Data Persistence

Overview

- Opening and closing files
- Reading data from file
- Writing data to files

Learning Objectives

- Identify the importance of various file handling procedures
- Implement file handling using Python

More to just files than reading and writing.

Also cover how Python handles files and how to prevent it from causing errors.

Input and Output

I / O

Input is the data you feed into your application.

Output is the new data that your application produces.

Introduce Input and Output. It coves more than just data persistence. Input() and Print() are also input and output.

Types of I / O

- Hardware such as a keyboard and monitor
- User commands and program text on the console
- Files
- Network

Ask for other types of I/O.

Saving data permanently

Any data that is not hardcoded into the application will be lost when the application stops

Data needs to be saved into a file for it to outlive the application.

File Handling

These are the main things you want to do with a file:

- Create
- Open
- Read
- Write
- Close

What else can we do with files? eg. delete, change file name.

Opening a File

```
file = open("my_file.txt", "r")
```

`open`: built-in function to create or open a file

`my_file.txt`: path to the file, relative to the program

`"r"`: file access mode (read)

There are more file modes we will look at in the next slide.

File Access Mode

A file access mode is a character that specifies the mode in which the file is opened. The default is `r`.

1. `r`: read-only
2. `r+`: both read and write
3. `w`: write-only - overwrites the file
4. `w+`: both read and write - overwrites the file
5. `a`: write-only - appends to the file
6. `a+`: both read and write - appends to the file

More exist but these are the ones we need to know for now.

Reading from a File

```
# hello.txt:
Hello!
```

```
# hello.py
file = open("hello.txt", "r")
contents = file.read()
print(contents)
```

```
$ python hello.py
Hello!
```

Demonstrate what happens if there is text on more than one line. Ask the learners to say what happens.

Quiz Time! 🤓

I have a file called `file.txt` which has a single line of text - **"hello world!"**. I open the file using `file.open("file.txt", "r+")`. I then immediately close the file. What is now the content of my file?

1. **"hello world!"**
2. **"hello world!"** \n
3. The file is blank
4. The file has been deleted

Answer: `1`

This question illustrates that opening with 'r+' does not alter the contents of the file - it just opens an existing file

---

I have a file called `file.txt` which has a single line of text - **"hello world!"**. I open the file using `file.open("file.txt", "w+")`. I then immediately close the file. What is now the content of my file?

1. **"hello world!"**
2. **"hello world!"** \n
3. The file is blank
4. The file has been deleted

Answer: `3`

This question illustrates that opening with 'w+' overwrites an existing file - so we have actually created a new file called `file.txt` and written over the old `file.txt`. As we have added no content to the new `file.txt`, the contents will be blank.

Worth also noting here that using 'w+' will create the file if it didn't already exist.

I have a file called `file.txt` which has a single line of text - **"hello world!"**. I open the file using `file.open("file.txt", "r+")`. I then run `contents = file.read()` followed by `print(contents)`. What is my output?

1. **"hello world!"**
2. A blank line
3. I will get an error

This question illustrates that, if you open with 'r+', you open at the beginning of the file. So using `file.read()` will read the first line of content

Answer: `1`

I have a file called `file.txt` which has a single line of text - **"hello world!"**. I open the file using `file.open("file.txt", "a+")`. I then run `contents = file.read()` followed by `print(contents)`. What is my output?

1. **"hello world!"**
2. A blank line
3. I will get an error

Answer: `2`

This question illustrates that, if you open with 'a+', you open at the end of the file. So using `file.read()` will result in just reading a blank line

---

Reading Line-by-Line

You can read the contents of the file line by line:

```
# people.txt
Lisa
John
Bob
```

```
# people.py
file = open("people.txt", "r")
lines = file.readlines()
for line in lines:
    print(line)
```

```
$ python people.py
Lisa
John
Bob
```

Exercise

1. Create a text file in your text editor and add some names of people on each line
2. Read the names from the file in your application and populate an empty list with the names
3. Print out the list!
4. Did you notice something weird?

Do a code-along or ask for a volunteer.

Answer

```
items = []
file = open("people.txt", 'r')
for line in file.readlines():
    items.append(line)

print(items)
```

```
$ python app.py
['Lisa\n', 'John\n', 'Bob\n']
```

Every line in the file is terminated by a newline character '\n'. We need to trim that off.

Exercise

Find a way to trim off the newline character from every line you read in your app.

Google is your friend here!

Another good way to engage learners for an answer.

Solution

Python has some built-in functions to help us with this:.

`rstrip()` will strip all trailing whitespace characters:

```
>>> 'test string \n \r\n\n\r \n\n'.rstrip()
'test string'
```

`strip()` will strip off all whitespace characters around any characters:

```
>>> s = "   \n\r\n  \n  abc   def \n\r\n  \n  "
>>> s.strip()
'abc   def'
```

\n is newline. \r is a return carriage.

Opening Files Safely

Remember our good friend, the `try-except` block?

```
try:
  file = open('file.txt', 'r')

except FileNotFoundError as fnfe:
  print('Unable to open file: ' + str(fnfe))

except Exception as e:
  print('An error occurred: ' + str(e))
```

Note: you can chain `except`s!

An example of catching errors programmatically. Where else could this be used?.

Writing to files

```
file = open('people.txt', 'w')
file.write('Susan')
```

What's the issue?

Open the file for the users.

Put a break point on the line after the write, so we've written data but we haven't closed the file off.

It will be blank as the file has not been released and the data is stored in memory.

Always Close Your Files

```
file = open('people.txt', 'w')
file.write('Susan\n')
file.close()
```

When we close a file, we release our handle on it, and push our changes to disk. This allows other programs to read from or write to the latest version of the document.

Repeat the example to show the data is written after a close.

---

try-except-finally

The `finally` keyword can be used with `try` or `try-except` to execute a block of code, no matter what the outcome is.

```
file = None

try:
  file = open(filepath, 'w')
  for item in items:
    file.write(item + '\n')

except FileNotFoundError as fnfe:
  print('Unable to open file: ' + str(fnfe))

finally:
  file.close()
```

Usually used to perform operations to release resources handling a file or a database connection.

File context managers

Allows for the previous slide to be made even easier by automatically setting up and tearing down resources. For this we use the `open()` operation:

```
try:
  with open(filepath, 'w') as items_file:
    for item in items:
      items_file.write(item + '\n')
except:
  print('Failed to open file')
```

Open

`open()` is a built-in function that allows us to open a file and read it's contents, returning us a file object.

`open()` takes two main arguments:

1. The name of the file
2. The 'file mode' which we will come on to

`as` takes whatever object is returned from the `open` function and aliases it to a temporary variable called `file`.

```
with open(filepath, 'w') as file:
    ...
```

Exercise

Distribute exercise file.

Learning Objectives Revisited

- Understand the importance of various file handling procedures
- Implement file handling using Python

Terms and Definitions Recap

Persistence: The characteristic of state of a system that outlives the process that created it.

Exception: Code that breaks the normal flow of execution and executes a pre-registered exception handler.

I / O: The communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system.

Further Reading

- Exceptions
- File Handling