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 of characters, so many of the tools that work with sequences work with strings
  - Similar to the list/tuples, we can use for repetition to iterate through individual characters and indexing to access to individual characters in a string
  - Use a `for` loop
    - Format: `for` *`character`* `in` *`string`*:
  - Use indexing
    - Format: *`character`* `=` *`my_string`*`[`*`i`*`]`

Iterating Over a String with `for` Loop p431

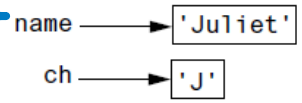
- **General Format**:

```
for variable in string:
    statement
    statement
    etc.
```

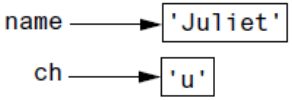
Example: Iterating through the characters in a string

```
name = 'Juliet'
for ch in name:
    print(ch)
```

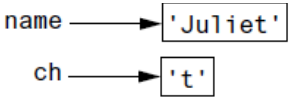
1st iteration



2nd iteration



6th iteration



Program Output

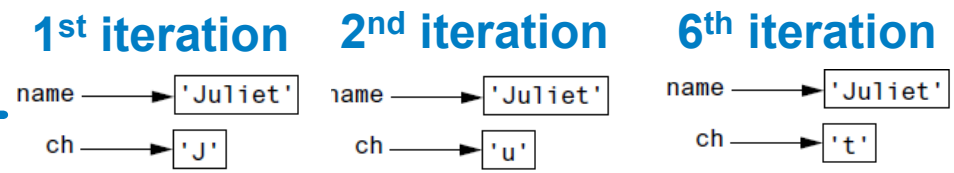
```
J
u
l
i
e
t
```

Iterating Over a String with `for` Loop p431

- **Let's try to modify the characters in a string by using for repetition**

Example:

```
name = 'Juliet'
print('Before:',name)
for ch in name:
      ch ='X'
print('After',name)
```



Because `ch` is referencing another place in the memory, setting `ch = 'X'` doesn't effect the value in the string. So we expect the name string should not change.

Program Output

```
Before: Juliet
After Juliet
```

Accessing the Individual Characters in a String (Indexing)

Another way that you can access the individual characters in a string is with an index. Each character in a string has an index that specifies its position in the string.

- Similar to list/tuples, first index is 0.
- Negative indexing can be used as well.

```
my_string = 'Roses are red'
ch = my_string[6]
```

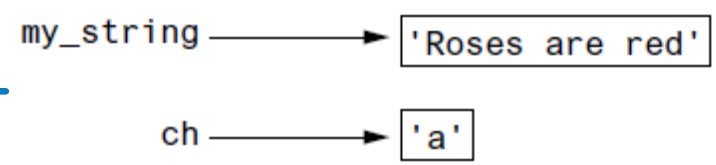
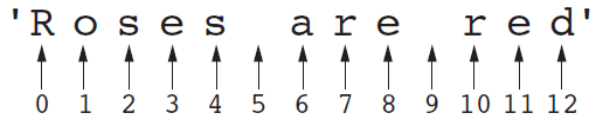


Figure 8-2 String indexes




```
my_string = 'Roses are red'
print(my_string[0], my_string[6], my_string[10])
```

Program Output

```
R a r
```

```
my_string = 'Roses are red
print(my_string[-1])
print(my_string[-3])
print(my_string[-13])
print(my_string[-15])
```

Program Output

```
d
r
R
Traceback (most recent call last):
  File "C:/Users/Fantom_E7000/sil.py",
    print(my_string[-15])
IndexError: string index out of range
```

Accessing the Individual Characters in a String (Indexing)

- **`IndexError` exception will occur if:**
  - You try to use an index that is out of range for 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

```
country = 'TURKEY'
for index in range(-1,-len(country)-1,-1):
    print(index,country[index])
```

Program Output

```
-1 Y
-2 E
-3 K
-4 R
-5 U
-6 T
```

```
country = 'TURKEY'
for index in range(len(country)):
    print(index,country[index])
```

Program Output

```
0 T
1 U
2 R
3 K
4 E
5 Y
```

String Concatenation

- **<u>Concatenation</u>: appending one string to the end of another string**
  - Use the + operator to produce a string that is a combination of its operands

```
str_1 = 'I love'
str_2 = 'you'
str_1 = str_1 + str_2
```

  - 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

```
str_1 += str_2
```

Strings Are Immutable

- **Remember: Strings are immutable,**

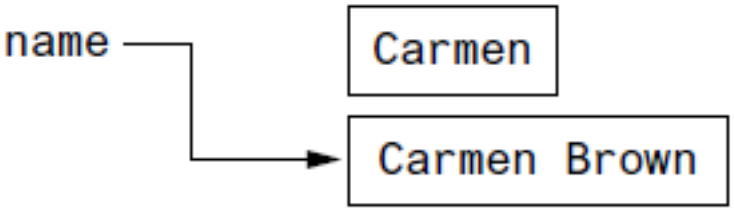
```
name = 'Carmen'
```

The string 'Carmen' assigned to name

  - 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

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

The string 'Carmen Brown' assigned to name



Strings Are Immutable

- Let's try changing a character/element in a string.

```
# Assign 'Bill' to friend.
friend = 'Bill'
# Can we change the first character to 'J'?
friend[0] = 'J'
```

Program Output: **Program run raises an exception error**

```
Traceback (most recent call last):
  File "C:/Python37/sil.py", line 9, in <module>
    friend[0] = 'J'
TypeError: 'str' object does not support item assignment
```

- Cannot use an expression of the form
- ~~*string[index] = new_character*~~ **Not Allowed!**
  - Statement of this type will raise an exception

8.1 Assume the variable `name` references a string. Write a `for` loop that prints each character in the string.

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

8.2 What is the index of the first character in a string?

Indexing goes as 0 , 1, 2, … so the first will be 0.

8.3 If a string has 10 characters, what is the index of the last character?

Indexing goes as 0 , 1, 2, … so the last will be 9 if length is 10.

8.4 What happens if you try to use an invalid index to access a character in a string?

`IndexError` exception will occur if we try to use an invalid index to access…

8.5 How do you find the length of a string?

By using `len` function. Alternatively, one can count the characters in a string as going through it b using for repetition.

8.6 What is wrong with the following code?

```
animal = 'Tiger'
animal[0] = 'L'
```

strings are immutable, using such type of statement will raise an exception error.

String Slicing

- **<u>Slice</u>: 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

String Slicing Examples


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

>>> letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> print(letters[0:26:2])
ACEGIKMOQSUWY
>>> print(letters[-3:])
XYZ
>>> print(letters[:3])
ABC
```

Testing Strings with `in` and `not in` p441

- **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

```
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.')
```

Program Output

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

- **Similarly you can use the `not in` operator to determine whether one string is not contained in another string**

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

Program Output

```
Pierre was not found.
```

String Methods

- **Recall that a method is a function that belongs to an object and performs some operation on that object.**
- **Strings in Python have numerous methods.**
  - Some of them will be covered in this course.

- **Strings in Python have many types of methods, divided into different types of operations**
  - General format:
    mystring.method(arguments)

- **We will cover some of the commonly used string methods.**
  - For a comprehensive list of string methods, see the Python documentation at www.python.org.

String Testing Methods

- **Some methods test a string for specific characteristics**
  - Generally Boolean methods, that return `True` if a condition exists, and `False` otherwise
  - The string methods shown in Table 8-1 test a string for specific characteristics.

Table 8-1 Some string testing methods

Page 442

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.

Example: Testing Methods 1/2 Page 442

`isdigit()`: Returns `true` if the string contains only numeric digits and is at least one character in length. Returns `false` otherwise.

Example 1:

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

Program Output

```
1200 contains only digits.
```

Example 2:

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

Program Output

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

Example: Testing Methods 2/2 Page 443

```
# This program demonstrates several string testing methods.

def main():
    # Get a string from the user.
    user_string = input('Enter a string: ')

    print('This is what I found about that string:')

    # Test the string.
    if user_string.isalnum():
        print('The string is alphanumeric.')
    if user_string.isdigit():
        print('The string contains only digits.')
    if user_string.isalpha():
        print('The string contains only alphabetic characters.')
    if user_string.isspace():
        print('The string contains only whitespace characters.')
    if user_string.islower():
        print('The letters in the string are all lowercase.')
    if user_string.isupper():
        print('The letters in the string are all uppercase.')

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

Program Output

```
Enter a string: abc
This is what I found about that string:
The string is alphanumeric.
The string contains only alphabetic characters.
The letters in the string are all lowercase.
```

String Modification Methods

- **Strings are immutable objects**

- **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
  - `islower` and `isupper` methods can be used for making case-insensitive string comparisons
  - `lower` and `upper` methods can be used for conversion from one case to another

String 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 lists several of these methods.

Table 8-2 String Modification Methods

Page 444

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.

Example: Modification Methods Page 444

`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.

```
letters = 'WXYZ'
print(letters, letters.lower())
```

Program Output

```
WXYZ wxyz
```

`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.

```
letters = 'abcd'
print(letters, letters.upper())
```

Program Output

```
abcd ABCD
```

Example: Modification Methods

`lstrip(), rstrip() and strip():` Returns a copy of the string with whitespace characters removed. Whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the beginning of the string.

- If any character is given as argument then it is removed.
- Each removes as l - left, r - right and s – both left and right.

```
str = "\n \t \t\nnice example....wow!!!      ";
print (str.lstrip())
print (str.rstrip())
print (str.strip())

str = "88888 nice example....wow!!!8888888";
print (str.lstrip('8'))
print (str.rstrip('8'))
print (str.strip('8'))
```

Program Output

```
nice example....wow!!!


nice example....wow!!!
nice example....wow!!!
 nice example....wow!!!8888888
88888 nice example....wow!!!
 nice example....wow!!!
```

Search and Replace Methods (cont'd.)

- **Programs commonly need to search for substrings, or strings that appear within other strings.**
  - Table 8-3 lists some of the Python string methods that search for substrings, as well as a method that replaces the occurrences of a substring with another string.

Table 8-3 Search and replace methods

Page 445

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*.

`endswith` and `startswith` Methods

- **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`

Example: `endswith` Method

- Write a program that determines the type of a given filename.
  - .txt is a Text .py is a Python file, .doc is a Word file etc.

```
filename = input('Enter the filename: ')
if filename.endswith('.txt'):
    print('This is a text file.')
elif filename.endswith('.py'):
    print('This is a Python source file.')
elif filename.endswith('.doc'):
    print('That is a Word document.')
else:
    print('Unknown file type.')
```

Program Output

```
Enter the filename: round.py
This is a Python source file.
```

Program Output

```
Enter the filename: chapter7.pdf
Unknown file type.
```

Program Output

```
Enter the filename: chapter3.doc
That is a Word document.
```

Program Output

```
Enter the filename: notes.txt
This is a text file.
```

Example: `startswith` Method

- Write a program that determines the city of a given plate number.
  - 01 - Adana, 06-Ankara, 34-Istanbul, 27-Gaziantep etc.

```
filename = input('Enter the plate number: ')
if filename.startswith('01'):
    print('This is Adana Plate.')
elif filename.startswith('06'):
    print('This is Ankara Plate.')
elif filename.startswith('27'):
    print('This is Gaziantep Plate.')
elif filename.startswith('34'):
    print('This is Istanbul Plate.')
else:
    print('Unknown Plate.')
```

Program Output

```
Enter the plate number: 34 JJ 147
This is Istanbul Plate.
```

Program Output

```
Enter the plate number: 27 GA 027
This is Gaziantep Plate.
```

Program Output

```
Enter the plate number: 02 TJK 24
Unknown Plate.
```

Program Output

```
Enter the plate number: 06 U 2567
This is Ankara Plate.
```

`find` and `replace` Methods

- **Programs commonly need to search for substrings**
- **Several methods to accomplish this (cont'd):**
  - `find(`*`substring`*`)`: searches for *`substring`* within the string
    - Returns lowest index of the substring
    - 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`*

Example: `find` Method

- Write a program that determines whether or not if a substring is found in a string.

Ex 1:

```
string = 'Four score and seven years ago'
position = string.find('seven')
if position != -1:
    print('The word "seven" was found at index', position)
else:
    print('The word "seven" was not found.')
```

Program Output

```
The word "seven" was found at index 15
```

Ex 2:

```
string = 'Four score and seven years ago'
subs=input('Enter a word to search:')
position = string.find(subs)
if position != -1:
    print('The word',subs,'was found at index', position)
else:
    print('The word',subs,'was not found.')
```

Program Outputs

```
Enter a word to search:and
The word and was found at index 11
```

```
Enter a word to search:Five
The word Five was not found.
```

```
Enter a word to search:four
The word four was not found.
```

Example: `replace` Method

- Write a program that replaces a substring in a string.

Ex 1:

```
string = 'Four score and seven years ago'
new_string = string.replace('years', 'days')
print(new string)
```

Program Output

```
Four score and seven days ago
```

Ex 2:

```
string = 'Four score and seven years ago'
print('"',string,'"',' to be modified.',sep='')
str1=input('Enter the word to be replaced:')
if string.find(str1) !=-1:
    str2=input('Enter the new word to be placed:')
    new_string = string.replace(str1, str2)
    print(new_string)
else:
    print(str1,'can not be found!')
```

Program Output

```
"Four score and seven years ago" to be modified.
Enter the word to be replaced:years
Enter the new word to be placed:days
Four score and seven days ago
```

Program Output

```
"Four score and seven years ago" to be modified.
Enter the word to be replaced:Seven
Seven can not be found!
```

The Repetition Operator *

- **<u>Repetition operator</u>: 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
  - <u>General format:</u> *`string_to_copy * n`*
  - Variable references a new string which contains multiple copies of the original string

```
>>> my_string = 'w' * 5
>>> print(my_string)
wwwww
>>> print(my_string * 3)
wwwwwwwwwwwwwwwww
>>>
>>> letters = 'abc'
>>> new = letters * 3
>>> print(new)
abcabcabc
>>>
>>> print( 4 * new )
abcabcabcabcabcabcabcabcabcabcabcabc
>>>
>>> print( 5 * 'a' )
aaaaa
```

Example: `repetition *` Operator

- Write a program that uses a repetition * operator to print a given character in triangular form.

```
# This program demonstrates the repetition operator.
# This program prints a character in triangular form.
def main():
    # Input the character to be printed
    ch=input('Enter a Character:')

    # Print 5 rows increasing in length.
    for count in range(1, 6):
        print( ch * count)

    # Print 4 rows decreasing in length.
    for count in range( 4, 0, -1):
        print( ch * count)

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

Program Output

```
Enter a Character:A
A
AA
AAA
AAAA
AAAAA
AAAA
AAA
AA
A
```

Splitting a String: `split` method

- **`split` method: returns a list containing the words in the string**
  - By default, uses space as separator

```
my_string = 'One two three four'
# Split and print the list.
print(my_string.split())
```

Program Output

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

  - Can specify a different separator by passing it as an argument to the `split` method

```
date_string = '25/10/2018'
date_list = date_string.split('/')
print('Day  :',date_list[0])
print('Month:',date_list[1])
print('Year :',date_list[2])
```

Program Output

```
Day  : 25
Month: 10
Year : 2018
```

```
# Inputting the hour from the user
hour = input('Enter time (hh:mm:ss)')
# Splitting hour with seperator ':'
hour_list = hour.split(':')
print('Hour  :',hour_list[0])
print('Minute:',hour_list[1])
print('Second:',hour_list[2])
```

Program Output

```
Enter time (hh:mm:ss)14:47:32
Hour  : 14
Minute: 47
Second: 32
```

8.11 Write code using the `in` operator that determines whether 'd' is in `mystring`.

```
if 'd' in mystring:
```

8.12 Assume the variable `big` references a string. Write a statement that converts the string it references to lowercase and assigns the converted string to the variable `little`.

```
little=big.lower()
```

8.13 Write an if statement that displays "Digit" if the string referenced by the variable `ch` contains a numeric digit. Otherwise, it should display "No digit."

```
 if ch.isdigit():
        print('Digit')
else:
        print('No digit')
```

8.14 What is the output of the following code?

```
ch = 'a'
ch2 = ch.upper()
print(ch, ch2)          a A
```

8.15 Write a loop that asks the user "Do you want to repeat the program or quit? (R/Q)". The loop should repeat until the user has entered an R or Q (either uppercase or lowercase).

```
answer='R'
while answer.upper == 'R':
        …Program will be here…
        answer=input('Do you want to repeat the program or quit? (R/Q)')
```

8.16 What will the following code display?

```
var = '$'
print(var.upper())
```

it displays $ because there is no upper form of symbols.

8.17 Write a loop that counts the number of uppercase characters that appear in the string referenced by the variable `mystring`.

```
count=0
for ch in mystring:
        if ch.isupper():
                count+=1
```

8.18 Assume the following statement appears in a program:

```
days = 'Monday Tuesday Wednesday'
```

Write a statement that splits the string, creating the following list:

```
['Monday', 'Tuesday', 'Wednesday']
days_list = days.split()
```

8.19 Assume the following statement appears in a program:

```
values = 'one$two$three$four'
```

Write a statement that splits the string, creating the following list:

```
['one', 'two', 'three', 'four']
values_list = values.split('$')
```

Review Questions

Page 453-454

Multiple Choice

2. This is the last index in a string.
   a. 1
   b. 99
   c. 0
   d. The size of the string minus one
3. This will happen if you try to use an index that is out of range for a string.
   a. A `ValueError` exception will occur.
   b. An `IndexError` exception will occur.
   c. The string will be erased and the program will continue to run.
   d. Nothing—the invalid index will be ignored.
5. This string method returns a copy of the string with all leading whitespace characters removed.
   a. `lstrip`
   b. `rstrip`
   c. `remove`
   d. `strip_leading`
6. This string method returns the lowest index in the string where a specified substring is found.
   a. `first_index_of`
   b. `locate`
   c. `find`
   d. `index_of`
7. This operator determines whether one string is contained inside another string.
   a. `contains`
   b. `is_in`
   c. `==`
   d. `in`

Algorithm Workbench

1. Assume `choice` references a string. The following `if` statement determines whether `choice` is equal to 'Y' or 'y':

   ```
   if choice == 'Y' or choice == 'y':
   ```

   Rewrite this statement so it only makes one comparison, and does not use the `or` operator. (*Hint: use either the* `upper` *or* `lower` *methods.*)

   ```
   if choice.upper() == 'Y':
   ```

2. Write a loop that counts the number of space characters that appear in the string referenced by `mystring`.

   ```
   count=0
   for c in mystring:
       if c.isspace():
           count += 1
   ```

5. Write a function that accepts a string as an argument and returns `true` if the argument starts with the substring `'https'`. Otherwise, the function should return `false`.

   ```
   def check( str ):
       return str.startswith('https')
   ```

7. Write a function that accepts a string as an argument and displays the string backwards.

   ```
   def reverse_string( str ):
       for i in range(len(str)-1,-1,-1):
           print(str[i],end='')
   ```

Programming Exercises

2. Sum of Digits in a String

Write a program that asks the user to enter a series of single-digit numbers with nothing separating them. The program should display the sum of all the single digit numbers in the string. For example, if the user enters 2514, the method should return 12, which is the sum of 2, 5, 1, and 4.

```
# Program Finds the total of the digits in a string

def main():
    # Get a string of numbers as input from the user.
    number_string = input('Enter a sequence of digits ' \
                          'with nothing separating them: ')

    # Call string_total method, and store the total.
    total = string_total(number_string)

    # Display the total.
    print('The total of the digits in the ' \
          'string you entered is', total)


# The string_total function receives a string and returns
# the total of all the digits contained in the string.
# It assumes that the string all digit characters
def string_total(string):
    # Local variables
    total = 0

    # Iterate through each character in the string.
    for i in string:
        # Add the value to the running total.
        total += int(i)

    # Return the total.
    return total

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

Program Output

```
Enter a sequence of digits with nothing separating them: 110456
The total digits is 17
```

Programming Exercises

3. Date Printer

Write a program that reads a string from the user containing a date in the form mm/dd/yyyy. It should print the date in the format March 12, 2018.

```
# This program writes a date in different format

def main():

    # Names of the months stored in a list
    month_list = ['January', 'February','March',
                  'April', 'May','June', 'July',
                  'August', 'September', 'October',
                  'November', 'December']

    # Get the date in mm/dd/yyyy format as input from the user.
    date_string = input('Enter a date in the format mm/dd/yyyy: ')

    # Split date_string.
    date_list = date_string.split('/')
    # Obtain month and day numbers.
    month_num = date_list[0]
    day = date_list[1]
    year = date_list[2]

    # Get month_name.
    month_name = month_list[int(month_num) - 1]

    # Create string for long date format.
    long_date = month_name + ' ' + day + ', ' + year

    # Display long date format.
    print(long_date)

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

Program Output

```
Enter a date in the format mm/dd/yyyy: 02/18/2019
February 18, 2019
```

Programming Exercises

4. Morse Code Converter

Morse code is a code where each letter of the English alphabet, each digit, and various punctuation characters are represented by a series of dots and dashes. **Table 8-4** shows part of the code. Write a program that asks the user to enter a string, then converts that string to Morse code.

Table 8-4 Morse code

Character	Code	Character	Code	Character	Code	Character	Code
space	*space*	6	−....	G	−−.	Q	−−.−
comma	−−..−−	7	−−...	H	....	R	.−.
period	.−.−.−	8	−−−..	I	..	S	...
question mark	..−−..	9	−−−−.	J	.−−−	T	−
0	−−−−−	A	.−	K	−.−	U	..−
1	.−−−−	B	−...	L	.−..	V	...−
2	..−−−	C	−.−.	M	−−	W	.−−
3	...−−	D	−..	N	−.	X	−..−
4	....−	E	.	O	−−−	Y	−.−
5	.....	F	..−.	P	.−−.	Z	−−..

Programming Exercises

4. Morse Code Converter`

```
# This program converts a given string to Morse Code
def main():
    # Store all Morse codes in a list
    morse_list = [' ', '--..--', '.-.-.-', '-----',
                  '.----', '..---', '...--', '....-',
                  '.....', '-....', '--...', '---..',
                  '----.', '.-', '-...', '-.-.', '-..',
                  '.', '..-.', '--.', '....', '..', '.---',
                  '-.-', '.-..', '--', '-.', '---', '.--.',
                  '--.-', '.-.', '...', '-', '..-', '...-',
                  '.--', '-..-', '-.-', '--..']
        # Get the string as input from the user.
    morse_string = input('Enter the string to be ' \
                         'converted to Morse code: ')

    # Process the entered string character by character
    for ch in morse_string:
        # Convert the character to uppercase.
        ch = ch.upper()
        # Determine the index of this character.
        if ch == ' ':
            index = 0
        elif ch == ',':
            index = 1
        elif ch == '.':
            index = 2
        elif ch == '?':
            index = 3
        elif ch == '0':
            index = 4
        elif ch == '1':
            index = 5
               ...
         elif ch == 'R':
             index = 31
         elif ch == 'S':
             index = 32
         elif ch == 'T':
             index = 33
         elif ch == 'U':
             index = 34
         elif ch == 'V':
             index = 35
         elif ch == 'W':
             index = 36
         elif ch == 'X':
             index = 37
         elif ch == 'Y':
             index = 38
         elif ch == 'Z':
             index = 39
         # Display the Morse code for this character.
         print (morse_list[index], ',', sep='', end='')

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

Table 8-4 Morse code

Character	Code	Character	Code	Character	Code	Character	Code
space	*space*	6	-....	G	--.	Q	--.-
comma	--..--	7	--...	H	....	R	.-.
period	.-.-.-	8	---..	I	..	S	...
question mark	..--..	9	----.	J	.---	T	-
0	-----	A	.-	K	-.-	U	..-
1	.----	B	-...	L	.-..	V	...-
2	..---	C	-.-.	M	--	W	.--
3	...--	D	-..	N	-.	X	-..-
4	....-	E	.	O	---	Y	-.-
5	.....	F	..-.	P	.--.	Z	--..

Program Output

```
Enter the string to be converted to Morse code: I Love you
.---, ,--,.--.,.--,..-., ,--..,.--.,...-,
```

Programming Exercises

10. Most Frequent Character: Write a program that lets the user enter a string and displays the character that appears most frequently in the string.

```
# Function displays the character that appears most frequently
# in the string. If several characters have the same highest
# frequency, displays the first character with that frequency.
def main():

    # Set up local variables
    count = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    # Receive user input.
    user_string = input('Enter a string: ')

    # Process each character in the string
    for ch in user_string:
        # Convert to upper case
        ch = ch.upper()
        # Determine which letter this character is.
        index = letters.find(ch)
        if index != -1:
            # Increase counting array for this letter.
            count[index] = count[index] + 1

    print(count)# Let's print the count list - frequency
    # Find the maximumg
    maximum=max(count)
    max_index=count.index(maximum)
    print('Most frequently character is', letters[max_index])

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

Program Output

```
Enter a string: Enter a string: Today we have started chapter 8 More about strings
[6, 1, 1, 2, 7, 0, 2, 2, 2, 0, 0, 0, 1, 3, 3, 1, 0, 6, 4, 8, 1, 1, 1, 0, 1, 0]
Most frequently character is T
```

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