Reading and writing files

We can interact with files in three modes

1. **read mode** ("r"):
   Read data from file into python variables. File remains unchanged.
2. **write mode** ("w"):
   Write data from python variables to file. Previous file contents is overwritten.
3. **append mode** ("a"):
   Add data to the end of an existing file.

Working with files is a three-step process

1. Open the file
2. Interact with the file
   (read from it, write to it)
3. Close the file

It is critical to always close every file you have opened!

We interact with files via file handles

- A file handle is a python object that allows us to interact with a file.
- Example 1. Open file for reading:

```
# the `open()` function opens the file
# and returns a handle
file_handle = open("file.txt", "r") # open in 'r' mode
contents = file_handle.read() # reads the entire file
file_handle.close() # always close at the end
```

We interact with files via file handles

- A file handle is a python object that allows us to interact with a file.
- Example 2. Open file for writing:

```
# the `open()` function opens the file
# and returns a handle
file_handle = open("file.txt", "w") # open in 'w' mode
# write one line
file_handle.write("New file contents.\n")
file_handle.close() # always close at the end
```

Intermission: The newline character ("\n")

The newline character ("\n")

```
In [1]: s = "String with newline.\n"
        # print() adds an additional "\n"
        print(s)
        print(s)

Out[1]: String with newline.

        String with newline.
```

The newline character ("\n")

```
In [1]: s = "String with newline.\n"
        # we can use `end` to suppress 2nd "\n":
        print(s, end='')
        print(s, end='')

Out[1]: String with newline.
        String with newline.
```

The newline character ("\n")

```
In [1]: s = "String with newline.\n"
        # or remove the "\n" using .rstrip():
        print(s.rstrip())
        print(s.rstrip())

Out[1]: String with newline.
        String with newline.
```

The newline character ("\n")

```
In [1]: s = "String with newline.\n"
        # using both eliminates all newlines:
        print(s.rstrip(), end='')
        print(s.rstrip(), end='')

Out[1]: String with newline.String with newline.
```

The newline character ("\n")

Unlike `print()`, the `.write()` function does not add a "`\n`".

Back to files

There are multiple ways to read a file

1. Read the whole file at once:

```
contents = file_handle.read()
# The variable `contents` now holds the
# entire file in one long string.
```

There are multiple ways to read a file

2. Read the file into a list of lines:

```
lines = file_handle.readlines()
# The variable `lines` now holds a list of
# strings, each corresponding to one line
# in the file
```

There are multiple ways to read a file

3. Iterate over the file in a `for` loop:

```
for line in file_handle:
    code block
# The code in the code block is executed
# once for each line in the file.
```

Let Python close the file for you: The `with` statement

```
# traditional open – work with file – close sequence
file_handle = open("file.txt", "r")
contents = file_handle.read()
file_handle.close()

# alternative form using `with`
with open("file.txt", "r") as file_handle:
    contents = file_handle.read()
# the file is closed automatically when the indented
# code-block ends.
```