8 Strings

8.1 String operations

8.1.1 Equality

Two strings are equal if they have *exactly* the same contents, meaning that they are both the same length and each character has a one-to-one positional correspondence. Many other languages compare strings by identity instead; that is, two strings are considered equal only if they occupy the same space in memory. Python uses the `is` operator[1] to test the identity of strings and any two objects in general.

Examples:

```
>>> a = 'hello'; b = 'hello' # Assign 'hello' to a and b.
>>> a == b                   # check for equality
True
>>> a == 'hello'             #
True
>>> a == "hello"             # (choice of delimiter is unimportant)
True
>>> a == 'hello '            # (extra space)
False
>>> a == 'Hello'             # (wrong case)
False
```

8.1.2 Numerical

There are two quasi-numerical operations which can be done on strings -- addition and multiplication. String addition is just another name for concatenation. String multiplication is repetitive addition, or concatenation. So:

```
>>> c = 'a'
>>> c + 'b'
'ab'
>>> c * 5
'aaaaa'
```

8.1.3 Containment

There is a simple operator 'in' that returns True if the first operand is contained in the second. This also works on substrings

1 Chapter 12.7 on page 63

```
>>> x = 'hello'
>>> y = 'ell'
>>> x in y
False
>>> y in x
True
```

Note that 'print x in y' would have also returned the same value.

8.1.4 Indexing and Slicing

Much like arrays in other languages, the individual characters in a string can be accessed by an integer representing its position in the string. The first character in string s would be s[0] and the nth character would be at s[n-1].

```
>>> s = "Xanadu"
>>> s[1]
'a'
```

Unlike arrays in other languages, Python also indexes the arrays backwards, using negative numbers. The last character has index -1, the second to last character has index -2, and so on.

```
>>> s[-4]
'n'
```

We can also use "slices" to access a substring of s. s[a:b] will give us a string starting with s[a] and ending with s[b-1].

```
>>> s[1:4]
'ana'
```

None of these are assignable.

```
>>> print s
>>> s[0] = 'J'
Traceback (most recent call last):
File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>> s[1:3] = "up"
Traceback (most recent call last):
File "<stdin>", line 1, in ?
TypeError: object does not support slice assignment
>>> print s
```

Outputs (assuming the errors were suppressed):

```
Xanadu
Xanadu
```

Another feature of slices is that if the beginning or end is left empty, it will default to the first or last index, depending on context:

```
>>> s[2:]
'nadu'
```

```
>>> s[:3]
'Xan'
>>> s[:]
'Xanadu'
```

You can also use negative numbers in slices:

```
>>> print s[-2:]
'du'
```

To understand slices, it's easiest not to count the elements themselves. It is a bit like counting not on your fingers, but in the spaces between them. The list is indexed like this:

```
Element:     1     2     3     4
Index:    0     1     2     3     4
         -4    -3    -2    -1
```

So, when we ask for the [1:3] slice, that means we start at index 1, and end at index 3, and take everything in between them. If you are used to indexes in C or Java, this can be a bit disconcerting until you get used to it.

8.2 String constants

String constants can be found in the standard string module such as; either single or double quotes may be used to delimit string constants.

8.3 String methods

There are a number of methods or built-in string functions:

- **capitalize**
- **center**
- **count**
- decode
- encode
- endswith
- **expandtabs**
- **find**
- **index**
- **isalnum**
- **isalpha**
- **isdigit**
- **islower**
- **isspace**
- **istitle**
- **isupper**
- **join**
- **ljust**

- **lower**
- **lstrip**
- **replace**
- **rfind**
- **rindex**
- **rjust**
- **rstrip**
- **split**
- **splitlines**
- startswith
- **strip**
- **swapcase**
- **title**
- translate
- **upper**
- zfill

Only emphasized items will be covered.

8.3.1 is*

isalnum(), isalpha(), isdigit(), islower(), isupper(), isspace(), and istitle() fit into this category.

The length of the string object being compared must be at least 1, or the is* methods will return False. In other words, a string object of len(string) == 0, is considered "empty", or False.

- **isalnum** returns True if the string is entirely composed of alphabetic and/or numeric characters (i.e. no punctuation).
- **isalpha** and **isdigit** work similarly for alphabetic characters or numeric characters only.
- **isspace** returns True if the string is composed entirely of whitespace.
- **islower** , **isupper** , and **istitle** return True if the string is in lowercase, uppercase, or titlecase respectively. Uncased characters are "allowed", such as digits, but there must be at least one cased character in the string object in order to return True. Titlecase means the first cased character of each word is uppercase, and any immediately following cased characters are lowercase. Curiously, 'Y2K'.istitle() returns True. That is because uppercase characters can only follow uncased characters. Likewise, lowercase characters can only follow uppercase or lowercase characters. Hint: whitespace is uncased.

Example:

```
>>> '2YK'.istitle()
False
>>> 'Y2K'.istitle()
True
>>> '2Y K'.istitle()
True
```

8.3.2 Title, Upper, Lower, Swapcase, Capitalize

Returns the string converted to title case, upper case, lower case, inverts case, or capitalizes, respectively.

The **title** method capitalizes the first letter of each word in the string (and makes the rest lower case). Words are identified as substrings of alphabetic characters that are separated by non-alphabetic characters, such as digits, or whitespace. This can lead to some unexpected behavior. For example, the string "x1x" will be converted to "X1X" instead of "X1x".

The **swapcase** method makes all uppercase letters lowercase and vice versa.

The **capitalize** method is like title except that it considers the entire string to be a word. (i.e. it makes the first character upper case and the rest lower case)

Example:

```
s = 'Hello, wOrLD'
print s              # 'Hello, wOrLD'
print s.title()      # 'Hello, World'
print s.swapcase()   # 'hELLO, WoRld'
print s.upper()      # 'HELLO, WORLD'
print s.lower()      # 'hello, world'
print s.capitalize() # 'Hello, world'
```

Keywords: to lower case, to upper case, lcase, ucase, downcase, upcase.

8.3.3 count

Returns the number of the specified substrings in the string. i.e.

```
>>> s = 'Hello, world'
>>> s.count('o') # print the number of 'o's in 'Hello, World' (2)
2
```

Hint: .count() is case-sensitive, so this example will only count the number of lowercase letter 'o's. For example, if you ran:

```
>>> s = 'HELLO, WORLD'
>>> s.count('o') # print the number of lowercase 'o's in 'HELLO, WORLD' (0)
0
```

8.3.4 strip, rstrip, lstrip

Returns a copy of the string with the leading (lstrip) and trailing (rstrip) whitespace removed. strip removes both.

```
>>> s = '\t Hello, world\n\t '
>>> print s
        Hello, world

>>> print s.strip()
Hello, world
>>> print s.lstrip()
Hello, world
```

```
        # ends here
>>> print s.rstrip()
        Hello, world
```

Note the leading and trailing tabs and newlines.

Strip methods can also be used to remove other types of characters.

```
import string
s = 'www.wikibooks.org'
print s
print s.strip('w')                  # Removes all w's from outside
print s.strip(string.lowercase)     # Removes all lowercase letters from outside
print s.strip(string.printable)     # Removes all printable characters
```

Outputs:

```
www.wikibooks.org
.wikibooks.org
.wikibooks.
```

Note that string.lowercase and string.printable require an import string statement

8.3.5 ljust, rjust, center

left, right or center justifies a string into a given field size (the rest is padded with spaces).

```
>>> s = 'foo'
>>> s
'foo'
>>> s.ljust(7)
'foo    '
>>> s.rjust(7)
'    foo'
>>> s.center(7)
'  foo  '
```

8.3.6 join

Joins together the given sequence with the string as separator:

```
>>> seq = ['1', '2', '3', '4', '5']
>>> ' '.join(seq)
'1 2 3 4 5'
>>> '+'.join(seq)
'1+2+3+4+5'
```

map may be helpful here: (it converts numbers in seq into strings)

```
>>> seq = [1,2,3,4,5]
>>> ' '.join(map(str, seq))
'1 2 3 4 5'
```

now arbitrary objects may be in seq instead of just strings.

8.3.7 find, index, rfind, rindex

The find and index methods return the index of the first found occurrence of the given subsequence. If it is not found, find returns -1 but index raises a ValueError. rfind and rindex are the same as find and index except that they search through the string from right to left (i.e. they find the last occurrence)

```
>>> s = 'Hello, world'
>>> s.find('l')
2
>>> s[s.index('l'):]
'llo, world'
>>> s.rfind('l')
10
>>> s[:s.rindex('l')]
'Hello, wor'
>>> s[s.index('l'):s.rindex('l')]
'llo, wor'
```

Because Python strings accept negative subscripts, index is probably better used in situations like the one shown because using find instead would yield an unintended value.

8.3.8 replace

Replace works just like it sounds. It returns a copy of the string with all occurrences of the first parameter replaced with the second parameter.

```
>>> 'Hello, world'.replace('o', 'X')
'HellX, wXrld'
```

Or, using variable assignment:

```
string = 'Hello, world'
newString = string.replace('o', 'X')
print string
print newString
```

Outputs:

```
Hello, world
HellX, wXrld
```

Notice, the original variable (`string`) remains unchanged after the call to `replace` .

8.3.9 expandtabs

Replaces tabs with the appropriate number of spaces (default number of spaces per tab = 8; this can be changed by passing the tab size as an argument).

```
s = 'abcdefg\tabc\ta'
print s
print len(s)
t = s.expandtabs()
```

```
print t
print len(t)
```

Outputs:

```
abcdefg abc     a
13
abcdefg abc     a
17
```

Notice how (although these both look the same) the second string (t) has a different length because each tab is represented by spaces not tab characters.

To use a tab size of 4 instead of 8:

```
v = s.expandtabs(4)
print v
print len(v)
```

Outputs:

```
abcdefg abc a
13
```

Please note each tab is not always counted as eight spaces. Rather a tab "pushes" the count to the next multiple of eight. For example:

```
s = '\t\t'
print s.expandtabs().replace(' ', '*')
print len(s.expandtabs())
```

Output:

```
****************
16
```

```
s = 'abc\tabc\tabc'
print s.expandtabs().replace(' ', '*')
print len(s.expandtabs())
```

Outputs:

```
abc*****abc*****abc
19
```

8.3.10 split, splitlines

The **split** method returns a list of the words in the string. It can take a separator argument to use instead of whitespace.

```
>>> s = 'Hello, world'
```

```
>>> s.split()
['Hello,', 'world']
>>> s.split('l')
['He', '', 'o, wor', 'd']
```

Note that in neither case is the separator included in the split strings, but empty strings are allowed.

The **splitlines** method breaks a multiline string into many single line strings. It is analogous to split('\n') (but accepts '\r' and '\r\n' as delimiters as well) except that if the string ends in a newline character, **splitlines** ignores that final character (see example).

```
>>> s = """
... One line
... Two lines
... Red lines
... Blue lines
... Green lines
... """
>>> s.split('\n')
['', 'One line', 'Two lines', 'Red lines', 'Blue lines', 'Green lines', '']
>>> s.splitlines()
['', 'One line', 'Two lines', 'Red lines', 'Blue lines', 'Green lines']
```

8.4 Exercises

1. Write a program that takes a string, (1) capitalizes the first letter, (2) creates a list containing each word, and (3) searches for the last occurrence of "a" in the first word.
2. Run the program on the string "Bananas are yellow."
3. Write a program that replaces all instances of "one" with "one (1)". For this exercise capitalization does not matter, so it should treat "one", "One", and "oNE" identically.
4. Run the program on the string "One banana was brown, but one was green."

8.5 External links

- "String Methods" chapter[2] -- python.org
- Python documentation of "string" module[3] -- python.org

2 http://docs.python.org/2/library/stdtypes.html?highlight=rstrip#string-methods
3 http://docs.python.org/2/library/string.html