JSON - Lab

Introduction

In this lab, you'll practice navigating JSON data structures.

Objectives

You will be able to:

- Practice using Python to load and parse JSON documents

Your Task: Find the Total Payments for Each Candidate

We will be using the same dataset, `nyc_2001_campaign_finance.json`, as in the previous lesson. Recall that the description is:

> A listing of public funds payments for candidates for City office during the 2001 election cycle

For added context, the Ciy of New York provides matching funds for eligible contributions made to candidates, using various ratios depending on the contribution amount (more details here). So these are not the complete values of all funds raised by these candidates, they are the amounts matched by the city. For that reason we expect that some of the values will be identical for different candidates.

Recall also that the dataset is separated into `meta`, which contains metadata, and `data`, which contains the actual campaign finance records. You will need to use the information in `meta` to understand how to interpret the information in `data`.

Your goal is to create a list of tuples, where the first value in each tuple is the name of a candidate in the 2001 election, and the second value is the total payments they received. The structure should look like this:

```
[
    ("John Smith", 62184.00),
    ("Jane Doe", 133146.00),
    ...
]
```

The list should contain 284 tuples, since there were 284 candidates.

Open the Dataset

Import the `json` module, open the `nyc_2001_campaign_finance.json` file using the built-in Python `open` function, and load all of the data from the file into a Python object using `json.load`.

Assign the result of `json.load` to the variable name `data`.

```
import json

with open('nyc_2001_campaign_finance.json') as f:
    data = json.load(f)
```

Recall the overall structure of this dataset:

```
print(f"The overall data type is {type(data)}")
print(f"The keys are {list(data.keys())}")
```

```
print()
print("The value associated with the 'meta' key has metadata, including all of these
print(list(data['meta']['view'].keys()))
print()
print(f"The value associated with the 'data' key is a list of {len(data['data'])} re
```

```
The overall data type is <class 'dict'>
The keys are ['meta', 'data']

The value associated with the 'meta' key has metadata, including all of these
attributes:
['id', 'name', 'attribution', 'averageRating', 'category', 'createdAt',
'description', 'displayType', 'downloadCount', 'hideFromCatalog',
'hideFromDataJson', 'indexUpdatedAt', 'newBackend', 'numberOfComments', 'oid',
'provenance', 'publicationAppendEnabled', 'publicationDate', 'publicationGroup',
'publicationStage', 'rowClass', 'rowsUpdatedAt', 'rowsUpdatedBy', 'tableId',
'totalTimesRated', 'viewCount', 'viewLastModified', 'viewType', 'columns',
'grants', 'metadata', 'owner', 'query', 'rights', 'tableAuthor', 'tags', 'flags']

The value associated with the 'data' key is a list of 285 records
```

Find the Column Names

We know that each record in the data list looks something like this:

```
data['data'][1]
```

```
[2,
 '9D257416-581A-4C42-85CC-B6EAD9DED97F',
 2,
 1315925633,
 '392904',
 1315925633,
 '392904',
 '{\n}',
 '2001',
 'B4',
 'Aboulafia, Sandy',
 '5',
 None,
 '44',
 'P',
```

```
 '45410.00',
 '0',
 '0',
 '45410.00']
```

We could probably guess which of those values is the candidate name, but it's unclear which value is the total payments received. To get that information, we need to look at the metadata.

Investigate the value of `data['meta']['view']['columns']`. It currently contains significantly more information than we need. Extract just the values associated with the `name` keys, so we have a list of the column names.

The result should look something like this:

```
[
    "sid",
    "id",
    "position",
    ...
]
```

Name this variable `column_names`.

```
# First, we look at data['meta']['view']['columns']
# What is the data type?

column_data = data['meta']['view']['columns']
type(column_data)
```

```
list
```

```
# With a list, it's often useful to look at the
# first entry, or first few entries
column_data[:3]
```

```
[{'id': -1,
  'name': 'sid',
  'dataTypeName': 'meta_data',
  'fieldName': ':sid',
  'position': 0,
```

```
  'renderTypeName': 'meta_data',
  'format': {},
  'flags': ['hidden']},
 {'id': -1,
  'name': 'id',
  'dataTypeName': 'meta_data',
  'fieldName': ':id',
  'position': 0,
  'renderTypeName': 'meta_data',
  'format': {},
  'flags': ['hidden']},
 {'id': -1,
  'name': 'position',
  'dataTypeName': 'meta_data',
  'fieldName': ':position',
  'position': 0,
  'renderTypeName': 'meta_data',
  'format': {},
  'flags': ['hidden']}]
```

```
# So, we have a list of dictionaries. We note that
# each dictionary has the key 'name' like was mentioned
# previously

# To extract the names, let's use a list comprehension
column_names = [info['name'] for info in column_data]
column_names
```

```
['sid',
 'id',
 'position',
 'created_at',
 'created_meta',
 'updated_at',
 'updated_meta',
 'meta',
 'ELECTION',
 'CANDID',
 'CANDNAME',
 'OFFICECD',
 'OFFICEBORO',
 'OFFICEDIST',
 'CANCLASS',
 'PRIMARYPAY',
 'GENERALPAY',
```

```
 'RUNOFFPAY',
 'TOTALPAY']
```

```
# There should be 19 names
assert len(column_names) == 19
# CANDNAME and TOTALPAY should be in there
assert "CANDNAME" in column_names and "TOTALPAY" in column_names
```

Ok, now we know what each of the columns represents.

The columns we are looking for are called `CANDNAME` and `TOTALPAY`. Now that we have this list, we should be able to figure out which of the values in each record lines up with those column names.

Loop Over the Records to Find the Names and Payments

The data records are contained in `data['data']`. Recall that the first (`0`-th) one is more of a header and should be skipped over.

Loop over the records in `data['data']` and extract the name and total payment from the city. Make sure you convert the total payment to a float, then make a tuple representing that candidate. Append the tuple to an overall list of results called `candidate_total_payments`.

```
# In theory we could just look at the list and
# count by hand to figure out the index of these
# strings, but Python can do it for us
name_index = column_names.index("CANDNAME")
total_payments_index = column_names.index("TOTALPAY")

print("The candidate name is at index", name_index)
print("The total payment amout is at index", total_payments_index)
```

```
The candidate name is at index 10
The total payment amout is at index 18
```

```
candidate_total_payments = []

# Loop over records starting at index 1 to skip header
for record in data['data'][1:]:
    name = record[name_index]
```

```
        total_payments = float(record[total_payments_index])
        candidate_total_payments.append((name, total_payments))

# Print the first five and last five
print(candidate_total_payments[:5])
print(candidate_total_payments[-5:])
```

```
[('Aboulafia, Sandy', 45410.0), ('Adams, Jackie R', 11073.0), ('Addabbo, Joseph
P', 149320.0), ('Alamo-Estrada, Agustin', 27400.0), ('Allen, William A',
62990.0)]
[('Wilson, John H', 0.0), ('Wooten, Donald T', 0.0), ('Yassky, David', 150700.0),
('Zapiti, Mike', 12172.0), ('Zett, Lori M', 0.0)]
```

```
# There should be 284 records
assert len(candidate_total_payments) == 284

# Each record should contain a tuple
assert type(candidate_total_payments[0]) == tuple

# That tuple should contain a string and a number
assert len(candidate_total_payments[0]) == 2
assert type(candidate_total_payments[0][0]) == str
assert type(candidate_total_payments[0][1]) == float
```

Now that we have this result, we can answer questions like: *which candidates received the most total payments from the city*?

```
# Print the top 10 candidates by total payments
sorted(candidate_total_payments, key=lambda x: x[1], reverse=True)[:10]
```

```
[('Green, Mark', 4534230.0),
 ('Ferrer, Fernando', 2871933.0),
 ('Hevesi, Alan G', 2641247.0),
 ('Vallone, Peter F', 2458534.0),
 ('Gotbaum, Betsy F', 1625090.0),
 ('Berman, Herbert E', 1576860.0),
 ('DiBrienza, Stephen', 1336655.0),
 ('Stringer, Scott M', 1223721.0),
 ('Markowitz, Marty', 1166294.0),
 ('Thompson, Jr., William C', 1096359.0)]
```

Since you found all of the column names, it is also possible to display all of the data in a nice tabular format using pandas. That code would look like this:

```
import pandas as pd

pd.DataFrame(data=data['data'][1:], columns=column_names)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	sid	id	position	created_at	created_meta	update
0	2	9D257416-581A-4C42-85CC-B6EAD9DED97F	2	1315925633	392904	131592
1	3	B80D7891-93CF-49E8-86E8-182B618E68F2	3	1315925633	392904	131592
2	4	BB012003-78F5-406D-8A87-7FF8A425EE3F	4	1315925633	392904	131592
3	5	945825F9-2F5D-47C2-A16B-75B93E61E1AD	5	1315925633	392904	131592

	sid	id	position	created_at	created_meta	update
4	6	9546F502-39D6-4340-B37E-60682EB22274	6	1315925633	392904	131592
...	...	...	...	...	...	...
279	281	C50E6A4C-BDE9-4F12-97F4-95D467013540	281	1315925633	392904	131592
280	282	04C6D19F-FF63-47B0-B26D-3B8F98B4C16B	282	1315925633	392904	131592
281	283	A451E0E9-D382-4A97-AAD8-D7D382055F8D	283	1315925633	392904	131592
282	284	E84BCD0C-D6F4-450F-B55B-3199A265C781	284	1315925633	392904	131592
283	285	5BBC9676-2119-4FB5-9DAB-DE3F71B7681A	285	1315925633	392904	131592

284 rows × 19 columns

Summary

Congratulations! You've started exploring some more JSON data structures used for the web and got to practice data munging and exploring!

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

- **Jupyter Notebook** 100.0%