module 1

July 29, 2021

1 Module 1

In this assignment, you will work with ufo sightings data.

- The data includes various data points about individual ufo sightings
- Data File(s): ufo-sightings.csv

[3]:
```
############################################################
### EXECUTE THIS CELL BEFORE YOU TO TEST YOUR SOLUTIONS ###
############################################################

import imp, os, sys
sol = imp.load_compiled("solutions", "./solutions.py")
sol.get_solutions("ufo-sightings.csv")
from nose.tools import assert_equal
```

[1]:
```
'''
1. Import the csv module. Load and read the UFO sightings data set, from the␣
 ↪ufo-sightings.csv file,
into a DictReader inside a with statement.  Assume the data file is in the same␣
 ↪directory as the code.

Print the field names of the data set. Iterate over the reader to put the data␣
 ↪into a list name "ufosightings".

'''

import csv
with open('ufo-sightings.csv','r') as csvfile:
    reader = csv.DictReader(csvfile)
    print(reader.fieldnames)



    ufosightings = []
    for row in reader:
        ufosightings.append(row)
```

```
['datetime', 'city', 'state', 'country', 'shape', 'duration (seconds)',
'duration (hours/min)', 'comments', 'date posted', 'latitude', 'longitude ']
```

[5]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(ufosightings, sol.ufosightings)
print("Success!")
```

```
Success!
```

[6]:
```
'''
2. How many sightings were there in total?  Put the count in␣
 ↪"ufosightings_count" and print the result.
'''
# your code here
ufosightings_count = len(ufosightings)
print(ufosightings_count)
```

```
80332
```

[7]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(ufosightings_count, sol.ufosightings_count)
print("Success!")
```

```
Success!
```

[10]:
```
'''
3. How many sightings were there in the US?  Put the US sightings in␣
 ↪"sightings_us" and print.

Hint: Check for ufo sightings where the country is 'us'.

'''

# your code here
sightings_us = [row for row in ufosightings if row["country"] == "us"]
print(sightings_us)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

```
Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

[11]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(sightings_us,  sol.sightings_us)
print("Success!")
```

```
Success!
```

[19]:
```
'''
4. Let's find the "fireball" sighting(s) that lasted more than ten seconds in␣
 ↪US.
Print the the datetime and state of each.  Put the data in "fball" and print␣
 ↪the result.

Note: Consider only the US sightings stored in "sightings_us".

- Cast the duration in seconds to a float (decimal).
- Check if duration is greater than 10.
- Check if the shape is "fireball".

'''

#First, define a Python function that checks if a given duration (seconds) is␣
 ↪"valid"
fball=[]
for row in ufosightings:
    if row['country'] == 'us' and row['shape'] == 'fireball' and␣
 ↪float(row['duration (seconds)'])>10.00:
        fball.append(row)
# your code here
```

[18]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(fball, sol.fball)
print("Success!")
```

```
Success!
```

[20]:
```
'''
5. Sort the above list by duration. What was the datetime and duration of the longest sighting?
Put the sorted list in "fballsorted" and print the result.

- Cast the duration in seconds to a float (decimal).
- Sort in reverse order.

'''

# your code here
fballsorted = []
fballsorted = sorted(fball, key = lambda x: float(x['duration (seconds)']),reverse = True)
print((fballsorted[0])["datetime"], (fballsorted[0])["duration (seconds)"])
```

```
8/3/2008 21:00 10526400
```

[21]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(fballsorted, sol.fballsorted)
print("Success!")
```

```
Success!
```

[22]:
```
'''
6. What state had the longest lasting "fireball"?  Put the state in "state" and print the result.

- Check if the shape is "fireball".
- Cast the duration in seconds to a float (decimal).
- Get the record with the largest (max) duration in seconds.
- Get the state for the record.

'''

# your code here
fballsorted = sorted(fball, key = lambda x: float(x['duration (seconds)']),reverse = True)
state = (fballsorted[0])['state']
```

[23]:
```
############################
### TEST YOUR SOLUTION ###
############################
```

```
assert_equal(state, sol.state)
print("Success!")
```

```
Success!
```

[24]:
```
'''
7. Let's assume that any sighting (of any shape) of 0 seconds is insignificant.
Write code to filter out these extraneous records and get the shortest sighting␣
 ↪overall now.
Put the minimum duration in "min_duration" and print the result.
Use ufosightings
Note: Consider all sightings stored in "ufosightings".

'''

# your code here
for row in ufosightings:
    fball.append(row)
fballsorted = sorted(fball, key = lambda x: (x['duration (seconds)']), reverse␣
 ↪= False)
min_duration = float((fballsorted[0])["duration (seconds)"])
min_duration
```

[24]:
```
0.001
```

[25]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(min_duration, sol.min_duration)
print("Success!")
```

```
Success!
```

[27]:
```
'''
8. What are the top 3 shapes sighted, and how many sightings were there for␣
 ↪each?

Note: Consider all sightings stored in "ufosightings".

- Create a new list "sightings_shapes" containing values from the "shape"␣
 ↪column in ufosightings.
- Create a new dictionary "count" with values of that column as keys and the␣
 ↪counts as values.
- Get a list of the dictionary keys and values using the items() method.  This␣
 ↪will return a list of key:value pairs.
```

```
Sort the list of key:value pairs in reverse order, from greatest (most sightings) to least.

Get the top 3 and store in "top3shapes".  Print the result.

'''

#Create a new list containing values from the "shape" column in ufosightings.
# your code here
sightings_shapes=[]
i=0
for dic in ufosightings:
    sightings_shapes.append((ufosightings[i])['shape'])
    i=i+1


#Create a new dictionary with values of that column as keys and the counts as values.
# your code here
count = {}
for shapes in sightings_shapes:
    if shapes not in count:
        count[shapes]=1
    else:
        count[shapes]=count[shapes]+1

#Get a list of the dictionary keys and values (use the items() method) and sort them in reverse order, from greatest (most sightings) to least.
#Get and print the top 3.
# your code here
top3shapes = sorted(count.items(), key = lambda x: x[1], reverse=True)[:3]
top3shapes
```

[27]:
```
[('light', 16565), ('triangle', 7865), ('circle', 7608)]
```

[28]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(sightings_shapes, sol.sightings_shapes)
print("Success!")
```

```
Success!
```

[29]:
```
############################
### TEST YOUR SOLUTION ###
############################
```

```
assert_equal(count, sol.count)
print("Success!")
```

```
Success!
```

[30]:
```
############################
### TEST YOUR SOLUTION ###
############################

assert_equal(top3shapes, sol.top3shapes)
print("Success!")
```

```
Success!
```

[ ]:
```

```