THE UNIVERSITY OF SYDNEY

INFO1110 / COMP9001 Week 6 Tutorial

File I/O

Reading a text file

Going back to Week 3, you will remember using the with keyword on a file. Now we are going to open files without using the with keyword and expand on the File API provided by python.

As part of python you already have explicit access to most file I/O functions that you will use.

Opening a file:

```
# We can use the open function and provide the
# filename of the file we want to use or create
# We can supply a string as a second argument to
# specify what mode we are going to open it in
# r = Read Only,
# w = Write,
# rb = Read Only Binary,
# wb = Write Binary,
# a = Append,
# ab = Append Mode Binary

file = open('filename', 'r')
```

Once a file has been opened we can use the function attached to the object such as `read` and `write` Writing to a file:

```
file = open('hello_file.txt', 'w')
file.write('Hello File!')
```

Reading from a file:

```
file = open('hello_file.txt', 'r')

#gets 50 characters or you can use read() to read in the whole thing
s = file.read(50)
# If there is < 50 characters it may encounter an end of file
print(s)
```

Question 1: Rewrite Less Command

We will be rewriting the less command in python. To use less, you can open a file with the following:

```
less <filename>
```

This will open the file for viewing.

Your program will need to navigate through the file line by line when the user presses enter. (You do not need to handle the case of going in reverse).

When the user presses `q`, your program should exit.

Example (using the file lorem.txt):

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit,<enter>
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.<enter>
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris<enter>
nisi ut aliquip ex ea commodo consequat.<enter>
Duis aute irure dolor in reprehenderit in voluptate velit<enter>
esse cillum dolore eu fugiat nulla pariatur.<enter>
Excepteur sint occaecat cupidatat non proident,<enter>
sunt in culpa qui officia deserunt mollit anim id est laborum.<enter>
```

Question 2: Subtitles File

You have been given a file called subtitles which includes 2 different sets of subtitles. You are tasked with extract each set into their own file.

The file you have been provided is called all.subtitle and you will need to split it into 3 different .subtitle files called

english.subtitle deutsch.subtitle nederlands.subtitle

This is a purely text based. Subtitle content follows it after its tags have been encountered. Once [<Language>] has been found, every line following it will correspond to it.

Format:

```
[<Language 1>]
<contents for language 1>
[<Language 2>]
<contents for language 2>
```

Example:

```
[English]

1
00:00:05,250 --> 00:00:50,500
Hello

[Deutsch]

1
00:00:05,250 --> 00:00:50,500
Hallo
```

Reading a binary File

Files can be stored in a way that is not human readable. We have source code which adheres to a human readable text encoding such as ASCII or UTF. We will need to be able to deal with binary files or non-human readable files. We can encode data from our programs into binary files that allow them to be easily loaded by our own programs but also by other programs.

Marshal

Marshalling is a way of serializing internal python structures, independent from the machine architecture (endianess being an obvious problem). However, marshalling cannot be used when it comes to user defined classes and instances.

Usage for `dump`:

```
import marshal
data = [1, 2, 3, 4, 5]
f = open('test.obj', 'wb')
marshal.dump(data, f)
```

Usage for `load`:

```
import marshal
f = open('test.obj', 'rb')
data = marshal.load(f)
print(data)
```

Pickling

Pickling and unpickling is used to serialize user defined structures and their instances. The pickling module is designed for general purpose where marshalling is strictly used for internal python object serialisation. Pickling can be used with user defined types.

Firstly we need to import the pickle module

```
import pickle
```

This will give us access to the `load` and `dump` functions from the module. (for more info `help(pickle)`) When using either `load` or `dump` function with a file, ensure that the file has the binary flag `'b'` set on it.

Usage for `dump`

```
import pickle

some_data = [1, 2, 3, 4, 5]
file = open('my_numbers.obj', 'wb')
pickle.dump(some_data, file)
```

Usage for `load`

```
import pickle

file = open('my_numbers.obj', 'rb')
some_data = pickle.load(file)
```

Question 3: Extract binary pokemon data

You will have been given a file that stores pokemon data called `poke.dex`. You will need to read this file and extract the stats of the pokemon and print them out in this format:

The poke.dex will store the data in a list where each element of the list is a tuple.

Pokemon Tuple Format:

```
kanto_no, integer (starts a 0)
level, integer
type_1, integer (index to type list)
type_2, integer
hp, integer,
max_hp, integer
name, string
```

Type list:

```
  "N/A", (reserve for when a pokemon does not have a second type)
  "Grass",
  "Poison",
  "Fire",
  "Flying",
  "Water",
  "Steel",
  "Ground",
  "Bug",
  "Normal",
  "Electric",
  "Fighting",
  "Psychic",
  "Ice",
  "Ghost",
  "Rock",
  "Dragon",
  "Dark",
```

Print out the pokemon from the .dex files that you have located. You are to write a program that will output this data.

```
#4
Charmander
Fire
N/A
Level: 3
HP: 40
MAX_HP: 50

#25
Pikachu
Eletric
N/A
Level: 32
HP: 450
MAX_HP: 700
```

JSON and other modules

It is to also note that other file formats such as JSON and CSV have their own module that operate in a very similar manner to pickling.

```
import json

data = { "name" : "Jim", "age" : 20 }
json.dump(data, open('person.json', "w"))
```

However, there is a difference, json and csv are text based formats and therefore, when opening a file it should not contain `b` unlike pickle and marshal.

Question 4: Getting data from an API Endpoint (JSON)

Very common with today's web apps, you will need to be able to retrieve data from a web api endpoint. One that is JSON based and extract the data and output as well as return the modified JSON data to the server with it's updates.

To run the server,

```
python vector_app.py
```

Using curl to check you can access the endpoint

```
curl -X POST -d '{"v":[1, 2, 3],"u":[1, 2, 3]}'
http://localhost:9001/crossproduct
```

This exposes a very common format called JSON and where it is commonly used. It allows you to easily probe end points used on websites such as Facebook and Ed and understand how data is being transferred between your client and the browser.

Output from: `curl -i -X POST -d '{"v":[1, 2, 3],"u":[3,2, 3]}' -H "Content-Type: application/json" http://localhost:9001/crossproduct`

```
HTTP/1.0 200 OK
Date: Tue, 17 Apr 2018 13:26:21 GMT
Server: WSGIServer/0.2 CPython/3.6.3
127.0.0.1 - - [17/Apr/2018 23:26:21] "POST /crossproduct HTTP/1.1" 200 10
Content-Length: 10
Content-Type: text/html; charset=UTF-8

[0, -6, 4]
```

Question 5: JSON Data

Use `urllib` module to retrieve the data from the server from a python program. This acts as a simple introduction into writing a client for JSON REST API.

As a hint, please refer to the documentation for urllib urllib documentation

You will need to use the request function to communicate to the end point specified.

Question 6: Moving through a tape

Given a file called `data.tape` you will need to only move through as much as the data as specified by the user. If the user wants to read through 1MB of the tape, your program will only need to read 1MB of the tape as well as jump segments of the file using the `seek()` function.

The file is a ASCII text file Example Text File:

```
abcdefghijabcdefghijabcdefghijabcdefghijabcdefghijabcdefghijabcdefghij
```

```
python tape.py file.txt
```

Usage:

```
MOVE 50
READ 10
abcdefghij
MOVE 5
READ 5
fghij
QUIT
<program ends>
```

Discussion

- When we use close() what occurs? If we wanted to immediately write to a file, what method would we use?
- If the class we are unpickling is incorrect, what behaviour would you observe? (Provide example)
- What are the advantages to a file buffer over unbuffered?
- What is line buffering and when have you observed it used?
- How are files laid out on the filesystem?
- Does the operating system or kernel know about what files are open?
- When you call the write function, does it immediately write to a file?