CS303E: Elements of Computers and Programming
Files

Dr. Bill Young
Department of Computer Science
University of Texas at Austin

Last updated: August 29, 2024 at 12:54

Value of Files

Files are a *persistent* way to store programs, input data, and output data.

Files are stored in the memory of your computer in an area allocated to the *file system*, which is typically arranged into a hierarchy of *directories*.

The *path* to a particular file details where the file is stored within this hierarchy.

Relative Pathnames

A path to a file may be *absolute* or *relative*.

If you just name the file, you're specifying that it is in the current working directory, i.e., *relative* to where you currently are in the file system hierarchy.

```
> pwd
/u/byoung/cs303e/slides
> ls -l MTable
-rw-r----- 1 byoung prof 812 Sep 21 13:11 MTable
> ls -l /u/byoung/cs303e/slides/MTable
-rw-r----- 1 byoung prof 812 Sep 21 13:11 /u/byoung/cs303e/
    slides/MTable
> ls syllabus303e.html
ls: cannot access 'syllabus303e.html': No such file or
    directory
> ls ../syllabus303e.html
../syllabus303e.html
```

File Paths

On Windows, a file path might be:

```
c:\byoung\cs303e\slides\slides11a-files.tex
```

On Linux or MacOS, it might be:

```
/home/byoung/cs303e/slides/slides11a-files.tex
```

Python passes filenames around as strings, which causes some problems for Windows systems, partly because Windows uses the "\" in filepaths. *Recall that backslash is an escape character, and including it in a string may require escaping it.*

Raw Strings

There is a way in Python to treat a string as a **raw string**, meaning that escaped characters are treated just as any other characters.

```
>>> print("abc\ndef")
abc
def
>>> print(r"abc\ndef")
abc\ndef
```

Prefix the string with an "r". You may or may not need to do this for Windows pathnames including "\"

Getting Your Bearings

Students often find that the file they want to run isn't in the directory where they're running Python. The following Python program shows the current directory and lists the files in it:

```
# Show the current directory and files in it.
import os
dir = os.getcwd()
print("Directory is: ", dir )
myfiles = os.listdir()
for file in myfiles:
    print( file )
```

```
> python showDirectory.py
Directory is:  /u/byoung/cs303e/python
ComputeAngles.py
ComputeCircleArea.py
ExamExample.py
FindMax.py
...
Project2-fibonacci-numbers.py
```

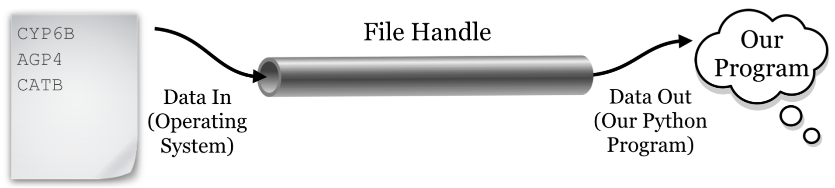
Managing Files in Python

Python provides a simple, elegant interface to storing and retrieving data in files.

open : establish a connection to the file and associate a local file *handle* with a physical file.
close : terminate the connection to the file.
read : input data from the file into your program.
write : output data from your program to a file.

Opening a File

Before your program can access the data in a file, it is necessary to *open* it. This returns a *file object*, also called a "handle," that you can use within your program to access the file.



It also informs the system how you intend for your program to interact with the file, the "mode," e.g., read or write.

Example of Opening a File

General Form:

```
fileHandle = open(filename, mode)
```

```
>>> outfile = open("MyNewFile", "w")
>>> outfile.write("My dog has fleas!\n")
18
>>> outfile.close()
>>>   # cntr-D out of interactive mode

> cat MyNewFile
My dog has fleas!
```

Here `outfile` is the *file handle* that you use to refer to the file within your program.

Opening a File: Modes

Here are some permissible modes for files:

Mode	Description
"r"	Open for reading.
"w"	Open for writing. If the file already exists the old contents are overwritten.
"a"	Open for appending data to the end of the file.
"rb"	Open for reading binary data.
"wb"	Open for writing binary data.

You also have to have necessary permissions from the operating system to access the files.

BTW: the mode defaults to reading, so `open(file)` is equivalent to `open(file, "r")`

Closing the File

General form:

```
fileHandle.close()
```

All files are closed by the OS when your program terminates. Still, it is very important to close any file you open in Python.

- the file will be locked from access by any other program while you have it open;
- items you write to the file may be held in internal buffers rather than written to the physical file;
- if you have a file open for writing, you can't read it until you close it, and re-open for reading;
- *it's just good programming practice.*

Modes r+ and w+

I didn't realize this until recently, but you actually can have a file open for both reading and writing simultaneously, using mode `r+` and `w+`.

However, it's pretty dangerous to do so, because writing occurs where the file pointer is, which may be at the start of the file.

It's easy to overwrite the file contents. **Don't use these modes in this class.**

Reading/Writing a File

There are various Python functions for reading data from or writing data to a file, given the file handle in variable `h`.

Function	Description
`h.read()`	Return entire remaining contents of file as a string.
`h.read(k)`	Return next k characters from the file as a string.
`h.readline()`	Return the next line as a string.
`h.readlines()`	Return all remaining lines in the file as a list of strings.
`h.write(str)`	Write the string to the file.

These functions advance an internal *file pointer* that indicates where in the file you're reading/writing. `open` sets it at the beginning of the file.

Use `readlines`, not `read`

Students have a tendency to use `read` on the contents of a file, because you already know how to manipulate strings. *In general, don't do that!*

A file can be many megabytes long and you don't want to create a string that long.

Instead use `readline` to read the file line by line (unless you know the file is very small.

That's much more scalable; it doesn't really matter how long the file is.

Testing File Existence

Sometimes you need to know whether a file exists, otherwise you may overwrite an existing file. Use the `isfile` function from the `os.path` module.

```
>>> import os.path
>>> os.path.isfile("slides11a-files.pdf")
True
>>> os.path.isfile("slides11a-files.png")
False
```

Here the filepath given is *relative* to the current directory.

Let's Take a Break

Example: Read Lines from File

```
import os.path

def main():
    """ Count lines in file. """
    # Does the file exist?
    if not os.path.isfile("gettysburg-address"):
        print("File does not exist")
        return
    # Open file for input
    gaFile = open("gettysburg-address", "r")
    lineCount = 0
    line = gaFile.readline()  # read the first line, if any
    while line:               # line is not empty string
        lineCount += 1
        print(format(lineCount, "3d"), ": ", \
              line.strip(), sep= "" )
        line = gaFile.readline()
    print("\nFound", lineCount, "lines.")
    gaFile.close()

main()
```

Example: Read Lines from File

```
> ls gettysburg-address
gettysburg-address
> wc gettysburg-address
  21  278 1475 gettysburg-address
> python ReadFile.py
  1: Four score and seven years ago our fathers brought
     forth on this
  2: continent, a new nation, conceived in Liberty, and
     dedicated to the
     ...
 21: freedom -- and that government of the people, by the
     people, for the
 22: people, shall not perish from the earth.

Found 22 lines.
```

Example: Write File

Recall our earlier example to generate and print a multiplication up to LIMIT. Below is the code to write the table to a file `MTable`.

```
LIMIT = 13

def main():
    """ Print a multiplication table to LIMIT - 1. """
    outfile = open("MTable", "w")
    outfile.write("Multiplication Table".center \
                       ( 6 + 4 * (LIMIT - 1)) + "\n")
    # Display the number title
    outfile.write("     |")
    for j in range(1, LIMIT):
        outfile.write(format(j, "4d"))
    outfile.write("\n")    # jump to a new line
    outfile.write("------" + "----"* (LIMIT - 1) + "\n")
```

Code continues next slide.

Example: Write File

Continued from previous slide.

```
    # Display table body
    for i in range(1, LIMIT):
        outfile.write( format(i, "3d") + " |")
        for j in range(1, LIMIT):
            # Display the product and align properly
            outfile.write( format( i*j, "4d"))
        outfile.write("\n")
    outfile.close()
```

There are some major differences between `print` and `write`:

1. `print` inserts a newline at the end of each line, unless you ask it not to. `write` does not do that.
2. `print` takes an arbitrary number of arguments and coerces them all to strings; `write` only takes one argument and it must be a string.

Example: Write File

```
> python MultiplicationTable2.py
> cat MTable
                 Multiplication Table
    |   1   2   3   4   5   6   7   8   9  10  11  12
------------------------------------------------------
  1 |   1   2   3   4   5   6   7   8   9  10  11  12
  2 |   2   4   6   8  10  12  14  16  18  20  22  24
  3 |   3   6   9  12  15  18  21  24  27  30  33  36
  4 |   4   8  12  16  20  24  28  32  36  40  44  48
  5 |   5  10  15  20  25  30  35  40  45  50  55  60
  6 |   6  12  18  24  30  36  42  48  54  60  66  72
  7 |   7  14  21  28  35  42  49  56  63  70  77  84
  8 |   8  16  24  32  40  48  56  64  72  80  88  96
  9 |   9  18  27  36  45  54  63  72  81  90  99 108
 10 |  10  20  30  40  50  60  70  80  90 100 110 120
 11 |  11  22  33  44  55  66  77  88  99 110 121 132
 12 |  12  24  36  48  60  72  84  96 108 120 132 144
```

Example: Reading One File, Writing Another

```python
import os.path

def CopyFile():
    """ Copy contents from file1 to file2. """
    # Ask user for filenames
    f1 = input("Source filename: ").strip()
    f2 = input("Target filename: ").strip()
    # Check if target file exists.
    if os.path.isfile( f2 ):
        print( f2 + " already exists" )
        return
    # Open files for input and output
    infile = open( f1, "r" )
    outfile = open( f2, "w" )
    # Copy from input to output a line at a time
    for line in infile:
        outfile.write( line )
    # Close both files
    infile.close()
    outfile.close()

CopyFile()
```

Running CopyFile

Here ls and cat are Linux/MacOS commands to list files and display the contents of a file, respectively.

```
> ls HelloWorld.py
HelloWorld.py
> cat HelloWorld.py
print( "Hello, World!" )
> ls NewHelloWorld.py
ls: cannot access 'NewHelloWorld.py': No
   such file or directory
> python CopyFile.py
Source filename: HelloWorld.py
Target filename: NewHelloWorld.py
> cat NewHelloWorld.py
print( "Hello, World!" )
```

Example: Reading and Writing File

It's dangerous to simultaneously read and write a file in Python. However, you can write a file, close it, and re-open it for reading.

In file WriteReadNumbers.py:

```python
from random import randint

def main():
    """ Write 100 random integers to a file, read them
        back, and print them, 10 per line. """
    # Open file for writing
    outfile = open("RandomNumbers.txt", "w")
    # Write 100 random integers in [0 .. 99] to file
    for i in range(100):
        outfile.write(str(randint(0, 99)) + " ")
    outfile.close()
```

Code continues on next slide.

Example: Reading and Writing File

Continued from previous slide.

```
    # Re-open the file for reading
    infile = open("RandomNumbers.txt", "r")
    # This will read the entire file into string.
    # We know this is a small file.
    string = infile.read()
    # Split string into numbers.
    numbers = [ int(x) for x in string.split() ]
    onLine = 0
    # Print them 10 per line.
    for num in numbers:
        print( format(num, "2d"), end = " ")
        onLine += 1
        if onLine == 10:
            print()
            onLine = 0
    infile.close()
```

Reading and Writing File

```
> python WriteReadNumbers.py
93  0 48 62 77 84 14 36 99 83
90 46 48 27 27 40 87 87 86 15
72  4 28 48 78 70 90 96 27 97
43 73 40 26 96 93 54 61 13 22
82 66 95 35 56 95 18 54 26 90
63 79  5 26 43 12 49 86 22 90
77 84 66 97 75 35 27 74 75  1
72  2 55 17 12 63 73 89  3 71
81 39 43 46 19 99 45 31 39 35
38 47 56 64 84 31 63 81  4 38
```

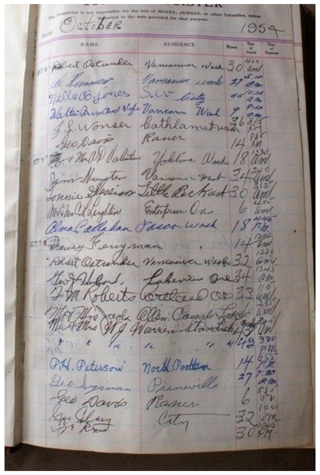
Append Mode

Opening a file in append mode "a", means that writing a value to the file appends it at the end of the file.

It *does not* overwrite the previous content of the file.

You might use this to maintain a log file of transactions on an account.

New transactions are added at the end, but all transactions are recorded.



Next stop: Tuples, sets and dictionaries.