

Computer Programming for Information Professionals

Working with Persistent Storage (File IO)

Persistent Storage

So far, we've been interacting with main memory. When our programs quit, everything is lost.

This is okay for a few applications (e.g., a calculator), but in general, we will want to save some or all of our results and load them back later.

To do this we need to be able read and write to files on the hard drive.

Types of File

There are two types of files:

- **Text Files**
 - **Can be opened in a text editor**
 - **A sequence of ASCII or Unicode characters**
 - **Python programs, HTML source code, CSV files, etc.**
- **Binary Files**
 - **Cannot be opened/read in a text editor**
 - **Stores data as in main memory (in binary: 10100010...)**
 - **MP3 files, image files, word documents, etc.**

We are going to focus on working with text files today

Overview of Working with Files

1. Open the file and associate the file object with a variable
2. Do stuff with the file (read/write data)
3. Close the file

Opening Files

- Prepares the file for reading:
 - Links a file variable with the physical file
 - I.e., references to the file variable are references to the physical file
 - Using our variable:box metaphor, our variable box now contains a file object
 - Note that this isn't the file as a string, more like an address to the file
- Positions the file pointer at the start of the file.

Format:

```
file_variable = open(file_name, mode)
```

Where

- **file_variable** is the name of the variable you want to use to reference your file object
- **file_name** is the name of the file that you want to open
- `open()` is a function that opens a file
- **mode** is an input parameter that specifies what you can do with the file.

For now, let's assume that the file is in the same directory/folder as the Python program.

And, let's also ignore error handling. It's important, but gets in the way of learning

Basic Modes

Mode	Description
"r"	Opens the file for reading. If the file doesn't already exist you will get <code>FileNotFoundError</code> error. 'r' is the default if mode is omitted
"w"	Opens the file for writing. If file doesn't exist, it will be created. If the file exists, then its data is destroyed/overwritten.
"a"	Opens the file for writing. If the file doesn't exist, it will be created. If the file already exists then it appends new data to the end of the file rather than destroying it.
"x"	Opens the file for writing. If the file already exists you will get <code>FileExistsError</code> error.

Additional Flags

These get combined with a main mode for further specification. E.g., `rb+` opens a binary file for reading and writing, preserving the existing file data, and starting at the top.

Mode	Description
"t"	Text mode 't' is the default if file_type is omitted.
"b"	Binary (e.g., images)
"+"	Open for updating (readings and writing) When combined with <code>r</code> prior data is preserved, pointer is at top of file. When combined with <code>w</code> prior data is erased, pointer is at top of file. When combined with <code>a</code> prior data is preserved, pointer is at end of file.

Examples: Opening a File

Example – Static file name

```
inputFile = open("data.txt", "r")    # Opens a text file for reading
inputFile = open("data.txt", "w")    # Opens a text file for writing
inputFile = open("data.txt", "rb")   # Opens a binary file for reading
inputFile = open("data.txt", "ab+")  # Opens a binary file for reading/
                                     # appending
```

Example – Variable file name (entered by user at runtime)

```
filename = input("Enter name of input file: ")
inputFile = open(filename, "r")
```

In practice this should be wrapped in a `try/except`, but we'll ignore that for now

Closing Files

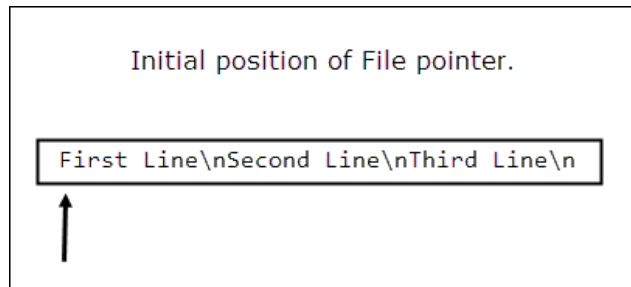
- Once done, close the file

```
filename.close()
```

- Releases valuable system resources
- If you forget, Python will close the file eventually (program ends, file object no longer referenced)
- But...! Can use significant resources
 - E.g., if your program is large, you're reading/writing multiple files
 - Best practice is to close a file as soon as you are done with it

Reading Files

When we open a file for reading, the pointer is at the start of the file



Reading the Entire File in One Go

- Remember that the file object returned by `open()` isn't the file itself but a pointer to the file
- Can read the entire contents of a file using `read()` method, e.g.,

```
filestuff = filename.read()
```

- Returns the entire file as a string
- Fine if your file is small, but remember that persistent storage is much (much) bigger than main memory and is shared across all applications
 - Can't always load the entire file
 - Putting too much stuff in main memory can causes slow downs and failures

Example: Reading an Entire File

In []:

```
inputFile = open("lyrics-short.txt", "r")
print(inputFile.read())
inputFile.close()
```

Reading a File in Chunks

- Can specify a number of characters as an input parameter, e.g.,


```
filename.read(4)
filename.read(10)
filename.read()
```

- Would read the first 4 characters, then the next 10, and then the rest of the file

This uses memory more efficiently, which is important for large files!


Initial position of File pointer.

First Line\nSecond Line\nThird Line\n




Position of File pointer after reading first 4 characters.

First Line\nSecond Line\nThird Line\n




Position of File pointer after reading next 10 characters.

First Line\nSecond Line\nThird Line\n



Position of File pointer after reading all the remaining data

First Line\nSecond Line\nThird Line\n



Example: Reading in Chunks

In []:

```
inputFile = open("lyrics-short.txt", "r")
inputFile.read(511)
print(inputFile.read(21))
print(inputFile.read(15))
print(inputFile.read(100))
print(inputFile.read(23))
print(inputFile.read())
inputFile.close()
```

Reading a File as a List of Lines

- Read gives us all the data as a single string.
- Often it is useful to have a list of lines
 - We saw this in Assignment 2 (and will see it again in Assignment 3)

```
filestuff = filename.readlines()
```

- Returns the entire file as a list of strings (each string is a line)
- Note in terms of memory usage this isn't any better than read
 - Handy for the stuff we are doing in class, but not useful for big data

Example: Reading a List of All Lines

In []:

```
inputFile = open("lyrics-short.txt", "r")
lines = inputFile.readlines()
#print(lines)
print(lines[0])
print("...\n")
print(lines[-1])
inputFile.close()
```

Reading One Line at a Time (with a Loop)

- Typically reading is done within the body of a loop
- Each execution of the loop will read a line from the file into a string
- More efficient use of memory

Format:

```
for next_line in file_object:
    Do_something_with_the_string_read_from_file
```

Example: Reading Lines with a Loop

In []:

```
inputFile = open("lyrics-short.txt", "r")
for line in inputFile:
    print(line[:-1])
inputFile.close()
```

Reading One Line at a Time (with a readLine)

- Sometimes you want to do different things with different lines
- Can use `readline()` to get each line in sequence without using a loop

Example

In []:

```
inputFile = open("lyrics-short.txt", "r")
print(inputFile.readline())
print(inputFile.readline().upper())
inputFile.close()
```

Writing to Files

- When we open files for writing, any existing data in the file is overwritten (deleted)
 - If you want to add data (to the end of a file) use append 'a' option
 - If you want to modify data, use 'r+'
- Use the `write()` method, e.g.,

```
filename.write(aString)
```

Example: Writing to a File

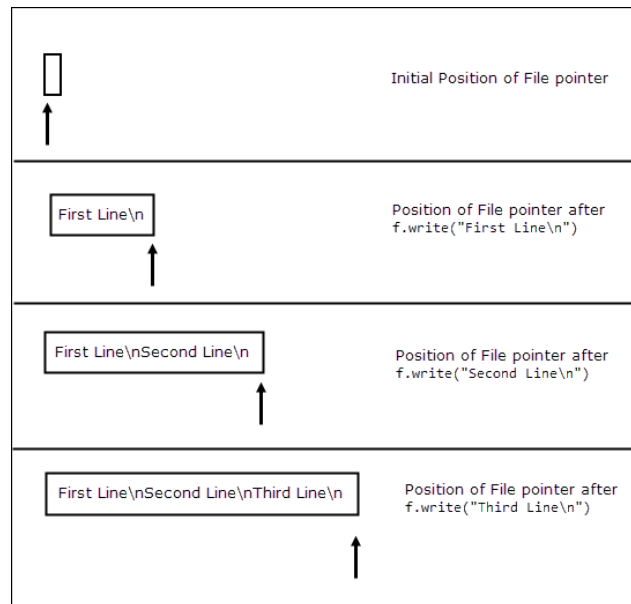
In [8]:

```
myfile = open("rhyme.txt", "w")
myfile.write("One for the money\nTwo for the show\nThree to make ready...\n...and four to go. \n")
myfile.close()
```

In [9]:

```
myfile = open("rhyme.txt", "r")
print(myfile.read())
myfile.close()
```

```
One for the money
Two for the show
Three to make ready...
...and four to go.
```



Appending to Files

- When we open files for writing, any existing data in the file is overwritten (deleted)
- Sometimes we want to open a file, write some data, close it, and then reopen it and write some more (either at different points in the program, or across runs of it)
- We can open the file for appending instead of writing
 - Append doesn't overwrite the existing data and starts from the bottom of the file

Example: Appending to a File

In [14]:

```
myfile = open("rhyme.txt", "a")           # Notice we use 'a' instead of
      'w'
myfile.write("\nTwiddle ")
myfile.write("Twoodle\n")
myfile.write("You're old hamster\n ")
myfile.write("...is now a noodle\n")
myfile.close()
```


In [15]:

```
myfile = open("rhyme.txt", "r")
print(myfile.read())
myfile.close()
```

```
One for the money
Two for the show
Three to make ready...
...and four to go.
```

```
Twiddle Twoodle
You're old hamster
...is now a noodle
```

```
Twiddle Twoodle
You're old hamster
...is now a noodle
```

```
Twiddle Twoodle
You're old hamster
...is now a noodle
```

Processing Data from Files

Files can be used to store complex data given that there exists a predefined format.

E.g.,

```
<score> <Review>      (From movieReviews.txt in Assignment 2)
```

Or

```
<ISBN>, <Year>, <Title>, <Author>      (From books.txt in Assignment
3)
```

We'll use an excerpt:

```
0517693119, 1989, The Hitchhiker's Guide To The Galaxy, Douglas Adam
s
0025002600, 1978, Watership Down, Richard Adams
0937247065, 1988, The Five People You Meet in Heaven, Mitch Albom
```

Example: Processing a data file

In [16]:

```
inputFile = open ("books.txt", "r")

print ("Reading from file books.txt")
for line in inputFile:
    line = line.strip()
    ISBN, year, title, author = line.split(', ')
    year = int(year)
    print(" " + title + " by " + author)

print ("Completed reading of file books.txt")
inputFile.close()
```

Reading from file books.txt

The Hitchhiker's Guide To The Galaxy by Douglas Adams

Watership Down by Richard Adams

The Five People You Meet in Heaven by Mitch Albom

Completed reading of file books.txt

Example: Reading and Writing

In [21]:

```
inputFile = open ("books.txt", "r")
outputFile = open("output.txt", "w")

outputFile.write ("Reading from file books.txt\n")
for line in inputFile:
    line = line.strip()
    ISBN, year, title, author = line.split(', ')
    year = int(year)
    outputFile.write(" " + title + " by " + author + "\n")           # Note thi
s line changed

outputFile.write ("Completed reading of file books.txt\n")
inputFile.close()
outputFile.close()
```

In [22]:

```
outputFile = open("output.txt", "r")
print(outputFile.read())
outputFile.close()
```

Reading from file books.txt

The Hitchhiker's Guide To The Galaxy by Douglas Adams

Watership Down by Richard Adams

The Five People You Meet in Heaven by Mitch Albom

Completed reading of file books.txt

File Methods

Method	Description
<code>read([num])</code>	Reads the specified number of characters from the file and returns them as string. If <code>num</code> is omitted then it reads the entire file.
<code>readline()</code>	Reads a single line and returns it as a string.
<code>readlines()</code>	Reads the content of a file line by line and returns them as a list of strings.
<code>write(str)</code>	Writes the string argument to the file and returns the number of characters written to the file.
<code>seek(offset, origin)</code>	Moves the file pointer to the given offset from the origin.
<code>tell()</code>	Returns the current position of the file pointer.
<code>close()</code>	Closes the file

Error Handling with Exceptions

- Exceptions are used to deal with extraordinary errors ('exceptional ones').
- Typically these are fatal runtime errors ("crashes" program)
- Example: trying to open a non-existent file
- Basic structure of handling exceptions

```
try:
    Attempt_something_where_exception_error_may_happen
except <exception_type>:
    React_to_the_error
else: # Not always needed
    What_to_do_if_no_error_is_encountered
finally: # Not always needed
    Actions_that_must_always_be_performed
```

Example: File Error Handling with Exceptions

In [23]:

```
inputOKFlag = False
while (inputOKFlag == False):
    try:
        inputFileName = input("Enter name of input file: ")
        inputFile = open(inputFileName, "r")
    except IOError:
        print("File", inputFileName, "could not be opened")
    else:
        print("Opening file", inputFileName, " for reading.")
        inputOKFlag = True
        fileData = inputFile.read()
        print ("Completed reading of file", inputFileName)
        inputFile.close()
        print ("Closed file", inputFileName)
```

```
Enter name of input file: sdkfjsdfj
File sdkfjsdfj could not be opened
Enter name of input file: books.txt
Opening file books.txt  for reading.
Completed reading of file books.txt
Closed file books.txt
```

Example: File Error Handling with Exceptions — Extended

In [24]:

```
inputFileOKFlag = False
while (inputFileOKFlag == False):
    try:
        inputFileName = input("Enter name of input file: ")
        inputFile = open(inputFileName, "r")
    except IOError:
        print("File", inputFileName, "could not be opened")
    else:
        print("Opening file", inputFileName, " for reading.")
        inputFileOKFlag = True
        fileData = inputFile.read()
        print ("Completed reading of file", inputFileName)
        inputFile.close()
        print ("Closed file", inputFileName)
    finally:
        if (inputFileOKFlag == True):
            print ("Successfully read information from file", inputFileName)
        else:
            print ("Unsuccessfully attempted to read information from file"
, inputFileName)
```

```
Enter name of input file: milk.txt
File milk.txt could not be opened
Unsuccessfully attempted to read information from file milk.txt
Enter name of input file: sdfkjdsf
File sdfkjdsf could not be opened
Unsuccessfully attempted to read information from file sdfkjdsf
Enter name of input file: books.txt
Opening file books.txt  for reading.
Completed reading of file books.txt
Closed file books.txt
Successfully read information from file books.txt
```

Example: Keyboard Input Error Handling with Exceptions

In [25]:

```
inputOKFlag = False
while (inputOKFlag == False):
    try:
        num = input("Enter a number: ")
        num = float(num)
    except ValueError:    # Can't convert to a number
        print("Non-numeric type entered", num)
    else:    # All characters are part of a number
        inputOKFlag = True
num = num * 2
print(num)
```

```
Enter a number: a
Non-numeric type entered a
Enter a number: sdfjhdsd
Non-numeric type entered sdfjhdsd
Enter a number: 88
176.0
```

Error Handling with with

Simplifies exception handling for common resources like file streams

```
with open('file_path', 'w') as file:
    file.write('hello world !')
```

Replaces

```
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()
```

There is no need to close the file as the resources are automatically released at the end of the with statement

Example: Reading a File with with

In [28]:

```
with open("books.txt", 'r') as inputFile:
    books = inputFile.readlines()

for bk in books:
    bk = bk.strip()
    ISBN, year, title, author = bk.split(', ')
    print(" " + title + " by " + author)
```

```
The Hitchhiker's Guide To The Galaxy by Douglas Adams
Watership Down by Richard Adams
The Five People You Meet in Heaven by Mitch Albom
```

Lesson Summary — Persistent Storage

You Should Now Know

- How to open a file for *reading*
- How to open a file for *writing*
- How to open a file for *appending*
- How information is read from and written to a file
- How to close a file and why it is good practice to do this explicitly
- How to read from a file of arbitrary size
- How to manipulate data from files
- How exceptions can be used in conjunction with file input and with invalid keyboard/console input
- How to use `with` to handle common file exceptions