LECTURE 13

FILES

STRING FORMATTING

MCS 260 Fall 2021

Emily Dumas

REMINDERS

- Project 2 description coming today
- Project 2 will be due Fri Oct 8
- Avoid the green "play" button in VS code
- Week 5 feedback survey open until 2pm Fri

FILES

A file is a named, ordered collection of data, usually in persistent storage (disks, flash, etc.).

Files are stored in a hierarchy of directories.

The OS provides functions that programs can use to access files, handling the lower-level details itself.

BASIC FILE OPERATIONS

- **open**: Request access to a file (by name)
- **read**: Get data from an open file
- **write**: Add or change data in an open file
- **close**: Relinquish access to an open file

The OS keeps track of a *file offset*, the place in the file where the next read or write operation will happen. This can also be moved with an operation called *seek*.

BYTES OR STRINGS?

Two ways to access files:

- **binary format**: Read and write bytes (as the OS itself does)
- **text format**: Read and write strings. Python translates to and from bytes using an encoding.

We'll only cover text format file operations for now.

ENCODINGS

An encoding is a way of turning a sequence of unicode characters into a sequence of bytes.

UTF-8, ISO-8859-1, and CP-1252 are examples of encodings.

E.g. translating `"Adiós"` into bytes:

- `0x41 0x64 0x69 0xc3 0xb3 0x73` in UTF-8
- `0x41 0x64 0x69 0xf3 0x73` in ISO-8859-1

We will use UTF-8 exclusively.

FILES IN PYTHON

`open(filename,[mode],[encoding=...])`

opens a file and returns an object representing it.

Methods of the file object allow you to read or write.

```
"""Write a string to a file"""
fout = open("out.txt","w",encoding="UTF-8")   # w means write
fout.write("Hello world")
fout.close() # Done with this file (OS does cleanup)

fin = open("out.txt","r",encoding="UTF-8")    # r means read only
s = fin.read() # Get entire file contents
fin.close()
print("Contents of file:",s)
```

The files you read/write this way can have any name you like; they don't need to end in ".txt".

MODES

- "r" - The default. Allows reading. File must exist.
- "w" - **Deletes the file if it exists**, creates it if not. Allows writing.
- "a" - Place offset at the end of the file if it exists. Allows writing (i.e. "appending").
- "r+" - Offset at beginning if file exists. Allows reading and writing.
- "w+" - **Deletes the file if it exists**, creates it if not. Allows reading and writing.

READING LINES

Often you want to process one line at a time. File objects are *iterable*, giving the lines. E.g. nl.py

```
"""Number the lines of a file specified on command line"""
import sys
fin = open(sys.argv[1],"r",encoding="UTF-8")
n = 0
for line in fin:
    n = n+1
    print(n,line,end="") # line usually has \n at the end
fin.close()
```

Sample output:

```
$ python nl.py nl.py
1 """Number the lines of a file specified on command line"""
2 import sys
...
```

Important: `file.write()` is not like `print()`. It doesn't add a newline, and it doesn't accept multiple arguments to print.

```
pet_type = "ducks"
print("I have",21,pet_type) # OK
fout.write("I have",21,pet_type) # FAILS
```

Must prepare a single string to write. The usual way is to use `str.format()`:

```
pet_type = "ducks"
fout.write("I have {} {}\n".format(21,pet_type)) # ok
```

STRING FORMATTING

`str.format()` has many features to create a string based on a **template** and some values. In the string, placeholders (`{}` or `{...}`) are replaced by arguments of `str.format()`.

```
>>> "{1} taught {0}".format("MCS 260","Dumas")  # give indices
'Dumas taught MCS 260'
>>> for x in range(98,101):
...     print("{:4}".format(x))  # specify width
...
  98
  99
 100
>>> "{:04}".format(42)  # pad to width with zeros
'0042'
```

```
>>> "{:8.2f}".format(42)  # f = float, width 8, 2 digits after .
'   42.00'
>>> "{:8x}".format(42)    # x = hex int, width 8
'      2a'
>>> "{:8d}".format(42)    # d = decimal int, width 8
'      42'
>>> "{:.2f}".format(13+2j) # f allows complex; no total width
'13.00+2.00j'
```

The general placeholder syntax is `{w:ot}` where w specifies **which** argument, o is a set of **options**, and t is the **type**.

`str.format()` has a lot of features we didn't discuss today.

REFERENCES

- In *Downey*:
  - Chapter 14 discusses files, especially Sections 14.1, 14.2, and 14.4.
  - Section 14.3 discusses a different, older way of formatting strings.
- This Introduction to String Formatters in Python 3 by Lisa Tagliaferri at DigitalOcean is a good reference for the topics in string formatting we covered today.

REVISION HISTORY

- 2021-09-22 Initial publication