CS303E: Elements of Computers and Programming

Files

Mike Scott
Department of Computer Science
University of Texas at Austin

Adapted from
Professor Bill Young's Slides

Last updated: June 23, 2022

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 (aka folders)*.

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 use the name of the file, you're assuming that it is in the current working directory.

```
plato% pwd
/u/scottm/314
plato% ls -l
total 8
drwx------ 17 scottm prof 4096 Sep 14  2020 grade
-rw-r--r--  1 scottm prof   42 Nov 25  2019 nums
-rw-r--r--  1 scottm prof   42 May  4 11:28 nums_sorted
-rw-r--r--  1 scottm prof   58 Nov 25  2019 simple.txt
drwx------  2 scottm prof 4096 Aug 19  2020 src
```

pwd -> print working directory
ls -l -> list the contents of the current directory in long form (with details)

Relative Pathnames

```
drwx------ 17 scottm prof 4096 Sep 14  2020 grade
-rw-r--r--  1 scottm prof   42 Nov 25  2019 nums
-rw-r--r--  1 scottm prof   42 May  4 11:28 nums_sorted
-rw-r--r--  1 scottm prof   58 Nov 25  2019 simple.txt
drwx------  2 scottm prof 4096 Aug 19  2020 src
plato% cat calculate_taxes.py
cat: calculate_taxes.py: No such file or directory
plato% cat src/calculate_taxes.py
def main():
    """Calculate US Federal Income Tax.

    Ask user for income and calculate US
    Federal income tax for 2021.
    Assumes user is filing single.
```

cat -> from con**cat**enate, synonym for append (in this case to standard output)
src/ means look for the file in the directory named src

File Paths

On Windows, a file path might be:

```
C:\Users\scottm\314\src\calculate_texas.py
```

On Linux or MacOS, it might be:

```
/home/scottm/314/src/calculate_texas.py
```

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 the for Windows pathnames including '\'

Python - Show the Current Working Directory

In CS303e when we open a file we will generally assume it is in the same directory as the running Python program.

When doing homework, how do you know what that is so you can put your data files in the same directory?

```
import os
print(os.getcwd())
```

```
print(os.getcwd()) # os already imported above
```

```
C:\Users\scottm\Documents\303e\_Su21\lecture_code\examples
```

Of course your output will be different.

Working with Files in Python

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

Functions for dealing with 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.

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.'

Example of Opening a File

General Form:

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

```
>>> outfile = open('test_file.txt', 'w')
>>> outfile.write('Testing can show the presence of bugs ...\n')
42
>>> outfile.write('but not prove their absence.\n')
29
>>> outfile.close()
```

What do you think the 42 and 29 (an int returned by the write function) represent above?

Notice we are calling a function (method) on a variable.
`outfile.write`

```
(lecture_code) C:\Users\scottm\Documents\303e\_Su21\lecture_code>type test_file.txt
Testing can show the presence of bugs ...
but not prove their absence.
```


Opening a File: Modes

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.

This semester we won't be using the binary modes.

In other words we are going to read from files assuming it is encoded as text. In binary we would read the raw 0s and 1s.

Closing the File

General form:

```
file_variable.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*.

Using the with statement

Although not in the textbook, the preferred way of opening a file is with the **with** statement. (Another Python keyword)

```
def demo_with(file_name):
    """Demonstrate creating file objects with the with keyword."""
    with open(file_name, 'r') as in_file:
        # Simply print the lines in the file
        for line in in_file:
            print(line, end='')
        print('Still in with. Is file closed? ',
              in_file.closed)
    # Is the file closed?
    print('After with block. Is file closed? ', in_file.closed)
```

```
Still in with. Is file closed?  False
After with block. Is file closed?  True
```


Reading/Writing a File

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

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

These functions advance an internal *file pointer (like a cursor in a word processing document or a program editor)* that indicates where in the file you're reading/writing. open sets the file pointer or cursor at the beginning of the file.

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.

```
>>> isfile('foo.txt')
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'isfile' is not defined
>>> import os.path
>>> os.path.isfile('foo.txt')
False
>>> os.path.isfile('test_file.txt')
True
```

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

Example: Read Lines from File

```
import os.path


def main():
    """Open the file. Print out all lines with a line number."""
    file_name = input('Enter file name: ')
    if not os.path.isfile(file_name):
        print(file_name, 'does not exist in the current directory.')
    else:
        file = open(file_name, 'r')
        line = file.readline()
        line_number = 0

        # Print out lines of file with line numbers.
        while line:
            line_number += 1
            print(format(line_number, '4d'), ': ',
                  line.strip(), sep='')
            line = file.readline()
        print('Found', line_number, 'lines.')
        print('Value of line that caused loop to stop:', line)
        file.close()
```

Example: Read Lines from File

```
Enter file name: lyrics.txt
   1: That's great, it starts with an earthquake
   2: Birds and snakes, and aeroplanes
   3: And Lenny Bruce is not afraid
   4:
   5: Eye of a hurricane, listen to yourself churn
    ...

  66: It's the end of the world as we know it (tim
  67: It's the end of the world as we know it (tim
  68: It's the end of the world as we know it and
Found 68 lines.
Value of line that caused loop to stop:
```


Example: Write File

Let's write out the flip of 10,000 coins to a file, H for heads, T for tails. 50 results per line separated by a space.

One major difference is that `print` inserts a newline at the end of each line, unless you ask it not to. `write` does not do that.

```
# Write out the results of coin flips to a file.
import random


def main():
    num_flips = 10_000
    flips_per_line = 50
    out_file = open('flip_results.txt', 'w')
    for i in range(1, num_flips + 1):
        side = 'H' if random.random() < 0.5 else 'T'
        out_file.write(side)
        if i % flips_per_line == 0:
            out_file.write('\n')
    out_file.close()
```

Part of Resulting File - Coin Flip Results

```
 1  THTHHTHHHHTTHHHHHHHHHTHTHTTHHTTTTHHTTTTTTTTTTTTTHH
 2  HHTHTTHTTHTTTHHTTTTHTHTHHTHTHHHHTHHTHHTTTHHHHTTHTH
 3  HHTHTTHHTTHTTTTTHTTTHTHTHHHTHHHHHTHTTHHHTTHHHTTHHH
 4  HHTHTHTHTHHHHTHTHTTHTHHHTTHTTHTTHTTTHTHTHTHTTHHHHH
 5  THTTTTTHTTHHTHHHHHTHHHHTTTHHHTTHHHTTHHHTHHTTTHTHTH
 6  HTHHTHTHHTHHTTHTHTTHHTTHHTHTTHTTHTHTTTHTHHHTTHTTTT
 7  HTTHHHTHTHTTHTHTTTTHHHHHHTTTHTTHHTTTHHTTHHHTHHHHHH
 8  THHTTHHTTTTTHHTHHTHHHTHTTHTTTHHTTTHHTTHTHTHTTTTHHT
 9  TTTHTHHTHHHTTHTTTTTHTHHTTHTHTTTTTTTTHHHHHTHHHHTTHT
10  TTTTTTTTHHHTHHHHTTHTHTHHHTHTTTTTHTHTHTHHHHHHTHTTT
11  THTHTHTTHHHHHHHHHTHTTHTTHHTHTHHTHTTHTHHTTHHHHHHTHH
12  HHHHHTHHTHHHHTTHHHHTHTTHHHHTHTHHTTHTHTTTHHTHHHTTTT
13  THTTHHHTTHTTTHHTTHTTTHHHTHHHTTHHHTTHTTTTTHTTTTHHT
14  THTTHTHHTTHTTTTHHHHTTHTHHHHTHTTTTHHHHHTTHHTHHHTHTH
15  TTTHHHTHTHHTTTHHTTHTHTHHTTTHTTHTHTHHTHHTTHTTTHHHHH
16  HHHTHTHTHHHHHTHTHHTHHTHTHTHTTHTHTHHHTTTTHTTHTTTHHH
17  HHTTHHTHHTHTTTHHHTHHHTTTHHHHHHTHTHTTHHTHHTHTTHHHTT
18  HTTHTHTTTTTHHHHTHHHHHHTTTTTTHTTTTHHHTHTTHHHTHHTHTT
19  TTHTHHHTTHTHTTHTTTHTHTTTTHTHTHTHTTHHHTHTTHTHTHTTHT
20  HHHTHHTHHTTHHTHTHHHTTTHHTHTHHTTHTTHTTHHHTTTHTHHHTH
```

Note, the line numbers are NOT part of the file. They are shown by the text editor I used.

Aside: Redirecting Output

There's another way to get the output of a program into a file.

When your file does a `print`, it sends the output to standard out, which is typically the terminal.

You can *redirect* the output to a file, using `> filename` on Linux systems. Anything that would have been printed on the screen goes into a file instead.

Notice that this happens at the OS level, not at the Python level. *Programmers know how to do things multiple ways!*

Can even redirect standard output inside of a Python program. This is part of how the auto grader works. Redirecting your program's standard output so we can compare it to what we expect the output to be.

Aside: Redirecting Output

```
plato% ls -l
total 36
drwxrwxr-x  3 scottm usl  4096 Feb  2  2001 00Fall
drwxr-x---  2 scottm usl  4096 Dec 18  2000 00Spring
drwx------  2 scottm prof 4096 Jun  2  2020 assignmn_solutions
drwxr-sr-x  2 scottm usl  4096 Feb  7  2001 Fall2000
drwx------ 16 scottm prof 4096 Jul 28  2020 grading
-rw-r--r--  1 scottm prof  422 Jun  3 10:54 hello_world.py
drwxr-sr-x  2 scottm usl  4096 May 20  2001 Quilt
drwxr-sr-x  2 scottm usl  4096 Feb 16  2001 Rock
drwxr-sr-x  2 scottm usl  4096 Feb  7  2001 Spring2000
plato% python hello_world.py > output.txt
plato% ls
00Fall    assignmn_solutions  grading         output.txt  Rock
00Spring  Fall2000            hello_world.py  Quilt       Spring2000
plato% cat output.txt
3
Hello World!
Hook 'em Horns!
362880
plato%
```


Example: Reading and Writing File

```
import os.path

def copy_file():
    ''' 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()

copy_file()
```

Notice the use of a for loop to read all the lines in the file.

Example: Reading and Writing File

One cannot simultaneously read and write a file in Python.
However, you can write a file, close it, and re-open it for reading.

```
import random


def main():
    """Write out 100 random integers to a file, then read the file."""
    outfile = open('random_nums.txt', 'w')
    for i in range(100):
        outfile.write(str(random.randint(0, 99)) + ' ')
    outfile.close()

    # Now read in the numbers and print 10 per line
    infile = open('random_nums.txt', 'r')
    nums = infile.read()
    print(nums, '\n')

    numbers = [int(x) for x in nums.split()]
    num_printed = 0
    for x in numbers:
        num_printed += 1
        print(format(x, '3d'), end='')
        if num_printed == 10:
            print()
            num_printed = 0
        else:
            print(' ', end='')
```

Reading and Writing File

```
 19  52  13  78  48  67  56  10   8  26
 34  54  75  80  16  85  83  97  40  70
 55   3  30  67  70  85   6  11  80   0
 56  56  28  57  67   2  57  52  90  52
 79  85  87  74  24  50  67  74  64  32
 71  42  97  22  75  57   7  18  77   1
 29  74  43  62  53  28  35  21  23  18
 48  82  22  71  62  23  84  98  53  36
 11  79  72  32  57  93   1  59  37  18
 42  27  43  54  11  50  12  77  80  43
```

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.