Lecture 10 - Files

Computer Programming

Robert Vincent & Samia Hilal

Marianopolis College

March 29, 2022

What are files?

- So far our programs' data comes from the source file itself, or from the user.
- In reality, most useful programs make use of *persistent* data from some external storage.
- Most operating systems organize storage in a series of files.
- A file is just a named object that consists of a sequence of bytes.
- Most programming languages have an interface for reading, creating, or writing files.

Some file basics

- For every file, the operating system stores information such as:
  - A name.
  - The current length (usually in bytes).
  - Creation date.
  - Owner.
  - Access rights.
- The data itself is stored in some arbitrary way, but the OS presents it to our software as if it were a simple list of bytes.

File paths

- Most operating systems implement a hierarchy of folders that contain other folders or files.
- A file name that includes folder information is called a *path*.
- Components are separated with a '/' or '\'.
- Pathname examples:
  - `/usr/local/bin/emacs`
  - `C:\Program Files\Microsoft\MSWord.exe`
  - `../Documents/myfile.doc`

Types of files

- Most files can be considered a simple sequence of bytes.
- The key is understanding how those bytes are to be interpreted.
  - Documents: MS Word, PDF, Text, HTML
  - Images: JPEG, PNG, GIF
  - Music: MP3
  - Archives: ZIP, RAR, 7Z, TAR
- File types may be distinguished by their names or by their contents.

File access

- Programs generally access files by giving the name of the file to the operating system and requesting read and/or write access.
- The request may fail because of security issues or because the named file might not exist.
- Files generally maintain a *current location* that is updated as we read or write data.
- We can't read past the 'end of file' (**EOF**).
- A program *closes* a file when it no longer needs access.

Text files

- Consist of ASCII (or Unicode) text, and are to some extent meant to be human-readable.
- These include Python (.py), plain text (.txt), or HTML (.html).
- Often organized as a series of lines ending with a newline (`'\n'`) character.
- Take note that the (`'\n'`) character cannot be seen in the text file but will be part of the data when you read from the file.
- File content can be treated as one or more Python strings.

Binary files

- Consist of a series of bytes in some arbitrary encoding.
- Include most picture, movie, or music formats.
- No concept of lines.
- The Python data type `bytes` is used to represent binary data.

Files in Python: open()

- Python programs request access to files using the built-in function `open()`.

```
fp = open("filename", mode='r', ... )
```

- `fp` is a variable name used to refer to the file.
- `open()` has only one required argument: the path of a file to open. If no path is given, file opens in the same folder as your .py file.
- Returns a Python object that supports input/output (I/O) operations.
- `mode` may be `'w'` to specify writing.
- There are other optional arguments which we will not use in this course.

The `mode` parameter

`'r'`	Read an existing file.
`'w'`	Write a new file.
`'a'`	Append to an existing file.
`'r+'`	Read/write an existing file.
`'w+'`	Read/write a new file.
`'a+'`	Append with read/write.

- Adding a `'b'` to any mode requests *binary* instead of *text* mode.
- If the file is open in binary mode, Python returns `bytes()` objects rather than `str()` objects.

Python file methods

`close()`	Close the file.
`read()`	Read data from the file.
`readline()`	Read the next line as a string.
`readlines()`	Return a list of lines.
`write()`	Write data to the file.
`writelines()`	Write iterable to the file.

The close() method

- Syntax: `fp.close()`
- Action: The file is closed. All data will be written to the file and operating system resources will be freed.
- It is very important to close a file when done writing to it or data will be lost.
- Returns: None
- Example: files_close.py

```
fp = open('log.txt', 'w')
fp.write('This is a test.\n')
fp.close()
# fp is no longer usable.
```

The `read()` method

- Syntax: `fp.read(size)`
- Action: Reads up to `size` bytes starting at the current file location. If `size` is omitted, reads to the end of the file.
- Returns: A `str()` or `bytes()` object.
- Example: read.py, alice.txt

```
fp = open('alice.txt', 'r')
txt1 = fp.read(100) # First 100 bytes
print(len(txt1)) # Print 100
txt2 = fp.read() # Rest of file.
print(len(txt2)) # Print 148440
fp.close()
```

The `readline()` method

- Syntax: `fp.readline()`
- Action: Reads the next line.
- Returns: A `str()` or `bytes()` object.
- Similar to input function.
- Example: readline.py

```
fp = open('alice.txt')
line = fp.readline()
while line:  # Until zero length.
    if len(line) >= 72:
        print(line.strip())
    line = fp.readline()
fp.close()
```

The `readline()` method

- Another Example
- Reads the next line and checks if empty
- Returns: A `str()` object.
- Remove end of line or other characters using the `strip` method
- Example: students_readline.py, students.txt

```
fp = open('students.txt')
while True: # keep reading
    line = fp.readline()
    if not line: break
    line=line.strip('\n') # remove EOLine
    # Insert code to process and store record
fp.close()
```

The `readlines()` method

- Syntax: `fp.readlines()`
- Action: Reads entire file as a list of lines.
- Returns: A list of the lines in the file.
- The list of files is stored in memory. Not recommended for long files.
- *Iterating over a file reads 1 line at a time.*
- Example: readlines.py

```
fp = open('alice.txt')
for line in fp.readlines(): # Or just fp:
    if len(line) >= 72:
        print(line.strip())
fp.close()
```

Reading a file - Another Method

The text file `alice.txt` contains the book "Alice's Adventures in Wonderland". The program counts lines and characters in the file.

- loop over the file object to read lines from a file.
- Fast & simple code that is also memory efficient.
- Example: files_ex1.py

```
in_file = open('alice.txt')
nl,nc,nw = 0,0,0 # Initialize counters.
for line in in_file: #in_file.readlines()
    nl += 1              # Count lines.
    nc += len(line)      # Count characters.
    nw += len(line.split())
print(nl,nc,nw)          # Print results.
in_file.close()          # Done with file.
```

The `write()` method

- Syntax: `fp.write(value)`
- Action: Writes the `str()` or `bytes()` value to the file, starting at the current file location.
- Returns: The number of characters (or bytes) written.
- Example: write.py

```
fp = open('test.txt', 'w')
n = fp.write("First line\n") # Need '\n'
m = fp.write("Second line\n")
print(n, m)                  # Prints 11 12
print(fp.tell())             # Prints 23
fp.close()
```

Printing to a file

- You can also use `print()` to write to a file.
- Unlike `write()`, `print()` takes multiple arguments and converts them to strings.
- Use the optional `file` parameter to print to a file. Default is to print to the screen.
- Example: print.py

```
fp = open('log.txt', 'w')
x, y = 5, 2
z = x + y
print(x, y, file = fp) # Write 5 2 on a line.
print(z, file = fp) # Write 7 on one line.
print("The end!", file = fp) # Final line.
fp.close()
```

Writing To a file

- Append a formatted message to a file
- Example: files_ex3.py

```
fp = open('log.txt', 'a') # Append
n = fp.tell()
print('Message starting at ', n, file = fp)
fp.close()
```

Conclusion

- The `open()` function returns an object that lets you access a file.
- Specialized modules exist for dealing with standard types of file data.
- You have seen the file `open()` function and the associated `close()` method.
- How to read the entire file at once or read one line at a time (readline()).
- Know how to write to a file using `print()`.
- Be sure you could `open()` a text file and read all of its lines.