Introduction to Data Science – Week 6

NETA LIVNEH

COURSE 55793

2019-2020

Last Week Recap

Sorting by different keys

Importing libraries

Using built-in and third-party libraries:

- Random
- Requests
- Pathlib

A few words about web pages

This Week

Reading and writing files

Using the csv library to read and write files

Sorting dictionaries by keys or values

Reading and Writing Files

Reading from text file:

1. Open file
2. Read strings from file
3. Close file

```
f = open(file_path, 'r')
f.read()
f.close()
```

Writing from text file:

1. Open file
2. Write strings to file
3. Close file

```
f = open(file_path, 'w')
f.write('My first line\n')
f.write('second line!\n')
f.close()
```


Opening file for reading

```
In [166]: with open(movies_path, 'r') as movies_data:
     ...:     for line in movies_data:
     ...:         name = line.split('\t')[0]
     ...:         print(name)
     ...:
name
2 Fast 2 Furious
28 Days Later
A Guy Thing
A Man Apart
A Mighty Wind
Agent Cody Banks
Alex & Emma
```

Mode	Description
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)

Reading to Text Files

Opens text file for reading:

- `file = open('file path', 'r')`
- return file object
- Mode should be 'rb' when data is not text

Reading entire file as sting (shouldn't be used with big files):

- `file_content = file.read()`
- Return string

Reading next line from file (as string).

- `line = file.readline()`
- \n is left at the end of the string.
- Blank line – only '\n' is returned
- End of file - Empty line

Reading the file as a list of strings, each representing a line.

- `Line_list = file.readlines()`

For reading lines from a file, you can loop over the file object.

- `For line in file:`
- Returns line one by one.
- This is memory efficient and fast

Writing to text file

```
In [171]: example = open(file_path, 'w')
    ...: example.write('This is my first file!' + '\n')
    ...: for i in range(10):
    ...:     example.write(f'the cube of {i} is {i**3}\n)
    ...: example.close()
```


Additional Parameters

Encoding

e*ncoding* is the name of the encoding used to decode or encode the file.

For example: utf-8, latin

Newline

newline controls how universal newlines mode works (it only applies to text mode). It can be None, '', '\n', '\r', and '\r\n'. It works as follows:

- When reading input from the stream, if *newline* is None, universal newlines mode is enabled. Lines in the input can end in '\n', '\r', or '\r\n', and these are translated into '\n' before being returned to the caller. If it is '', universal newlines mode is enabled, but line endings are returned to the caller untranslated. If it has any of the other legal values, input lines are only terminated by the given string, and the line ending is returned to the caller untranslated.
- When writing output to the stream, if *newline* is None, any '\n' characters written are translated to the system default line separator, os.linesep. If newline is '' or '\n', no translation takes place. If newline is any of the other legal values, any '\n' characters written are translated to the given string.

Reading files with the csv library

CSV – comma-separated values

csv.reader(csvfile): returns an object that can be used to iterate through the file. Each row is parsed and a *list of strings* is returned.

```
csvfile = open(csv_file, 'r', newline='') # open the file as we have seen with text.
my_reader = csv.reader(csvfile)
for row in my_reader:
    print(row)
csvfile.close()
```


Example: CSV Reader

```
def my_csv_reader(filename):
    dir_path = Path.cwd()
    file_name = dir_path.joinpath(filename + '.csv‘)
    movies = []
    with open(file_name, 'r‘, newline=‘’) as csv_file:
        csv_reader = csv.reader(csv_file)
        column_names = next(csv_reader)  # reads one line (the first)
        for current_row in csv_reader:  # reading each line in turn
            movies.append([current_row[0]])
    print ('went over {0:d} rows'.format(len(movies)))
   Return movies
```


Reading to files with the csv library

csv.writer(csvfile): returns a writer object that converts a list of values into a CSV file:

```
my_writer = csv.writer(csvfile)
```

```
# Examples:
my_writer.writerow((1, 'is smaller than', 2)) # write a single line
my_writer.writerows([('a', 1), ('b', 2), ('c', 3)]) # write multiple lines
```


Example: CSV Writer

```
def my_csv_writer(filename, movies):
    dir_path = Path.cwd()
    file_name = dir_path.joinpath(filename + '.csv‘)
    with open(file_name, 'w') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(["Movies“])   # write headline
        csv_writer.writerows(movies) # write multiple lines
```


Using DictReader

csv.DictReader(csvfile): loads each row into a dictionary with named fields. Reads field names from the first row.

```
import csv

with open('names.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['first_name'], row['last_name’])
```


Using DictWriter

csv.DictWriter(csvfile, fieldnames): write the first row according to fieldnames and write each dict element write these fieldnames to the file.

```
import csv

with open('names.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'first_name': 'Baked', 'last_name': 'Beans'})
    writer.writerow({'first_name': 'Lovely', 'last_name': 'Spam'})
    writer.writerow({'first_name': 'Wonderful', 'last_name': 'Spam'})
```


Sorted Function – Dictionary example

The built-in function sorted(iterable, key, reverse) builds a new sorted list from an iterable.

It has two optional parameters: key, receives a function that determines the comparison key, and reverse, which sorts the list in a reversed order.

Sorting by keys:

```
In [1]: my_dict = {1: 'c', 2: 'a', 3: 'f' , \
   ...:  4:'b'}
In [2]: for i in sorted(my_dict):
   ...:     print(i,my_dict[i])

1 c
2 a
3 f
4 b
```

Sorting by values:

```
In [3]: for key, value in sorted(my_dict.items(), \
   ...:  key = lambda d: d[1]):
   ...:     print(key,value)

2 a
4 b
1 c
3 f
```