1. An "abstract" view of the stack:

\[ \text{LIFO} \]

Using an array implementation would look something like:

```
items: [a, b, c, d, 0, 1, 2, 3]  \text{(max-1)}
```

```
top: 2 \text{ max: 100}
```

```
return top without removing
tag
```

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

<table>
<thead>
<tr>
<th></th>
<th>push(item)</th>
<th>pop()</th>
<th>peek()</th>
<th>size()</th>
<th>isEmpty()</th>
<th>isFull()</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big-oh</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td></td>
</tr>
</tbody>
</table>

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

```python
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.

```
myStack = Stack()
mystack.push('a')
```

```
if mystack.isEmpty():
    raise Exception("Cannot pop an empty stack")
```

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

<table>
<thead>
<tr>
<th></th>
<th>push(item)</th>
<th>pop()</th>
<th>peek()</th>
<th>size()</th>
<th>isEmpty()</th>
<th><em>init</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Big-oh</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td></td>
</tr>
</tbody>
</table>

b) Which operations should have what preconditions?

- **Precond.** Pop - stack is not empty
- **Precond.** Peek - ...
3. The text’s alternative stack implementation also using a Python list is:

```python
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:

<table>
<thead>
<tr>
<th>&quot;Abstract&quot; Stack</th>
<th>&quot;alternate&quot; Stack Object</th>
<th>list Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td></td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c b a</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>push(item)</th>
<th>pop()</th>
<th>peek()</th>
<th>size()</th>
<th>isEmpty()</th>
<th><strong>init</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Big-oh</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

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? ("(([])))")

Scan string from left-to-right

Push opening symbols (',]', ')

On closing symbols pop stack and compare for match. If don't match,
then not balanced.

Note: if stack not empty when string runs out then not balanced.
1. The Node class (in node.py) is used to dynamically create storage for a new item added to the stack. The LinkedList class (in linked_stack.py) uses this Node class. Conceptually, a LinkedList 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
```

### 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():
Method to implement "linked" operation

1. Draw the normal-case picture if non-empty
2. Update picture to reflect the after change
3. # the update to reflect the order of steps
4. Write the normal-case code

5. Consider "special cases"
   (a) empty stack
   (b) ? full