Strings. Implement the following function.

```python
def peel(markers, text):
    """Returns a new string where the `markers` have been removed from the beginning and end of `text`"
    Examples:
    peel( "()", "(abc)" ) --> "abc"
    peel( "()", "(1(+1)" ) --> "1(+1"
    peel( "<()>", "<(>.<)>" ) --> ">.<"
    peel( "ab", "ab" ) --> ""
    Preconditions:
    markers: string of even length (0 is allowed)
    text: any-length string that starts w/ 1st half of `markers`, ends w/ 2nd half.
    """
    # REMINDER: in a slice expression like s[n:m], n and m must be ints, not floats
    marker_len = len(markers)//2
    text_len = len(text)
    # This solution avoids using rindex/rfind by subtracting from len(text).
    return text[marker_len:text_len-marker_len]
```

Remember that because / is a float operator, the result of x / 2 will be a float even if x is an even int:

```
>>> test = ['a', 'b']
>>> len(test)
2
>>> len(test)/2
1.0
>>> test[len(test)/2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices, not float
```

So, one must use either // or do an explicit cast to an int for this question.

Alternate solutions:

```python
def peel2(markers, text):
    m1= markers[:len(markers) //2]
```
m2 = markers[len(markers)//2:]
start_inside = text.index(m1)+len(m1)
end_inside = text.rindex(m2)-1
return text[start_inside:end_inside+1]

# Another alternate solution
def peel3(markers, text):
    m1= markers[:len(markers) //2]
m2 = markers[len(markers)//2:]
start_inside = text.index(m1)+len(m1)
start_outside = text.rindex(m2)
return text[start_inside:start_outside]

Note: it does not suffice to set start_outside = text.index(m2, start_inside) because there could be occurrences of the second marker(s) before the final occurrence, as happens with the second test case we gave.
2. [8 points] **Lists.** Implement the following function.

```python
def swap2(a_list, j, k):
    """Modifies a_list by swapping the two elements of a_list starting
    at index j with the 2 entries of a_list starting at index k.
    Examples:
    swap2([100, 101, 102, 103, 104, 105, 106, 107, 108, 109], 1, 6)
    changes a_list to
    [100, 106, 107, 103, 104, 105, 101, 102, 108, 109]
    -------- --------
    swap2([100, 101, 102, 103, 104, 105, 106, 107, 108, 109], 0, 4)
    changes a_list to
    [104, 105, 102, 103, 100, 101, 106, 107, 108, 109]
    -------- --------
    swap2(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'], 0, 4)
    changes a_list to
    ['e', 'f', 'c', 'd', 'a', 'b', 'g', 'h', 'i', 'j']
    -------- --------
    Preconditions:
    j and k are valid indices (positive, < len(a_list))
    j + 2 <= k (the elements you're swapping don't overlap in a_list)
    k + 2 <= len(a_list)
    """
    temp1 = a_list[j]
    temp2 = a_list[j+1]
    a_list[j] = a_list[k]
    a_list[j+1] = a_list[k+1]
    a_list[k] = temp1
    a_list[k+1] = temp2
```

---

Page 3
3. Some truths are self evident. Some are learned in CS 1110.

(a) [2 points] True or False? The drawing below accurately depicts the value of variable x in Global Memory after the code below is executed in Python:

```python
def double_x(input):
    return input * 2
x = 5
x = double_x(x)
```

Circle One:

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
</table>

Correct Answer: True

(b) [2 points] True or False? The drawing below accurately depicts the value of variable x in Global Memory after the code below is executed in Python:

```python
def double_x(input):
    x = input * 2
x = 5
double_x(x)
```

Circle One:

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
</table>

Correct Answer: False. Global x stays 5.

(c) [2 points] True or False? The drawing below accurately depicts the value of variable x in Global Memory after the code below is executed in Python:

```python
def double_x(input):
    x = input * 2
    print(str(x))
x = 5
double_x(x)
```

Circle One:

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
</table>

Correct Answer: False. Global x stays 5.

(d) [2 points] True or False? The drawing below accurately depicts the value of variable x in Global Memory after the code below is executed in Python:

```python
def double_x(input):
    x = input * 2
    print(str(x))
    x = 5
    x = double_x(x)
```

Circle One:

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
</table>

Correct Answer: False. x would be None
4. [24 points] Time for dinner! Place is an object with 3 attributes: spoon, fork, and knife. A call of the form Place(s, f, k) creates a new Place object with attribute spoon set to s, fork set to f, and knife set to k. Assume that class Place is accessible within the given code. Simulate running all 27 lines of code and draw the memory diagram as seen in class and Assignment 2.

```python
def soup(p):
    p.spoon = p.spoon + 1
    drawer.spoon = drawer.spoon - 1

def salad(p):
    p.fork = p.fork + 1
    drawer.fork = drawer.fork - 1
    p2.knife = p2.knife + 2
    drawer.knife = drawer.knife - 1

def dinner(p, with_soup, with_salad):
    if with_soup:
        soup(p)
    if with_salad:
        salad(p)

def dessert(p, name):
    if name == "ice cream":
        n_spoons = 2
    else:
        n_spoons = 0
    p.fork = p.fork + 1
    p.spoon = p.spoon + n_spoons
    return n_spoons

p1 = Place(1, 2, 0)
p2 = Place(1, 2, 0)
drawer = Place(6, 4, 8)
dinner(p1, False, True)
n_spoons = dessert(p2, "ice cream")
drawer.spoon = drawer.spoon - n_spoons
```
5. [8 points] **Testing, Testing, 1, 2, 3, Testing!**

Consider the following function specification, which you might use if you want to distribute the cost of dinner amongst you and your friends.

```python
def batch_withdraw(balance_list, withdraw_amount):
    """balance_list is a list of floats representing the balances of multiple bank accounts

    Pre-condition:
    withdraw_amount is a float with value >= 0.

    Return a new list of the same length as balance_list, where every value is the corresponding value in balance_list minus withdraw_amount. If any value in balance_list is less than withdraw_amount (i.e., there is not enough in the account to withdraw), return the empty list. """
```

Here is an example of one set of sample inputs and an expected output:

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Inputs</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>balance_list: [20.0, 30.0, 40.0, 50.0], withdraw_amount: 10.0</td>
<td>return value: [10.0, 20.0, 30.0, 40.0]</td>
</tr>
</tbody>
</table>

Provide **two** more conceptually distinct test cases, using the same format. Include a short statement (1-2 sentences) explaining what situation each of your test cases represents.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Inputs</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>balance_list: [70.0, 10.0, 80.0], withdraw_amount: 40.0</td>
<td>return value: []</td>
</tr>
<tr>
<td>3</td>
<td>balance_list: [30.0], withdraw_amount: 30.0</td>
<td>return value: [0.0]</td>
</tr>
<tr>
<td></td>
<td>balance_list: [], withdraw_amount: 20.0</td>
<td>return value: []</td>
</tr>
</tbody>
</table>

Test Case 2 covers the following situation:

Test Case 3 covers the following situation:

Some possibilities:

- **balance_list: [70.0, 10.0, 80.0], withdraw_amount: 40.0**
  - return value: []
  - tests case where one value in the **balance_list** is < the **withdraw_amount** and so should return the empty list

- **balance_list: [30.0], withdraw_amount: 30.0**
  - return value: [0.0]
  - tests case where value in the **balance_list** is equal to the **withdraw_amount** and so should be zeroed out (but not return empty list)

- **balance_list: [], withdraw_amount: 20.0**
  - return value: []
  - tests case where **balance_list** is empty and so should return the empty list
6. **The eyes have it.** Assume objects of class `Point` have two attributes: `x` and `y`; both are `ints`. Assume objects of new class `Face` have three `Point` attributes: `left_eye`, and `right_eye`, and `nose`. `Face` attributes should have the following relationships to be considered **proportionate**:

- `left_eye` and `right_eye` have the same `y` attribute values (they are the same height)
- `left_eye` and `right_eye` are centered across the y-axis (`left_eye`’s `x` attribute is negative and `right_eye`’s `x` attribute is positive)
- `nose` always sits on the y-axis (x=0)
- `nose` is always lower than the eyes by the distance that the eyes are from the y-axis. Example: if the eyes are 2 units from the y-axis, the nose will be 2 units below the eyes.

![Graph showing proportionate face](image)

(a) [6 points] Implement the following function.

```python
def set_face(f, right_x, right_y):
    """Given ints right_x and right_y (which are the desired values for the x and y coordinates of the right eye of Face f), sets the left_eye, right_eye and nose attributes of Face f, so that Face f is proportionate.

    Precondition: right_x and right_y are non-negative ints."
    f.right_eye.x = right_x
    f.right_eye.y = right_y
    f.left_eye.x = -right_x
    f.left_eye.y = right_y
    f.nose.x = 0
    f.nose.y = right_y - right_x
```

# Alternate solution
```python
def set_face2(f, right_x, right_y):
    f.right_eye.x = right_x
    f.right_eye.y = right_y
    f.left_eye.x = -f.right_eye.x
    f.left_eye.y = f.right_eye.y
    f.nose.x = 0
    f.nose.y = right_y - right_x
```
(b) [9 points] Implement the following function.

```python
def is_proportionate(f):
    """Return True if the locations of the eyes and nose of Face f make the face `proportionate`, based on the definition at the beginning of this question. If any of the x,y attributes of the elements of Face f are not in proportion, return False. """
    # check eyes
    if f.right_eye.x != -f.left_eye.x:
        return False
    if f.right_eye.y != f.left_eye.y:
        return False

    # check nose
    if f.nose.x != 0:
        return False
    if (f.right_eye.y - f.right_eye.x) != f.nose.y:
        return False
    return True
# END REMOVE
```

# BEGIN REMOVE
# Alternate solution
return (f.right_eye.x == -f.left_eye.x and
        f.right_eye.y == f.left_eye.y and
        f.nose.x == 0 and
        f.nose.y == f.right_eye.y - f.right_eye.x)
# END REMOVE

(c) [6 points] Implement the following function.

```python
def eyes_wider(first, second):
    """Return True if the eyes of Face `first` are wider apart than the eyes of Face `second`. Otherwise return False. Also return False if either face is not proportionate. """
    if not is_proportionate(first) or not is_proportionate(second):
        return False
    return first.right_eye.x > second.right_eye.x
# END REMOVE
```

# BEGIN REMOVE
# Alternate solution
if is_proportionate(first) and is_proportionate(second):
    return first.right_eye.x > second.right_eye.x
else:
    return False
# END REMOVE