Chapter 8 More About Strings

8.3 Testing, Searching, and Manipulating Strings

Concept:

Python provides operators and methods for testing strings, searching the contents of strings, and getting modified copies of strings.

Testing Strings with in and not in

In Python you can use the in operator to determine whether one string is contained in another string.

The general format:

string1 in *string2*

Example:

```
text = 'Four score and seven years ago'
if 'seven' in text:
   print('The string "seven" was found.')
else:
   print('The string "seven" was not found.')
```

Output:

```
The string "seven" was found.
```

You can use also the not in operator.

Example:

```
names = 'Bill Joanne Susan Chris Juan Katie'
if 'Pierre' not in names:
   print('Pierre was not found.')
else:
   print('Pierre was found.')
```

Output:

Pierre was not found.

String Methods

String methods for performing the following types of operations:

- Testing the values of strings
- Performing various modifications
- Searching for substrings and replacing sequences of characters

The general format of a string method call:

stringvar.method(*arguments*)

The isdigit method returns true if the string contains only numeric digits. Otherwise, it returns false.

Example:

```
string1 = '1200'
if string1.isdigit():
   print(string1, 'contains only digits.')
else:
   print(string1, 'contains characters other than digits.')
```

Output:

```
1200 contains only digits.
```

Example:

```
string2 = '123abc'
if string2.isdigit():
   print(string2, 'contains only digits.')
else:
   print(string2, 'contains characters other than digits.')
```

Output:

```
123abc contains characters other than digits.
```

Table 8-1 Some string testing methods

Method	Description
`isalnum()`	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
`isalpha()`	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
`isdigit()`	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
`islower()`	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
`isspace()`	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (`\n`), and tabs (`\t`).
`isupper()`	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

Modification Methods

Although strings are immutable, meaning they cannot be modified, they do have a number of methods that return modified versions of themselves.

Table 8-2 String Modification Methods

Method	Description
`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the beginning of the string.
`lstrip (char)`	The `char` argument is a string containing a character. Returns a copy of the string with all instances of `char` that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the end of the string.
`rstrip (char)`	The `char` argument is a string containing a character. The method returns a copy of the string with all instances of `char` that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(char)`	Returns a copy of the string with all instances of `char` that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

Searching and Replacing

Programs commonly need to search for substrings, or strings that appear within other strings. For example, suppose you have a document opened in your word processor, and you need to search for a word that appears somewhere in it. The word that you are searching for is a substring that appears inside a larger string, the document.

Table 8-3 Search and replace methods

Method	Description
`endswith(substring)`	The `substring` argument is a string. The method returns true if the string ends with `substring`.
`find(substring)`	The `substring` argument is a string. The method returns the lowest index in the string where `substring` is found. If `substring` is not found, the method returns –1.
`replace(old, new)`	The `old` and `new` arguments are both strings. The method returns a copy of the string with all instances of `old` replaced by `new`.
`startswith(substring)`	The `substring` argument is a string. The method returns true if the string starts with `substring`.

The Repetition Operator

The repetition operator (*) works with strings as well.

The general format:

*string_to_copy * n*

The repetition operator creates a string that contains *n* repeated copies of *string_to_copy*.

Example:

```
my_string = 'w' * 5
```

After this statement executes, my_string will reference the string 'wwwww'.

Splitting a String

Strings in Python have a method named split that returns a list containing the words in the string.

Example:

```
my_string = 'One two three four'

word_list = my_string.split()

print(word_list)
```

Output:

```
['One', 'two', 'three', 'four']
```

By default, the split method uses spaces as separators (that is, it returns a list of the words in the string that are separated by spaces).

Example:

```
date_string = '11/26/2014'

date_list = date_string.split('/')
```

Output:

```
['11', '26', '2014']
```