Introduction to Programming in Python

File I/O

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

Last updated: June 4, 2021 at 11:05

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.

```
> 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 the for Windows pathnames including "\"

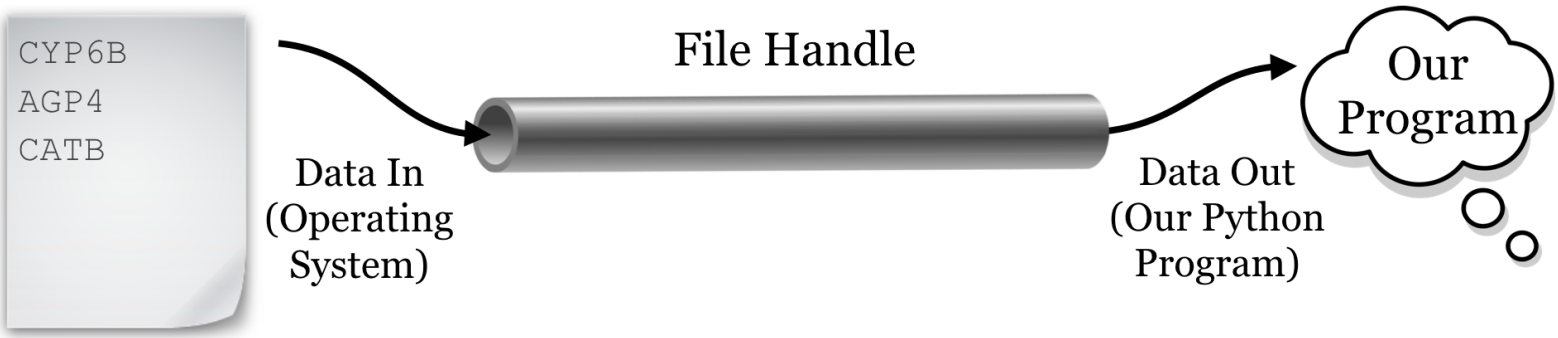
Managing Files in Python

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."

Example of Opening a File

General Form:

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

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

> cat MyNewFile     # OS command to show file
My dog has fleas!
```

Opening a File: Modes

Here are the 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.

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

There are also some *binary* modes, but we won't use them.

Closing the File

General form:

```
fileVariable.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*.

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 `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* that indicates where in the file you're reading/writing. `open` sets it 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.

```
>>> 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.

Example: Read Lines from File

```
import os.path

FILENAME = "Raven.txt"

def main():
    """ Print lines from file, keeping a count. """
    if not os.path.isfile( FILENAME ):
        print("File does not exist")
        return

    # Open file for input
    ravenFile = open( FILENAME, "r")
    line = ravenFile.readline()
    lineCount = 0
    while line:                    # line is not empty string
        lineCount += 1
        print(format(lineCount, "3d"), ": ", \
              line.strip(), sep= "" )
        line = ravenFile.readline()
    print("\nFound", lineCount, "lines.")
    ravenFile.close()

main()
```

Example: Read Lines from File

```
> python readFromFile.py
  1: Once upon a midnight dreary, while I pondered, weak and
       weary,
  2: Over many a quaint and curious volume of forgotten lore
       .
  3: While I nodded, nearly napping, suddenly there came a
       tapping,
  4: As of some one gently rapping, rapping at my chamber
       door.
  5: "'Tis some visitor," I muttered, "tapping at my chamber
       door,
  6: Only this and nothing more."
  7:
  8: Ah, distinctly I remember it was in the bleak December;
  9: And each separate dying ember wrought its ghost upon
       the floor.
...
 19: Some late visitor entreating entrance at my chamber
     door;
 20: This it is and nothing more."

Found 20 lines.
```

Example: Write File

Recall our earlier example to generate a Student Grade Report:

```
Grades for Susie Q.
  Exam1: 75.0
  Exam2: 94.44
  Exam3: 87.69
Exam average: 85.71
  Proj1: 95.0
  Proj2: 75.0
Proj average: 85.0
Course average: 85.43
```

Example: Write File

We used code like this:

```
    # Left out the code to generate the variables:

    print("Grades for", student)
    print("  Exam1:", round(exam1Norm, 2))
    print("  Exam2:", round(exam2Norm, 2))
    print("  Exam3:", round(exam3Norm, 2))
    print("Exam average:", round(examAvg, 2))

    print("  Proj1:", round(proj1Norm, 2))
    print("  Proj2:", round(proj2Norm, 2))
    print("Proj average:", round(projAvg, 2))

    print("Course average:", round(courseAvg, 2))
```

Example: Write File

Here's the code if we're writing the output to a file instead.

```
# Print the student's report to file GradeReport.txt:
of = open("GradeReport.txt", "w")
f1.write("Grades for " + student + "\n")

f1.write("  Exam1: " + str(round(exam1Norm, 2)) + "\n")
f1.write("  Exam2: " + str(round(exam2Norm, 2)) + "\n")
f1.write("  Exam3: " + str(round(exam3Norm, 2)) + "\n")
f1.write("Exam average: " + str(round(examAvg, 2)) + "\n")

f1.write("  Proj1: " + str(round(proj1Norm, 2)) + "\n")
f1.write("  Proj2: " + str(round(proj2Norm, 2)) + "\n")
f1.write("Proj average: " + str(round(projAvg, 2)) + "\n")

f1.write("Course average: " \
             + str(round(courseAvg, 2)) + "\n")
f1.write("\n")

f1.close()
```

Unlike `print`, for every `write` the argument must be a single string.