Reading and Writing to Files

Why is this important?

Files are fundamental part of programming. They allow us to store data persistently, which means that the data will still be there even after the program has stopped running. This is important for many applications, such as web applications, games, and data analysis.

Doesn't matter if it's a text file, a csv file, or a json file, being able to read and write files is a fundamental skill that every programmer should have.

What are we going to do?

We're going to create a journal application that will allow us to read and write entries to a file.

Steps

1. Let's open our `journal.txt` file and read the contents.

First we need to know the page of the file we want to read.

In this case, we want to read the `journal.txt` file which is in the same directory as our script.

With files you need to use `open()` to open the file, and then you can read it using `read()`, `readline()`, or `readlines()`.

Let's create a `read_journal` function that will read the contents of the file and return the lines as a list.

```
def read_journal(file_path):
    """Read the journal entries from a file."""
    with open(file_path, 'r') as file:
        # option 1: read all of the file at once
        entries = file.readlines()
    return entries
```

Let's breakdown the code:

- We define a function `read_journal` that takes a file path as an argument.
- We use the `with` statement to open the file in read mode (`'r'`). This ensures that the file is properly closed after we're done with it. The options for opening a file are:
    - `'r'`: read mode (default)
    - `'w'`: write mode (overwrites the file)
    - `'a'`: append mode (adds to the end of the file)
    - `'b'`: binary mode (for binary files)
- We use `file.readlines()` to read all the lines in the file and store them in the `entries` variable. This returns a list of lines.
    - The other options are:

    - `file.read()`: reads the entire file as a single string.
    - `file.readline()`: reads a single line from the file.
- Finally, we return the `entries` list.

Note: the `with` statement is a context manager that automatically closes the file when we're done with it, even if an error occurs.

2. Let's create the `__main__` function to read the journal entries and print them.

Let's use this read function to read the `journal.txt` file and print the entries.

```
# ... read_journal function ...

def main():
    file_path = 'journal.txt'
    entries = read_journal(file_path)
    print(F"Journal Entries of {file_path}:\n")
    for index, entry in enumerate(entries):
        print(entry)

if __name__ == "__main__":
    main()
```

Now if you execute the script `python journal_app.py`, this will be the output.

```
Journal Entries of journal.txt:

I've been learning python and it's pretty fun so far!

I'm really excited to learn how to use pip and virtualenv, and learn more
about the python ecosystem.
```

3. Let's see what happens when we try to read a file that doesn't exist.

Change the `file_path` to a file that doesn't exist, for example `non_existent_file.txt`.

Now run the script again `python journal_app.py` and you will see this error.

```
$ python journal_app.py
Traceback (most recent call last):
  File
"C:\Users\dmouris\course_development\sdev1001_programming_fundamentals\sde
v1001-master-
course\files_reading_and_writing\read_and_write_fundamentals_end\journal_a
pp.py", line 19, in <module>
    main()
  File "C:\path\to\journal_app.py", line 12, in main
```

```
    entries = read_journal(file_path)
              ^^^^^^^^^^^^^^^^^^^^^^
  File "C:\path\to\journal_app.py", line 5, in read_journal
    with open(file_path, 'r') as file:
         ^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory:
'non_existant_file.txt'
```

When you try to read a file that doesn't exist, Python raises a `FileNotFoundError`. This is an exception that indicates that the file could not be found.

External files are something that can change, and aren't guaranteed to be there so let's use a `try` and `except` block to handle this error gracefully.

4. Let's handle the `FileNotFoundError` exception.

Now we will modify the `read_journal` function to handle the `FileNotFoundError` exception. This way, if the file does not exist, we can return a friendly error message instead of crashing the program.

```
def read_journal(file_path):
    """Read the journal entries from a file."""
    try:
        with open(file_path, 'r') as file:
            # option 1: read all of the file at once
            entries = file.readlines()
        return [entry.strip() for entry in entries if entry.strip()]
    except FileNotFoundError:
        return ["Error: The file does not exist. Please check the file
path."]

# ... main function ...
```

Let's rerun the script again with the `file_path` set to `non_existent_file.txt`.

```
$ python journal_app.py
Journal Entries of non_existant_file.txt:

Error: The file does not exist. Please check the file path.
```

Now, instead of crashing the program, we get a friendly error message that tells us the file does not exist.

5. Let's write to the journal file.

Now that we can read from the journal file, let's add a function to write entries to the journal file.

```
def write_journal(file_path, entry):
    """Write a journal entry to a file."""
```

```
    with open(file_path, 'a') as file:  # 'a' mode to append
        file.write(F"{entry}\n")
```

Let's break down the code:

- We define a function `write_journal` that takes a file path and an entry as arguments.
- We use the `with` statement to open the file in append mode (`'a'`). This means that if the file does not exist, it will be created, and if it does exist, the new entry will be added to the end of the file.
- We use `file.write()` to write the entry to the file, followed by a newline character (`\n`) to ensure that each entry is on a new line.

6. Let's modify our `main` function to promprt the user if they want to add a new entry or read the journal entries.

First change the `file_path` back to `journal.txt`.

```
# ... read_journal and write_journal functions ...

def main():
    file_path = 'journal.txt'
    while True:
        action = input("Do you want to (r)ead the journal or (w)rite a new
entry? (q to quit): ").strip().lower()
        if action == 'r':
            entries = read_journal(file_path)
            print(F"Journal Entries of {file_path}:\n")
            for index, entry in enumerate(entries):
                print(entry)
        elif action == 'w':
            new_entry = input("Enter your journal entry: ")
            write_journal(file_path, new_entry)
            print("Entry added to the journal.")
        elif action == 'q':
            print("Goodbye!")
            break
        else:
            print("Invalid option. Please try again.")
```

You can see that we add a loop that will keep asking the user if they want to read or write to the journal until they choose to quit.

Here's a sample run of the program:

```
$ python journal_app.py
Do you want to (r)ead the journal or (w)rite a new entry? (q to quit): w
Enter your journal entry:  Here's my first entry from the journal app.
Entry added to the journal.
Do you want to (r)ead the journal or (w)rite a new entry? (q to quit): w
Enter your journal entry: Wow I can see the files being changed if I open
```

```
it in vscode!
Entry added to the journal.
Do you want to (r)ead the journal or (w)rite a new entry? (q to quit): r
Journal Entries of journal.txt:

I've been learning python and it's pretty fun so far!
I'm really excited to learn how to use pip and virtualenv, and learn more
about the python ecosystem.
Here's my first entry from the journal app.
Wow I can see the files being changed if I open it in vscode!
Do you want to (r)ead the journal or (w)rite a new entry? (q to quit): q
Goodbye!
```

Now if you open the `journal.txt` file in a text editor, you will see the new entries added to the file.

Conclusion

In this lesson, we learned how to read and write files in Python. We created a simple journal application that allows us to read and write entries to a file. We also learned how to handle exceptions when reading files that do not exist.