Computer Science 111

Lab 6: Strings

1 Introduction

This lab shows you all the fun things you can do with strings. **A handy list of Python string methods is attached to the back of this lab.** It is also about practicing combining your control structures.

Start up Python and start by trying out some code in the interpreter.

Operators Remember there are two string operators: `+` is for concatenating two strings together, and `*` is for creating multiple copies of strings. Try typing these commands.

```
>>>txt = "The quick brown fox jumps over lazy dogs."   #Whatever
>>>dubble = txt + txt
>>>wowza = txt  * 5
>>>dubble
>>>wowza
```

Any questions?

String indexing. You can use the indexing notation, with square brackets, to refer to sections of a string. If a string is of length 10, its indices are numbered 0 ..9.

Take a look at the notation below. If there is **one index**, it is the start position. If there are *two indices* they refer to the start and the end minus 1. If there are *three indices* they are start, end, and skip. This is exactly the same the notation for the `range()`. Try it!

```
>>> len(txt)
>>> txt[0]          # the character at index 0
>>> txt[2:5]        # the characters in index range 2..(5-1)
>>> txt[2:5:2]      # charatcers in the range, skipping by 2
>>> txt[:5]         # from 0 to (5-1)
>>> txt[5:]         # from 5 to end
>>> txt[::2]
>>> for ix in range (len(txt)):
>>>    print (ix, txt[ix]))
>>> for ch in txt:        # another way to loop without an index
>>>     print (ch)
```

Some more variations on the index notation are shown at the end of this lab. Try them out!

Note: This bracket notation **cannot** be used on the left side of any assignment statement! In fact there is no notation in Python that allows you to modify part of a string. New word for the day: ***strings are immutable***. Try it:

```
>>> txt[3]  = 'a'
```

See? It doesn't work. You get a syntax error.

String methods. There is no way in Python to modify a part of a string (because they're immutable, duh). The only way to change the value of `txt` is to create a whole new string and then re-assign it to `txt`.

There are lots of ways to make new strings by modifying old ones, using ***methods***. A method is a function, except it is attached to a variable with a dot, and it refers to the variable it is attached to. For example `turtle.forward(10)` is an example of a method: it is a function attached to the turtle variable with a dot, that moves the turtle forward according to its parameters.

String methods can also be used for boolean tests on strings and for finding substrings within a string. A handy table of some string methods appears at the end of this lab.

Try using these string methods.

```
>>>myname = "Jeffrey Amherst"                 #use your own name
>>>myame.upper()
>>>myname.title()
>>>myname                       # See? Methods don't change the original.
>>>myname = myname.lower()       # To change it, you have to re-assign it
```

Read through the big list of methods on the back page and try out some of the others to see what they can do.

Strings in loops. Programs that work with text (text editors, search engines, etc.) often work by iterating through strings and processing their components. The first task in string processing is to identify the component parts of the string: this is called *parsing*. The rest of the lab lets you practice various approaches to parsing string variables.

Point your browser to `www.cs.amherst.edu/∼ccm/cs111/lab6.py` and download a copy. The tasks for this lab are described in the file.

2 Python Strings

Notation and escape symbols.

- You can use the single quote character `'`, the double quote character `"` or triple quotes (three copies of either `'` or `"`, to delimit strings.
- You can refer to parts of a string using the index notation. Suppose you have a string `mystring = "abcdefghij"`. This string is of length 10, with indices 0 .. 9.

Index	Example	Meaning
$[s]$	mystring[4] is e	Character in position s
$[s : e]$	mystring[3 : 5] is de	Characters in range s to e-1
$[s : e : t]$	mystring[0 : 9 : 2] is acegi	Characters in range, striding by t
$[: e]$	mystring[: 5] is abcde	From 0 to e-1
$[s :]$	mystring[5 :] is fghij	From s to end of string
$-x$	mystring[-1] is j	Negative indices count back from end.
	mystring[-9] is a	
len()	len(mystring) is 10	Function for string length

String methods. You can attach a method to a string variable (or any string value) with a dot: it returns the result of calling the method using that dotted string value, perhaps with additional parameters. Strings are **immutable** – there is no method that actually changes the value of a string, it only creates a new one.

We assume that string `s` in the table below has already been initialized.

	String Methods
Syntax	Meaning
s.capitalize()	Capitalize the letter at index 0 (if there is one)
s.center(w)	Center the s in a new string of width w
s.center(w,ch)	As above, but pad with ch, not spaces
s.count(t)	Count the number of occurrences of string t in s
s.count(t, st, en)	Count occurrences of t in range (st, en)
s.endswith(x)	Return true if s ends with string x
s.endswith(x, st, en)	Return true if the part in range ends with x
s.find(t)	Return the starting position of t in s, or -1 if not found
s.find(t, st,en)	Look for t in this range
s.rfind()	Like find, but returns the ending (rightmost) position
s.isalnum()	True if s is nonempty and every character is alphanumeric
s.isalpha()	True if s is nonempty and every character is alphabetic
s.isdeicmal()	True if s is nonempty and every character is a digit
s.islower()	True if s is nonempty, has at least one letter, and all letters are lowercase
s.isnumeric()	True if nonempty and all characters are digits
s.isspace()	True if nonempty and all characters are whitespace
	including space, tab, newline
s.isupper()	Can you guess?
s.join (a,b,c...)	Concatenates a,b,c ...
s.lower()	Lower-case all the letters
s.replace(t,r)	Replace all substrings t with r
s.replace(t,r,n)	Replace at most n copies of t with r
s.startswith(t)	True if s starts with t
s.strip()	Remove leading and trailing whitespace
s.swapcase()	Upper to lower and vice versa
s.title()	Capitalize the first letter of each word
s.upper()	Uppercase all the letters