1 Strings

1.1 Creating String Literals

We have seen how to create string literals, by placing some text between pairs of single quotes or pairs of double quotes, e.g. “hi” and ’ho’. Once a string has been created in Python, Python “forgets” which quotes were used to delimit it. In other words, the quotes are just used to create it and don’t determine how the string is stored. For example, if we type

\[
\text{myString} = "hi"
\]

at the Python interpreter prompt (i.e. we use double quotes), then type

\[
\text{myString}
\]

it will print out ’hi’ (single quotes).

However, if we actually call \texttt{print}, it won’t print the quotes at all, e.g.

\[
\text{print myString}
\]

results in hi.

1.1.1 Escape Characters

Strings can have special characters like tabs and carriage returns encoded with escape sequences. Escape sequences begin with a backslash. Table 1
shows a few that are useful when reading text that has been read in from a file.

Table 1: Selected Escape Sequences

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>t</td>
<td>tab</td>
</tr>
<tr>
<td>n</td>
<td>newline (return in Unix)</td>
</tr>
<tr>
<td>r</td>
<td>carriage return</td>
</tr>
</tbody>
</table>

An escape sequence is considered as a single character – e.g. `len('h\th')` is 3.
Carriage returns, newlines, tabs and spaces are all considered whitespace.

1.2 String Operations

We have seen how to concatenate two strings, e.g. “hi” + “ho” is “hiho”.
There are several other operators that apply to strings shown in Table 2.
Here are some examples illustrating the operations (with the command typed at the Python prompt and the result that is displayed):

- "man" in 'human' ⇒ True
- "woman" not in 'human' ⇒ True
- "hi" = "ho" ⇒ 'hiho'
- "ho " * 3 ⇒ 'ho ho ho '
- 'blast'[0] ⇒ 'b'
- 'blast'[1:5] ⇒ 'last'
Table 2: String Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x in s</td>
<td>True if an item of s is equal to x, else False</td>
</tr>
<tr>
<td>x not in s</td>
<td>False if an item of s is equal to x, else True</td>
</tr>
<tr>
<td>s + t</td>
<td>the concatenation of s and t</td>
</tr>
<tr>
<td>s * n, n * s</td>
<td>n shallow copies of s concatenated</td>
</tr>
<tr>
<td>s[i]</td>
<td>ith item of s, origin 0</td>
</tr>
<tr>
<td>s[i:j]</td>
<td>slice of s from i to j (but not including the jth item)</td>
</tr>
<tr>
<td>s[i:j:k]</td>
<td>slice of s from i to j with step k (but not including the jth item)</td>
</tr>
<tr>
<td>len(s)</td>
<td>length of s</td>
</tr>
</tbody>
</table>

- 'blast'[0:5:2] ⇒ 'bat'
- len('blast') ⇒ 5
- min('blast') ⇒ 'a'
- max('blast') ⇒ 't'

And all of these examples produce identical results if they are applied to a variable containing a string, e.g. after the line

```
myString = 'blast'
```

we reproduce the same results:

- myString[0] ⇒ 'b'
- myString[1:5] ⇒ 'last'
- myString[0:5:2] ⇒ 'bat'
- len(myString) ⇒ 5

This is true because the operations are performed on the value itself – symbols just let us get to the values.
1.3 String Methods

Guess what? Strings are objects, so they have methods.¹

One of the most useful string methods is find, which searches for the first occurrence of one string within another and returns its location. Its syntax, along with that of several other string methods is in Table 3.

Table 3: Selected String Methods

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.upper()</td>
<td>returns a copy of s in upper case</td>
</tr>
<tr>
<td>s.lower()</td>
<td>returns a copy of s in lower case</td>
</tr>
<tr>
<td>s.split()</td>
<td>Returns a list of words in the string, using whitespace as the delimiter string: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace</td>
</tr>
<tr>
<td>s.split(sep)</td>
<td>Return a list of the words in the string, using sep as the delimiter string</td>
</tr>
<tr>
<td>s.split(sep, maxsplit)</td>
<td>Return a list of the words in the string, using sep as the delimiter string with at most maxsplit done</td>
</tr>
</tbody>
</table>

Here are several examples using the string methods:

- "ho ho ho".upper() ⇒ "HO HO HO"
- "ho ho ho".split() ⇒ ['ho', 'ho', 'ho']
- "ho,ho,ho".split(\',\') ⇒ ['ho', 'ho', 'ho']

¹In fact, all data types in Python are actually objects. Yes, even numeric types are objects, but there are no commonly used methods on numeric types. For the purposes of CS151, there is no need to think of numbers as objects, so we ignore that fact. New in Python 2.6 is a method called hex which returns the hexadecimal representation of a float
• "ho,ho,ho".split(’,’,1) ⇒ [’ho’, ’ho,ho’]

For a complete list of string methods, see
http://docs.python.org/library/stdtypes.html#id4

1.4 Strings Methods Return New Strings

Notice that all of these methods return a new string with the result. None of
them are changing the original string. This is because string are immutable
in Python. Once a string has been made, it will always be the same. But
that doesn’t mean we can’t just reuse a variable, e.g.

str = ’defenestration of prague’
str = str.title()

Now str contains the string ’Defenestration Of Prague’.

2 Quiz Yourself

1. How would you use the slice operation to retrieve the substring “can”
from the string “catatonic” (recall that the square brackets are used to
slice strings)?

2. How would you separate the string ”2.3e-9” into two strings – one
containing the mantissa and the other containing the exponent?

3. Familiarize yourself with the String Methods at http://docs.python.org/library/stdtypes.html
How would you remove the trailing spaces from a string?

4. How would you check to see if all characters in a string are numbers?

5. How would you count the number of times “ACCT” appears in “AC-
CTTGGCACCT”?