Data Structures (CS 1520) Lecture 4 Name:_____

1. An "abstract" view of the stack:

d
c
b
a

top

bottom

Using an array implementation would look something like:

	0	1	2	3			(max-1)
items:	a	b	c	d			

top: 3 max: 100

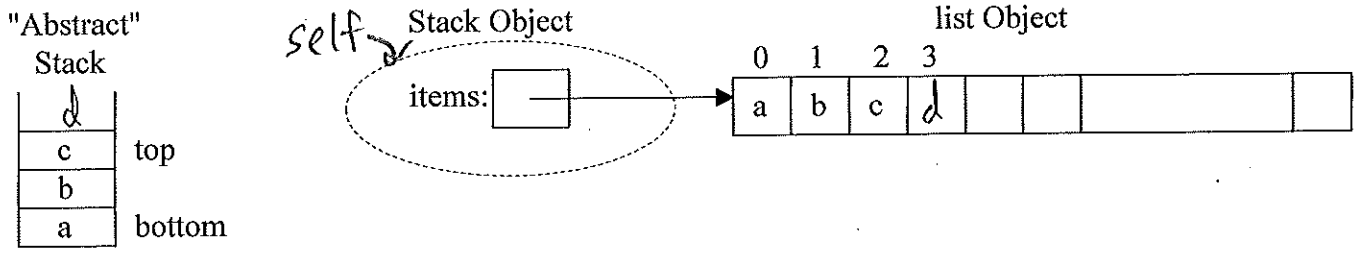
Complete the big-oh notation for the following stack methods assuming an array implementation: ("n" is the # items)

	push(item)	pop( )	peek( )	size( )	isEmpty( )	isFull( )	Constructor
Big-oh	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	

2. Since Python does not have a (directly accessable) built-in array, we can use a list.

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

Since Python uses an array of references (pointers) to list items in their implementation of a list.



a) Complete the big-oh notation for the stack methods assuming this Python list implementation: ("n" is the # items)

	push(item)	pop( )	peek( )	size( )	isEmpty( )	__init__
Big-oh	O(1) average	O(1)	O(1)	O(1)	O(1)	O(1)

b) Which operations should have what preconditions?

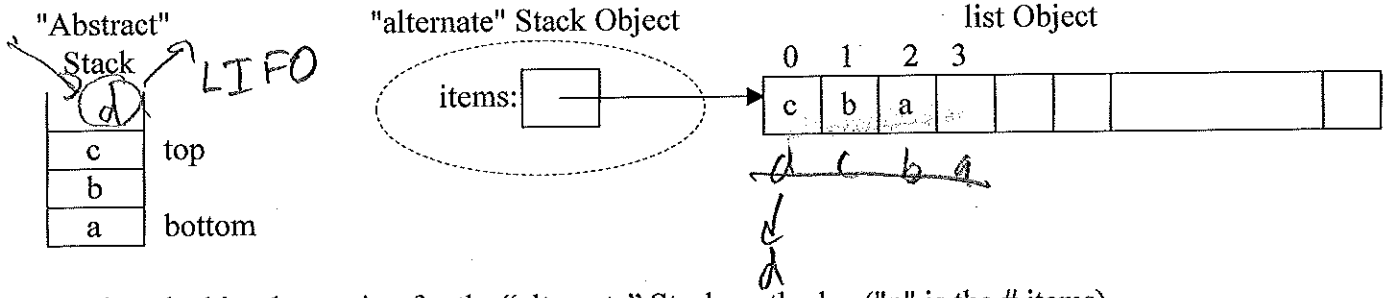
pop - stack is not empty

peek - " " " "

3. The text's alternative stack implementation also using a Python list is:

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.insert(0,item)

    def pop(self):
        return self.items.pop(0)

    def peek(self):
        return self.items[0]

    def size(self):
        return len(self.items)
```

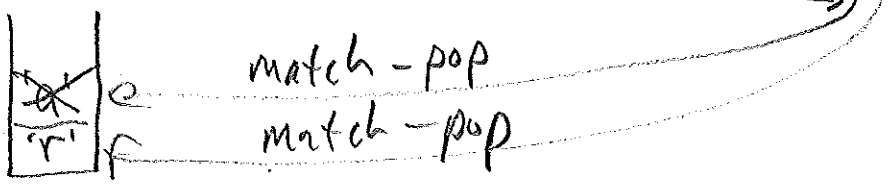
Since an array is used to implement a Python list, the alternate Stack implementation using a list:



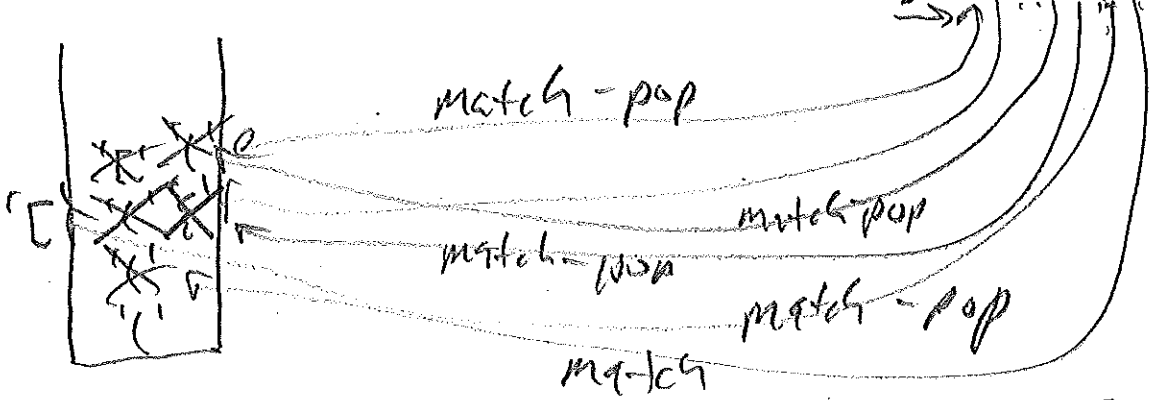
a) Complete the big-oh notation for the "alternate" Stack methods: ("n" is the # items)

	push(item)	pop( )	peek( )	size( )	isEmpty( )	__init__
Big-oh	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)

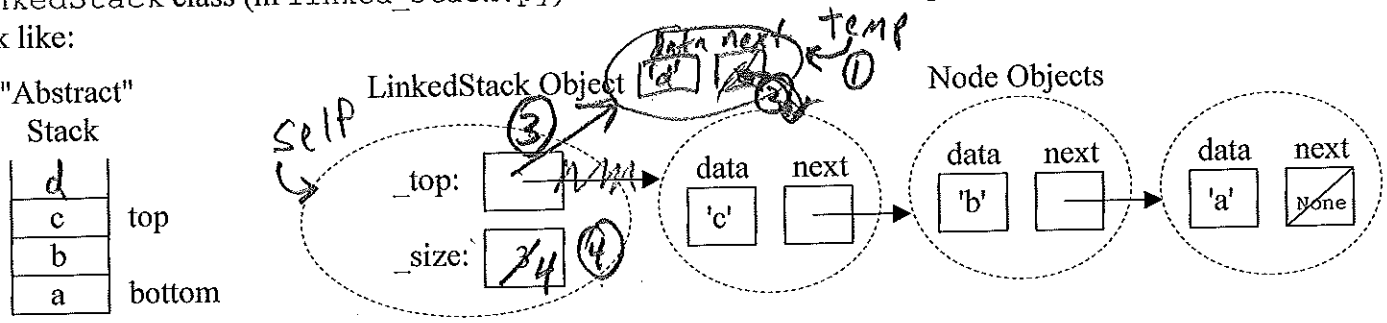
4. How could we use a stack to check if a word is a palindrome (e.g., radar, toot)?



5. How could we check to see if we have a balanced string of nested symbols? ("(([])){()}[])")



1. The `Node` class (in `node.py`) is used to dynamically create storage for a new item added to the stack. The `LinkedStack` class (in `linked_stack.py`) uses this Node class. Conceptually, a `LinkedStack` object would look like:




```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext
```

accessor methods

mutators methods

a) Complete the `push`, `pop`, and `__str__` methods.

b) Stack methods big-oh's? (Assume "n" items in stack)

- constructor `__init__`:
- `push(item)`:
- `pop()`
- `peek()`
- `size()`
- `isEmpty()`
- `str()`

```
class LinkedStack(object):
    """ Link-based stack implementation."""

    def __init__(self):
        self._top = None
        self._size = 0

    def push(self, newItem):
        """Inserts newItem at top of stack."""
        temp = Node(newItem)
        temp.setNext(self._top)
        self._top = temp
        self._size += 1

    def pop(self):
        """Removes and returns the item at top of the stack.
        Precondition: the stack is not empty."""



    def peek(self):
        """Returns the item at top of the stack.
        Precondition: the stack is not empty."""
        return self._top.getData()

    def size(self):
        """Returns the number of items in the stack."""
        return self._size

    def isEmpty(self):
        return self._size == 0

    def __str__(self):
        """Items strung from top to bottom."""
```

Steps for writing "linked" methods

(1) Draw normal (non empty) case picture

(2) Draw the updated picture after method runs

(3) Number steps in drawing

(4) write normal-case code from picture

(5) Consider "special cases"

(a) empty stack <"run" normal case code / Draw picture