JSON

Outline

- What is JSON?
  - Syntax
  - Example
  - Schema
- Validating JSON file
- Using JSON with R
- Using JSON with Python
  - Writing a JSON file
  - Reading a JSON
- Accessing JSON Properties in Python

What is JSON?

- "JSON" stands for "JavaScript Object Notation"
  - Lightweight data-interchange format
  - Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs
  - Structured representation of data object
  - Can be parsed with most modern languages
- JSON Schema can be used to validated a JSON file

JSON Syntax Rules

- JSON is almost identical to python dictionary except for
  - In JSON, `true` and `false` are not capitalized
  - In JSON, `null` is used instead of `None`
- Uses key/value pairs: {“name”: “John”}
- Uses double quotes around KEY and VALUE
- Must use the specified types
- File type is “.json”
- A *value* can be: A string, a number, true, false, null, an object, or an array
- *Strings* are enclosed in double quotes, and can contain the usual assortment of escaped characters

JSON Example

```
{
        "name": "John Smith",
        "age": 35,
        "address": {
                "street": "5 main St.",
                "city": "Austin"
        },
        "children": ["Mary", "Abel"]
}
```

JSON Schema

- A JSON Schema allows you to specify what type of data can go into your JSON files.
- It allows you to restrict the type of data entered.

JSON Schema

```
{
 "type": "object",
 "properties": {
  "name": {
   "type": "string"
  },
  "age": {
   "type": "integer"
  },
```

```
"address": {
    "type": "object",
    "properties": {
     "street": {
      "type": "string"
     },
     "city": {
      "type": "string"
     }
    }
},
```

```
"children": {
    "type": "array",
    "items": [
     {
      "type": "string"
     }
    ]
}
}
```

Validating JSON file

- The following website can be used to validate a JSON file against a schema

  https://www.jsonschemavalidator.net/

- Paste both the schema and the corresponding JSON file

Using JSON with R

```
install.packages("rjson")

# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file = ”input.json")

# Print the result.
print(result)

# Convert JSON file to a data frame.
json_data_frame <- as.data.frame(result)

print(json_data_frame)
```

Input file

- { "ID":["1","2","3","4","5","6","7","8"],
  "Name":["Rick","Dan","Michelle","Ryan","Gary","Nina","Simon","Guru"],
  "Salary":["623.3","515.2","611","729","843.25","578","632.8","722.5"], "StartDate":[
  "1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013", "7/30/2013","6/17/2014"], "Dept":[
  "IT","Operations","IT","HR","Finance","IT","Operations","Finance"] }

Convert JSON file to a data frame

```
# Convert JSON file to a data frame.
json_data_frame <- as.data.frame(result)

print(json_data_frame)
```

Data frame

	id,	name,	salary,	start_date,	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	NA	Gary	843.25	2015-03-27	Finance
6	6	Nina	578.00	2013-05-21	IT
7	7	Simon	632.80	2013-07-30	Operations
8	8	Guru	722.50	2014-06-17	Finance

Using JSON with Python

- To work with JSON (string, or file containing JSON object), you can use Python's json module.

```
import json
```

Loading JSON data from a file

- Example:

```
def load_json(filename):
    with open(filename) as file:
        jsn = json.load(file)
        file.close()
    return jsn
person = load_json('person.json')
```

- This command parse the above person.json using json.load() method from the json module. The result is a Python dictionary.

Writing JSON object to a file

- Example:

```
person = { "name": "John Smith", "age": 35,
"address": {"street": "5 main St.", "city":
"Austin"}, "children": ["Mary", "Abel"]}

with open('person_to_json.json', 'w') as fp:
    json.dump(person, fp, indent=4)
```

- Using json.dump(), we can convert Python Objects to JSON file.

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property "name"

- `Print(person["name"])`
- `John Smith`

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property “age"

- `person["age"]`
- `35`

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property “street"

- `print(person["address"]["street"])`
- `5 main St.`

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property “street"

- `print(person["address"]["city"])`
- `Austin`

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property “street"

- `print(person["children"][0])`
- `Mary`

Accessing JSON Properties in Python

- Example:

Assume that you already loaded your person.json as follows.

```
person = load_json('person.json')
```

To access the property “street"

- `print(person["children"][1])`
- `Abel`

Python – JSON Objects

Python	JSON Equivalent
`dict`	object
`list`, `tuple`	array
`str`	string
`int`, `float`, `int`	number
`True`	true
`False`	false
`None`	null

Credit:

- https://www.youtube.com/watch?v=wI1CWzNtE-M
- https://www.programiz.com/python-programming/json