Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Processing Data Files

Name: Phua Yan Han

IC Number: 050824070059

Date : 24/6/23

Introduction : Learning different type of data file

Conclusion : Learned what are the ways you can read write and append data files

Module P05: Processing Data Files

Processing data is an essential starting point towards any data science / machine learning work. Without data, nothing can be done. This module looks into how files can be read and written by a Python program. We start with some generic reading and writing operations, and then move to JSON and CSV files, two popular data file types.

Reading and Writing to Files

A portion of this content uses official python tutorial (https://docs.python.org/3.6/tutorial/inputoutput.html).

`open()` returns a file object, and is most commonly used with two arguments: `open(filename, mode)` .

```
f = open('workfile', 'w')
```

The first argument is a string containing the filename.

The second argument is another string containing a few characters describing the way in which the file will be used.

mode can be:

- `'r'` read (by default)
- `'w'` write
- `'a'` append

Normally, files are opened in **text mode**, that means, you read and write **strings** from and to the file, which are encoded in a specific encoding. If encoding is not specified, the default depends on your platform.

`'b'` appended to the mode opens the file in **binary mode**: now the data is read and written in the form of **bytes** objects.

Example:

```
f = open('workfile', 'wb')
```

It is good practice to use the **`with`** keyword when dealing with file objects. The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point. Using `with` is also much shorter than writing equivalent `try-finally` blocks:

```
with open('workfile') as f:
    read_data = f.read()
```

Methods of file objects

Files can be read as a file object **`f`**:

- **`f.write(s)`** write **`s`** in **`f`**
- **`f.read()`** reads the whole file
- **`f.readline()`** reads one line
- to read a file line by line, you can loop over the file object itself:

```
with open('workfile', 'r') as f:
    for line in f:
        print(line)
```

Writing files

Let's create a file example_file.txt

In [4]:
```
filename = 'example_file.txt'
```

In [5]:
```
# We can pass a string directly
with open(filename, 'w') as my_file:
    my_file.write("Here is some example text\n")

    # More often, we want to write data from our Python program to file.
    n = 100
    example_string = "The value of n is %d.\n"%n
    my_file.write(example_string)

    # Here, we write to a file from within a loop
    my_file.write("\nAnd now, here's a song about 99 ipsum lorem lines:\n")
    for nlines in range(1, n, 1):
        line = "%d ipsum lorem lines printed here. %d ipsum lorem lines. One is
        my_file.write(line)
```

Using the Jupyter notebook directory page on your web browser, locate the file and open it. You can also open it using your OS's file manager. Verify that the contents make sense by comparing them to the code above.

Reading files

Let's read the file in Python. Again, we will use the `open()` function, this time with the `'r'` parameter to indicate that we are reading.

In [ ]:
```
# Again the open() function gives us a variable of type 'file'
with open(filename, 'r') as input_file:

    # The read() function puts the contents of the file into a string object
    file_contents = input_file.read()

    # Let's see what we got
print(file_contents)
```

Parsing

It will often be necessary to "parse" the contents of a file. This means we will extract information from the string into Python variables that we can operate on. When the file was not designed to be parsed conveniently, this can get messy!

As an example, let's locate all words that appear in 25 of Pythagoras' famous sayings. Instead of reading the entire file all-at-once, in this example we'll choose to read it line-by-line so that we can perform additional processing to each line, i.e. splitting and collecting them into another list. This is done by iterating over the file object.

In [2]:
```
with open('..\\Data files\\pythagoras.txt','r') as input_file:

    # Throw away the first 3 lines. They do not contain the quotes.
    # Notice that we are reading the first 3 lines but we are not doing anythir
    # Using the _ name is a Python convention for a variable name that we are r
    for j in range(3):
        _ = input_file.readline()

    words = []
    i = 0
    # Iterate over remaining lines
    for line in input_file:

        # Action: think about this line. What does it do?
        words.extend(line.split())

        # Action: what about this line. what does it do?
        words = [w.lower() for w in words]

        i = i + 1

    print((words))
    print(len(words))
    print(i)   # sanity check to ensure it actually went thru 25 lines
```

```
['as', 'long', 'as', 'man', 'continues', 'to', 'be', 'the', 'ruthless', 'dest
royer', 'of', 'lower', 'living', 'beings,', 'he', 'will', 'never', 'know', 'h
ealth', 'or', 'peace.', 'for', 'as', 'long', 'as', 'men', 'massacre', 'animal
s,', 'they', 'will', 'kill', 'each', 'other.', 'indeed,', 'he', 'who', 'sow
s', 'the', 'seed', 'of', 'murder', 'and', 'pain', 'cannot', 'reap', 'joy', 'a
nd', 'love.', 'be', 'silent', 'or', 'let', 'thy', 'words', 'be', 'worth', 'mo
re', 'than', 'silence.', 'if', 'there', 'be', 'light,', 'then', 'there', 'i
s', 'darkness;', 'if', 'cold,', 'heat;', 'if', 'height,', 'depth;', 'if', 'so
lid,', 'fluid;', 'if', 'hard,', 'soft;', 'if', 'rough,', 'smooth;', 'if', 'ca
lm,', 'tempest;', 'if', 'prosperity,', 'adversity;', 'if', 'life,', 'death.',
'no', 'one', 'is', 'free', 'who', 'has', 'not', 'obtained', 'the', 'empire',
'of', 'himself.', 'no', 'man', 'is', 'free', 'who', 'cannot', 'command', 'him
self.', 'there', 'is', 'geometry', 'in', 'the', 'humming', 'of', 'the', 'stri
ngs.', 'there', 'is', 'music', 'in', 'the', 'spacing', 'of', 'the', 'sphere
s.', 'do', 'not', 'say', 'a', 'little', 'in', 'many', 'words,', 'but', 'a',
'great', 'deal', 'in', 'few!', 'educate', 'the', 'children', 'and', 'it', "wo
n't", 'be', 'necessary', 'to', 'punish', 'the', 'men.', 'rest', 'satisfied',
'with', 'doing', 'well,', 'and', 'leave', 'others', 'to', 'talk', 'of', 'yo
u', 'as', 'they', 'please.', 'in', 'anger', 'we', 'should', 'refrain', 'bot
h', 'from', 'speech', 'and', 'action.', 'the', 'oldest,', 'shortest', 'word
s', 'â€”', 'yes', 'and', 'no', 'â€”', 'are', 'those', 'which', 'require', 'th
e', 'most', 'thought.', 'a', 'man', 'is', 'never', 'as', 'big', 'as', 'when',
'he', 'is', 'on', 'his', 'knees', 'to', 'help', 'a', 'child.', 'as', 'long',
'as', 'men', 'massacre', 'animals,', 'they', 'will', 'kill', 'each', 'othe
r.', 'indeed,', 'he', 'who', 'sows', 'the', 'seeds', 'of', 'murder', 'and',
'pain', 'cannot', 'reap', 'the', 'joy', 'of', 'love.', 'silence', 'is', 'bett
er', 'than', 'unmeaning', 'words.', 'above', 'all', 'things,', 'respect', 'yo
urself.', 'let', 'no', 'one', 'persuade', 'you', 'by', 'word', 'or', 'deed',
'to', 'do', 'or', 'say', 'whatever', 'is', 'not', 'best', 'for', 'you.', 'n
o', 'man', 'is', 'free', 'who', 'cannot', 'control', 'himself.', 'it', 'is',
'only', 'necessary', 'to', 'make', 'war', 'with', 'five', 'things;', 'with',
'the', 'maladies', 'of', 'the', 'body,', 'the', 'ignorances', 'of', 'the', 'm
ind,', 'with', 'the', 'passions', 'of', 'the', 'body,', 'with', 'the', 'sedit
ions', 'of', 'the', 'city', 'and', 'the', 'discords', 'of', 'families.', 'cho
ose', 'rather', 'to', 'be', 'strong', 'of', 'soul', 'than', 'strong', 'of',
'body.', 'reason', 'is', 'immortal,', 'all', 'else', 'mortal.', 'salt', 'is',
'born', 'of', 'the', 'purest', 'parents:', 'the', 'sun', 'and', 'the', 'se
a.', 'number', 'rules', 'the', 'universe.', 'we', 'ought', 'so', 'to', 'behav
e', 'to', 'one', 'another', 'as', 'to', 'avoid', 'making', 'enemies', 'of',
'our', 'friends,', 'and', 'at', 'the', 'same', 'time', 'to', 'make', 'friend
s', 'of', 'our', 'enemies.', 'anger', 'begins', 'in', 'folly,', 'and', 'end
s', 'in', 'repentance.', 'choose', 'always', 'the', 'way', 'that', 'seems',
'the', 'best,', 'however', 'rough', 'it', 'may', 'be;', 'custom', 'will', 'so
on', 'render', 'it', 'easy', 'and', 'agreeable.', 'as', 'soon', 'as', 'laws',
'are', 'necessary', 'for', 'men,', 'they', 'are', 'no', 'longer', 'fit', 'fo
r', 'freedom.']
420
25
```

Side topic: Bytes and strings

When you talk about “text” you’re probably thinking of “characters and symbols on my computer screen.”

But computers don’t deal in characters and symbols; they deal in **bits and bytes**.

Every piece of text you've ever seen on a computer screen is actually stored in a particular **character encoding**.

Very roughly speaking, the character encoding provides a mapping between the stuff you see on your screen and the stuff your computer actually stores in memory and on disk.

There are many different character encodings, some optimized for particular languages like Russian, Hindi or Chinese or English, and others that can be used for multiple languages.

A very common **encoding** is **UTF-8**.

Python has two different classes: **`bytes`** and **`string`** .

To transform bytes in string, you need to **decode** it, specifying the **encoding**.

To transform string in bytes, you need to **encode** it, specifying the **encoding**.

Examples

In [9]:
```
s1 = '义勇军进行曲'
print(s1)
s2 = 'जन गण मन'
print(s2)
```

```
义勇军进行曲
जन गण मन
```

In [10]:
```
b1 = s1.encode()
print(b1)
b2 = s2.encode()
print(b2)
```

```
b'\xe4\xb9\x89\xe5\x8b\x87\xe5\x86\x9b\xe8\xbf\x9b\xe8\xa1\x8c\xe6\x9b\xb2'
b'\xe0\xa4\x9c\xe0\xa4\xa8 \xe0\xa4\x97\xe0\xa4\xa3 \xe0\xa4\xae\xe0\xa4\xa8'
```

The function `encode()` encodes the characters into byte form, as you may have noticed a `b'` in front of the code shown. By default, it uses 'utf-8' or 8-bit Unicode encoding.

In [18]:
```
type(b1), type(b2)
```

Out[18]:
```
(bytes, bytes)
```

In [19]:
```
b1.decode(), b2.decode()
```

Out[19]:
```
('义勇军进行曲', 'जन गण मन')
```

In [20]:
```
help(b1.decode)
```

```
Help on built-in function decode:

decode(encoding='utf-8', errors='strict') method of builtins.bytes instance
    Decode the bytes using the codec registered for encoding.

    encoding
      The encoding with which to decode the bytes.
    errors
      The error handling scheme to use for the handling of decoding errors.
      The default is 'strict' meaning that decoding errors raise a
      UnicodeDecodeError. Other possible values are 'ignore' and 'replace'
      as well as any other name registered with codecs.register_error that
      can handle UnicodeDecodeErrors.
```

The text in English that we are familiar with, can be encoded with ASCII as well. In this case, the a

In [21]:
```
s3 = 'china'
s3.encode('ASCII')
```

Out[21]:
```
b'china'
```

Some encodings do not work for certain types of characters!

In [11]:
```
'义勇军进行曲'.encode('ASCII')
```

```
---------------------------------------------------------------------------
UnicodeEncodeError                        Traceback (most recent call last)
Cell In[11], line 1
----> 1 '义勇军进行曲'.encode('ASCII')

UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-5: or
dinal not in range(128)
```

In [23]:
```
'abc'.encode().find(b'c')
```

Out[23]:
```
2
```

In [34]:
```
with open("file_china_chars", 'bw') as f:
    f.write(s1.encode())
```

In [35]:
```
!type file_china_chars
!del file_china_chars
```

```
ä¹‰å‹‡å†›è¿›è¡Œæ›²
```

JSON Files

JSON (JavaScript Object Notation) is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.

Many APIs work with JSON format to provide and receive data. Popular alternatives to JSON are YAML and XML. JSON is often compared to XML because its usage is similar. JSON has now become an alternative to XML, which is older.

Normalised in RCF7159 (https://tools.ietf.org/html/rfc7159) and ECMA404 (http://www.ecma-international.org/publications/standards/Ecma-404.htm).

Description

JSON's data types are:

- **Number**
- **String**
- **Boolean**: **true** / **false**
- **Array**: ordered list of values. Each of which may be of any type. Arrays use square bracket **[ ]** notation with elements being **comma-separated ,** .
- **Object**: an unordered collection of name/value pairs where the names (also called keys) are strings. Objects are delimited with **curly brackets {}** and use **commas to separate each pair**, while within each pair the **colon ':' character separates the key or name from its value**.
- **null**: An empty value, using the word **null**.

Python	JSON	Example in JSON
int or float	Number	34
string	String	"foo bar"
Bool	Boolean	true
list	Array	[1, 2, 3]
dict	Object	{"type": "home", "number": "212 555-1234"}

Example

Here's an example from wikipedia (https://en.wikipedia.org/wiki/JSON#Data_types.2C_syntax_and_example)

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
```

In [3]:
```
import json
```

Note! Indentation doesn't matter during creation of data, as long as the specifications are correct.

In [4]:
```
json_data = '''{
            "type": "home",
        "number": "212 555-1234",
    "active": true,
            "status": [1, 0, 4],
    "user": {
        "ID": "benley83",
                "passcode": "432Kd3D"
    }
}'''

data = json.loads(json_data)  # Loads the JSON data
print(data)
print(type(data))
```

```
{'type': 'home', 'number': '212 555-1234', 'active': True, 'status': [1, 0,
4], 'user': {'ID': 'benley83', 'passcode': '432Kd3D'}}
<class 'dict'>
```

In [5]:
```
with open("example.json", 'w') as f:
    json.dump(data, f)
```

In [4]:
```
!type example.json
# 'cat' instead of 'type' for Linux
```

```
{"type": "home", "number": "212 555-1234", "active": true, "status": [1, 0,
4], "user": {"ID": "benley83", "passcode": "432Kd3D"}}
```

In [5]:
```
with open("example.json", 'r') as f:
    data2 = json.load(f)

print(data2)
```

```
{'type': 'home', 'number': '212 555-1234', 'active': True, 'status': [1, 0,
4], 'user': {'ID': 'benley83', 'passcode': '432Kd3D'}}
```

This library `json.tool` is used with the `python` command line to indent a json file

In [6]:
```
!python -m json.tool example.json
```

```
{
    "type": "home",
    "number": "212 555-1234",
    "active": true,
    "status": [
        1,
        0,
        4
    ],
    "user": {
        "ID": "benley83",
        "passcode": "432Kd3D"
    }
}
```

This does not make any changes to the original JSON file formatting (check the file), but only prints it pretty for your checking purposes. Normally, we want the JSON file to be as compact and lightweight as possible, so having no whitespaces is actually good.

We now load some data containing the top rated movies in Sweden (`top-rated-movies-01.json'). First, try to understand how the data is structure inside this JSON file.

In [6]:
```
with open('../Data files/top-rated-movies-01.json') as json_data:
    d = json.load(json_data)
    print(d)
```


```
[{'title': 'Nyckeln till frihet', 'year': '1994', 'genres': ['Crime', 'Dram
a'], 'ratings': [8, 8, 6, 10, 2, 3, 4, 5, 4, 9, 3, 9, 6, 10, 4, 8, 10, 1,
2, 8, 1, 9, 5, 4, 4, 2, 4, 6, 9, 10], 'poster': 'MV5BODU4MjU4NjIwNl5BMl5Ban
BnXkFtZTgwMDU2MjEyMDE@._V1_SY500_CR0,0,336,500_AL_.jpg', 'contentRating':
'15', 'duration': 'PT142M', 'releaseDate': '1995-03-03', 'averageRating':
0, 'originalTitle': 'The Shawshank Redemption', 'storyline': "Chronicles th
e experiences of a formerly successful banker as a prisoner in the gloomy j
ailhouse of Shawshank after being found guilty of a crime he did not commi
t. The film portrays the man's unique way of dealing with his new, torturou
s life; along the way he befriends a number of fellow prisoners, most notab
ly a wise long-term inmate named Red.                Written by\nJ-S-Golde
n", 'actors': ['Tim Robbins', 'Morgan Freeman', 'Bob Gunton'], 'imdbRatin
g': 9.3, 'posterurl': 'https://images-na.ssl-images-amazon.com/images/M/MV5
BODU4MjU4NjIwNl5BMl5BanBnXkFtZTgwMDU2MjEyMDE@._V1_SY500_CR0,0,336,500_AL_.j
pg'}, {'title': 'Gudfadern', 'year': '1972', 'genres': ['Crime', 'Drama'],
'ratings': [6, 10, 4, 10, 1, 3, 7, 2, 3, 3, 3, 4, 2, 5, 6, 9, 10, 8, 7, 4,
8, 9, 9, 10, 4, 6, 2, 9, 7, 7], 'poster': 'MV5BZTRmNjQ1ZDYtNDgzMy00OGE0LWE4
N2YtNTkzNWQ5ZDhlNGJmL2ltYWdlL2ltYWdlXkEyXkFqcGdeQXVyNjU0OTQ0OTY@._V1_SY500_
CR0,0,352,500_AL_.jpg', 'contentRating': '15', 'duration': 'PT175M', 'relea
```


You probably want to take a peek at the first entry's keys. Too many entries and values printed just makes it hard to read!

In [7]:
```
with open('../Data Files/top-rated-movies-01.json') as json_data:
    d = json.load(json_data)
    print(d[0].keys())
```

```
dict_keys(['title', 'year', 'genres', 'ratings', 'poster', 'contentRating',
'duration', 'releaseDate', 'averageRating', 'originalTitle', 'storyline', 'ac
tors', 'imdbRating', 'posterurl'])
```

Quick Exercise 1 Print the the number of movies, followed by the full list of movies with their respective IMDB ratings.

In [8]:
```
with open('../Data Files/top-rated-movies-01.json') as json_data:
    d = json.load(json_data)
    for i,x in enumerate(d):
        arr = x.get('ratings')
        average =sum (arr) / len(arr)
        print(f"{i+1} Title: {x.get('title')}  Rating {x.get('imdbRating')}")

    # write your code after this line
```

```
1 Title: Nyckeln till frihet  Rating 9.3
2 Title: Gudfadern  Rating 9.2
3 Title: Gudfadern del II  Rating 9.0
4 Title: The Dark Knight  Rating 9.0
5 Title: 12 edsvurna män  Rating 8.9
6 Title: Schindler's List  Rating 8.9
7 Title: Pulp Fiction  Rating 8.9
8 Title: Sagan om konungens återkomst  Rating 8.9
9 Title: Den gode, den onde, den fule  Rating 8.9
10 Title: Fight Club  Rating 8.8
11 Title: Sagan om ringen: Härskarringen  Rating 8.8
12 Title: Rymdimperiet slår tillbaka  Rating 8.8
13 Title: Forrest Gump  Rating 8.8
14 Title: Inception  Rating 8.8
15 Title: Sagan om de två tornen  Rating 8.7
16 Title: Gökboet  Rating 8.7
17 Title: Maffiabröder  Rating 8.7
18 Title: Matrix  Rating 8.7
19 Title: De sju samurajerna  Rating 8.7
```

CSV Files

A Comma-Separated Values (CSV) file stores tabular data (numbers and text) in plain text. Most often than not, the CSV file can be comfortably parsed and opened in spreadsheet programs such as MS Excel, OpenOffice Calc and etc.

Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

Basic Rules

- CSV is a delimited data format that has fields/columns separated by the comma character and records/rows terminated by newlines.
- A CSV file does not require a specific character encoding, byte order, or line terminator format (some software does not support all line-end variations).
- A record ends at a line terminator. However, line-terminators can be embedded as data within fields, so software must recognize quoted line-separators (see below) in order to correctly assemble an entire record from perhaps multiple lines.
- All records should have the same number of fields, in the same order.

- Adjacent fields must be separated by a single comma. However, "CSV" formats vary greatly in this choice of separator character. In particular, in locales where the comma is used as a decimal separator, semicolon, TAB, or other characters are used instead.

Example

In [11]:
```
import csv

with open('test.csv', 'w') as csvfile:
    writer = csv.writer(csvfile, lineterminator = '\n')
    writer.writerow(['Year', 'Make', 'Model'])
    writer.writerow(['2009', 'Proton', 'Exora'])
    writer.writerow(['2009', 'Perodua', 'Alza'])
```

In [12]:
```
!type test.csv
```

```
Year,Make,Model
2009,Proton,Exora
2009,Perodua,Alza
```

Read a file and store csv records row-by-row into a nested list (i.e. each inner list holds a row of record):

In [13]:
```
import csv

def load_csv(filename, delim=','):
    data = []
    with open(filename, 'r') as f:
        reader = csv.reader(f, delimiter=delim)
        for row in reader:
            data.append(row)
    return data

data = load_csv('test.csv')
print(data)
```

```
[['Year', 'Make', 'Model'], ['2009', 'Proton', 'Exora'], ['2009', 'Perodua',
'Alza']]
```

TASK We have obtained information from Malaysia's Open Data Portal on the number of vehicles in Malaysia over the period of 2008-2015. Examine the contents of the CSV file carefully.

1. Find the total number of vehicles in Malaysia per year.
2. What is generally the ratio of active to inactive vehicles in Malaysia (across all states)?

In [14]:
```
vehicle = load_csv('../Data Files/kenderaan.csv', delim=';')

print(vehicle[0])   # header stuff
print(vehicle[1])

# need to remove header before processing further
vehicle = vehicle[1:]
```

```
['STATE', 'STATUS', 'YEAR', 'NO']
['PERLIS', 'ACTIVE', '2008', '56,557']
```

Tip: Noticed how the number of vehicles have a comma separator. You have to remove this comma before the number can be further processed arithmetically.

In [15]:
```
total = 0
active = 0
inactive = 0
currentYear = "2008"

print("2008")
for i, x in enumerate(vehicle):
    if type(x[3])!= int and len(x[3]) > 3:
        tempNum = int(x[3].replace(",", ""))
    else:
        tempNum = int(x[3])
    if currentYear != x[2]:
        currentYear = x[2]
        print(f"Total Number of Cars: {total}")
        print(f"Ratio of active to inactive cars: {round(active/inactive, 2)}")
        active=0
        inactive=0
        total=0
        print(currentYear)
    if x[2] == currentYear:
        total+=tempNum
        if x[1] == "ACTIVE":
            active += tempNum
        else:
            inactive += tempNum

print(f"Total Number of Cars: {total}")
print(f"Ratio of active to inactive cars: {round(active/inactive,2)}")
```

```
2008
Total Number of Cars: 17970297
Ratio of active to inactive cars: 3.09
2009
Total Number of Cars: 19015088
Ratio of active to inactive cars: 3.01
2010
Total Number of Cars: 20188565
Ratio of active to inactive cars: 2.93
2011
Total Number of Cars: 21401269
Ratio of active to inactive cars: 2.89
2012
Total Number of Cars: 22702221
Ratio of active to inactive cars: 2.7
2013
Total Number of Cars: 24032666
Ratio of active to inactive cars: 2.61
2014
Total Number of Cars: 25101192
Ratio of active to inactive cars: 2.55
2015
Total Number of Cars: 26301952
Ratio of active to inactive cars: 2.42
```