# Learning Objectives - Reading

- **Demonstrate how to open a file in read mode**
- **Explain what happens when you read from a file that does not exist**
- **Iterate through a file line by line (for loop and while loop)**
- **Explain what represents the end of a file**
- **Demonstrate reading a file more than once**
- **Differentiate between “read and write” and “read and append”**
- **Read from one file and write to another**
- **Find a keyword with the `seek` method**

# Read Mode

## Reading a Non-Existing File

Read mode works much like write or append mode. Use a `"r"` when opening a file. Notice, there a file named `read_practice.txt`. Remove the underscore from the filename to see if a new file will be created like with the write and append modes.

```
read_file = open("student_folder/text/readpractice.txt", "r")
read_file.close()
```

Read mode does not create a new file if the file does not exist. You can only read previously existing files.

## Reading a File

Return the underscore to the filename so that it is `read_practice.txt`. Use a print statement to print the contents of the file to the screen.

```
read_file = open("student_folder/text/read_practice.txt", "r")
print(read_file)
read_file.close()
```

Notice that Python does not print the contents of the text file. Instead it prints information about the file object (name, mode, and encoding). If you want to print the contents of a text file, use the `readlines` method.

▼ **UTF-8**
There are many characters that go beyond letters and numbers. For example, punctuation, accents, symbols, emojis, etc. There needs to be a way to use these characters in a way that everybody can understand. Encoding is an agreed upon method for representing these characters. UTF-8 is by far the most popular way to encode text today. If you want to know more, check out this article.

```
read_file = open("student_folder/text/read_practice.txt", "r")
print(read_file.readlines())
read_file.close()
```

Notice how the text is a list of strings, and the newline character (`\n`) is a part of the output.

# File Iteration - For Loop

## File Iteration - For Loop

The `readlines` method returns all of the text at once. A loop allows you to deal with each line individually. To print out the text as it would normally appear, use a for loop to print each line of the text.

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    for line in read_file.readlines():
      print(line)
```

▼ **Reading a file multiple times**

When Python reaches the end of a file with `readlines`, it will not start back at the beginning until you close the file and then reopen it.

challenge

### What happens if you:

- Change the print statement to `print(line.upper())`?
- Change the print statement to `print(line.replace("a", "\U0001F61C"))`?

## Extra Lines

You probably noticed that the printed text has an empty line between lines of text. If you open the original text file, there are no empty lines. The original text has a newline character at the end of each line. The print command also adds a newline character by default. So Python prints two new lines: the first one goes to the next line, the second one creates the blank line. Use `end=""` in the print statement to remove the blank line. This replaces the newline character with an empty string.

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    for line in read_file.readlines():
      print(line, end="")
```

challenge

## What happens if you:

- Change the print statement to `print(line, end="!")`?
- Change the print statement to `print(line, end="\n\n\n")`?

# File Iteration - While Loop

## Readline vs. Readlines

The `readlines` method reads all of the lines at once and returns them in a list. There is another method called `readline`. The `readline` method returns only one line from the text file. Python automatically keeps track of which lines have been read, and which have not.

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    print(read_file.readline())
```

challenge

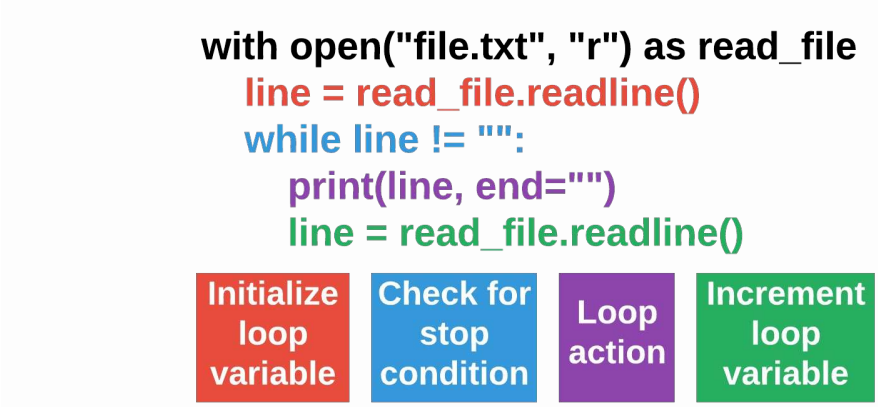
### What happens if you:

- Change `readline()` to `readlines()`?
- Change the code to look like this:

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    print(read_file.readline(), end="")
    print(read_file.readline(), end="")
```

## While Loop

While loops can also iterate over a text file. Remember, while loops require a stop condition to keep it from becoming an infinite loop. Each text file ends with an empty string (`""`). That is how Python knows it has reached the end of a file. So a while loop should each line of the text file until it reaches the empty string. Because of this, iterating over a text with a while loop will use `readline` instead of `readlines`. It is important to read the first line of the text file **before** starting the while loop. And, read the next line of the text file **inside** the while loop as well.




Reading a File with a While Loop

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    line = read_file.readline()
    while line != "":
        print(line)
        line = read_file.readline()
```

challenge

## What happens if you:

- Chage the first `line = read_file.readline()` to `readlines()`?
- Remove the first `line = read_file.readline()`?
- Put back the first `line = read_file.readline()` and remove the second one?

# Seek Method

## Seek Method

The seek method takes an integer as a parameter, and causes Python to go to a specific character in the text file. The integer is the index for the text file. So `seek(0)` is the first character of the file, `seek(1)` is the second character, etc.

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    read_file.seek(30)
    print(read_file.readline())
    read_file.seek(0)
    print(read_file.readline())
```

challenge

### What happens if you:

- Change a `seek` to `180`?
- Change a `seek` to `1000`?
- Change a `seek` to `-1`?

## Read a File Multiple Times

It was previously stated that a file cannot be read multiple times. That is true when trying to use `readlines` twice. You must close and then open the file to read it again. However, the `seek` method can also be used to move Python back to the beginning of the text file.

```
with open("student_folder/text/read_practice.txt", "r") as
        read_file:
    print("First Time")
    for line in read_file.readlines():
        print(line, end="")
    read_file.seek(0)
    print("\n\nSecond Time")
    for line in read_file.readlines():
        print(line, end="")
```

# Read & Write - Two Files

## Read from One File and Write to Another

It is possible to open a file in read mode and another in write mode. Information can be passed from the reading file to the writing file. Like previous examples, `with open` will be used to open both files at once. The code below will open the file `read_practice` in read mode, create `destination.txt` in write mode, read the lines from the source file, and write these lines to the destination file.

```
with open("student_folder/text/read_practice.txt", "r") as
        source, open("student_folder/text/destination.txt", "w")
        as dest:
    for line in source.readlines():
        dest.write(line)
```

▼ **`write` vs `writelines`**

In the code above the `write` method is being used to write text to a file. Previously, the method `writelines` was used to write text to a file. What's the difference? `writelines` can accept a single string or a list of strings as shown in previous lessons. `write`, however, can only accept a single string.

challenge

## What happens if you:

- Change the for loop to:

```
for line in source.readlines():
    dest.write(line.upper())
```

- Change the for loop to:

```
for line in source.readlines():
    line += "\n"
    dest.write(line)
```

# Formative Assessment 1

# Formative Assessment 2