Lecture 5: Sequences and Loops*

Today we'll talk about looping (iteration) over **sequences** such as `str` and `list`.

1. Motivating example: counting vowels

Assume we are given different words, such as "Williams", "Boston", "Eephs", "Berkshires", and so on, and want to count the number of vowels in each word, or to count how many times a particular letter occurs in the word.

How do we do that? We can utilize the predicate, `isVowel` defined below.

Predicate: `isVowel`

[1]:
```
def isVowel(char):
    """Predicate that returns true only when a letter is a vowel."""
    return char.lower() in 'aeiou'
```

2. Use indices to access elements in a string

We can access each character (or element) in a string `word` by using **indices**. These are integers from `0` up to one less than the length of `word`.

- This is a computer science way of counting—starting at 0!

Length of a sequence. Python has an in-built `len()` function that computes the length of a sequence such as a string or a list.

[10]:
```
word = "Boston"
word[0] # character at 0th index?
```

[10]:
```
'B'
```

[11]:
```
len(word) # returns length of a word
```

[11]:
```
6
```

[12]:
```
word[1] # character at 1st index?
```

[12]:
```
'o'
```

*Acknowledgement. This notebook has been adapted from the Wellesley CS111 Spring 2019 course materials (http://cs111.wellesley.edu/spring19).

[7]:
```
word[3]
```

[7]:
```
't'
```

Question: Will the following expression work?

[13]:
```
word[6]
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-13-8fed510628cf> in <module>
----> 1 word[6]


IndexError: string index out of range
```

Question: What about this one, will this work?

[16]:
```
word[-1]
```

[16]:
```
'n'
```

Question: Can you access the character "s" by using a negative index? Write it below to test:

[17]:
```
word[-4]
```

[17]:
```
's'
```

Sequence indexing: Refer to the lecture slides to see an explanation of the indexing scheme for sequences such as lists and strings.

3. Sequential vs. Chained conditionals

How can we count the vowels in a word? We can use `isVowel` and the indices to test each character and keep track of vowels through a counter variable.

But, how do we write the conditionals to test for each character?

Scenario 1: A series of `if` statements

Can you predict the result?

[18]:
```
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)
```

2

Scenario 2: Chained conditionals

Can you predict the result?

[19]:
```
word = 'Boston'
counter = 0
if isVowel(word[0]):
    counter += 1
elif isVowel(word[1]):
    counter += 1
elif isVowel(word[2]):
    counter += 1
elif isVowel(word[3]):
    counter += 1
elif isVowel(word[4]):
    counter += 1
elif isVowel(word[5]):
    counter += 1
print(counter)
```

1

Summary. Be careful to know when to use a bunch of if statements verses if-elif-else blocks.

Try the code with another word

We always strive to write code that is generic, what will happen when we run it with a new string?

[20]:
```
word = 'Anna'
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)
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-20-d51454b9d9c9> in <module>
      9 if isVowel(word[3]):
     10     counter += 1
---> 11 if isVowel(word[4]):
     12     counter += 1
     13 if isVowel(word[5]):


IndexError: string index out of range
```

Summary. The above logic of using conditionals to manually check every letter does not generalize to arbitrary strings. We need a way to to ''run through" all the letters of a string (of some length `n`) and be able to do the same check for every character of the string.

6. `for` loops

We can solve the problem of vowel-counting using a `for` loop. A loop is an execution mechanism to repeat a series of steps for a certain number of times.

In the following example, the for block repeats itself length(word) number of times. `char` is the loop variable which first takes the value of `word[0]`, then it takes the value of `word[1]`, and so on, on the final iteration it takes the value of `word[len(word)-1]` (the last character of the string) after which the loop is completed and the control flow exits the for block.

[21]:
```
def countAllVowels(word):
    '''Returns number of vowels in the word'''
    count = 0 # initialize the counter
    for char in word: # iterate over the word one character at a time
        if isVowel(char):
            count += 1
    return count
```

[22]:
```
countAllVowels('Williams')
```

[22]: 3

[23]:
```
countAllVowels('Eephs')
```

[23]: 2

[24]:
```
# countAllVowels()  # insert a word with a lot of vowels?
```

Notice how our for loop approach works for words of any length! The for loop automatically finished after we run out of characters of `word`, even though we have not computed the length of word manually. This is the beauty of Python—it lets us iterate directly over sequences.

Tracing the loop executation. You can trace the execution of the loop, and how the variables within its body are changing using print statements that display their content during each iteration.

[25]:
```
def traceCountAllVowels(word):
    '''Traces the execution of countAllVowels function'''
    count = 0 # initialize the counter
    for char in word: # iterate over the word one character at a time
        print('char, count: ('+ char + ' , ' + str(count) +')')
        if isVowel(char):
            print('Incrementing counter')
            count += 1
    return count
```

[26]:
```
traceCountAllVowels('Williams')
```

```
char, count: (W , 0)
char, count: (i , 0)
Incrementing counter
char, count: (l , 1)
char, count: (l , 1)
char, count: (i , 1)
Incrementing counter
char, count: (a , 2)
Incrementing counter
char, count: (m , 3)
char, count: (s , 3)
```

[26]: 3

[27]:
```
traceCountAllVowels('Queue')
```

```
char, count: (Q , 0)
char, count: (u , 0)
Incrementing counter
```

```
char, count: (e , 1)
Incrementing counter
char, count: (u , 2)
Incrementing counter
char, count: (e , 3)
Incrementing counter
```

[27]: 4

Summary. As you can see, the loop variable `char` takes the value of every character in the string one by one until the last character. Inside the loop, we check if `char` is a vowel and if so we increment the counter.

7. Exercise: `countChar`

Let us do an exercise. Define a function `countChar` that takes two arguments – a character and a word – and returns the number of times that character appears in the word.

[28]:
```
# your function definition for countChar

def countChar1(char, word):
    '''Counts the number of times a character appears in a word'''
    count = 0
    for letter in word:
        if char == letter:
            count += 1
    return count
```

[29]:
```
countChar1('h', 'character')
```

[29]: 1

[30]:
```
countChar1('a', 'Alabama')  # there are 4 a's in Alabama but our function says␣
↪3, how do we fix this?
```

[30]: 3

[31]:
```
# your function definition for countChar

def countChar2(char, word):
    '''Counts the number of times a character appears in a word'''
    count = 0
    for letter in word:
        if char.lower() == letter.lower():
            count += 1
    return count
```

[32]:
```
countChar2('a', 'Alabama')
```

[32]: 4

```
[33]: countChar2('E', 'Eephs')
```

[33]: 2

Summary. When comparing two characters or strings, make sure that they are in the *same form* (in this case both are lowercase or uppercase letters.

9. `for` loops with lists

Let us use `for` loops with another sequence, lists. We can loop over lists the same way we loop over strings. The loop variable iteratively takes on the values of each item in the list, starting with the first item, then second, and finally the last item of the list.

Similar to strings, we can index lists as well. See examples:

```
[34]: phrase = ["A", "lovely", "Fall", "day"]  # define a new list
```

```
[35]: phrase[0]  # first item of a sequence is indexed at 0
```

[35]: 'A'

```
[36]: phrase[1] # second item in list phrase
```

[36]: 'lovely'

```
[39]: phrase[-1]  #predict what this does! Review how indexing in
      # sequences works in the lecture slides
```

[39]: 'day'

```
[40]: # for loop over a list
      for word in phrase:
          print(word)
```

```
A
lovely
Fall
day
```

Exercise: `wordStartEnd`

You can define a function that iterates over a given list, the same way we did with strings, and keeps track of some property, perhaps, this time for fun, let's count the number of words in the given list that start and end with the same letter.

[41]:
```
def wordStartEnd(wordList):
    '''Takes a list of words and counts the
    number of words in it that start and end
    with the same letter'''
    count = 0 #initilize counter
    for word in wordList:
        if len(word):  #why do we need this?
            if word[0].lower() == word[-1].lower():  # will this work?
                # debugging print here perhaps
                # print(word)
                count += 1

    return count
```

[42]:
```
wordStartEnd(['Exude', 'Eerie', 'Soup', 'knack', 'snack'])
```

[42]:
```
3
```

[47]:
```
wordStartEnd(['Scared', 'Sure', 'Sars', 'Viral', 'viv', 'stills'])
```

[47]:
```
3
```

Range function

Python provides an easy way to iterate over common numerical sequences through the `range` function.

In Python 3, we don't have an easy way to display the numbers stored in a range object. If we want to examine the contents of a range object, we may pass the object into the function list() which returns a list of the numbers in the range object.

[48]:
```
range(0,10)
```

[48]:
```
range(0, 10)
```

[49]:
```
type(range(0, 10))
```

[49]:
```
range
```

[50]:
```
list(range(0, 10))
```

[50]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Notice. The `range(firNum, secNum)` represents all numbers from `firNum` through `secNum - 1`. If the `firNum` is 0, we can omit. For example:

[51]:
```
list(range(10))
```

[51]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Looping over ranges. Range functions provides us with an iterable sequence, which we can loop over, just like we did with strings and list.

What are some reasons we might want to loop over a range of numbers?

[52]:
```
for i in range(1, 11):  # simple for loop that prints numbers 1-11
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

[53]:
```
for i in range(5):  # for loops to print patterns
    print('$' * i)
for j in range(5):
    print('*' * j)
```

```
$
$$
$$$
$$$$

*
**
***
****
```

Introducing `while` loops

`for` loops iterate over a pre-determined sequence and stop at the end of the sequence.

`while` loops are useful when we don't know in advance when to stop. A while loop will keep iterating until the condition in the parenthesis is satisfied and will halt if the condition fails to hold.

Example of a `while` loop that depends on user input

[54]:
```
prompt = 'Please enter your name (type quit to exit): '
name = input(prompt)

while (name.lower() != 'quit'):
    print('Hi,', name)
    name = input(prompt)

print('Goodbye')
```

```
Please enter your name (type quit to exit): Shikha
Hi, Shikha
Please enter your name (type quit to exit): Emma
Hi, Emma
Please enter your name (type quit to exit): Aamir
Hi, Aamir
Please enter your name (type quit to exit): quit
Goodbye
```

Example of a `while` loop that depends on argument value

[55]:
```
def printHalves(n):
    while n > 0:
        print(n)
        n = n//2

printHalves(100)
```

```
100
50
25
12
6
3
1
```

[56]:
```
printHalves(22)
```

```
22
11
5
2
1
```

Gotcha: infinite loop

We sometimes might by accident write an infinite loop, one that never ends (it happens to the best of us!). In these cases, use Ctrl+C (keyboard interrupt) to break out of the loop.

[57]:
```
def printHalves2(n):
    """Attempts to print positive successive halves of n.
    """
    while n > 0:
        print(n)
    n = n//2
```

NOTE: In the Notebook itself, it might not be possible sometimes to break the loop, even with Kernel -> Interrupt. In this case, close the tab and open it again.

[ ]:
```
printHalves2(22)  # not executed here because it leads to an large output
```

Testing Functions

Suppose we want to test a function we have written. There are several ways to do so. You can test using interactively python by importing the function from checking to see if it returns the correct output when called on a bunch of different values.

Testing using doctests. Python's doctest module allows you to embed your test cases and expected output directly into a functions docstring. To use the doctest module we must import it. To make sure the test cases are run when the program is run as a script from the terminal, we need to call doctest.testmod(). To make sure that the tests are not run in an interactive shell or when the functions from the module are imported, we should place the command within a guarded if `__name__ == "__main__":` block. See slides for more explanation.

Testing `isVowel` using doctests

The `doctest` module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown. There are several common ways to use doctest:

[67]:
```
def isVowel(char):
    """Predicate that returns true only when a letter is a vowel.
    >>> isVowel('d')
    False
    >>> isVowel('e')
    True
    """
    return char.lower() in 'aeiou'
```

[68]:
```
import doctest
doctest.testmod(verbose = True)

# Task:  try this out as a script and
```

```
# run from the terminal us try this out
```

```
Trying:
    isVowel('d')
Expecting:
    False
ok
Trying:
    isVowel('e')
Expecting:
    True
ok
9 items had no tests:
    __main__
    __main__.countAllVowels
    __main__.countChar1
    __main__.countChar2
    __main__.printHalves
    __main__.printHalves2
    __main__.traceCountAllVowels
    __main__.vowelWordsAccumulator
    __main__.wordStartEnd
1 items passed all tests:
   2 tests in __main__.isVowel
2 tests in 10 items.
2 passed and 0 failed.
Test passed.
```

[68]: `TestResults(failed=0, attempted=2)`