JSON

Introduction

In this lesson, you'll continue investigating new formats for data. Specifically, you'll investigate one of the most popular data formats for the web: JSON files.

Objectives

You will be able to:

- Describe features of the JSON format and the Python `json` module
- Use Python to load and parse JSON documents

JSON Format

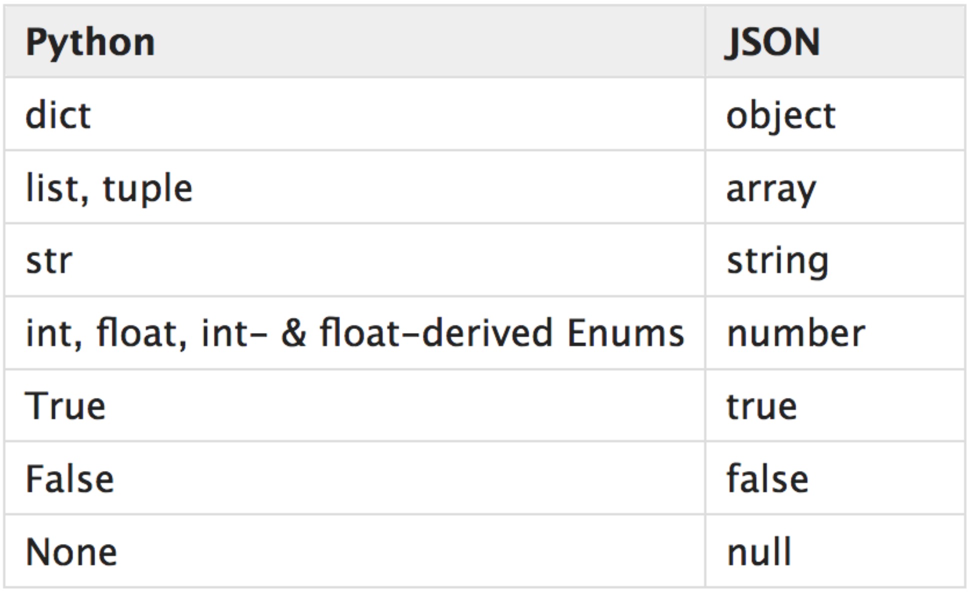
JSON stands for JavaScript Object Notation. Similar to CSV, JSON is a **plain text** data format. However the structure of JSON — based on the syntax of JavaScript — is more complex.

Here's a brief preview of a JSON file:

```
{
  "meta" : {
    "view" : {
      "id" : "k3cd-yu9d",
      "name" : "2001 Campaign Expenditures",
      "attribution" : "Campaign Finance Board (CFB)",
      "averageRating" : 0,
      "category" : "City Government",
      "createdAt" : 1315950581,
      "description" : "A listing of  expenditures for candidates for City office during the 2001 election cycle",
      "displayType" : "table",
      "downloadCount" : 1314,
      "hideFromCatalog" : false,
      "hideFromDataJson" : false,
      "indexUpdatedAt" : 1536596254,
      "newBackend" : false,
      "numberOfComments" : 0,
      "oid" : 25997384,
      "provenance" : "official",
      "publicationAppendEnabled" : false,
      "publicationDate" : 1506007637,
      "publicationGroup" : 240369,
      "publicationStage" : "published",
      "rowClass" : "",
      "rowsUpdatedAt" : 1506007571,
      "rowsUpdatedBy" : "d5dp-fses",
      "tableId" : 14211791,
      "totalTimesRated" : 0,
      "viewCount" : 527,
      "viewLastModified" : 1536605711,
      "viewType" : "tabular",
      "columns" : [ {
        "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" ]
```

```
"tags": ["hidden"]
```

As you can see, JSON is not a tabular format with one set of rows and one set of columns. JSON files are often nested in a hierarchical structure and will have data structures analogous to Python dictionaries and lists. Here's all of the built-in supported data types in JSON and their counterparts in Python:



Python	JSON
dict	object
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True	true
False	false
None	null

`json` Module

In theory we could write our own custom code to split that string on `{` , `"` , `:` etc. and parse the contents of the file into the appropriate Python data structures.

Instead, we'll go ahead and use a pre-built Python module designed for this purpose. It will give us a powerful starting point for accessing and manipulating the data in JSON files. This module is called `json` .

You can find full documentation for this module here (https://docs.python.org/3/library/json.html).

To use the `json` module, start by importing it:

In [1]:
```
import json
```

`json.load`

To load data from a JSON file, you first open the file using Python's built-in `open` function. Then you pass the file object to the `json.load` function, which returns a Python object representing the contents of the file.

In the cell below, we open the campaign finance JSON file previewed above:

In [2]:
```
with open('nyc_2001_campaign_finance.json') as f:
    data = json.load(f)
print(type(data))
```

```
<class 'dict'>
```

As you can see, this loaded the data as a dictionary. You can begin to investigate the contents of a JSON file by using our traditional Python methods.

Parsing a JSON File

Since we have a dictionary, check its keys:

In [4]:
```
data.keys()
```

Out[4]:
```
dict_keys(['meta', 'data'])
```

Investigate what data types are stored within the values associated with those keys:

In [5]:
```
for v in data.values():
    print(type(v))
```

```
<class 'dict'>
<class 'list'>
```

Parsing Metadata

Then we can dig a level deeper. What are the keys of the nested dictionary?

In [7]:
```
data['meta'].keys()
```

Out[7]:
```
dict_keys(['view'])
```

And what is the type of the value associated with that key?

In [8]:
```
type(data['meta']['view'])
```

Out[8]:
```
dict
```

Again, what are the keys of that twice-nested dictionary?

```
In [7]: data['meta']['view'].keys()
```

```
Out[7]: dict_keys(['id', 'name', 'attribution', 'averageRating', 'category', 'createdA
t', 'description', 'displayType', 'downloadCount', 'hideFromCatalog', 'hideFrom
DataJson', 'indexUpdatedAt', 'newBackend', 'numberOfComments', 'oid', 'provenan
ce', 'publicationAppendEnabled', 'publicationDate', 'publicationGroup', 'public
ationStage', 'rowClass', 'rowsUpdatedAt', 'rowsUpdatedBy', 'tableId', 'totalTim
esRated', 'viewCount', 'viewLastModified', 'viewType', 'columns', 'grants', 'me
tadata', 'owner', 'query', 'rights', 'tableAuthor', 'tags', 'flags'])
```

That is a lot of keys! One way we might try to view all of that information is using the `pandas` package to make a table.

In [9]:
```
import pandas as pd
pd.set_option("max_colwidth", 120)
pd.DataFrame(
    data=data['meta']['view'].values(),
    index=data['meta']['view'].keys(),
    columns=["value"]
)
```

Out[9]:

	value
id	8dhd-zvi6
name	2001 Campaign Payments
attribution	Campaign Finance Board (CFB)
averageRating	0
category	City Government
createdAt	1315950830
description	A listing of public funds payments for candidates for City office during the 2001 election cycle
displayType	table
downloadCount	1470
hideFromCatalog	False
hideFromDataJson	False
indexUpdatedAt	1536596254
newBackend	False
numberOfComments	0
oid	4140996
provenance	official
publicationAppendEnabled	False
publicationDate	1371845179
publicationGroup	240370
publicationStage	published
rowClass	
rowsUpdatedAt	1371845177
rowsUpdatedBy	5fuc-pqz2
tableId	932968
totalTimesRated	0
viewCount	233
viewLastModified	1536605717
viewType	tabular
columns	[{'id': -1, 'name': 'sid', 'dataTypeName': 'meta_data', 'fieldName': ':sid', 'position': 0, 'renderTypeName': 'meta_...

	value
grants	[{'inherited': False, 'type': 'viewer', 'flags': ['public']}]
metadata	{'rdfSubject': '0', 'rdfClass': '', 'attachments': [{'filename': 'Data_Dictionary_Public_Funds_Payments_FINAL.xlsx',...
owner	{'id': '5fuc-pqz2', 'displayName': 'NYC OpenData', 'profileImageUrlLarge': '/api/users/5fuc-pqz2/profile_images/LARG...
query	{}
rights	[read]
tableAuthor	{'id': '5fuc-pqz2', 'displayName': 'NYC OpenData', 'profileImageUrlLarge': '/api/users/5fuc-pqz2/profile_images/LARG...
tags	[finance, campaign finance board, cfb, nyccfb, campaign finance, elections, contributions, politics, campaign, funding]
flags	[default, restorable, restorePossibleForType]

So, it looks like the information under the `meta` key is essentially all of the metadata about the dataset, including the category, attribution, tags, etc.

Now let's look at the main data.

Parsing Data

This time, let's look at the value associated with the `data` key. Recall that we previously identified that this had a `list` data type, so let's look at the length:

In [10]:
```
len(data['data'])
```

Out[10]:
```
285
```

Now let's look at a couple different values:

In [11]:
```
data['data'][0]
```

Out[11]:
```
[1,
 'E3E9CC9F-7443-43F6-94AF-B5A0F802DBA1',
 1,
 1315925633,
 '392904',
 1315925633,
 '392904',
 '{\n  "invalidCells" : {\n    "1519001" : "TOTALPAY",\n    "1518998" : "PRIMAR
YPAY",\n    "1519000" : "RUNOFFPAY",\n    "1518999" : "GENERALPAY",\n    "15189
94" : "OFFICECD",\n    "1518996" : "OFFICEDIST",\n    "1518991" : "ELECTION"\n
}\n}',
 None,
 'CANDID',
 'CANDNAME',
 None,
 'OFFICEBORO',
 None,
 'CANCLASS',
 None,
 None,
 None,
 None]
```

In [12]:
```
data['data'][1]
```

Out[12]:
```
[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']
```

```
In [13]: data['data'][2]
```

```
Out[13]: [3,
          'B80D7891-93CF-49E8-86E8-182B618E68F2',
          3,
          1315925633,
          '392904',
          1315925633,
          '392904',
          '{\n}',
          '2001',
          '445',
          'Adams, Jackie R',
          '5',
          None,
          '7',
          'P',
          '11073.00',
          '0',
          '0',
          '11073.00']
```

This looks more like some kind of tabular data, where the first (`0` -th) row is some kind of header. Again, let's use pandas to make this into a more-readable table format:

In [14]:
```
pd.DataFrame(data['data'])
```

Out[14]:

	0	1	2	3	4	5	6	7	8
0	1	E3E9CC9F-7443-43F6-94AF-B5A0F802DBA1	1	1315925633	392904	1315925633	392904	{\n "invalidCells" : {\n "1519001" : "TOTALPAY",\n "1518998" : "PRIMARYPAY",\n "1519000" : "RUNOFFPAY",\n ...	None
1	2	9D257416-581A-4C42-85CC-B6EAD9DED97F	2	1315925633	392904	1315925633	392904	{\n}	2001
2	3	B80D7891-93CF-49E8-86E8-182B618E68F2	3	1315925633	392904	1315925633	392904	{\n}	2001
3	4	BB012003-78F5-406D-8A87-7FF8A425EE3F	4	1315925633	392904	1315925633	392904	{\n}	2001
4	5	945825F9-2F5D-47C2-A16B-75B93E61E1AD	5	1315925633	392904	1315925633	392904	{\n}	2001
...	...	...	...	...	...	...	...	...	...
280	281	C50E6A4C-BDE9-4F12-97F4-95D467013540	281	1315925633	392904	1315925633	392904	{\n}	2001
281	282	04C6D19F-FF63-47B0-B26D-3B8F98B4C16B	282	1315925633	392904	1315925633	392904	{\n}	2001
282	283	A451E0E9-D382-4A97-AAD8-D7D382055F8D	283	1315925633	392904	1315925633	392904	{\n}	2001
283	284	E84BCD0C-D6F4-450F-B55B-3199A265C781	284	1315925633	392904	1315925633	392904	{\n}	2001
284	285	5BBC9676-2119-4FB5-9DAB-DE3F71B7681A	285	1315925633	392904	1315925633	392904	{\n}	2001

285 rows × 19 columns

We still have some work to do to understand what all of the columns are supposed to mean, but now we have a general sense of what the data looks like.

Extracting a Value from a JSON File

Now, let's say that our task is:

> Extract the description of the dataset

We know from our initial exploration that this JSON file contains `meta` and `data` , and that `meta` has this kind of high-level information whereas `data` has the actual records relating to campaign finance.

Let's look at the keys of `meta` again:

In [15]:
```
data['meta']['view'].keys()
```

Out[15]:
```
dict_keys(['id', 'name', 'attribution', 'averageRating', 'category', 'createdA
t', 'description', 'displayType', 'downloadCount', 'hideFromCatalog', 'hideFrom
DataJson', 'indexUpdatedAt', 'newBackend', 'numberOfComments', 'oid', 'provenan
ce', 'publicationAppendEnabled', 'publicationDate', 'publicationGroup', 'public
ationStage', 'rowClass', 'rowsUpdatedAt', 'rowsUpdatedBy', 'tableId', 'totalTim
esRated', 'viewCount', 'viewLastModified', 'viewType', 'columns', 'grants', 'me
tadata', 'owner', 'query', 'rights', 'tableAuthor', 'tags', 'flags'])
```

Ok, `description` is the 7th one! Let's pull the value associated with the `description` key:

In [16]:
```
data['meta']['view']['description']
```

Out[16]:
```
'A listing of public funds payments for candidates for City office during the 2
001 election cycle'
```

Great! This is the general process you will use when extracting information from a JSON file.

Summary

As you can see, there's a lot going on here with the deeply nested structure of JSON data files. In the upcoming lab, you'll get a chance to practice loading files and continuing to parse and extract the data as you did here.