File Input/Output

Most data is stored in files, not input by the user every time. In this activity, you'll learn the basics of reading and writing plain text files.

Content Learning Objectives

After completing this activity, students should be able to:

- Create a new text file, and output several lines to it.
- Open an existing file, and append several lines to it.
- Read a text file line by line, and extract data from it.

Process Skill Goals

During the activity, students should make progress toward:

- Justifying answers based on the results of an experiment. (Critical Thinking)

Facilitation Notes

Be aware of what editors and operating systems your students are using. They might need help changing directories to find the input/output files used in this activity. Model 1 may take a little more time for setup and discussion. The source code should be provided (see `write.py` in the repository).

Model 2 switches from using an Editor to using the Shell. Help students figure out that the `write` method returns the number of characters written. Depending on the environment, they might not see the output file contents until after calling the `close` or `flush` method. If it comes up, discuss how buffering works when reporting out.

Notice that the input file for Model 3 is the `out.txt` file from Model 1 and Model 2. If students corrupt their `out.txt` file, some of their answers might be incorrect. If needed, you can give them a correct version of `out.txt` before they complete the Model 3 table. Report out after most teams reach Question #12, so that teams will have adequate time to answer the last question.

By the way, the `out.txt` data is in FASTA format, a standard in bioinformatics for representing sequences. See https://en.wikipedia.org/wiki/FASTA_format.

Model 1 Writing to a File

The following example creates a new file (in the current/default folder) named `out.txt` and writes several lines of output to it. Run the code, and examine the contents of the resulting `out.txt` file. In the space below, write the contents of `out.txt` to the right of the code.

```python
outfile = open("out.txt", "w")
outfile.write("Example ")
outfile.write("output ")
outfile.write("text file\n")
outfile.write("xyz Coordinates\n")
outfile.write("MODEL\n")
outfile.write("ATOM %3d" % 1)
seq = "n %5.1f%5.1f%5.1f" % (0, 1, 2)
outfile.write(seq)
outfile.write("\n")
outfile.close()
```

out.txt

```
Example output text file
xyz Coordinates
MODEL
ATOM   1n   0.0  1.0  2.0
```

Questions (15 min)

Start time: ________

1. Based on the Python code:

   a) How many arguments are passed to `open`? What are their types? two strings

   b) What variable stores the ***file object*** returned by the `open` function? outfile

   c) Identify the names of all methods used on this file object in the code. write, close

   d) What type of data does the `write` method require for its argument? string

2. Based on the `out.txt` file:

   a) How many times was the `write` method called to create the first line of text? 3

   b) How many times was the write method called to create the second line of text? 1

   c) What does the `"\n"` character do? It ends the current line.

   d) How is the `write` method different from the `print` function? Doesn't append a newline.

3. Write a program that creates a file named `lines.txt` and writes 100 lines like this:

```
Line #1
Line #2
Line #3
Line #4
...
```

```python
outfile = open("lines.txt", "w")
for i in range(1, 101):
    outfile.write("Line #%d\n" % i)
outfile.close()
```

Model 2 Appending to a File

The second argument of `open` specifies the ***mode*** in which the file is opened. When writing output to a file, there are two basic modes:

- The write (`"w"`) mode will overwrite/replace the file contents.
- The append (`"a"`) mode will add new data to the end of the file.

Either mode will create the file automatically if it does not already exist. Enter the following lines into a Python Shell, and record the output at each step.

Python code	**Shell output**
`afile.write("new line\n")`	NameError: name 'afile' is not defined
`afile = open("out.txt", "a")`	
`afile.write("new line\n")`	9
`afile.write(2.0)`	TypeError: write() argument must be str, not float
`afile.write("2.0")`	3
`afile.close()`	
`afile.write("new line\n")`	ValueError: I/O operation on closed file.

Questions (10 min) Start time: ________

4. Explain what happens as a result of the line: `afile = open("out.txt", "a")`

The existing out.txt file (from Model 1) is open for appending.

5. How do the arguments passed to the `open` function differ for writing a new file in comparison to appending an existing file?

The second argument is `"a"` instead of a `"w"`.

6. What does the `write` method return? Run `help(afile.write)` to check your answer.

The number of characters written (which is always equal to the length of the string).

7. Explain the reason for the error observed after entering:

a) the first line of code: `afile.write("new line\n")` the file wasn't open

b) the last line of code: `afile.write("new line\n")` the file wasn't open

c) the statement: `afile.write(2.0)` you can only write strings to files

Model 3 Reading from a File

Programs often require input data from an external file source. Not surprisingly, there are methods for reading the contents of files. Enter the following lines into a Python Shell.

Python code	Shell output
`infile = open("out.txt", "r")`	
`infile.readline()`	'Example output text file\n'
`infile.readline()`	'xyz Coordinates\n'
`infile.readlines()`	list of the remaining lines
`infile.readline()`	''
`infile.close()`	
`infile = open("out.txt", "r")`	

`for line in infile:`
`    print(line)`
	prints each line double spaced

`infile.close()`	
`infile = open("out.txt", "r")`	

`for i in range(3):`
`    infile.readline()`	

`line = infile.readline()`	
`line`	'ATOM 1n 0.0 1.0 2.0\n'
`print(line[0])`	A
`print(line[0:5])`	ATOM
`words = line.split()`	
`words`	['ATOM', '1n', '0.0', '1.0', '2.0']
`print(words[0])`	ATOM
`infile.close()`	

Questions (20 min) Start time: ________

8. Based on the output above:

a) What type of data does the `readline` method return? string

b) What type of data does the `readlines` method return? list of strings

9. Why did the `readline` method return different values each time?

Each time `readline` is called, it returns the next line of the file.

10. What happens if you try to read past the end of the file? Justify your answer.

From then on, `readline` returns `''`, and `readlines` returns an empty list.

11. What is the difference between the two `for` loops in Model 3?

The first loop iterates and prints every line of the file. The second for loop reads only the first three lines, making it possible to assign the fourth line in the next statement.

12. Consider the output of the first `for` loop:

a) Why does the program display the file as if it were double spaced?

Each line in the file ends with a newline character \n, and `print` adds another one.

b) How would you change the code to avoid printing extra blank lines?

Change the function: `print(line, end='')` Or change the string: `print(line[:-1])`

13. Based on the second half of Model 3:

a) Why was it necessary to open the file again? The data had already been read previously.

b) Write code that would output 1.0 using `line` print(line[17:20])

c) Write code that would output 1.0 using `words` print(words[3])

14. Consider a file `names.txt` that contains first and last names of 100 people, with one name per line (e.g., "Anita Borg"). Write a program that prints all the last names (the second word of each line) in the file.

```
infile = open("names.txt", "r")
for line in infile:
    words = line.split()
    print(words[1])
infile.close()
```