assignment-Strings_and_Collection_Types_Assignment

June 24, 2018

1 Assignment 3 - Strings and Collection Types

1.1 Instructions

For this assignment we will be working with the `avengers.csv` dataset used for the Joining The Avengers Is As Deadly As Jumping Off A Four-Story Building. You can find the original data in the fivethirtyeight/data Github repository. Copy this data into the same directory as your assignment notebook.

Follow the instructions for submitting a Jupyter Notebook assignment in the submitting assignments documentation.

1.2 1. Reading data from a file (5 points)

The `avengers.csv` is encoded with ISO-8859-1 character encoding. This is common in CSV files generated by Microsoft Excel.

Try opening the the file `avengers.csv` and reading the data. What error message do you receive? What does it mean?

QUESTION: What error message do you receive? What does it mean?

ANSWER: Pandas is a library to read csv file. If I try to read the avengers.csv file which is encoded with ISO-8859-1 character encoding I get the error message 'UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe6 in position 184: invalid continuation byte'

```
import pandas as pd
data = pd.read_csv('avengers.csv')
```

1.3 2. Changing file encodings (5 points)

In order to work with the data in `avengers.csv`, we need to change its character encoding to `utf-8`.

Open the `avengers.csv` as a binary file, decode the binary data from `ISO-8859-1`, and write the `utf-8` encoded data to `avengers_utf.csv`.

```
In [19]: import csv

         #Using csv library, first 'avengers.csv' is opened to as read mode and reading, decodin
         #each line and writing each line to 'avengers_utf.csv' encoded as 'utf-8' character enc
```

```
with open('avengers.csv', 'r',newline='') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    with open('avengers_utf.csv', 'w',encoding='utf-8') as new_file:
        #List that contains the header of the csv file
        fieldnames = ['URL','Name/Alias','Appearances','Current?','Gender','Probationar

        csv_writer = csv.DictWriter(new_file, fieldnames=fieldnames, delimiter=',')

        csv_writer.writeheader()

        #writing one line at a time in the new csv file
        for line in csv_reader:
            csv_writer.writerow(line)
```

1.4 3. Read and count the lines (3 points)

Open the `resouces/avengers_utf8.csv` data and read the lines from the file. Assign the lines to the variable `lines`.

How many lines does the file contain?

```
In [23]: import csv

         filename = 'avengers_utf.csv'
         lines = []

         #The new 'avengers_utf.csv' file is then opened again as read mode
         with open(filename,'r') as csv_file:
             #each line from the file is read and added to a list named 'lines'
             for line in csv_file:
                 lines.append(line)

         #Printing the number of rows in lines list
         print('There are ',len(lines),' in the csv file')
```

```
There are  348  in the csv file
```

1.5 4. Parse the header row (12 Points)

The first row of the CSV file is the header row.

1. Using list slicing, assign the header row to the variable `header_row`.

2. CSV files use commas to separate fields. Create a list of header fields from the `header_row` variable using the string `split` method.

3. Using list slicing, make a list that contains last two fields in the header.

4. Using list slicing, make a list that contains the 3rd through 6th elements (Hint: Remember that Python uses zero-based indexing).

```
In [24]: # For  List
         # Index position
         #  0    1st
         #  1    2nd
         #  2    3rd

         #1. We access the header with in the list lines as 0 since header is the first row in t
         header_row = lines[0]

         #Printing the header
         print('The header: ',header_row)

         #2. The header labels are seperated by ',' so we can split this using split() method of
         header_fields  = header_row.split(",")

         #3. To access last two elements -2: (list slicing) is used

         # For  List
         # Index position
         #  -1    last
         #  -2    2nd last

         sub_header1 =  header_fields[-2:]

         #4. To access the 3rd through 6th elements 2:6 list slicing is used since indexing star
         sub_header2 = header_fields[2:6]
```

```
The header:  URL,Name/Alias,Appearances,Current?,Gender,Probationary Introl,Full/Reserve Avenger
```

```
In [8]: import csv

        filename = 'avengers_utf.csv'
        lines = []

        with open(filename,'r') as csv_file:
            csv_reader = csv.DictReader(csv_file)
            for line in csv_reader:
                lines.append(line)

        print('There are ',len(lines),' in the csv file')
```

```
There are  173  in the csv file
```

1.6 5. Working with Tuples (6 Points)

Using the `header` variable created in the previous problem, we are going to create a tuple from that header and assign it to the variable `header_tuple`. We also create a copy of the original header and assign it to `header_copy`.

```
import copy

header_tuple = tuple(header)
header_copy = copy.copy(header)
```

1. In the original header, the last value in the list has an extra newline. Strip the newline character from the last header field and reassign..
2. In the `header_copy` list, append the value `More info` to the end of the list. Verify the value has been added.
3. Change the value of the first item in the header list from `URL` to `url`. Verify the value has been changed.
4. Try steps 1 and 2 on `header_tuple` instead of `header`. What happens? Why can we change `header`, but not `header_tuple`?

```
In [9]: import copy
        header_tuple = tuple(header_fields)
        header_copy = copy.copy(header_fields)

        #1. Stripping the newline character from the last header field and reassign.
        # strip method of <str class> removes '\n' ,'\r' or '\t' from the string
        header_fields[-1] = header_fields[-1].strip()
        print('After Stripping header: ',header_fields)

        #2. Appending the value More info to the end of the list. '-1' means the last index
        header_copy.insert(-1,'More info')

        print('Verifying...')
        print('header_tuple :',header_tuple)
        print('header_copy: ',header_copy)

        #3. Changing the value of the first item in the header list from URL to url
        header_fields[0] = 'url'
```

```
After Stripping header:  ['URL', 'Name/Alias', 'Appearances', 'Current?', 'Gender', 'Probationar
Verifying...
header_tuple : ('URL', 'Name/Alias', 'Appearances', 'Current?', 'Gender', 'Probationary Introl',
header_copy:  ['URL', 'Name/Alias', 'Appearances', 'Current?', 'Gender', 'Probationary Introl',
```

QUESTION: What happens? Why can we change header, but not header_tuple?

ANSWER: We can change header, but not header_tuple in 'tuple' all elements are treated as a single object whereas in list elements are stored like an array so inorder to change it, the tuple must be get back to list using list() function and after striping the list is change back to tupple again

“‘python temp_list = list(header_tuple) temp_list[-1] = temp_list[-1].strip()
header_tuple = tuple(temp_list) print(header_tuple)“‘

1.7 6. Parsing Row Data (9 points)

From the `lines` variable you set in part 3, set the sixth line to the variable `line`. This should be the entry for *Thor Odinson*. From this entry, we will create a dictionary with information from this line in the CSV file.

1. Create a dictionary record

Given the field definitions defined below, create a dictionary called `record` from the string data in the `line` variable. The record should have keys corresponding to the `name` variable and type corresponding with the `type` variable. The `index` variable gives the index position of field value when the comma separated line is parsed into a list.

```
fields = [
    {'index': 0, 'name': 'url', 'type': str},
    {'index': 1, 'name': 'name_alias', 'type': str},
    {'index': 2, 'name': 'appearances', 'type': int},
    {'index': 3, 'name': 'is_current', 'type': bool},
    {'index': 4, 'name': 'gender', 'type': str},
    {'index': 7, 'name': 'year', 'type': int},
    {'index': 8, 'name': 'years_since_joining', 'type': int},
]
```

2. Since this is from an older dataset, the `years_since_joining` value is no longer accurate. Update the `years_since_joining` using the current year.

3. Using the `name_alias` field, add two new names to the record called `first_name` and `last_name`.

```
In [12]: #The line is read as sorted dictionary
         line = lines[4]
         i = 0
         fields = []
         #1. Creating a dictionary record
         #To access an element in dictionary data structure you use key
         # i.e.; In a dictionary of defination dict = {'index': 0, 'name': 'url', 'type': str}
         # the values can be accessed by their i.e.; key = 'index' of dict['key] will give you 0
         for key in line:
             if key in ['URL','Name/Alias','Gender']:
                 # For ['URL','Name/Alias','Gender'] we set 'type' as str
                 record = {}
                 record['index'] = i
                 record[key] = line[key]
                 record['type'] = str
                 #we increament only when we insert a record in feilds list
                 i += 1
                 fields.append(record)
             elif key in ['Appearances','Year','Years since joining']:
```

```
                # For ['Appearances','Year','Years since joining'] we set 'type' as int
                record = {}
                record['index'] = 7 if key=='Year' else 8
                record[key] = line[key]
                record['type'] = int
                i += 1
                fields.append(record)
            elif key in ['Current?']:
                # For ['Current?'] we set 'type' as bool
                record = {}
                record['index'] = i
                record[key] = line[key]
                record['type'] = bool
                i += 1
                fields.append(record)



        print('The dictionary: ',fields)

        #2. Updating the years_since_joining using the current year

        # Loop each element of list containing dictionaries {'index': idx, 'name': 'url', 'type
        # and setting feild['Years since joining'] with 2018
        for field in fields:
            field['Years since joining'] = '2018'
        #3.  Using the name_alias field, added record called first_name and last_name

        #The 2nd elemetnof field list contains 'Name/Alias' and we split the names using split(
        #which creates new temp list of strings separated by whitespace
        temp = fields[1]['Name/Alias'].split()

        #From temp list assignning first name to 7th index and second name to 8th index
        fields.append({'index':5,'first_name':temp[0],'type':str})
        fields.append({'index':6,'second_name':temp[1],'type':str})

        print()
        print('The modified dictionary: ',fields)
```

```
The dictionary:  [{'index': 0, 'URL': 'http://marvel.wikia.com/Thor_Odinson_(Earth-616)', 'type'

The modified dictionary:  [{'index': 0, 'URL': 'http://marvel.wikia.com/Thor_Odinson_(Earth-616)
```