Lecture notes October 9, 2019

Practice writing and debugging functions

We looked at 2 ways to solve problem 5 on the recursion handout.

> Problem 5: Write a recursive function named countChars that will accept two parameters: one string named sentence and one char named c. The function should return the number of times that c appears in sentence.

Version 1:

We started with this code.

```
def countChars( sentence, c ):
   list = sentence.split( )
   for item in list:
      if item == c:
         answer = answer + 1
   return answer
```

Debugging is not finished

```
def countChars( sentence, c ):
    answer = 0
    list = sentence.split()
    for item in list:
        if item == c:
            answer = answer + 1
    return answer
```

Debugging Notes:

1. answer must be initialized to 0
2. split separates the string into pieces using whitespace as the separator
3. this version of the function is not recursive

Version 2:

We started with this code.

```
def countChars( sentence, c ):
  if c not in sentence:
     return 0
  if len(sentence) <= 1:
     return
  if sentence[0]==c:
     return 1 + countChars( sentence[1:-1], c)
```

Debugging is finished

```
def countChars( sentence, c ):
  if c not in sentence:
     return 0
  if len(sentence) == 1:
     return 1
  if sentence[0] == c:
     return 1 + countChars(sentence[1 : ], c)
  else:
     return countChars(sentence[1 : ], c)
```

Debugging Notes:

1. the first if statement is an "error case"
2. the second if statement is a "base case"
3. each return statement must return an int
4. the third if statement requires an else
5. this version of the function is recursive

Chapter 7 - Brief notes

String Slicing

myString[start : end]	It takes characters from start to end - 1.
myString[: end]	Start is left blank, takes characters from the beginning of the string to end - 1.
myString[start :]	End is left blank, takes characters from start to the end of the string.
myString[:]	Makes a copy of the entire string.

Slicing doesn't change the original string. It makes a new string object and copies characters into the new string.

If start >= len(myString) or start >= end, the result is an empty string.

stride (or step) is an optional 3rd parameter.
myString[start : end : stride]

String methods

myString.replace(old, new)	Creates a new string object with all occurrences of old replaced by new.
myString.replace(old, new, count)	Similar but replaces count occurrences of old by new.

Example: newString = myString.replace ("a", "e", 3)

Example: myString = myString.replace("x", "y")

find method

find (x)	Returns the index of the first occurrence of x or -1 if x was not found.
find (x, start)	Starts searching at start index.
find (x, start, end)	Starts search at start index and steps searching at end - 1.

rfind method

rfind(x) - reverse find, starts searching at the end of the string

count method

count(x)

Strings can be compared using < > <= >= == !=

Strings can be compared with the operators: in not in

Caution: You can use the operators **is** and **is not** with strings. However, these operators are looking to see if the strings are the same object (stored in the same memory location). That is usually not what you want your program to do.

Predicate methods for strings

isalnum
isdigit
islower
isupper
isspace - returns true if the string contains only whitespace (space, tab, or newline)
startswith(x)
endswith(x)

strip - creates a new string with all leading and trailing whitespace removed

split - creates a list with the string broken up into pieces based on a separator
The default separator is whitespace.