We all use files every day. Files are used to store data persistently so that you can access that data again and again. When you store data in a file, it resides on your computer's hard drive (or other storage medium: USB stick, SSD, DVD, etc.). You can then open that file any time. Files are typically used by specific programs and have custom formats suited to that program. For example, Microsoft Word creates files that contain all of the important formatting information for a document along with the actual text of that document. Programs other than Microsoft Word would not be able to understand the contents of the file in order to use it (although there are a number of other programs that now work with Microsoft Word files to some extent). Other types of files are meant to be used across different programs. For example, plain text files can be created and read by a large variety of text editors. Also, PDF files can be created by numerous word processing programs and displayed by a variety of different PDF reader applications.

Regardless of whether a file is specific to a particular program or not, all files are created, read, written, and deleted by computer programs. The program might write files in a well known format so that it is compatible with other programs or it might write files in a proprietary way that have no meaning to any other program. In either case, the contents of the file will remain persistent on disk so that the program, or another program, can use that file again in the future.

The file's *extension* (.pdf, .docx, .txt, .jpg, .mp3 etc.) at the end of the filename indicates the type of file. While extensions are not strictly necessary, they help the operating system and other programs infer the format of the contents of the file. A file with a ".txt" extension at the end is assumed to be a plain text file that can be read and modified by any text editor. A file with a ".pdf" extension is assumed to be a PDF file. And so on. Note, however, that a user could change the extension on any file (using a computer program, often one that seems invisible because you are using it via a graphical user interface that allows you to explore files and folders on your computer) making the extension not match the format of the contents. Therefore, extensions are really only hints about the contents of the file that might be incorrect.

What is a file?

A file is just a sequence of data. It is entirely up to the program to interpret that data. Files can either be stored in *text* or *binary* form. A text file is a sequence of characters that can be used to store any text. A binary file is a sequence of bytes that can be interpreted in any way the computer would like. Binary data is just raw numbers that can be interpreted by a computer as any type of data. For example, you can store the number 3426 in either a text file or a binary file. To store it in a text file, you would simply put the string with the characters representing these numbers ("3426") into the file. This would store the ASCII or Unicode representation of those four characters in the file. To store the number 3426 in a binary file, you would instead put the actual number 3426 into the file, not a string. Text files, therefore, can be thought of as one long string. In contrast, binary files can contain anything, which includes combinations of different types of data.

In this class, and throughout the specialization, we will focus on text files.

File Operations

Conceptually, computer programs can do four operations on files: open, read, write, and close. Let's consider each operation in turn:

Open. Before a program can actually do anything with a file it has to open it. The process of opening the file involves actually finding it on the disk, making sure the program has the proper permissions to access the file, and then returning an open file object to the program that will allow it to later read from or write to the file. If any of these steps fail, the open operation will cause some sort of an error. Because the open operation is checking that the program has the proper permissions to operate on the file, one must typically specify whether the program intend to perform read, write, or both types of operations on the file when opening the file.

Read. Once a program has access to an open file object, it then may read from it. For text files, this involves getting a certain number of characters as a string. Python provides a variety of different methods to allow programs to read from files. In almost all cases, each read from a file gets the contents of the file that appear right after the text that was returned from the previous read operation. While it is possible to change where in the file the program wants to read (instead of reading the file sequentially), the program must explicitly do that.

Write. Once a program has access to an open file object, it may then write to it. For text files, this involves passing a string to the file object. Python again provides a variety of different methods to allow programs to write to files. As with reading files, in almost all cases, each write to a file places the new text right after the text that was written by the previous write operation. It is also possible to specify where to write into the file (instead of writing the file sequentially), but the program must explicitly do that. When writing to a file that already existed, the new contents can either overwrite the old contents of the file or be added to the end of the file. Whether the file should be overwritten or extended must be specified when the file is opened.

Close. After the program is done reading or writing the file, it must close the open file object. Operating systems typically limit the number of files that can be open at any given time, since open files consume resources in memory. This includes both bookkeeping data and the file contents themselves (although the entire file contents may not be in memory at any given time, just some of it). When the file is closed, the operating system makes sure that all of the file data is copied back onto the disk. So, if you forget to close a file after writing it, all of the data may not appear on the disk if an error occurs later in your program.

Python

In Python, files are opened using the `open` function:

```python
# The name of the file to open
filename = "starwars.txt"

# The mode in which to open it (read, text)
mode = "rt"

# Actually open the file
openfile = open(filename, mode)
```

The `open` function takes two arguments, which are both strings: a filename and a mode. It then returns an open file object that can be used to access the file. In the snippet above the file is opened in `rt` mode, with indicates **r**ead, **t**ext. Other relevant modes will be discussed in the videos. Because this file was opened in read mode, the `openfile` object can only be used to read the file, it will not allow writes.

Open file objects have many different read methods, but the most basic is intuitively called `read`:

```python
openfile = open("gonewiththewind.txt", "rt")

filedata = openfile.read()
print(filedata)
```

The `read` method simply returns the entire contents of the file as a string. You can then use it in your program as you would any other Python string. Note that `read` does take an optional size argument that limits the returned string to a maximum number of characters. This can be useful when reading large files. See the Python documentation for more details.

Similarly, open file objects have many different write methods, but the most basic is intuitively called **`write`**:

```
openfile = open("mymoviescript.txt", "wt")

openfile.write('I wish I had an idea for a movie...\n')
```

The **`write`** method takes a string and writes the text to the file. It returns the number of characters that were written to the file, which should always be the length of the string.

Finally, never forget to close the file! To do so, open file objects have a **`close`** method:

```
openfile = open("emptyfile.txt")

openfile.close()
```

It is good practice to close the file right after the last read or write operation. It is sometimes tempting to begin to process the data from the file before closing it, for instance, but you should resist that temptation. Get in the habit of closing the file promptly. In most cases, it will not matter, but occasionally (especially when there are unexpected errors in the program) it will make a big difference.

Mark as completed