More on Classes and Objects
The line class

- The goal is to define a class called `line` such that instances of this class represent line segments that connect two points.
- Two fields: `point1` and `point2`, each being an instance of the `point` class we defined earlier.
- Besides the constructor method, we will define methods such as:
  - length
  - slope
from point import *

class line:
    # The constructor for the line class; constructs the line segment defined by the two given points p1 and p2
    def __init__(self, p1, p2):
        self.point1 = p1
        self.point2 = p2
# The length and slope methods

# Returns the Euclidean length of the line segment
def length(self):
    return (self.point1).distance(self.point2)

# Returns the slope of the line segment
def slope(self):
    if self.point1.x == self.point2.x:
        return None
    return (self.point1.y - self.point2.y)/(self.point1.x - self.point2.x)
A method for “displaying” line instances

# Returns a str the represents a line segment. This function is called whenever a
# string representation of the object is needed
def __repr__(self):
    return "({}+str(self.point1.x) + "," + str(self.point1.y) + ")---("+ str(self.point2.x) + ","+ str(self.point2.y) + ")"

• When the name of a line instance is typed at the Python shell prompt, this function is called and the string returned by this function is displayed.
In Homework 8, we needed a data structure that represented a collection of items that allowed us access in “first in first out” (FIFO) order.

That is, whenever we ask for an item from the collection, it should return to us the earliest item in the collection.

This is a standard data structure in computer science called the *queue* data structure.

Typically, it supports two operations:
- `enqueue(item)`: inserts an item into the collection
- `dequeue(item)`: removes the oldest item in the collection and returns it.
A simple list based implementation

# Implements a queue that allows access to a collection of items in a "first in first out" (FIFO) manner. A queue is implemented as a list with elements arriving at the front of the list into slot 0 and elements leaving from the list from the back.

class queue:
    
    # Constructs an empty queue
    def __init__(self):
        self.L = []

    def enqueue(self, item):
        self.L.insert(0, item)

    def dequeue(self):
        return self.L.pop()

    def __repr__(self):
        return str(self.L)
The enqueue method runs in linear time.

Question: can we implement both methods in constant time?

Yes...we can use either lists to do this or dictionaries to do this.

We will now see a method that uses dictionaries.
Efficient *queue* class implementation: Main Idea

- We maintain a clock – whenever a new item arrives wanting to be inserted into the queue, we increment the time on the clock.
- We use the time on the clock to “time stamp” each item as it arrives.
- We insert an element $e$ that arrives at time $t$ as $t : e$ into a dictionary.
- Finally, we keep track of the time stamps corresponding to the oldest and the most recent elements in the dictionary.