Sequences and Loops

Motivation: How to count the number of vowels in a word?
- You’re given words like 'boston', 'wellesley', 'needham', 'lynn', etc.
- Tasks:
  - count the number of vowels in a word.
  - count the number of times a certain vowel appears in a word

```python
def countAllVowels(word):
    # body here
    ?
```
```python
def countVowel(word, vowel):
    # body here
    ?
```

Reminder from last lecture: `isVowel`
```python
def isVowel(s):
    return (s == 'a' or s == 'e' or s == 'i' or s == 'o' or s == 'u' or s == 'A' or s == 'E' or s == 'I' or s == 'O' or s == 'U')
```
```python
def isVowel2(s):
    l = s.lower()
    return (l == 'a' or l == 'e' or l == 'i' or l == 'o' or l == 'u')
```

How will the function `isVowel` be useful for solving our “counting vowels” problem?
Indices: accessing characters in a string

In [1]: word = 'boston'
In [2]: word[0]
Out[2]: 'b'
In [3]: word[1]
Out[3]: 'o'
In [4]: word[2]
Out[4]: 's'
In [5]: word[3]
Out[5]: 't'
In [6]: word[4]
Out[6]: 'o'
In [7]: word[5]
Out[7]: 'n'

Notice
- 0, 1, 2, etc. are the indices (plural of index).
- Indices start at 0.
- Indices go from 0 to \( \text{len}(\text{word}) - 1 \).

A possible solution: which is correct?

```python
word = 'boston'
counter = 0
if isVowel(word[0]):
    counter += 1
if isVowel(word[1]):
    counter += 1
if isVowel(word[2]):
    counter += 1
if isVowel(word[3]):
    counter += 1
if isVowel(word[4]):
    counter += 1
print counter
```

Does our solution work for all words?

- Do you think the right solution from 6-6 will work for all words: 'wellesley', 'needham', 'lynn', etc.?
- What happens if we use an index that's greater than or equal to the length of the word?

```python
In [1]: word = 'lynn'
In [2]: word[4]
IndexError: string index out of range
```

Creating a list of indices with `range`

When the `range` function is given two integer arguments, it returns a list of all integers starting at the first and up to, but not including, the second.

```python
In [1]: range(0, 10)
Out[1]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```python
In [2]: range(3, 7)
Out[2]: [3, 4, 5, 6]
```

```python
In [3]: range(3, 2)
Out[3]: []
```

```python
In [4]: range(3, 3)
Out[4]: []
```

```python
In [5]: range(3)  # missing first argument defaults to 0
Out[5]: [0, 1, 2]
```
Properties of the `range` function

An optional third argument to `range` controls the step size between elements (which defaults to 1).

```
In [1]: range(1, 10, 2)
Out[1]: [1, 3, 5, 7, 9]
```

```
In [2]: range(3, 70, 10)
Out[2]: ?
```

```
In [3]: range(9, 0, -1)
Out[3]: ?
```

```
In [4]: range(9, 0, -2)
Out[4]: ?
```

```
In [5]: range(63, 0, -10)
Out[5]: ?
```

A new value type: lists

`range()` returns values of type `list`

```
In [1]: type(range(0, 10))
Out[1]: list
```

`list()` converts a string into lists of characters

```
In [2]: list("Wendy Wellesley")
Out[2]: ['W', 'e', 'n', 'd', 'y', ' ', 'W', 'e', 'l', 'l', 'e', 's', 'l', 'e', 'y']
```

Can also specify a list directly as a comma separated list of values

```
In [3]: phrase = ["a", "lovely", "autumn", "day"]
```

Back to our vowel counting problem

```
word = 'boston'
counter = 0
if isVowel(word[0]):
    counter += 1
if isVowel(word[1]):
    counter += 1
if isVowel(word[2]):
    counter += 1
if isVowel(word[3]):
    counter += 1
if isVowel(word[4]):
    counter += 1
if isVowel(word[5]):
    counter += 1
print counter
```

```
In [1]: word = 'boston'
In [2]: range(len(word))
Out[2]: [0, 1, 2, 3, 4, 5]
In [3]: word = 'wellesley'
In [4]: range(len(word))
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
In [5]: word = 'lynn'
In [6]: range(len(word))
Out[6]: [0, 1, 2, 3]
```

```
range() solves our indexing problem, by generating the correct list of indices.
```

Loops to the rescue!

```
word = 'boston'
counter = 0
if isVowel(word[0]):
    counter += 1
if isVowel(word[1]):
    counter += 1
if isVowel(word[2]):
    counter += 1
if isVowel(word[3]):
    counter += 1
if isVowel(word[4]):
    counter += 1
if isVowel(word[5]):
    counter += 1
print counter
```

```
word = 'boston'
counter = 0
for i in range(len(word)):
    if isVowel(word[i]):
        counter += 1
print counter
```

Try it out in the notebook to check that we get the same result.
Iterating Over Sequences with **for** Loops

One of the most common ways to manipulate a sequence is to perform some action for each element in the sequence. This is called **looping** or **iterating** over the elements of a sequence, and in Python is accomplished with a **for** loop.

```
for var in sequence:
    # Body of the loop
    statements using var
```

### Define `countAllVowels` and `countVowel`

- In the notebook, write definitions for `countAllVowels` and `countVowel` using for-loops
  
  - `countAllVowels("america")` should return 4
  - `countVowel("america", "a")` should return 2

#### for loops without **range**

- The **range** function provides a list of indices

  ```python
  phrase = ["an", "autumn", "day"]
  for i in range(len(phrase)):
      print phrase[i] + '!
  
an!
autumn!
day!
  ```

- We can also loop directly over any list. The code below produces the **same output**.

  ```python
  phrase = ["an", "autumn", "day"]
  for word in phrase:
      print word + '!
  ```

#### When is it better to use **range** instead of directly looping?

- Let's modify the previous example to print both the index and the item for each item in the list.

  ```python
  for i in range(len(phrase)):
      print(str(i) + ' ' + phrase[i] + '!
  
  0 an!
  1 autumn!
  2 day!
  ```

- Notice this would NOT be possible if we directly looped over the list.
Strings and lists are both sequences

A sequence is an “abstract” type, which serves as template for “concrete” types such as string or list.

Operations in Sequences

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x in seq</code></td>
<td>True if an item of seq is equal to x</td>
</tr>
<tr>
<td><code>x not in seq</code></td>
<td>False if an item of seq is equal to x</td>
</tr>
<tr>
<td><code>seq1 + seq2</code></td>
<td>The concatenation of seq1 and seq2</td>
</tr>
<tr>
<td><code>seq*n, n*seq</code></td>
<td>n copies of seq concatenated</td>
</tr>
<tr>
<td><code>seq[i]</code></td>
<td>i’th item of seq, where origin is 0</td>
</tr>
<tr>
<td><code>seq[i:j]</code></td>
<td>slice of seq from i to j</td>
</tr>
<tr>
<td><code>seq[i:j:k]</code></td>
<td>slice of seq from i to j with step k</td>
</tr>
<tr>
<td><code>len(seq)</code></td>
<td>length of seq</td>
</tr>
<tr>
<td><code>min(seq)</code></td>
<td>smallest item of seq</td>
</tr>
<tr>
<td><code>max(seq)</code></td>
<td>largest item of seq</td>
</tr>
</tbody>
</table>

Looping over a string

```
In [1]: word = 'boston'
In [2]: counter = 0
for i in range(len(word)):
    if isVowel(word[i]):
        counter += 1
print counter
```

A sequence is an “abstract” type, which serves as template for “concrete” types such as string or list.

The Slicing operation

```
In [1]: word = 'boston'
In [2]: word[2]  # Output: 's'
In [3]: word[:3] # Output: 'bos'
In [4]: word[0:6:2] # Output: 'bso'
In [5]: word[3:10] # Output: 'ton'
In [6]: word[:3]  # Output: 'boso'
In [7]: word[0:6:2] # Output: 'notso'
```

```
In [1]: digits = range(5,10)
In [3]: digits[2:4] # Output: [7, 8]
In [4]: digits[:3] # Output: [5, 6, 7]
In [5]: digits[3:10] # Output: [8, 9]
In [6]: digits[3:7:2] # Output: [8, 9]
In [7]: digits[0:5:2] # Output: [8, 9, 7, 5]
In [8]: digits[::1] # Output: [5, 7, 9]
```

Can we avoid range in this code as well as we did in 6-15?

Rewrite the code without using range.

```
word = 'boston'
counter = 0
for i in range(len(word)):
    if isVowel(word[i]):
        counter += 1
print counter
```
How do indices work?

Indices in Python are both positive and negative. Everything outside these values will cause an IndexError.

**In [7]:** word[::-1]
**Out[7]:** 'notsob'

This means: start at 0 until the end of sequence with step -1. And it works because of the negative indices.

IMPORTANT: Nested loops

```python
verse = "Two roads diverged in a yellow wood"
for word in verse.split():
    counter = 0
    for letter in word:
        if isVowel(letter):
            counter += 1
    print 'Vowels in', word, '->', counter
```

Vowels in Two -> 1
Vowels in roads -> 2
Vowels in diverged -> 3
Vowels in in -> 1
Vowels in a -> 1
Vowels in yellow -> 2
Vowels in wood -> 2

Graphics Examples with for Loops

We can use for loops in conjunction with the range function and the cs1graphics module to create complex pictures with repeated subpatterns that are transformed by scaling, rotation, etc.

Each of these pictures is created by using a loop to create multiple copies of a simple shape (ellipse, circle, square) that differ in their rotation, size, and/or color.
A simple flower

```python
from cslgraphics import *

paper = Canvas(800, 700, 'skyBlue', 'Rotation Designs')

for i in range(12):
    petal = Ellipse(150, 30)
    petal.setFillColor('yellow')
    # 75 is one half 150; try -95 instead
    petal.adjustReference(-75,0)
    petal.rotate(i*30)  # 30 is 360/12
    petal.moveTo(200, 200)
    paper.add(petal)
```

Abstracting over our flower with `makeFlower`

Define a function `makeFlower` that takes as arguments (1) the number of petals (2) the color of each petal (3) the width of each petal and (4) the height of each petal and returns a `Layer` object with an appropriately constructed flower object.

```python
makeFlower(12, 'yellow',30, 150)
makeFlower(10, 'brown', 90, 150)
makeFlower(30, 'magenta', 20, 150)
```

A simple nautilus shell

```python
for i in range(50):
    ring = Circle(100)
    ring.setFillColor('white')
    # adjust by radius size
    ring.adjustReference(-100, 0)
    # 10 is just a small amount
    ring.rotate(i*10)  # 0.95 just makes it smaller by a tad
    ring.scale(0.95**i)
    ring.moveTo(500, 150)
    paper.add(ring)
```

Parameterize it: `makeNautilus`

```python
shell = makeNautilus(50, 100, 10, .95, 'pink')
shell.moveTo(200,200)
paper.add(shell)
```

```python
def makeNautilus(num, size, angle, shrink, color):
    nautilus = Layer()
    for i in range(num):
        ring = Circle(size)
        ring.setFillColor(color)
        ring.adjustReference(-size, 0)
        ring.rotate(i*angle)
        ring.scale(shrink**i)
        nautilus.add(ring)
    return nautilus
```
Make it fancy:
makeColorfulNautilus

colorfulShell = makeColorfulNautilus(50,100,10,.95,['pink','blue','green','magenta'])
colorfulShell.moveTo(400,400)
paper.add(colorfulShell)

def makeColorfulNautilus(num, size, angle, shrink, colorList):
    nautilus = Layer()
    for i in range(num):
        ring = Circle(size)
        ring.setFillColor(colorList[i % len(colorList)])
        ring.adjustReference(-size, 0)
        ring.rotate(i*angle)
        ring.scale(shrink**i)
        nautilus.add(ring)
    return nautilus

Rotated squares

for i in range(16):
    s = Square(200, Point(200, 500))
    s.rotate(6*i)
    s.scale(0.9**i)
    s.setFillColor(Color.randomColor())
paper.add(s)

Rose-colored squares

# Helper function
def makeColor(redFraction, greenFraction, blueFraction):
    return Color((255.0*redFraction, 255.0*greenFraction, 255.0*blueFraction))

for i in range(25):
    s = Square(10 + 10*i, Point(600,500))
    s.rotate(15*i)
    s.setDepth(i)
    s.setFillColor(makeColor(i/24.0, 0.5, 0.5))
paper.add(s)