1 More Practice with for Loops

Example 1:

```python
def count_vowels(s):
    ''' (str) -> int

    Return the number of vowels in s.

    >>> count_vowels('Anna')
    2
    >>> count_vowels('UofT')
    1
    >>> count_vowels('1234')
    0
    '''
```

Example 2:

Note that \( i f \) in the all_vowels description below stands for \( i f \ and \ only \ if \). This description also means that you need to return False in all other cases.

```python
def all_vowels(s):
    ''' (str) -> bool

    Return True iff all characters in s are vowels.

    >>> all_vowels('Anna')
    False
    >>> all_vowels('Toronto')
    False
    >>> all_vowels('AI')
    True
    '''
```
2 Lists

2.1 Intro and Indexing

So far we have seen single item types: `int`, `bool`, `float`. We cannot iterate over these. We’ve also seen `strings` which are a collection of chars. We can iterate over the chars because they are a collection. It can be very useful to store other types (such as `ints`, `bools` or even a mixture of types) as collections. For example,

```python
measurement1 = 45.27
measurement2 = 45.26
measurement3 = 45.24
...
```

If we do 100 measurements, we don’t want 100 variables! We would like one variable that contains all 100 measurements. Python lets us do this with a `list`.

```python
>>> measurements = [45.27, 45.26, 45.24]
>>> measurements
[45.27, 45.26, 45.24]
```

Q. How do you think you access the value 45.27?

A.

Q. What does `measurements[3]` give?

A.

Lists can contain more than just numbers, and even have `mixed types`. For example,

```python
instructors = ["Anna", "Anya", "Joe"]
student = ["Jon Reed", "Trinity College", 812391236, 3.45]
```

Q. What does `mutable` mean?

A.

Lists are mutable:

```python
instructors[2] = "Jen"    # You can change an element
instructors.append("Anya") # You can add to a list
```

Q. What kind of thing is `append()`?

A.
2.2 List functions

Lists come with some useful functions (recall the difference between functions and methods).

Fill in the table below for the lists measurements and instructors defined above:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(instructors)</td>
<td>instructors[len(instructors) - 1]</td>
<td></td>
</tr>
<tr>
<td>max(measurements)</td>
<td>min(measurements)</td>
<td></td>
</tr>
<tr>
<td>sum(measurements)</td>
<td>min(instructors)</td>
<td></td>
</tr>
<tr>
<td>max(instructors)</td>
<td>sum(instructors)</td>
<td></td>
</tr>
</tbody>
</table>

2.3 List Methods

Q. How can we get a list of all the methods for type list?

A.

Let’s use the list L == ['a', 'b', 'c', 'd']. Then the following methods can be applied to L:

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.append('e')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.insert(2, 'new')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.insert(len(L), 'z')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.sort()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.4 for Loop over a List

The same as looping over strings:

for item in list:
    statements involving item

For example,

# Print the items in the list measurements
for value in measurements:
    print(value)
Q. What will this print?

```python
L = [1, 2, 3]
for item in L:
    item = item * 2
print(L)
```

A.

Q. Why?

A.

Q. How can we iterate through a list then **and** change the value of the list items?

A. We loop over the list *indices* and use those to change the list itself.

We loop over the indices of a list by using a function called `range()`.

Q. Now what does the following code print?

```python
L = [1, 2, 3]
for i in range(len(L)):
    L[i] = L[i] * 2
print(L)
```

A.

### 2.5 Ranges

The function `range(x, y, step)` can be used to iterate over values `x`, `x + step`, `x + 2 * step`, ..., up to and excluding `y`. If you omit `step`, the default is 1. You can also omit the first argument; the default is 0.

Q. Now what does the following code print?

```python
for i in range(10, 18, 2):
    print(i)
```

A.

We can also create a list out of the object that function `range` returns by using a function `list`. For example,

```python
>>> list(range(10, 40, 2))
[10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38]
```

The step can also be negative:

```python
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

```python
>>> list(range(40, 20, -3))
[40, 37, 34, 31, 28, 25, 22]
```