

LING115 Lecture Notes

Session #4: Methods for string and list

1. Introduction

Last time we learned strings and lists among others. Today, we will learn *methods* for strings and lists.

A method is a function that is specific to a data-type. For example, a string method is only meant to work for strings and not for numbers, for example. We refer to a method for an object of a particular by `<object>.<method>` as in the following example:

```
>>> a='this is a string'  
>>> a.split(' ')
```

In the first line, we created a string object and stored it in a variable named `a`. In the next line, we used a string method called `split`, which splits a string by the delimiter we specify inside the parentheses.

There are many methods that Python provides for each data type and Python is constantly being updated, introducing a number of new methods every time it is updated. It is hard to remember them all. Enter `dir(<object>)` (e.g. `dir('abc')`, where `'abc'` is a string object) in the Python interactive mode to see a list of methods that you can use with the object.

2. String methods

Below are some useful string methods. For a more comprehensive list of string methods, see 6.6.1 of <http://docs.python.org/library/stdtypes.html>.

2.1. strip

Text may include control characters: characters that control how the text must be printed in the output devices such as a screen or a printer, but which do not count as written symbols. Example control characters that we will frequently encounter include tab (`\t`), line feed (`\n`), carriage return (`\r`).

It is sometimes necessary to remove control characters from text. Suppose we wanted to extract words from lines that looked like the following on the terminal screen:

```
This is the only sentence in the first paragraph.  
This is the first sentence in the second paragraph.
```

The first line, for example, is really a sequence of characters that looks like the following:

```
\tThis is the only sentence in the first paragraph.\n
```

We can remove such unwanted characters from string using the following methods in Python.

`<string>.strip(<characters>)`

Return a copy of `<string>` with all *leading* and *trailing* characters specified in `<characters>` removed. The stripping process applies in both directions: left-to-right and right-to-left. The process in each direction stops as soon as a character not specified in `<characters>` is encountered. Characters specified in `<characters>` must be enclosed in single quotes as in `'\t\n'`. If `<characters>` is omitted, white-space characters are removed: blank space plus the control characters mentioned above.

```
>>> a='\tThis is the only sentence in the first paragraph.\n'
>>> a.strip('\t\n')
'This is the only sentence in the first paragraph.'

>>> a='\tThis is the only sentence in the first paragraph.\n'
>>> a.strip()
'This is the only sentence in the first paragraph.'

>>> b='ling115'
>>> b.strip('\n15')
'ing'
```

`<string>.lstrip(<characters>)`

Return a copy of `<string>` with all *leading* characters specified in `<characters>` removed. The stripping process applies left-to-right. The process stops as soon as a character not specified in `<characters>` is encountered.

```
>>> a='\tThis is the only sentence in the first paragraph.\n'
>>> a.lstrip('\t\n')
'This is the only sentence in the first paragraph.\n'

>>> b='ling115'
>>> b.lstrip('\n15')
'ing115'
```

`<string>.rstrip(<characters>)`

Return a copy of `<string>` with all *trailing* characters specified in `<characters>` removed. The stripping process applies right-to-left. The process stops as soon as a character not specified in `<characters>` is encountered.

```
>>> a='\tThis is the only sentence in the first paragraph.\n'
>>> a.rstrip('\t\n')
'\tThis is the only sentence in the first paragraph.'

>>> b='ling115'
>>> b.rstrip('\n15')
'ling'
```

2.2. split

We sometimes want to split a line of text into pieces and list its components. For example, we can split a line by white space to list all the words in the line. We can use the `split` method in Python to do this.

`<string>.split(<separator>)`

Split `<string>` by `<separator>` and return the list of words delimited by `<separator>`, where `<separator>` is a string.

```
>>> a='This is an example'
>>> a.split(' ')
>>> ['This', 'is', 'an', 'example']

>>> b='The separator can be any string'
>>> b.split('separator')
>>> ['The ', ' can be any string']
```

2.3. join

We may also want to string together words in a sequence data-type (e.g. strings, lists).

```
<separator>.join(<sequence>)
```

Concatenate the words in the sequence data-type with `<separator>` inserted between every two words, where `<separator>` can be any string as in the split method. See the following example:

```
>>> a=['This', 'is', 'an', 'example']
>>> ' '.join(a)

>>> b=['The', 'can be any string']
>>> ' separator '.join(b)

>>> c='Insertspaceaftereachletter'
>>> ' '.join(c)
```

2.4. Case

Python provides several methods related to case.

```
<string>.upper()
```

```
>>> 'convert to upper-case'.upper()
```

```
<string>.lower()
```

```
>>> 'CONVERT TO LOWER-CASE'.lower()
```

```
<string>.capitalize()
```

Convert the first character of the string in upper-case.

```
>>> 'this is a sentence'.capitalize()
```

```
<string>.isupper()
```

Return True if all cased characters are in upper-case. Otherwise, return False.

<string>.islower()

Return True if all cased characters are in lower-case. Otherwise, return False.

3. List methods

Below are some useful methods that belong to list objects. For a more comprehensive list, see section 5.1 at <http://docs.python.org/tutorial/datastructures.html> .

3.1. Adding and removing

To add a new entry to a list, we use the following methods.

<list>.append(x)

Add *x* to the end of <list>.

```
>>> a=['a','b','c']
>>> a.append('d')
```

<list>.insert(i,x)

Insert *x* before the member indexed *i*. Remember that members are indexed from zero in Python.

```
>>> a=['a','c','d']
>>> a.insert(1,'b')
```

To remove an entry from a list, we frequently use the following methods.

<list>.pop(i)

Pop the entry in <list> whose index is *i*. If index is omitted, the last entry will be removed.

```
>>> a=['a','b','c']
>>> a.pop(1)
```

```
>>> a=['a','b','c']
>>> a.pop()
```

<list>.remove(x)

Remove the first member whose value is *x*. That is, if there are multiple members with the same value, only the first member will be removed from the list. If there is no entry with the specified value, the method returns an error.

```
>>> a=['a','b','c','a','b','c']
>>> a.remove('b')
```

3.2. Indexing and counting

There are methods for identifying the index of a member of the list that matches a value and counting how many members match the value.

<list>.index(x)

Return the index of the *first* member whose value is *x*. The method returns error if none of the members matches the value.

```
>>> a=['a','b','c']
>>> a.index('b')

>>> a=['a','b','c','a','a','b']
>>> a.index('b')
```

<list>.count(x)

Return the number of members whose value is *x*.

```
>>> a=['a','b','c','a','a','b']
>>> a.count('b')
```

3.3. Sorting and reversing

We can sort the list or reverse its order using the following two methods.

<list>.sort()

```
>>> a=[7,1,2,59,23,9]
>>> a.sort()
>>> a

>>> a=['blah','ablah','good','wow','perfect']
>>> a.sort()
>>> a
```

<list>.reverse()

```
>>> a=[7,1,2,59,23,9]
>>> a.reverse()
>>> a
```

4. Exercise: using string and list methods to compile a word list

Let's practice what we have learned so far by writing a Python program that prints out a list of words extracted from standard input. Here is what we will do:

- Read lines from standard input into a list called `lines`.
- Create an empty list called `vocabulary`.
- For each `line` in `lines`, do the following:
 - Remove any leading or trailing white space characters.

- Split `line` by white space and store it as a list of words called `words`.
- For each `word` in `words`, do the following:
 - Convert `word` into lower-case.
 - If `word` is not in `vocabulary`, add it to `vocabulary`.
- For each `word` in `vocabulary`, do the following:
 - Print out `word`.

4.1. Reading in lines from standard input

To direct standard input as input to our Python program, we need to include the following two lines at the beginning of our code:

```
import sys
lines=sys.stdin.readlines()
```

We will learn what the above lines mean when we discuss Python modules and functions. But to give you a brief introduction, the first line `import sys` means we will use functions defined in a module called `sys`. You can think of a module as a file containing definitions of various functions and objects. One of the objects – we will learn about this later – defined in `sys` is `stdin`. Roughly speaking, the Python interpreter captures the standard input as if it were a file. You may recall the following lines from homework:

```
f=open(filename)
lines=f.readlines()
```

In `f=open(filename)`, we are opening a file with `filename` and creating a file object called `f`. Like the string methods we saw above, file objects have methods of their own. One of them is `readlines()`, which reads all the lines of the file and stores them as a list of lines.

So, `sys.stdin.readlines()` is basically equal to creating a file object that corresponds to the standard input and running the `readlines()` method to that file object. That is, read all lines in the standard input and store them as a list of lines.

4.2. Creating an empty list

An empty list is denoted `[]` in Python. So,

```
vocabulary=[]
```

4.3. Extracting words from lines

We will extract words line by line, so we start a for-loop.

```
for line in lines:
```

For each line, we should remove any leading or trailing white space characters. That is,

```
line=line.strip()
```

Note that we are overwriting the value of `line`: it now stores a line with the white space removed.

We then split the line by white space and store the list of words. That is,

```
words=line.split()
```

Alternatively, we can specify the stripping and splitting processes in a single line of code as follows:

```
words=line.strip().split()
```

Now we need to manipulate the case of each word and add it to vocabulary if it has not been already added. So we will start another for-loop nested in the for-loop mentioned above.

```
for word in words:
```

We convert word to lower-case as follows:

```
word=word.lower()
```

We then check if vocabulary already contains word or not. If not, we add it to vocabulary.

```
if not word in vocabulary:  
    vocabulary.append(word)
```

To summarize, the block of code for extracting words from lines should look like the following:

```
for line in lines:  
    words=line.strip().split()  
    for word in words:  
        word=word.lower()  
        if not word in vocabulary:  
            vocabulary.append(word)
```

4.4. Print out words in vocabulary

This should be easy:

```
for word in vocabulary:  
    print word
```

4.5. Summary

So your source code should look something like the following:

```
import sys  
lines=sys.stdin.readlines()  
vocabulary=[]  
for line in lines:  
    words=line.strip().split()  
    for word in words:  
        word=word.lower()  
        if not word in vocabulary:  
            vocabulary.append(word)  
for word in vocabulary:  
    print word
```