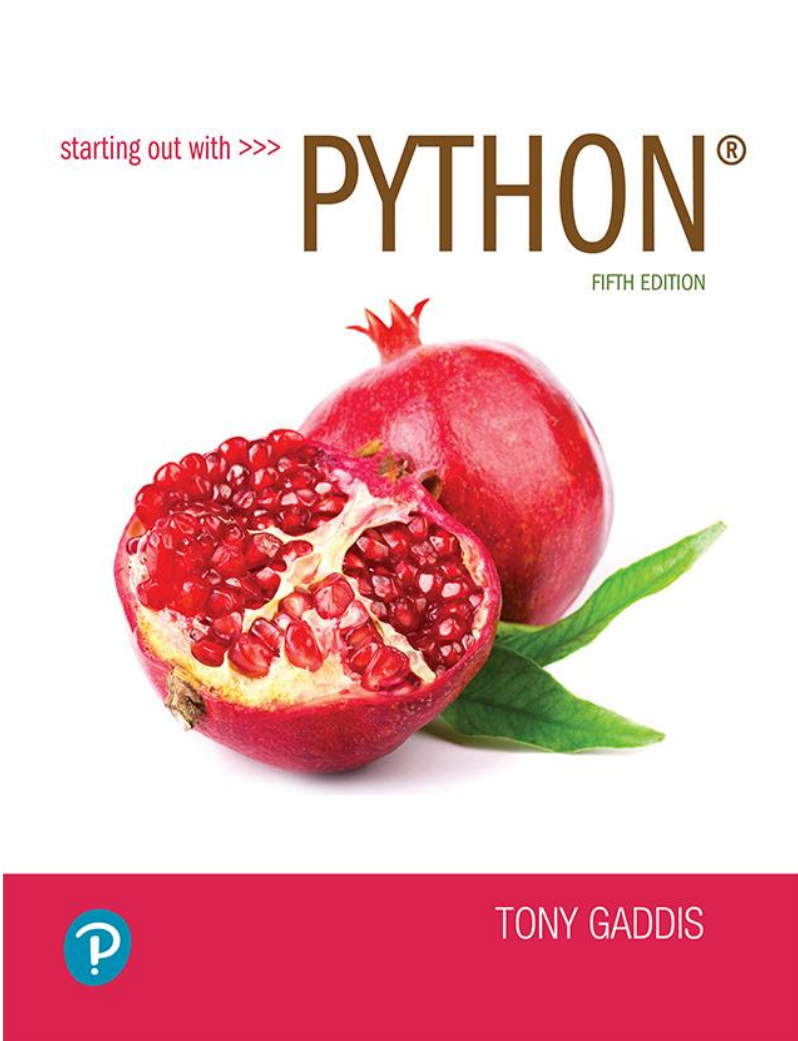


Starting out with Python

Fifth Edition



Chapter 8

More About Strings

Topics

- Basic String Operations
- String Slicing
- Testing, Searching, and Manipulating Strings

Basic String Operations

- Many types of programs perform operations on strings
- In Python, many tools for examining and manipulating strings
 - Strings are sequences, so many of the tools that work with sequences work with strings

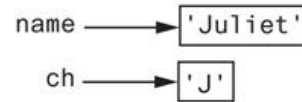
Accessing the Individual Characters in a String (1 of 4)

- To access an individual character in a string:
 - Use a `for` loop
 - Format: `for character in string:`
 - Useful when need to iterate over the whole string, such as to count the occurrences of a specific character
 - Use indexing
 - Each character has an index specifying its position in the string, starting at 0
 - Format: `character = my_string[i]`

Accessing the Individual Characters in a String (2 of 4)

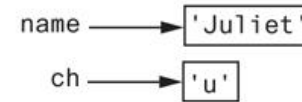
1st Iteration

```
for ch in name:  
    print(ch)
```



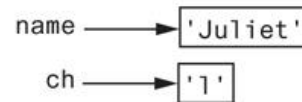
2nd Iteration

```
for ch in name:  
    print(ch)
```



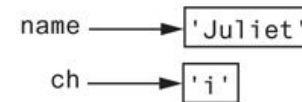
3rd Iteration

```
for ch in name:  
    print(ch)
```



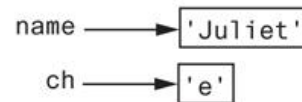
4th Iteration

```
for ch in name:  
    print(ch)
```



5th Iteration

```
for ch in name:  
    print(ch)
```



6th Iteration

```
for ch in name:  
    print(ch)
```

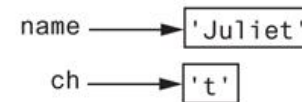


Figure 8-1 Iterating over the string 'Juliet'

Accessing the Individual Characters in a String (3 of 4)

'R o s e s a r e r e d'

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

0 1 2 3 4 5 6 7 8 9 10 11 12

Figure 8-2 String indexes

my_string → 'Roses are red'

ch → 'a'

Figure 8-3 Getting a copy of a character from a string

Accessing the Individual Characters in a String (4 of 4)

- `IndexError` exception will occur if:
 - You try to use an index that is out of range for the string
 - Likely to happen when loop iterates beyond the end of the string
- `len(string)` function can be used to obtain the length of a string
 - Useful to prevent loops from iterating beyond the end of a string

String Concatenation

- Concatenation: appending one string to the end of another string
 - Use the + operator to produce a string that is a combination of its operands
 - The augmented assignment operator += can also be used to concatenate strings
 - The operand on the left side of the += operator must be an existing variable; otherwise, an exception is raised

Strings Are Immutable (1 of 2)

- Strings are immutable
 - Once they are created, they cannot be changed
 - Concatenation doesn't actually change the existing string, but rather creates a new string and assigns the new string to the previously used variable
 - Cannot use an expression of the form
 - `string[index] = new_character`
 - Statement of this type will raise an exception

Strings Are Immutable (2 of 2)

```
name = 'Carmen'
```



Figure 8-4 The string 'Carmen' assigned to name

```
name = name + ' Brown'
```

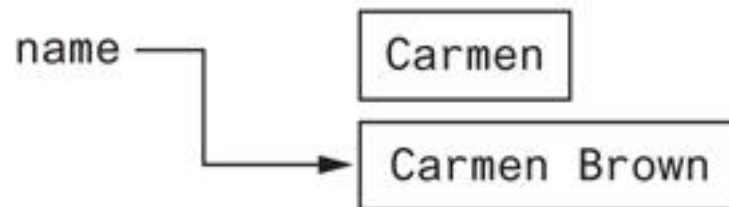


Figure 8-5 The string 'Carmen Brown' assigned to name

String Slicing

- Slice: span of items taken from a sequence, known as *substring*
 - Slicing format: `string[start : end]`
 - Expression will return a string containing a copy of the characters from `start` up to, but not including, `end`
 - If `start` not specified, 0 is used for start index
 - If `end` not specified, `len(string)` is used for end index
 - Slicing expressions can include a step value and negative indexes relative to end of string

Testing, Searching, and Manipulating Strings

- You can use the `in` operator to determine whether one string is contained in another string
 - General format: `string1 in string2`
 - `string1` and `string2` can be string literals or variables referencing strings
- Similarly you can use the `not in` operator to determine whether one string is not contained in another string

String Methods (1 of 7)

- Strings in Python have many types of methods, divided into different types of operations
 - General format:
mystring.method(arguments)
- Some methods test a string for specific characteristics
 - Generally Boolean methods, that return `True` if a condition exists, and `False` otherwise

String Methods (2 of 7)

Table 8-1 Some string testing methods

Method	Description
<code>isalnum()</code>	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
<code>isalpha()</code>	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
<code>isdigit()</code>	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
<code>islower()</code>	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.
<code>isspace()</code>	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 (<code>\n</code>), and tabs (<code>\t</code>).
<code>isupper()</code>	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.

String Methods (3 of 7)

- Some methods return a copy of the string, to which modifications have been made
 - Simulate strings as mutable objects
- String comparisons are case-sensitive
 - Uppercase characters are distinguished from lowercase characters
 - `lower` and `upper` methods can be used for making case-insensitive string comparisons

String Methods (4 of 7)

Table 8-2 String Modification Methods

Method	Description
<code>lower()</code>	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.
<code>lstrip()</code>	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (<code>\n</code>), and tabs (<code>\t</code>) that appear at the beginning of the string.
<code>lstrip(char)</code>	The <i>char</i> argument is a string containing a character. Returns a copy of the string with all instances of <i>char</i> that appear at the beginning of the string removed.
<code>rstrip()</code>	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (<code>\n</code>), and tabs (<code>\t</code>) that appear at the end of the string.
<code>rstrip(char)</code>	The <i>char</i> argument is a string containing a character. The method returns a copy of the string with all instances of <i>char</i> that appear at the end of the string removed.
<code>strip()</code>	Returns a copy of the string with all leading and trailing whitespace characters removed.
<code>strip(char)</code>	Returns a copy of the string with all instances of <i>char</i> that appear at the beginning and the end of the string removed.
<code>upper()</code>	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.

String Methods (5 of 7)

- Programs commonly need to search for substrings
- Several methods to accomplish this:
 - endswith(*substring*): checks if the string ends with *substring*
 - Returns True or False
 - startswith(*substring*): checks if the string starts with *substring*
 - Returns True or False

String Methods (6 of 7)

- Several methods to accomplish this (cont'd):
 - `find(substring)`: searches for *substring* within the string
 - Returns lowest index of the substring, or if the substring is not contained in the string, returns -1
 - `replace(substring, new_string)`:
 - Returns a copy of the string where every occurrence of *substring* is replaced with *new_string*

String Methods (7 of 7)

Table 8-3 Search and replace methods

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

The Repetition Operator

- Repetition operator: makes multiple copies of a string and joins them together
 - The * symbol is a repetition operator when applied to a string and an integer
 - String is left operand; number is right
 - General format: *string_to_copy* * *n*
 - Variable references a new string which contains multiple copies of the original string

Splitting a String (1 of 2)

- split method: returns a list containing the words in the string
 - By default, uses space as separator
 - Can specify a different separator by passing it as an argument to the `split` method

Splitting a String (2 of 2)

- Examples:

```
>>> my_string = 'One two three four'
>>> word_list = my_string.split()
>>> word_list
['One', 'two', 'three', 'four']
>>>
```

```
>>> my_string = '1/2/3/4/5'
>>> number_list = my_string.split('/')
>>> number_list
['1', '2', '3', '4', '5']
>>>
```

String Tokens (1 of 4)

- Sometimes a string contains substrings that are separated by a special character

- Example:

`'peach raspberry strawberry vanilla'`

- This string contains the substrings *peach*, *raspberry*, *strawberry*, and *vanilla*
 - The substrings are separated by the space character
 - The substrings are known as *tokens* and the separating character is known as the *delimiter*

String Tokens (2 of 4)

- Example:

`'17;92;81;12;46;5'`

- This string contains the tokens 17, 92, 81, 12, 46, and 5
- The delimiter is the ; character

String Tokens (3 of 4)

- *Tokenizing* is the process of breaking a string into tokens
- When you tokenize a string, you extract the tokens and store them as individual items
- In Python you can use the `split` method to tokenize a string

String Tokens (4 of 4)

- Examples:

```
>>> str = 'peach raspberry strawberry vanilla'
>>> tokens = str.split()
>>> tokens
['peach', 'raspberry', 'strawberry', 'vanilla']
>>>
```

```
>>> my_address = 'www.example.com'
>>> tokens = my_address.split('.')
>>> tokens
['www', 'example', 'com']
>>>
```

Summary

- This chapter covered:
 - String operations, including:
 - Methods for iterating over strings
 - Repetition and concatenation operators
 - Strings as immutable objects
 - Slicing strings and testing strings
 - String methods
 - Splitting a string