Think Python 2e, Chapter 14 Notes

Files

October 20, 2022

Persistence

- Most data in a computer program is **transient**.
- When the program ends, the data is lost.
- **Text files** are one way to make **persistent** data.
- Another way is a **database**.

Writing to files

```
>>> fout = open('output.txt', 'w')
```

If the file already exists, opening it in write mode deletes the old data, so be careful! If the file doesn't exist, a new one is created.

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
24
```

The return value is the number of characters that were written. The file object keeps track of where it is, so if you call write again, it adds the new data to the end of the file.

```
>>> line2 = "the emblem of our land.\n"
>>> fout.write(line2)
24
```

Closing a file

When you are done writing, you should close the file.

```
>>> fout.close()
```

If you don't close the file, it is closed when the program ends.

Writing with format strings

```
>>> camels = 42
>>> ’I have spotted %d camels.’ % camels
’I have spotted 42 camels.’
>>> ’In %d years I saw %g %s.’ % (3, 0.1, ’camels’)
’In 3 years I saw 0.1 camels.’
```

The os module

- os stands for "operating system".

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/home/dinsdale'
```

- Files are organized into **directories** also called **folders**
- cwd stands for "current working directory"
- A string like '/home/dinsdale' that identifies a file or directory is called a **path**.

Relative and absolute paths

- A simple filename, like `memo.txt` is also considered a path, but it is a **relative path** because it relates to the current directory.
- If the current directory is `/home/dinsdale`, the filename `memo.txt` would refer to `/home/dinsdale/memo.txt`.
- A path that begins with `/` does not depend on the current directory.
- It is called an **absolute path**.
- To find the absolute path to a file, you can use:

```
>>> os.path.abspath('memo.txt')
'/home/dinsdale/memo.txt'
```

os.path

```
>>> os.path.exists(’memo.txt’)
True
>>> os.path.isdir(’memo.txt’)
False
>>> os.path.isdir(’/home/dinsdale’)
True
>>> os.path.isfile(’memo.txt’)
True
>>> os.listdir(cwd)
[’music’, ’photos’, ’memo.txt’]
```

A lot of things can go wrong with files

```
>>> fin = open('bad_file')
FileNotFoundError: [Errno 2] No such file or directory: 'bad_file'
>>> fout = open('/etc/passwd', 'w')
PermissionError: [Errno 13] Permission denied: '/etc/passwd'
>>> fin = open('/home')
IsADirectoryError: [Errno 21] Is a directory: '/home'
```

These are called **exceptions**.

Catching exceptions

```
try:
    fin = open(’bad_file’)
except:
    print(’Something went wrong.’)
```

Databases

- A **database** is a file that is organized for storing data.
- Many databases are organized as key/value pairs, like dictionaries.
- Databases are on disk, so they are persistent.
- The module `dbm` provides a simple database.

dbm

```
>>> import dbm
>>> db = dbm.open('captions', 'c')
```

The option `'c'` means create if it does not exist.
It can now be used like a dictionary.

```
>>> db['cleese.png'] = 'Photo of John Cleese.'
>>> db['cleese.png']
b'Photo of John Cleese.'
```

Note the `b` in front of the string.
This is a **bytes object**. For now, it behaves just like a string.
`dbm` keys and values must be strings or bytes.

db.keys and db.values are iterators

```
for key in db.keys():
    print(key, db[key])
```

What could you do if you wanted a list of keys?

Don't forget to close!

```
>>> db.close()
```

Pickling

Because `dbm` only handles strings or bytes, Python has a module to turn anything into a byte.

```
>>> import pickle
>>> t = [1, 2, 3]
>>> pickle.dumps(t)
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

You can get the object back:

```
>>> t1 = [1, 2, 3]
>>> s = pickle.dumps(t1)
>>> t2 = pickle.loads(s)
>>> t2
[1, 2, 3]
```

Use of `dbm` and `pickle` together is very common.

Pipes

Read: `http://www.ee.surrey.ac.uk/Teaching/Unix/`

```
>>> cmd = 'ls -l'
>>> fp = os.popen(cmd)
```

The `fp` object behaves like a file pointer.

You can use `fp.read()` or `fp.readline()`

Close a pipe like a file:

```
>>> stat = fp.close()
>>> print(stat)
None
```

If `None` is returned, that means it closed with no errors.

popen is deprecated

- `popen` is deprecated now.
- Which means we are supposed to stop using it and start using the `subprocess` module.
- For simple cases, `subprocess` is more complicated than necessary.
- `popen` may be taken away someday, but for now feel free to use it.

Writing modules

Suppose we write a module called `wc.py`, with the following contents.

```
def linecount(filename):
    count = 0
    for line in open(filename):
        count += 1
    return count

print(linecount(’wc.py’))
```

If we run this program, it reads itself and prints 7, the number of lines in the file.

Writing modules

You can also import this file

```
>>> import wc
7
>>> wc.linecount(’wc.py’)
7
>>> wc.linecount(’../chap13/greeneggsandham.txt’)
137
```

The problem is every time we import this file, it runs the test code at the bottom. Programs imported as modules use the following idiom to incorporate test code:

```
if __name__ == ’__main__’:
    print(linecount(’wc.py’))
```

`__name__` is `’__main__’` only when **not** imported.

Writing modules: `import` only works once

- Warning: If you import a module that has already been imported, Python does nothing. It does not re-read the file, even if it has changed.
- If you want to reload a module, you can use the built-in function reload, but it can be tricky, so the safest thing to do is restart the interpreter and then import the module again.

Debugging

When you are reading and writing files, you might run into problems with whitespace. These errors can be hard to debug because spaces, tabs and newlines are normally invisible:

```
>>> s = '1 2\t 3\n 4'
>>> print(s)
1 2  3
 4
```

`repr` takes any object as an argument and returns a string representation of the object. For strings, it represents whitespace characters with backslash sequences:

```
>>> print(repr(s))
'1 2\t 3\n 4'
```

This can be helpful for debugging.

Vocabulary

persistent: Pertaining to a program that runs indefinitely and keeps at least some of its data in permanent storage.

format operator: An operator, %, that takes a format string and a tuple and generates a string that includes the elements of the tuple formatted as specified by the format string.

format string: A string, used with the format operator, that contains format sequences.

format sequence: A sequence of characters in a format string, like %d, that specifies how a value should be formatted.

Vocabulary

text file: A sequence of characters stored in permanent storage like a hard drive.

directory: A named collection of files, also called a folder.

path: A string that identifies a file.

relative path: A path that starts from the current directory.

absolute path: A path that starts from the topmost directory in the file system.

Vocabulary

catch: To prevent an exception from terminating a program using the try and except statements.

database: A file whose contents are organized like a dictionary with keys that correspond to values.

bytes object: An object similar to a string.

shell: A program that allows users to type commands and then executes them by starting other programs.

pipe object: An object that represents a running program, allowing a Python program to run commands and read the results.