\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CSC121 PYTHON Programming**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lesson 10 Text Files

# **Objectives**

In this lesson, students will learn:

- How to write code to read from a text file

- How to write code to write to a text file

- How to apply sequence operations on strings

- How to use string methods

# **10.1 Overview**

A text file is a file containing characters, structured as individual lines of text. They can be directly viewed and created using a text editor such as Notepad. In Python, we can write code to read data from text files and store output in text files.

# **10.2 Reading Text files**

To read from a text file, you need to follow three steps:

Step 1: Open the file for reading

Step 2: Read lines of text as strings

Step 3: Close the file

To open a file for reading, the built-in open function is used. Example:

input\_file = open('myfile.txt', 'r')

The statement above creates a file objects input\_file. The open function takes two string arguments. The first argument is the file name to be opened, 'myfile.txt'. The second argument, 'r', indicates that the file is to be opened for reading. If no path is provided in the first argument, the computer will search for the file in the current directory. If the file is located in a different directory, you need to specify the path. For example,

input\_file = open('C:/mypythonfiles/data/myfile.txt', 'r')

Once a file is opened for reading, you can use the readline method of the file object to read one line of text as a string. Use the readline method multiple times to read the text one line after another.

When there is no more text to read, you can close the file with the close method.

The following is an example.

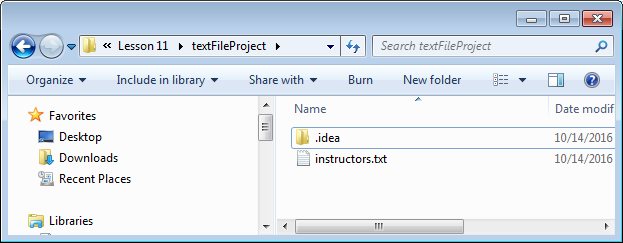
Create a new PyCharm project named textFileProject. Before we write Python code for this program, let’s use Notepad to create a text file:

Peter Chen

Joe Park

Al Molin

Name this file instructor.txt. Save this file in the folder of the textFileProject.



Add a new Python file to the project. Type the Python code in the Python file:

**def** main():  
   
 *""" Display strings read from text file """* input\_file = open(**'instructors.txt'**, **'r'**)  
 empty\_str = **''** line = input\_file.readline()   
 **while** line != empty\_str:  
 print(line)  
 line = input\_file.readline()

input\_file.close()  
  
main()

This program first opens a file for reading. A loop is used to read each line of text as a string from the file. Once a line is read, it is displayed in the console window immediately. This loop stops when there is no more text to read, i.e., when the input string is empty. The following is the output:

Peter Chen

Joe Park

Al Molin

A blank line is displayed right after each line of text. This needs some explanation. When we type text in a text file, we press the “Enter” key to go to the next line right after we have finished typing one line. In fact, a “new line” character is stored at the end of the line. We do not see this new line character because it is a hidden character. Its function is to move the cursor to the next line. When we use the readline method to read a line of text from a text file, this new line character is read and saved along with other characters in the same line. In other words, when the first line of the instructors.txt file is read, it reads and stores the characters “Peter Chen” and the new line character in the string variable line. This becomes clear if we use the len function to display the length of the string.

**def** main():  
  
 *""" Display strings read from text file """* input\_file = open(**'instructors.txt'**, **'r'**)  
 empty\_str = **''**

line = input\_file.readline()  
 **while** line != empty\_str:  
 length = len(line)  
 **print**(**"Number of characters in the string:"**, length)  
 **print**(line)  
 line = input\_file.readline()

input\_file.close()  
  
main()

Output of the program:

Number of characters in the string: 11

Peter Chen

Number of characters in the string: 9

Joe Park

Number of characters in the string: 8

Al Molin

The length of the first line read from the file is 11. There are 5 characters in “Peter” and 4 characters in “Chen”. There is one space between “Peter” and “Chen”. Together they are 10 characters. The length is 11 because there is a hidden new line character stored as the last character of the string. When this string is displayed, the characters “Peter Chen” are displayed, and the new line character actually moves the cursor to the next line. Then the print function moves the cursor down one more line just like it usually does when it finishes its task. Therefore, in the output we see a blank line displayed after each instructor’s name. Later you will see how this is fixed.

The program can be shortened by using a for loop instead of a while loop:

**def** main():  
  
 *""" Display strings read from text file """* input\_file = open(**'instructors.txt'**, **'r'**)   
 **for** line **in** input\_file:  
 print(line)  
 input\_file.close()  
  
main()

This program generates the same output as before.

If data stored in the text file are numerical values, we can convert them to numbers. For example, suppose we have four test scores stored in a text file scores.txt:

95

87

90

79

We can write the following program to read the scores and calculate their total:

**def** main():  
  
 *""" Find total of numbers stored in text file """* input\_file = open(**'scores.txt'**, **'r'**)  
 total = 0

**for** line **in** input\_file:  
 number = float(line)  
 print(number)  
 total = total + number

input\_file.close()  
 print(**"Total: "**, total)  
  
main()

The following is the output:

95.0

87.0

90.0

79.0

Total: 351.0

If we want to, we can store all the scores in a list right after they are read from the text file before we do anything with them:

**def** main():  
  
 *""" Find total of numbers stored in text file """* input\_file = open(**'scores.txt'**, **'r'**)  
 score\_list = []  
 **for** line **in** input\_file:  
 number = float(line)  
 score\_list.append(number)  
  
 total = 0  
 **for** score **in** score\_list:  
 print(score)  
 total = total + score

input\_file.close()  
 print(**"Total: "**, total)  
  
main()

An empty list score\_list is created before data are read from the text file. Right after a score is read from the file, it is appended to the list. A separate for loop is later used to display the scores and calculate their total. This program generates the same output as before.

# **10.3 Writing Text files**

To write to a text file, you need to follow three steps:

Step 1: Open the file for writing

Step 2: Write texts to the file one line at a time

Step 3: Close the file

To open a file for writing, use the open function with 'w' as the second argument:

output\_file = open('mynewfile.txt', 'w')

If the file does not exist, the file will be created to store output data. If the file already exists, it will be overwritten (starting with the first line of the file). If you want to retain old data, use 'a' as the second argument. Example:

output\_file = open('mynewfile.txt', 'a')

In this case, new data will be appended to the end of old data.

The following is an example.

**def** main():  
  
 *""" Store three departments in a text file """* output\_file = open(**'departments.txt'**, **'w'**)  
 output\_file.write(**'Psychology'**)  
 output\_file.write(**'History'**)  
 output\_file.write(**'Sociology'**)  
 output\_file.close()  
  
main()

This program writes three strings to the file ‘departments.txt’. The following is the content of the text file:

PsychologyHistorySociology

The write method of the file object is called three times to write three strings to the text file. The three strings are stored in the same line since no new line characters are written to the text file to send the cursor to the next line. If we want to separate the three strings to three lines, we need to write a new line character after every string. In Python, we can use '\n' to represent a new line character.

**def** main():  
  
 *""" Store three departments in a text file """* output\_file = open(**'departments.txt'**, **'w'**)  
  
 output\_file.write(**'Psychology'**)  
 output\_file.write(**'\n'**) *# write a new line character* output\_file.write(**'History'**)  
 output\_file.write(**'\n'**) *# write a new line character* output\_file.write(**'Sociology'**)  
 output\_file.write(**'\n'**) *# write a new line character* output\_file.close()  
  
main()

The following is the content of the text file:

Psychology

History

Sociology

Suppose the file ‘departments.txt’ already exists before we run the program and we have the following data stored there:

Computer Programming  
Data Science  
Game development  
Networking  
Web Development

If we run the same program, old data will be erased and overwritten by new data. If we want to retain the old data, we need to use ‘a’ instead of ‘w’ as the second argument in the open function.

**def** main():  
  
 *""" Store three departments in a text file """* output\_file = open(**'departments.txt'**, **'a'**)  
  
 output\_file.write(**'Psychology'**)  
 output\_file.write(**'\n'**) *# write a new line character* output\_file.write(**'History'**)  
 output\_file.write(**'\n'**) *# write a new line character* output\_file.write(**'Sociology'**)  
 output\_file.write(**'\n'**) *# write a new line character* output\_file.close()  
  
main()

The following is the content of the data file:

Computer Programming

Data Science

Game development

Networking

Web Development

Psychology

History

Sociology

We can open files for reading and files for writing in the same program. The following is an example:

**def** main():  
  
 *""" Assign letter grades to scores """* input\_file = open(**'scores.txt'**, **'r'**)  
 output\_file = open(**'scores\_grades.txt'**, **'w'**)  
  
 **for** line **in** input\_file:  
 score = float(line)  
 **if** 100 >= score >= 90:  
 grade = **'A'  
 elif** 90 > score >= 80:  
 grade = **'B'  
 elif** 80 > score >= 70:  
 grade = **'C'  
 elif** 70 > score >= 60:  
 grade = **'D'  
 elif** 60 > score >= 0:  
 grade = **'F'** output\_string = str(score) + **' '** + grade + **'\n'** output\_file.write(output\_string)  
  
 input\_file.close()  
 output\_file.close()  
  
main()

This program opens a text file to read test scores, determines corresponding letter grades and write scores and grades to another text file. This is done one score at a time. We concatenate the score, a blank space, the letter grade and the new line character into a combined string before writing it to output file. We need to use the str function str(score) to convert the score from a floating point number to a string before we can use the ‘+’ operator to concatenate it with other strings. The following is the content of the file ‘scores\_grades.txt’.

95.0 A

87.0 B

90.0 A

79.0 C

# **10.4 String Processing**

The information in a text file, as with all information, is most likely going to be searched, analyzed, and/or updated. Collectively, the operations performed on strings are called string processing. In this section, we revisit sequence operations that apply to strings, and look at additional string-specific methods.

## **String Traversal**

In an earlier lesson we learned how any sequence can be traversed, including strings. This is usually done by the use of a for loop. Example:

book = **"Introduction to Computer Science using Python"**num\_spaces = 0  
**for** chr **in** book:  
 **if** chr == **' '**:  
 num\_spaces = num\_spaces + 1  
print(**'Number of spaces:'**, num\_spaces)

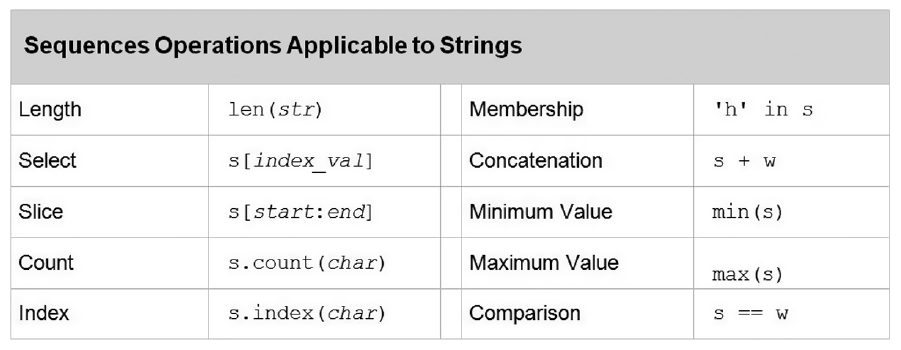
In this program, we use a for loop to examine every character in the string to count how many of them are spaces. Output:

Number of spaces: 5

## **String-Applicable Sequence Operations**

Because strings (unlike lists) are immutable, sequence-modifying operations are not applicable to strings. For example, one cannot directly add, delete, or replace characters of a string. Therefore, all string operations that “modify” a string return a new string that is a modified version of the original string.

Figure 8-4 of the textbook lists sequence operations applicable to strings:



The len function returns the length of a string. Example:

college = **"Wake Tech"**length = len(college)  
print(**"Length of string:"**, length)

Output:

Length of string: 9

We use an index to access a character in a string. Example:

college = **"Wake Tech"**third\_char = college[2]  
print(**"Third character in the string:"**, third\_char)

Output:

Third character in the string: k

Slicing returns a substring. The slice operator s[start:end] returns the substring starting with index start, up to *but not including* index end. Example:

college = **"Wake Tech"**first\_word = college[0:4]  
second\_word = college[5:9]  
print(**"First word in the string:"**, first\_word)  
print(**"Second word in the string:"**, second\_word)

Output:

First word in the string: Wake

Second word in the string: Tech

s.count(char) returns the number of occurrences of char in the string s. Example:

college = **"Wake Tech"**num\_e = college.count(**"e"**)  
print(**"Number of e's in 'Wake Tech':"**, num\_e)

Output:

Number of e's in 'Wake Tech': 2

s.index(chr) returns the index of the first occurrence of chr in s. Example:

college = **"Wake Tech"**first\_e = college.index(**"e"**)  
print(**"Index number of first e in 'Wake Tech':"**, first\_e)

Output:

Index number of first e in 'Wake Tech': 3

The in operator can be used to test whether a substring is in a string. Example:

college = **"Wake Tech"  
if 'e' in** college:  
 print(**"The letter e is found in 'Wake Tech'"**)  
**else**:  
 print(**"The letter e is not found in 'Wake Tech'"**)

Output:

The letter e is found in 'Wake Tech'

The + operator can be used to concatenate two strings. Example:

college = **"Wake Tech"**state = **"NC"**college\_state = college + **", "** + state  
print(college\_state)

Output:

Wake Tech, NC

The min function as applied to strings returns the smallest character based on the underlying Unicode encoding. All lowercase letters have larger Unicode values than all uppercase letters. Example:

college = **"WakeTech"**min\_char = min(college)  
print(**"Character in 'WakeTech' with smallest Unicode value:"**, min\_char)

Output:

Character in 'WakeTech' with smallest Unicode value: T

The max function returns the largest character based on the underlying Unicode encoding. Example:

college = **"WakeTech"**max\_char = max(college)  
print(**"Character in 'WakeTech' with largest Unicode value:"**, max\_char)

Output:

Character in 'WakeTech' with largest Unicode value: k

The == operator tests whether two strings have exactly the same characters. Example:

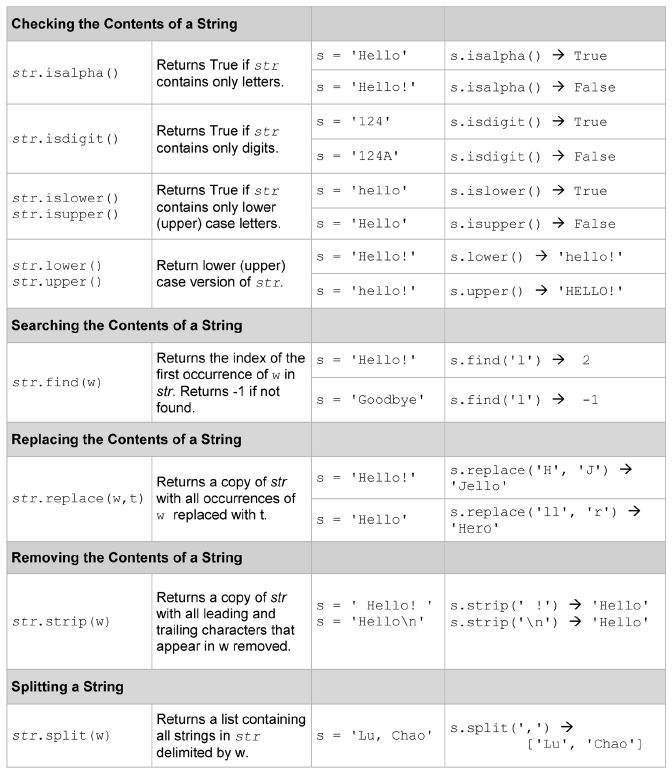
college1 = **"Wake Tech"**college2 = **"WAKE TECH"  
if** college1 == college2:  
 print(**"'Wake Tech' and 'WAKE TECH' are same string."**)  
**else**:  
 print(**"'Wake Tech' and 'WAKE TECH' are different strings."**)

Output:

'Wake Tech' and 'WAKE TECH' are different strings.

## **String Methods**

There are a number of methods specific to strings in addition to the general sequence operations. Figure 8-5 of the textbook lists these methods.



Sometimes we want to check every character in string to see whether they all fulfill certain requirements. For example, to perform error checking on an entered credit card number, we may want to check whether every character in the string is a digit. Since checking for uppercase/lowercase and digit characters is common in programming, Python provides string methods isalpha, isdigit, isupper, and islower (among others).

The method isdigit returns True if and only if each character in the string is a digit and the string is at least one character in length. False is returned otherwise. Example:

card\_num = input(**"Please enter your 16-digit card number: "**)  
**while not** card\_num.isdigit():  
 print(**"Invalid card number."**)  
 card\_num = input(**"Please enter your 16-digit card number: "**)  
print(**"Card number entered:"**, card\_num)

Sample test run:

Please enter your 16-digit card number: 123456781234rtyu

Invalid card number.

Please enter your 16-digit card number: 1234567812345678

Card number entered: 1234567812345678

The method isalpha returns True if and only if each character in the string is an alphabetic letter and the string is at least one character in length. False is returned otherwise. Example:

string1 = input(**"Please enter a string: "**)  
**if** string1.isalpha():  
 print(**"Every character in the string is a letter."**)  
**else**:  
 print(**"Not every character in the string is a letter."**)

Sample test run:

Please enter a string: Raleigh

Every character in the string is a letter.

Another sample test run:

Please enter a string: excellent!

Not every character in the string is a letter.

The isalpha method returns False because ‘!’ is not a letter.

Another sample test run:

Please enter a string: Wake Tech

Not every character in the string is a letter.

The isalpha method returns False because the space character is not a letter.

The method islower returns True if all of the alphabetic letters in the string are lowercase and the string contains at least one alphabetic letter. False is returned otherwise. Example:

string1 = input(**"Please enter a string: "**)  
**if** string1.islower():  
 print(**"This string has at least one letter and every letter is lowercase."**)  
**else**:  
 print(**"This string has no letters or some letters are not lowercase."**)

Sample test run:

Please enter a string: csc121

This string has at least one letter and every letter is lowercase.

Another sample test run:

Please enter a string: Csc121

This string has no letters or some letters are not lowercase.

Another sample test run:

Please enter a string: 123456

This string has no letters or some letters are not lowercase.

The method isupper returns True if all of the alphabetic letters in the string are uppercase and the string contains at least one alphabetic letter. False is returned otherwise. Example:

string1 = input(**"Please enter a string: "**)  
**if** string1.isupper():  
 print(**"This string has at least one letter and every letter is uppercase."**)  
**else**:  
 print(**"This string has no letters or some letters are not uppercase."**)

Sample test run:

Please enter a string: CSC121

This string has at least one letter and every letter is uppercase.

Another sample test run:

Please enter a string: cSc121

This string has no letters or some letters are not uppercase.

Another sample test run:

Please enter a string: 123456

This string has no letters or some letters are not uppercase.

The methods upper and lower return the uppercase and lowercase versions of the string, respectively. Example:

college = **"wake Tech"**college\_upper = college.upper()  
print(**"All uppercase:"**, college\_upper)  
college\_lower = college.lower()  
print(**"All lowercase:"**, college\_lower)

Sample test run:

All uppercase: WAKE TECH

All lowercase: wake tech

The method find returns the starting index of the first occurrence of a target substring. If the target substring is not found, -1 is returned. Example:

state = **"Mississippi"**first\_s = state.find(**'s'**)  
print(**"Index of first 's':"**, first\_s)  
first\_k = state.find(**'k'**)  
print(**"Index of first 'k':"**, first\_k)  
first\_iss = state.find(**'iss'**)  
print(**"Index of first 'iss':"**, first\_iss)

Sample test run:

Index of first 's': 2

Index of first 'k': -1

Index of first 'iss': 1

There are four occurrences of the letter ‘s’ in the string. The method state.find(**'s'**) returns 2 because the first ‘s’ is found in the position with index 2. -1 is returned by state.find(**'k'**) because ‘k’ is not found in the string. 1 is returned by state.find(**'iss'**) because the substring 'iss' is first found in positions with indices 1, 2 and 3. The starting index of this substring is 1.

The replace method produces a new string with every occurrence of a given substring within the original string replaced with another. Example:

word = **"common"**word2 = word.replace(**'m'**, **'t'**)  
print(**"All m's replaced with t's:"**, word2)

Sample test run:

All m's replaced with t's: cotton

The strip method removes leading and trailing characters from a string. If no argument is passed to the method, all leading and trailing white space characters (i.e. spaces, new lines and tabs) are removed. Example:

course = **" CSC 121 "**course2 = course.strip()  
print(**"Leading and trailing white spaces removed:"**, course2)

Sample test run:

Leading and trailing white spaces removed: CSC 121

If a substring is passed to the method, all leading and trailing occurrences of that substring are removed. Example:

course = **"...CSC.121..."**course2 = course.strip(**'.'**)  
print(**"Leading and trailing periods removed:"**, course2)

Sample test run:

Leading and trailing periods removed: CSC.121

The strip method can be used to remove the new line character when we read data from a text file. Earlier we saw that when a line of text is read from a file and stored as a string, the new line character is read and stored along with other characters. We can use the strip method to remove the new line character.

**def** main():  
  
 *""" Display strings read from text file """* input\_file = open(**'instructors.txt'**, **'r'**)  
 **for** line **in** input\_file:  
 line = line.strip()  
 **print**(line)  
 input\_file.close()  
  
main()

The following is the output of the program. There is no more blank line between instructor names because the new line characters are removed from the strings.

Peter Chen

Joe Park

Al Molin

The split method returns a list containing all substrings delimited by a delimiter. If no argument is passed to the method, the default delimiter is space. Example:

my\_string = **'One two three four'**word\_list = my\_string.split()  
print(word\_list)

The split method separates the string my\_string into four substrings and stores them as four elements in the list word\_list. Output of the program:

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

Another example:

date\_string = '11/26/2014'

date\_list = date\_string.split('/')

print('Month:', date\_list[0])

print('Day:', date\_list[1])

print('Year:', date\_list[2])

The split method separates the string date\_string into three substrings and stores them as three elements in the list date\_list. Output of the program:

Month: 11

Day: 26

Year: 2014

Before we end this section, let’s see one more example showing the use of string processing techniques in reading text files.

Suppose we have some information about departments in a college stored in a text file “department\_locations.txt”.

Psychology, Davie Hall, 1075  
Sociology, Hamilton Hall, 0548  
English, Green Hall, 2817  
Computer Science, Sitterson Hall, 0614

Each line in the file contains three pieces of data separated by commas: department name, building where the department is located, and a campus mail box number.

The following program reads the data from the text file. The three pieces of data of each department are stored in a list of three strings, and the four department lists are stored in a list of lists.

**def** main():  
  
 *""" Read comma delimited text file """* input\_file = open(**'dept\_locations.txt'**, **'r'**)  
 dept\_list = []  
 **for** line **in** input\_file:  
 dept = line.split(**','**)  
 dept[0] = dept[0].strip()  
 dept[1] = dept[1].strip()  
 dept[2] = dept[2].strip()  
 dept\_list.append(dept)  
  
 **for** dept **in** dept\_list:  
 print(**"Department:"**, dept[0])  
 print(**"Building:"**, dept[1])  
 print(**"Mail box:"**, dept[2])  
 print()  
  
 input\_file.close()  
  
main()

An empty list dept\_list is created before data are read into the program. This is going to be a list of lists. Each element in this list will be a department, which itself is a list of three data items. In the first for loop, we read a line of text from the file and use comma as a delimiter to split the strings into three substrings. Since each substring may contain leading and trailing white spaces, we use the strip method to clean it up. Each department is then appended to dept\_list. A second for loop is used to display the data. Output of the program:

Department: Psychology

Building: Davie Hall

Mail box: 1075

Department: Sociology

Building: Hamilton Hall

Mail box: 0548

Department: English

Building: Green Hall

Mail box: 2817

Department: Computer Science

Building: Sitterson Hall

Mail box: 0614

# **10.5 Further Readings**

Please read Section 8.1, 8.2 and 8.3 of the textbook. Sections 8.1 and 8.2 are about text files. Section 8.3 is about strings.