Chapter 8
Working with Sequences: Strings and Lists

STARTING OUT WITH

Python

First Edition

by
Tony Gaddis

8.1 Sequences

Concept:

A sequence is an object that holds multiple items of data, stored one after the other. You can perform operations on a sequence, to examine and manipulate the items stored in it.

8.2 Working with Strings

Concept:

Python provides several ways to access the individual characters in a string. Strings also have methods that allow you to perform operations on them.

8.2 Working with Strings

Accessing the Individual Characters in a String

Iterating Over a String with the `for` Loop

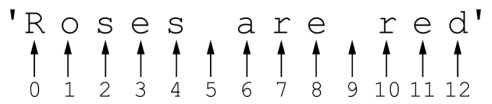
Program 8-1
(count_Ts.py)

```
# This program counts the number times
# the letter T (uppercase or lowercase)
# appears in a string.

def main():
    # Create a variable to use to hold the count.
    # The variable must start with 0.
    count = 0

    # Get a string from the user.
    my_string = raw_input('Enter a sentence: ')

    # Count the Ts.
    for ch in my_string:
        if ch == 'T' or ch  == 't':
            count += 1

    # Print the result.
    print 'The letter T appears', count, 'times.'

# Call the main function.
main()
```


8.2 Working with Strings

Accessing the Individual Characters in a String
Indexing

Figure 8-2 String indexes




```
my_string = 'Roses are red'
print my_string[0], my_string[6], my_string[10]
print my_string[-1], my_string[-2], my_string[-3]
```

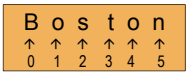

8.2 Working with Strings

Accessing the Individual Characters in a String
`IndexError` Exceptions

`IndexError` exception will if an out of range index is accessed

For example:

```
city = 'Boston'
index = 0
while index < 7:
    print city[index]
    index += 1
```



8.2 Working with Strings

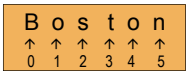
Accessing the Individual Characters in a String

The `len` Function

`len` returns the length of a sequence

For example:

```
city = 'Boston'
index = 0
while index < len(city):
    print city[index]
    index += 1
```



8.2 Working with Strings

Strings Are Immutable

Program 8-2 (concatenate.py)

```
1  # This program concatenates strings.
2
3  def main():
4      name = 'Carmen'
5      print 'The name is', name
6      name = name + ' Brown'
7      print 'Now the name is', name
8
9  # Call the main function.
10 main()
```

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

```
name = 'Carmen'
```

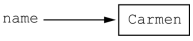
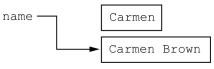


Figure 8-5 The string 'Carmen Brown' assigned to `name`

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



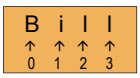
8.2 Working with Strings

Strings Are Immutable

Immutable means that once they are created they cannot be changed.

For example:

```
friend = 'Bill'
friend[0] = 'J'   # No, this will cause an error!
```



8.2 Working with Strings

String Slicing

Slice is a span of items that are taken from a sequence.

String slices are also called ***substrings***.

```
string[start : end]
```


8.2 Working with Strings

String Slicing


```
full_name = 'Patty Lynn Smith'
middle_name = full_name[6:10]   # middle_name <= 'Lynn'
first_name = full_name[:5]      # first_name <= 'Patty'
last_name = full_name[11:]      # last_name <= 'Smith'
my_string = full_name[:]        # my_string <= 'Patty Lynn Smith'
my_string = full_name[0:len(full_name)] # my_string <= 'Patty Lynn Smith'
last_name = full_name[-5:]      # last_name <= 'Smith'
```


8.2 Working with Strings

String Slicing


```
string[start : end : step]
```

```
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print letters[0:26:2]          # 'ACEGIKMOQSUWY'
```

8.2 Working with Strings

Testing Strings with `in` and `not in`

string1* `in` *string2

`in` operator determines whether one string is contained in another string

```
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.'
```

string1* `in` *string2

`not in` operator determines whether one string is contained in another string

```
name = 'Bill Joanne Susan Chris Juan Katie'
if 'Pierre' not in text:
    print 'Pierre was not found.'
else:
    print 'Pierre was found.'
```


8.2 Working with Strings

String Methods

String Testing Methods

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.

8.2 Working with Strings

String Methods

Modification Methods

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.

8.2 Working with Strings

The Repetition Operator ... *

Program 8-8 (repetition_operator.py)

```
1  # This program demonstrates the repetition operator.
2
3  def main():
4      # Print 9 rows increasing in length.
5      for count in range(1, 10):
6          print 'Z' * count
7
8      # Print 9 rows decreasing in length.
9      for count in range(8, 0, -1):
10         print 'Z' * count
11
12 # Call the main function.
13 main()
```


8.3 Lists

Concept:

A list is an object that contains multiple data items. Lists are mutable, which means that their contents can be changed during a program's execution. Lists are dynamic data structures, meaning that items may be added to them or removed from them. You can use indexing, slicing, and various methods to work with lists in a program

8.3 Lists

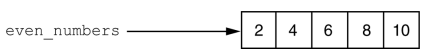
- **A list is an object that contains multiple data items.**
- **Each item that is stored in the list is called an element.**

  For example:

  ```
  even_numbers = [2, 4, 6, 8, 10]
  ```

- **Elements are enclosed in brackets and separated by commas.**

Figure 8-6 A list of integers



8.3 Lists

For example:

```
numbers = [5, 10, 15, 20]
print numbers          # displays [5, 10, 15, 20]

number range(5)        # returns a list of integers in the range of 0 up to (but not including) 5

numbers = range(1, 10, 2)  # list [1, 3, 5, 7, 9] is assigned to numbers

numbers = [0] * 5      # list [0, 0, 0, 0, 0] is assigned to numbers
```


8.3 Lists

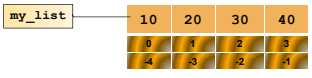
Iterating Over a List with the `for` Loop

```
numbers = [99, 100, 101, 102]
for n in numbers:
  print n
```


8.3 Lists

Indexing

```
my_list = [10, 20, 30, 40]
```

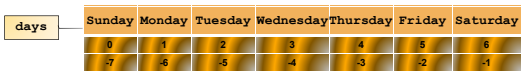



```
index = 0
while index < 4
  print my_list[index]
  index += 1
```


8.3 Lists

Slicing

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday', 'Saturday']
```




```
mid_days = days[2:5]  # from indexes 2 up to but not including 5
                      # ['Tuesday', 'Wednesday','Thursday']
```


8.3 Lists

Finding Items in a List with `in` and `not in`

Program 8-9 (in_list.py)

```
# This program demonstrates the in operator
# used with a list.

def main():
    # Create a list of product numbers.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']

    # Get a product number to search for.
    search = raw_input('Enter a product number: ')

    # Determine whether the product number is in the list.
    if search in prod_nums:
       print search, 'was found in the list.'
    else:
       print search, 'was not found in the list.'

# Call the main function.
main()
```


8.3 Lists

List are Mutable

- List are mutable – means elements can be changed
- ***`List[index]`*** can appear on the left side of an assignment operator

Program 8-10
(sales_list.py)

```
# The NUM_DAYS constant holds the number of
# days that we will gather sales data for.
NUM_DAYS = 5

def main():
    # Create a list to hold the sales
    # for each day.
    sales = [0] * NUM_DAYS

    # Create a variable to hold an index.
    index = 0

    print 'Enter the sales for each day.'

    # Get the sales for each day.
    while index < NUM_DAYS:
        sales[index] = input('Day #' + str(index+1) + ': ')
        index += 1

    # Display the values entered.
    print 'Here are the values you entered:'
    for value in sales:
        print value

# Call the main function.
main()
```

8.3 Lists

List Methods

Table 8-4 A few of the list methods

Method	Description
`append(item)`	Adds `item` to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts `item` into the list at the specified `index`. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of `item` from the list. A `ValueError` exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

8.3 Lists

The `del` Statement

- Removes an element from a specific index

For example:

```
my_list = [1, 2, 3, 4, 5]
print 'Before deletion:' my_list
del my_list[2]
print 'After deletion:' my_list
```


8.3 Lists

The `min` and `max` Functions

- **`min`** function accepts a sequence and returns the item that has the lowest value
- **`max`** function accepts a sequence and returns the item that has the highest value

8.3 Lists

Concatenating Lists

For example:

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list3 = list 1 + list2   # [1, 2, 3, 4, 5, 6, 7, 8]
```


8.3 Lists

Copying Lists

For example:

```
list1 = [1, 2, 3, 4]      # Creates a list with values
list2 = [] + list1        # Creates a copy of list1
```


8.3 Lists

Processing Lists

Program 8-15
(barista_pay.py)

```
# This program calculates the gross pay for
# each of Megan's baristas.

# NUM_EMPLOYEES is used as a constant for the
# size of the list.
NUM_EMPLOYEES = 6

def main():
    # Create a list to hold employee hours.
    hours = [0] * NUM_EMPLOYEES

    # Get each employee's hours worked.
    for index in range(NUM_EMPLOYEES):
        print 'Enter the hours worked by employee',
        hours[index] = input(str(index + 1) + ': ')

    # Get the hourly pay rate.
    pay_rate = input('Enter the hourly pay rate: ')

    # Display each employee's gross pay.
    for index in range(NUM_EMPLOYEES):
        gross_pay = hours[index] * pay_rate
        print 'Gross pay for employee', index + 1, 'is',
        print '$%.2f' % gross_pay

# Call the main function.
main()
```

8.3 Lists

Totaling the Values in a List

Program 8-16
(total_list.py)

```
# This program calculates the total of the values
# in a list.

def main():
    # Create a list.
    numbers = [2, 4, 6, 8, 10]

    # Create a variable to use as an accumulator.
    total = 0

    # Calculate the total of the list elements.
    for value in numbers:
        total += value

    # Display the total of the list elements.
    print 'The total of the elements is', total

# Call the main function.
main()
```


8.3 Lists

Averaging the Values in a List

Program 8-17
(average_list.py)

```
# This program calculates the average of the values
# in a list.

def main():
    # Create a list.
    scores = [2.5, 8.3, 6.5, 4.0, 5.2]

    # Create a variable to use as an accumulator.
    total = 0.0

    # Calculate the total of the list elements.
    for value in scores:
        total += value

    # Calculate the average of the elements.
    average = total / len(scores)

    # Display the total of the list elements.
    print 'The average of the elements is', average

# Call the main function.
main()
```


8.3 Lists

Passing a List as an Argument to a Function

Program 8-18
(total_function.py)

```
# This program uses a function to calculate the
# total of the values in a list.

def main():
    # Create a list.
    numbers = [2, 4, 6, 8, 10]

    # Display the total of the list elements.
    print 'The total is', get_total(numbers)

# The get_total function accepts a list as an
# argument returns the total of the values in
# the list.
def get_total(value_list):
    # Create a variable to use as an accumulator.
    total = 0

    # Calculate the total of the list elements.
    for num in value_list:
        total += num

    # Return the total.
    return total

# Call the main function.
main()
```


8.3 Lists

Returning a List from a Function

Program 8-19
(return_list.py)

```
# This program uses a function to create a list.
# The function returns a reference to the list.

def main():
    # Get a list with values stored in it.
    numbers = get_values()

    # Display the values in the list.
    print 'The numbers in the list are:'
    print numbers

# The get_values function gets a series of numbers
# from the user and stores them in a list. The
# function returns a reference to the list.
def get_values():
    # Create an empty list.
    values = []

    # Create a variable to control the loop.
    again = 'Y'

    # Get values from the user and add them to
    # the list.
    while again.upper() == 'Y':
        # Get a number and add it to the list.
        num = input('Enter a number: ')
        values.append(num)

        # Want to do this again?
        print 'Do you want to add another number?'
        again = raw_input('Y = yes, anything else = no: ')
        print

    # Return the list.
    return values

# Call the main function.
main()
```


8.3 Lists

Working with Lists and Files … Saving a list as a File

Program 8-21
(writelines.py)

```
# This program uses the writelines method to save
# a list of strings to a file.

def main():
    # Create a list of strings.
    cities = ['New York', 'Boston', 'Atlanta', 'Dallas']

    # Open a file for writing.
    outfile = open('cities.txt', 'w')

    # Write the list to the file.
    outfile.writelines(cities)

    # Close the file.
    outfile.close()

# Call the main function.
main()
```


8.3 Lists

Working with Lists and Files … Reading a list from a File

Program 8-23
(read_list.py)

```
# This program reads a file's contents into a list.

def main():
    # Open a file for reading.
    infile = open('cities.txt', 'r')

    # Read the contents of the file into a list.
    cities = infile.readlines()

    # Close the file.
    infile.close()

    # Strip the \n from each element.
    index = 0
    while index < len(cities):
        cities[index] = cities[index].rstrip('\n')
        index += 1

    # Print the contents of the list.
    print cities

# Call the main function.
main()
```