Files

CSE 1310 – Introduction to Computers and Programming
Vassilis Athitsos
University of Texas at Arlington

The Need for Files

- Suppose that we have to write a program that:
  - takes a book (or a set of books) as an input.
  - identifies the most frequent words in that book or set of books.
- Can you think of example applications for such a program?

The Need for Files

- Suppose that we have to write a program that:
  - takes a book (or a set of books) as an input.
  - identifies the most frequent words in that book or set of books.
- Can you think of example applications for such a program?
  - identifying the most important words to introduce, in a foreign language class.
  - identifying the language in which a book was written.
  - identifying and comparing style of different authors, newspapers, centuries, etc.

A Book as Program Input

- How can our program go through a whole book?
- Based on what we have learned so far, we would have to type the book into the program.
- Luckily, Python (like typical programming languages) has a much better alternative, which is **file input/output.**

Another Motivating Application

- Consider a phonebook application, that allows:
  - Making a new entry (new name and phone number).
  - Modifying an existing entry.
  - Deleting an entry.
  - Looking up the phone given the name.
  - Looking up the name given the phone.
- What can we do and what can we not do, using what we have learned so far?

Another Motivating Application

- Consider a phonebook application, that allows:
  - Making a new entry (new name and phone number).
  - Modifying an existing entry.
  - Deleting an entry.
  - Looking up the phone given the name.
  - Looking up the name given the phone.
- We can do all five things listed above. However, at the end of the program, all information vanishes.
- Again, file input/output provides a solution:
  - data can be saved into files, and read again from those files when needed.

Example: Reading a File

```
my_file = open("file1.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- Function **open** creates a connection between:
  - variable **my_file**.
  - the content of file **file1.txt.**
- To use function **open**, you need to specify three things:
  - A variable name to be associated with this file (e.g., **my_file).**
  - The name (or full path) of the file.

Example: Reading a File

```
my_file = open("file1.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- To use function **open**, you need to specify three things:
  - A variable name to be associated with this file (e.g., **my_file).**
  - The name (or full path) of the file (e.g., **"file1.txt"**).
    - If the file is on the same directory as the code you are executing, the name is sufficient. Otherwise, you will need to specify a path like "c:/users/vassilis/file1.txt").
  - A **mode of access:** e.g., "r" for reading, "w" for writing.

Example: Reading a File

```
my_file = open("file1.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- Important: **ALWAYS CLOSE A FILE THAT YOU HAVE OPENED, WHEN YOU DO NOT NEED IT ANYMORE.**

A Closer Look

```
my_file = open("file1.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- Function **open** creates a connection between variable **my_file** and the content of file **file1.txt.**
- What is this connection?

A Closer Look

```
my_file = open("file1.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- Function **open** creates a connection between variable **my_file** and the content of file **file1.txt.**
- What is this connection?
  - **my_file** becomes a **stream**, from which we can access lines one at a time, till we reach the end of the file.
  - We can access these lines in different ways.
  - One way (as shown above): **`for line in my_file`**

Each Line Is Read Only Once

```
my_file = open("hello2.txt", "r")

for line in my_file:
    print(line)

for line in my_file:
    print(line)

my_file.close()
```

- Each line in the file is read only once.
- The first for-loop reads all the lines of the file.
- Thus, the second for-loop will not read anything (we **will not see** the file printed twice).

To Access Lines Multiple Times

```
my_file = open("hello2.txt", "r")

lines = my_file.readlines()

for line in lines:
    print(line)

for line in lines:
    print(line)

my_file.close()
```

- Approach 1: Use the **readlines** method to store all lines into a list of strings.

To Access Lines Multiple Times

```
my_file = open("hello2.txt", "r")
for line in my_file:
    print(line)
my_file.close()

my_file = open("hello2.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- Approach 2: Open and close the file twice.

A Second Example: Length of a File

```
my_file = open("file1.txt", "r")
for line in my_file:
    print(line)
my_file.close()
```

- Modify the above program, to obtain a program that computes the length of file "file1.txt", i.e., the number of characters in "file1.txt".

A Second Example: Length of a File

```
my_file = open("file1.txt", "r")
total = 0

for line in my_file:
    total = total + len(line)

my_file.close()
print("the total length is:", total)
```

- The above program computes the length of file "file1.txt", i.e., the number of characters in "file1.txt".

Converting to a Function

```
my_file = open("file1.txt", "r")
total = 0

for line in my_file:
    total = total + len(line)

my_file.close()
print("the total length is:", total)
```

- Modify the above program, so that it defines (and uses) a function file_length(filename), that:
  - takes as argument **filename** the name of a file.
  - returns the number of characters in that file.

Converting to a Function

```
def file_length(filename):
    my_file = open(filename, "r")
    result = 0

    for line in my_file:
        result = result + len(line)

    my_file.close()
    return result

def main():
    total = file_length("file1.txt")
    print("the total length is:", total)

main()
```

Counting Words

- Modify the previous program to also count the number of words in the file.
- Useful string method: **split**
  - my_string.split() returns a **list** of words in a string.
  - More specifically, it returns a list of substrings that are separated by white space.
  - Example:

```
>>> string1 = "today is Monday"
>>> b = string1.split()
>>> b
['today', 'is', 'Monday']
```

Counting Words: Solution

```
def file_words(filename):
    my_file = open(filename, "r")
    result = 0
    for line in my_file:
        words = line.split()
        result = result + len(words)
    my_file.close()
    return result

def main():
    name = "file1.txt"
    number_of_words = file_words(name)
    print("the number of words is:", number_of_words)

main()
```

Writing a File: An Example

```
out_file = open("hello2.txt", "w")
print("writing a line to a file", file=out_file)
print("writing a second line", file=out_file)
out_file.close()
```

- Opening a file for writing is similar to opening a file for reading, except that we use "w" instead of "a" as the second argument.
- To write a line to a file, we use a **print** command, putting **file=xxx** as the last argument.
  - **xxx** is just the name of the variable associated with the output file.

Writing a File: An Example

```
out_file = open("hello2.txt", "w")
print("writing a line to a file", file=out_file)
print("writing a second line", file=out_file)
out_file.close()
```

- The four lines above create a text file called **hello2.txt**, with the following content:

```
writing a line to a file
writing a second line
```

- What happens if file **hello2.txt** already existed?

Writing a File: An Example

```
out_file = open("hello2.txt", "w")
print("writing a line to a file", file=out_file)
print("writing a second line", file=out_file)
out_file.close()
```

- The four lines above create a text file called **hello2.txt**, with the following content:

```
writing a line to a file
writing a second line
```

- What happens if file **hello2.txt** already existed?

Its previous contents are **lost forever**.

Exercise: Copying a File

- Write a function **copy_file(name1, name2)** that:
  - Takes two strings as arguments, **name1** and **name2**.
  - Copies the contents of existing file **name1** into a new file **name2**.

Exercise: Copying a File

```
def copy_file(in_name, out_name):
    in_file = open(in_name, "r")
    out_file = open(out_name, "w")

    for line in in_file:
        print(line, file=out_file, end="")

    in_file.close()
    out_file.close()

def main():
    copy_file("hello2.txt", "hello3.txt")
    print("done converting to upper case")

main()
```

Note: Avoiding The "\n" Character

- In the previous program, we used this line:

```
print(line, file=out_file, end="")
```

- The **`end=""`** argument tells Python to **NOT** put a newline character (the "\n" character) at the end of the line that it prints.

Exercise: Convert to Upper Case

- Write a function **convert_to_upper_case(name1, name2)** that:
  - Takes two strings as arguments, **name1** and **name2**.
  - Converts the contents of existing file **name1** to uppercase, and saves the converted contents into a new file **name2**.

Exercise: Convert to Upper Case

```
def convert_to_upper_case(in_name, out_name):
    in_file = open(in_name, "r")
    out_file = open(out_name, "w")

    for line in in_file:
        converted_to_upper = line.upper()
        print(converted_to_upper, file=out_file, end="")

    in_file.close()
    out_file.close()

def main():
    convert_to_upper_case("file1.txt", "file2.txt")
    print("done converting to upper case")

main()
```

Reading Individual Lines

- To read an individual line from a file, use the readline() method.

```
in_file = open("phonebook.txt", "r")

first_name = in_file.readline()
first_name = first_name.strip()
first_number = in_file.readline()
first_number = first_number.strip()

print(first_name + ": " + first_number)

in_file.close()
```

Different File Access Modes

- Function **open** takes an *access mode* as second argument. The possible access modes are:

Mode	Description	If File Exists	If File Does Not Exist
'r'	read-only	Opens that file	Error
'w'	write-only	Clears the file contents	Creates and opens a new file
'a'	write-only	File contents left intact and new data appended at file's end	Creates and opens a new file
'r+'	read and write	Reads and overwrites from the file's beginning	Error
'w+'	read and write	Clears the file contents	Creates and opens a new file
'a+'	read and write	File contents left intact and read and write at file's end	Creates and opens a new file

Different File Access Modes

- In this course, we will only use the 'r' and 'w' modes, the other modes are provided only for reference.

Mode	Description	If File Exists	If File Does Not Exist
'r'	read-only	Opens that file	Error
'w'	write-only	Clears the file contents	Creates and opens a new file
'a'	write-only	File contents left intact and new data appended at file's end	Creates and opens a new file
'r+'	read and write	Reads and overwrites from the file's beginning	Error
'w+'	read and write	Clears the file contents	Creates and opens a new file
'a+'	read and write	File contents left intact and read and write at file's end	Creates and opens a new file

Checking if a File Exists

- In our phonebook application, we save data to a file.
- When the application starts, it reads data from the file.
- What happens the first time we use the application?

```
in_file = open("phonebook.txt", "r")
```

Checking if a File Exists

- In our phonebook application, we save data to a file.
- When the application starts, it reads data from the file.
- What happens the first time we use the application?

```
in_file = open("phonebook.txt", "r")
```

- If phonebook.txt does not exist, the above line will generate an error and crash the program.
- How can we avoid that?

Checking if a File Exists

- Before opening a file for reading, we need to check if a file exists, using **os.path.isfile**.

```
import os

file_lines = []

if (os.path.isfile('phonebook.txt')):
    in_file = open('phonebook.txt')
    file_lines = in_file.readlines()
    in_file.close()

print("the number of lines read was",
len(file_lines))
```