

```
1  import random
2  class Task:
3      def __init__(self,time):
4          self.timestamp = time
5          self.pages = random.randrange(1,21)
6
7      def getStamp(self):
8          return self.timestamp
9
10     def getPages(self):
11         return self.pages
12
13     def waitTime(self, currenttime):
14         return currenttime - self.timestamp
```

Printer Queue Simulation–The Main Simulation I

```
1  from queue import *
2  from printer import *
3  from task import *
4
5  import random
6
7  def simulation(numSeconds, pagesPerMinute):
8
9      labprinter = Printer(pagesPerMinute)
10     printQueue = Queue()
11     waitingtimes = []
12
13     for currentSecond in range(numSeconds):
14
15
```

Printer Queue Simulation–The Main Simulation II

```
16          if newPrintTask():
17              task = Task(currentSecond)
18              printQueue.enqueue(task)
19
20          if (not labprinter.busy()) and \
21                      (not printQueue.isEmpty()):
22            nexttask = printQueue.dequeue()
23            waitingtimes.append( \
24                nexttask.waitTime(currentSecond))
25            labprinter.startNext(nexttask)
26
27          labprinter.tick()
28
29      averageWait=sum(waitingtimes)/float(len(waitingtimes))
30      print "Average Wait Time%6.2f seconds"%(averageWait),
31      print "Tasks Remaining %3d"%(printQueue.size())
```

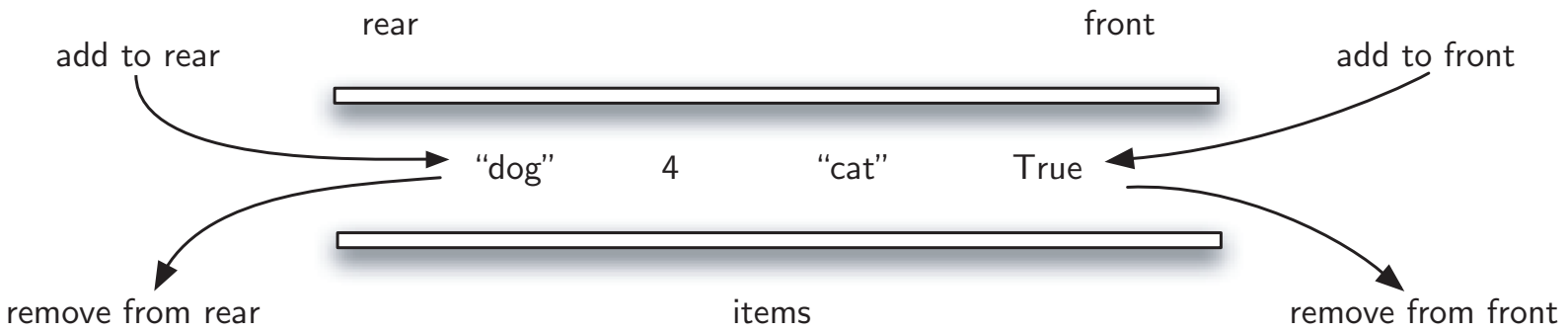
Printer Queue Simulation–The Main Simulation III

```
32
33
34  def newPrintTask():
35      num = random.randrange(1,181)
36      if num == 180:
37          return True
38      else:
39          return False
```

- What if enrollment goes up and the average number of students increases by 20?
- What if it is Saturday and students are not needing to get to class? Can they afford to wait?
- What if the size of the average print task decreases since Python is such a powerful language and programs tend to be much shorter?

Outline

A Deque of Python Data Objects



Outline

- `Deque()` creates a new deque that is empty. It needs no parameters and returns an empty deque.
- `addFront(item)` adds a new item to the front of the deque. It needs the item and returns nothing.
- `addRear(item)` adds a new item to the rear of the deque. It needs the item and returns nothing.
- `removeFront()` removes the front item from the deque. It needs no parameters and returns the item. The deque is modified.
- `removeRear()` removes the rear item from the deque. It needs no parameters and returns the item. The deque is modified.
- `isEmpty()` tests to see whether the deque is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the deque. It needs no parameters and returns an integer

Outline

Deque Implementation in Python I

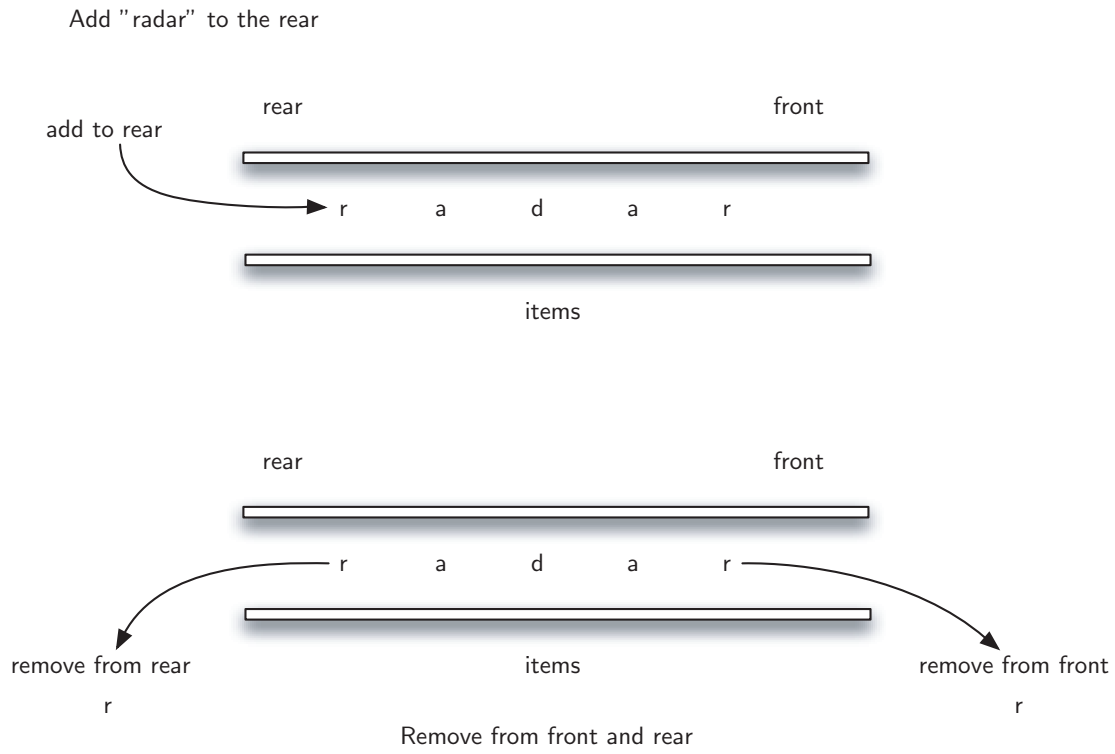
```
1  class Deque:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def addFront(self, item):
9          self.items.append(item)
10
11     def addRear(self, item):
12         self.items.insert(0,item)
13
14     def removeFront(self):
15         return self.items.pop()
```

Deque Implementation in Python II

```
16
17      def removeRear(self):
18          return self.items.pop(0)
19
20      def size(self):
21          return len(self.items)
```

Outline

A Deque


Add "radar" to the rear
rear
front
add to rear
r a d a r
items
rear
front
r a d a r
remove from rear
r
items
remove from front
r
Remove from front and rear

Palindrome Checker

```
1  def palchecker(aString):
2      chardeque = Deque()
3
4      for ch in aString:
5          chardeque.addRear(ch)
6
7      stillEqual = True
8
9      while chardeque.size() > 1 and stillEqual:
10         first = chardeque.removeFront()
11         last = chardeque.removeRear()
12         if first != last:
13             stillEqual = False
14
15     return stillEqual
```