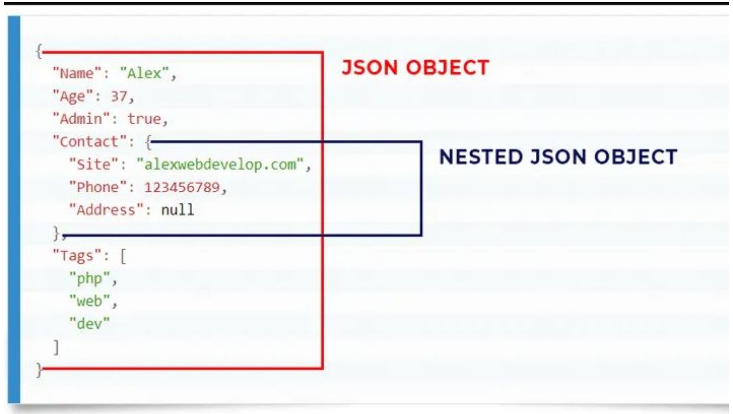
Advanced Data Processing

THE KNOWLEDGE HOUSE

Celebrating 10 Years of Diversifying Tech

Agenda - Schedule

1. Warm-Up
2. Hierarchy of Data Needs
3. Data Formats & JSON
4. Break
5. Lab



```
{
  "Name": "Alex",
  "Age": 37,
  "Admin": true,
  "Contact": {
    "Site": "alexwebdevelop.com",
    "Phone": 123456789,
    "Address": null
  },
  "Tags": [
    "php",
    "web",
    "dev"
  ]
}
```

JavaScript Object Notation (JSON) is an open-standard data format or interchange for semi-structured data. It is text-based and readable by humans and machines. [*https://www.snowflake.com/guides/what-is-json*](https://www.snowflake.com/guides/what-is-json)

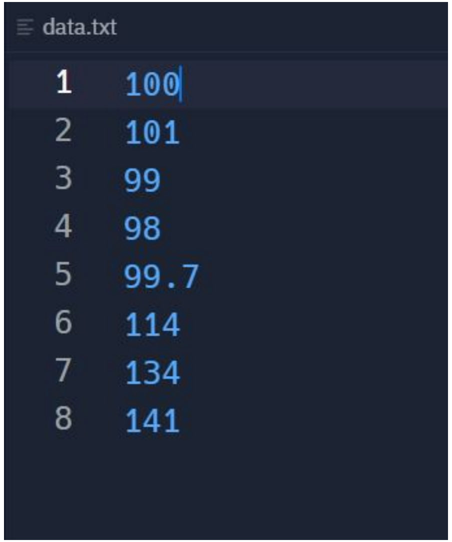
Agenda - Goals

- Review the different data formats you will work with during the fellowship
- Understand how to interpret a JSON file
- Get introduced to working with JSON files in Python

Warm-Up

```
def evaluate(obj) -> int:
    count = 0
    for d in obj:
        count += 1
    return count

obj = open("data.txt")
print(evaluate(obj))
```



data.txt

```
100
101
99
98
99.7
114
134
141
```

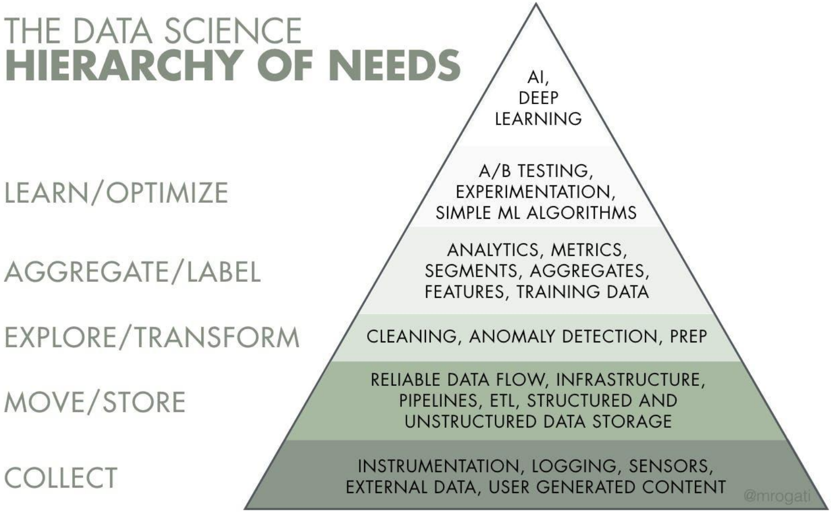
Work together to figure out what will occur when we run this code.

Hierarchy of Data Needs - Review

Hierarchy of Data Needs

Let's see **how what we've learned (and what we will learn) applies to this pyramid.**

First let's list off the hierarchies again...


THE DATA SCIENCE
HIERARCHY OF NEEDS
LEARN/OPTIMIZE
AGGREGATE/LABEL
EXPLORE/TRANSFORM
MOVE/STORE
COLLECT
AI, DEEP LEARNING
A/B TESTING, EXPERIMENTATION, SIMPLE ML ALGORITHMS
ANALYTICS, METRICS, SEGMENTS, AGGREGATES, FEATURES, TRAINING DATA
CLEANING, ANOMALY DETECTION, PREP
RELIABLE DATA FLOW, INFRASTRUCTURE, PIPELINES, ETL, STRUCTURED AND UNSTRUCTURED DATA STORAGE
INSTRUMENTATION, LOGGING, SENSORS, EXTERNAL DATA, USER GENERATED CONTENT
@mrogati

Hierarchy of Data Needs

The data science process involves:

- **Collecting data:**
- **Storing data:**
- **Exploring data:**
- **Aggregating data:**
- **Learning about data:**

SQL
pandas
Python
sklearn
keras
word2vec

We've learned about many different technologies/concepts. Can you name their appropriate categories?

Hierarchy of Data Needs

- Collecting data: Python
- Storing data: SQL
- Exploring data: pandas, SQL
- Aggregating data: pandas, SQL
- Learning about data: sklearn, keras, word2vec

Roughly speaking, these are the categories that each tool fits into. For the remainder of this fellowship, we will explore ways we can collect data using the base Python language.

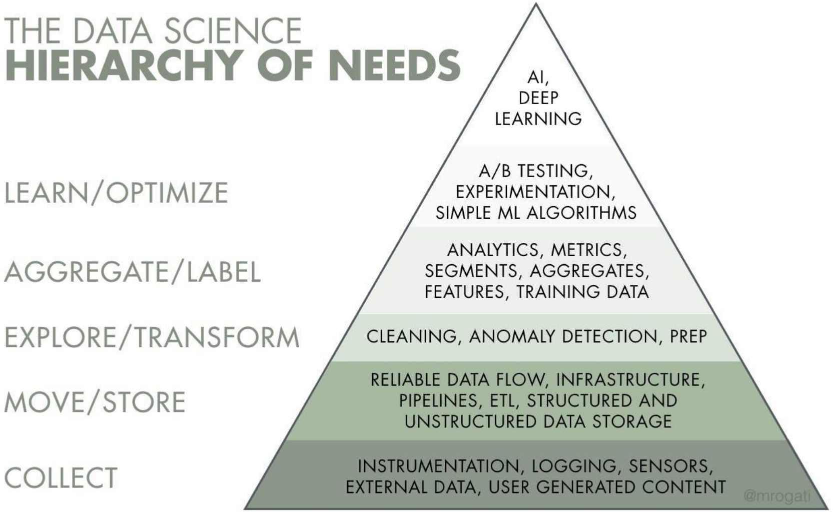
Hierarchy of Data Needs

Tools are **not the focus** of this fellowship.

The tools you use **might change** from job to job.

The concepts however, will **not change**.

The only way you can learn these concepts is **by writing your own code**.



THE DATA SCIENCE
HIERARCHY OF NEEDS

LEARN/OPTIMIZE

AGGREGATE/LABEL

EXPLORE/TRANSFORM

MOVE/STORE

COLLECT

AI, DEEP LEARNING

A/B TESTING, EXPERIMENTATION, SIMPLE ML ALGORITHMS

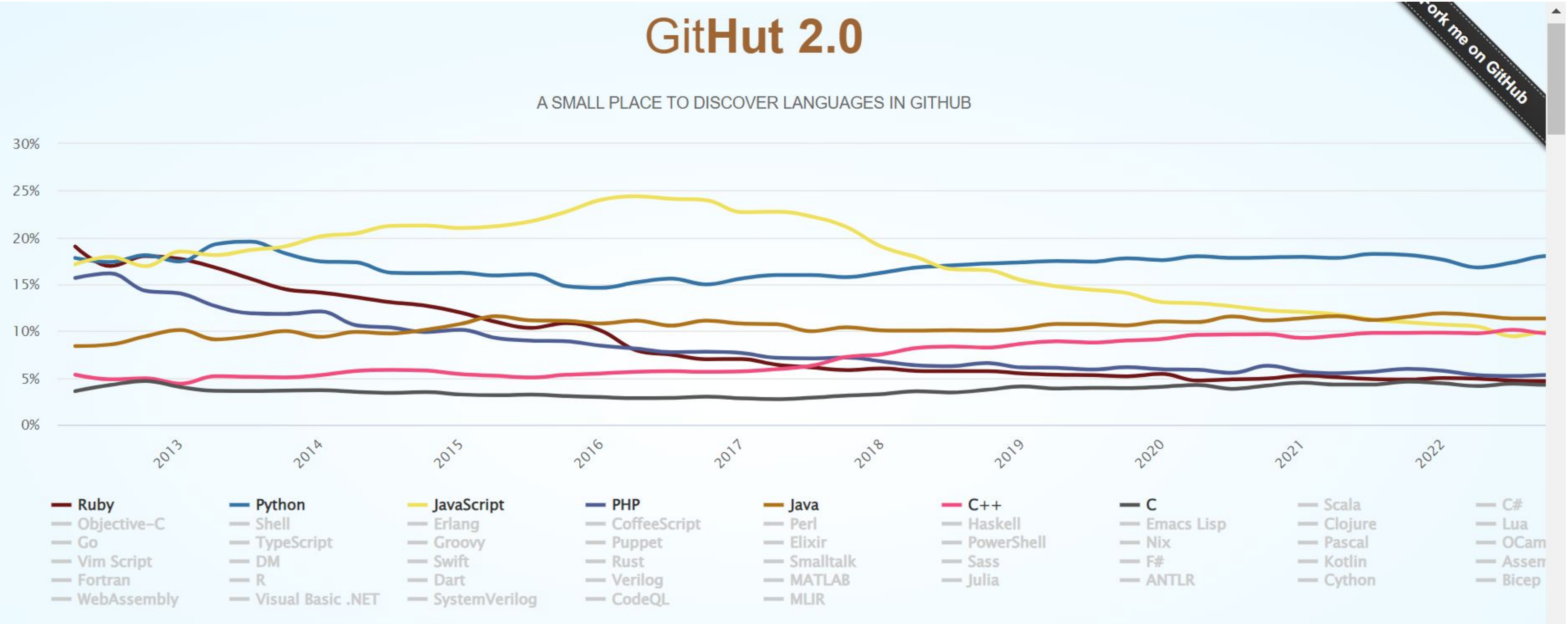
ANALYTICS, METRICS, SEGMENTS, AGGREGATES, FEATURES, TRAINING DATA

CLEANING, ANOMALY DETECTION, PREP

RELIABLE DATA FLOW, INFRASTRUCTURE, PIPELINES, ETL, STRUCTURED AND UNSTRUCTURED DATA STORAGE

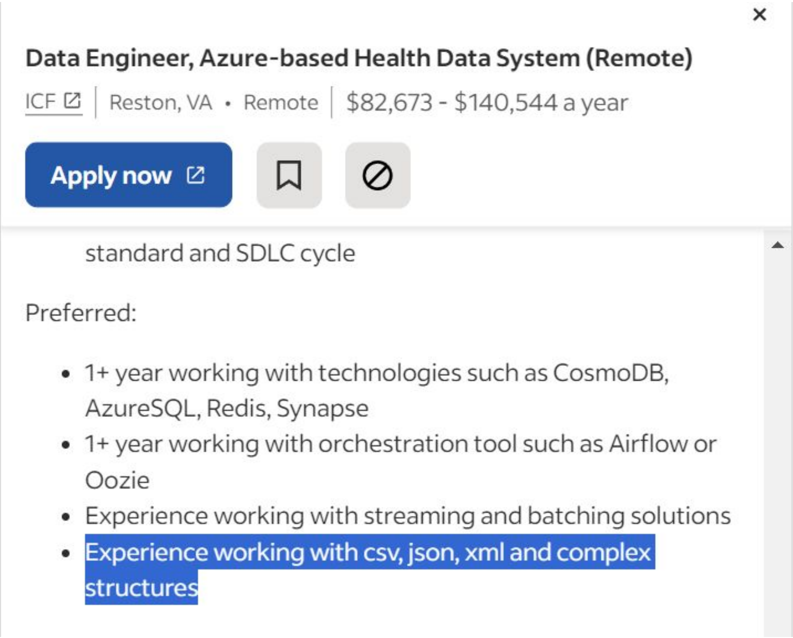
INSTRUMENTATION, LOGGING, SENSORS, EXTERNAL DATA, USER GENERATED CONTENT

@mrogati



https://madnight.github.io/githut/#/pull_requests/2023/4

This graph is a bit more applicable to web development than data science, however the idea remains the same: ***Technologies are replaceable, skills are not.***



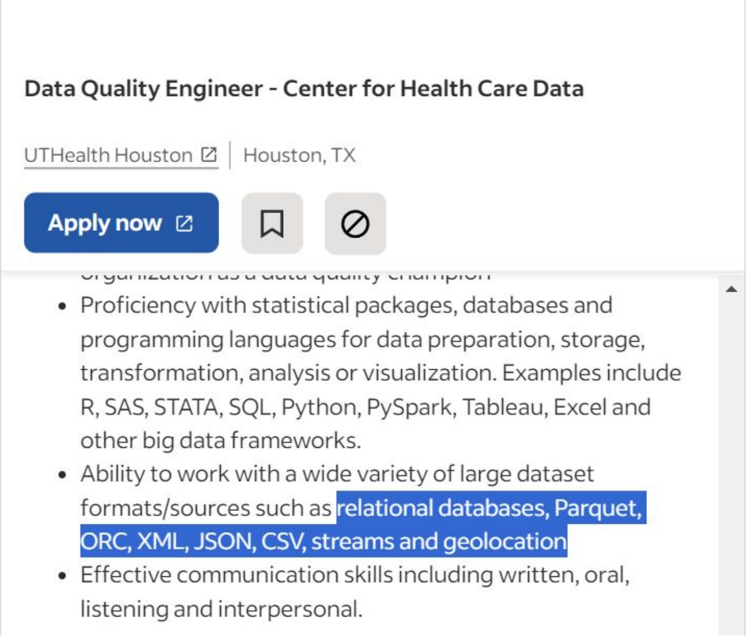
Data Engineer, Azure-based Health Data System (Remote)

ICF | Reston, VA • Remote | $82,673 - $140,544 a year

standard and SDLC cycle

Preferred:

- 1+ year working with technologies such as CosmoDB, AzureSQL, Redis, Synapse
- 1+ year working with orchestration tool such as Airflow or Oozie
- Experience working with streaming and batching solutions
- Experience working with csv, json, xml and complex structures



Data Quality Engineer - Center for Health Care Data

UTHealth Houston | Houston, TX

organization as a data quality champion

- Proficiency with statistical packages, databases and programming languages for data preparation, storage, transformation, analysis or visualization. Examples include R, SAS, STATA, SQL, Python, PySpark, Tableau, Excel and other big data frameworks.
- Ability to work with a wide variety of large dataset formats/sources such as relational databases, Parquet, ORC, XML, JSON, CSV, streams and geolocation
- Effective communication skills including written, oral, listening and interpersonal.

Data engineering as it applies to basic data formats is an **essential skill of data science** (as we see above). Let’s get to know a few of these formats.

Data Formats

Data Formats

So far, we've been **loading** in our data via the `open( )` function.

However, are **text** files the only data format?

If not, can you think of any examples of other forms of data?

Think about the **forms of data you observe when browsing the internet.**

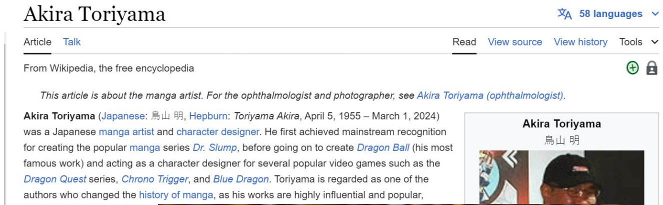
Data Formats

Data can exist in **many different forms and structures**.

This includes **websites, geo-data, and Web API’s**.

As data scientists, we should be able to **ingest and restructure** these forms of data.

Only then can we easily apply **statistical analysis** & **machine learning algorithms**.



Akira Toriyama

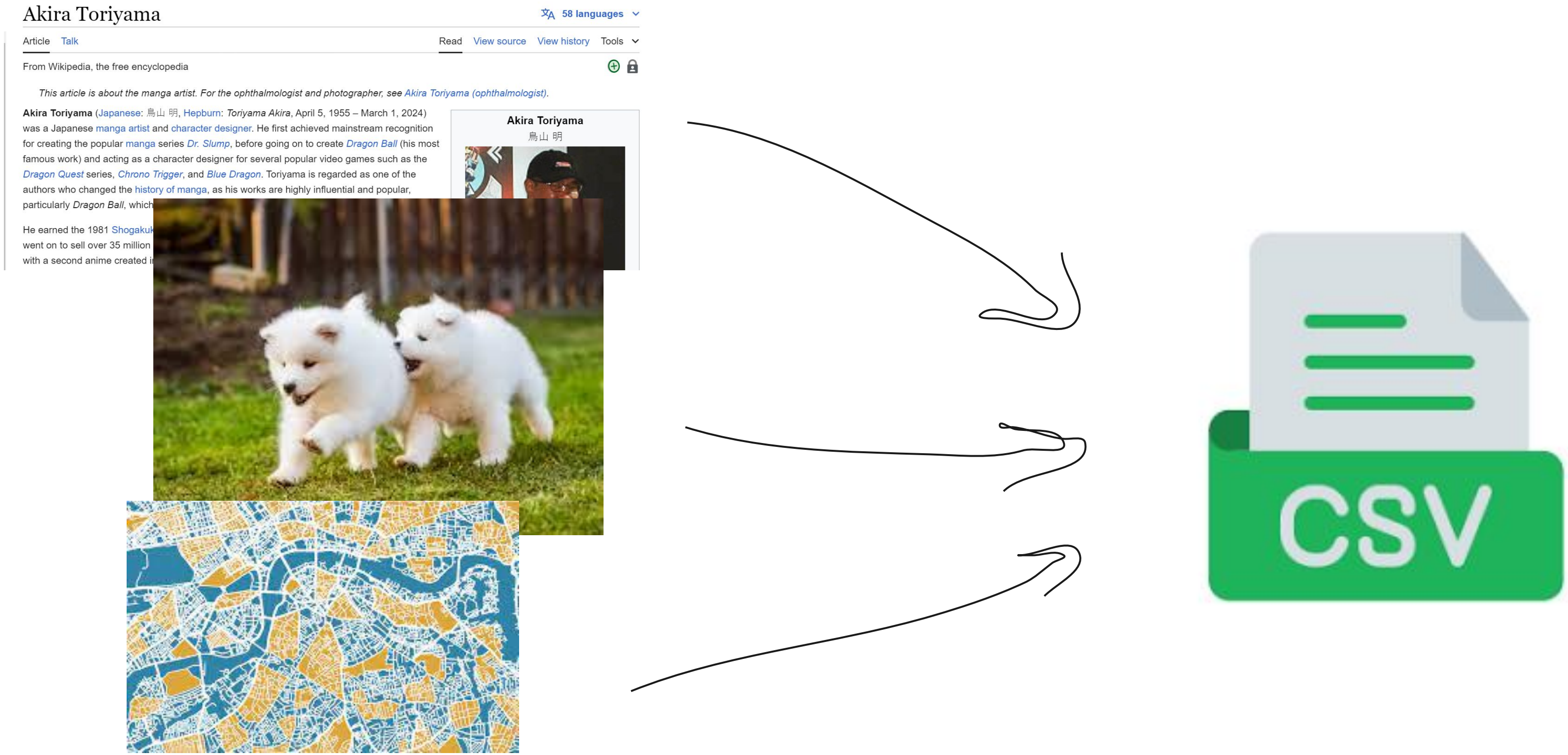
From Wikipedia, the free encyclopedia

This article is about the manga artist. For the ophthalmologist and photographer, see Akira Toriyama (ophthalmologist).

Akira Toriyama (Japanese: 鳥山 明, Hepburn: *Toriyama Akira*, April 5, 1955 – March 1, 2024) was a Japanese manga artist and character designer. He first achieved mainstream recognition for creating the popular manga series *Dr. Slump*, before going on to create *Dragon Ball* (his most famous work) and acting as a character designer for several popular video games such as the *Dragon Quest* series, *Chrono Trigger*, and *Blue Dragon*. Toriyama is regarded as one of the authors who changed the history of manga, as his works are highly influential and popular, particularly *Dragon Ball*, which

He earned the 1981 Shogakuk
went on to sell over 35 million
with a second anime created i

Akira Toriyama
鳥山 明



Akira Toriyama

From Wikipedia, the free encyclopedia

This article is about the manga artist. For the ophthalmologist and photographer, see Akira Toriyama (ophthalmologist).

Akira Toriyama (Japanese: 鳥山 明, Hepburn: *Toriyama Akira*, April 5, 1955 – March 1, 2024) was a Japanese manga artist and character designer. He first achieved mainstream recognition for creating the popular manga series *Dr. Slump*, before going on to create *Dragon Ball* (his most famous work) and acting as a character designer for several popular video games such as the *Dragon Quest* series, *Chrono Trigger*, and *Blue Dragon*. Toriyama is regarded as one of the authors who changed the history of manga, as his works are highly influential and popular, particularly *Dragon Ball*, which

He earned the 1981 Shogakuk
went on to sell over 35 million
with a second anime created i

Akira Toriyama
鳥山 明

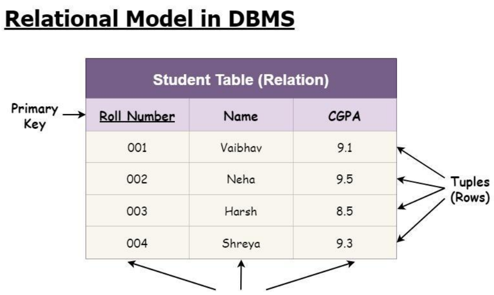
Most of our development work will entail figuring out how we express the concept of a dog into a CSV file.

Data Formats

Before learning about the different ways we can work with these data-files, let’s review some broad categories of **data-formats**

- Structured
- Semi-Structured
- Unstructured

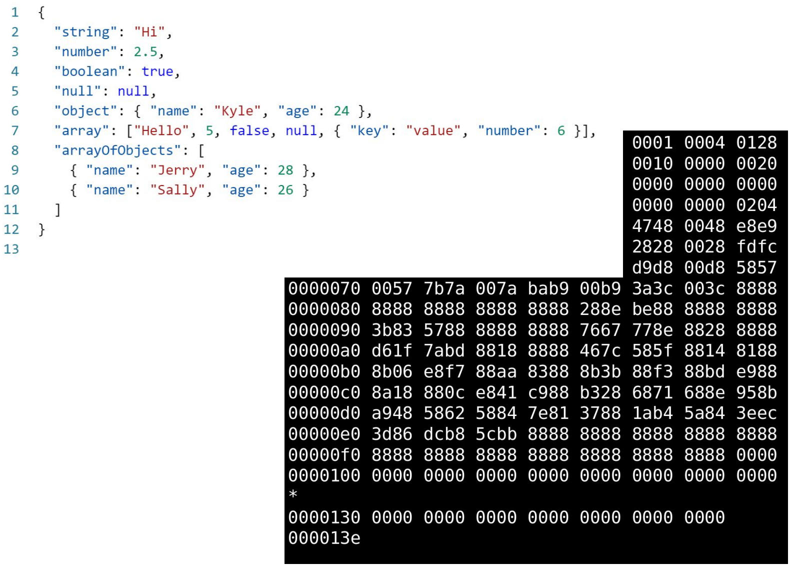
What do you think is the **difference** amongst these 3 formats?



Relational Model in DBMS

Student Table (Relation)		
Roll Number	Name	CGPA
001	Vaibhav	9.1
002	Neha	9.5
003	Harsh	8.5
004	Shreya	9.3

Primary Key
Tuples (Rows)



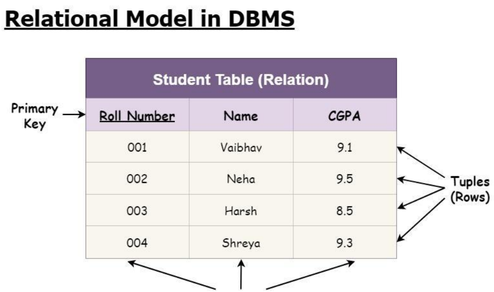
```
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}
```

```
                                         0001 0004 0128
                                         0010 0000 0020
                                         0000 0000 0000
                                         0000 0000 0204
                                         4748 0048 e8e9
                                         2828 0028 fdfc
                                         d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Data Formats

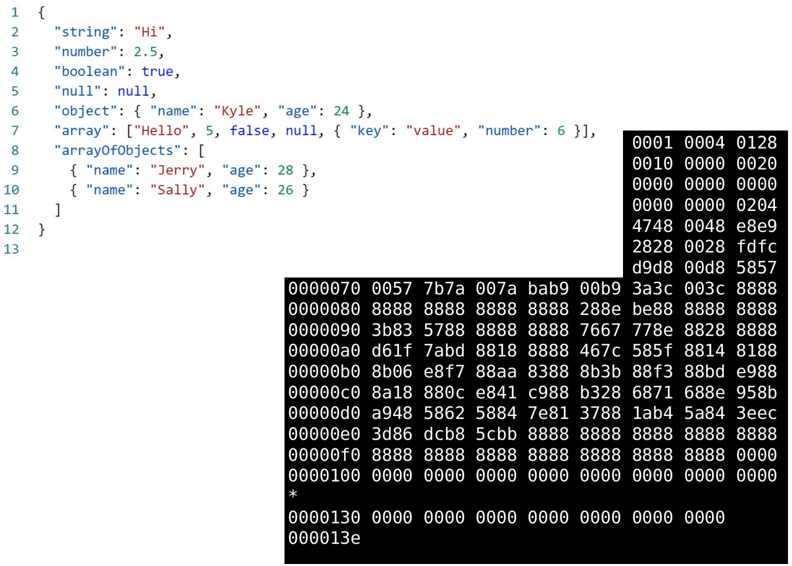
- **Structured**
  - Data adheres to a **fixed** structure
- **Semi-Structured**
  - Structure **varies**, but is **mostly consistent**
- **Unstructured**
  - Data abides by **no structure**

Can you think of any examples of these 3 formats?



Relational Model in DBMS

Student Table (Relation)		
Roll Number	Name	CGPA
001	Vaibhav	9.1
002	Neha	9.5
003	Harsh	8.5
004	Shreya	9.3



```
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}
```

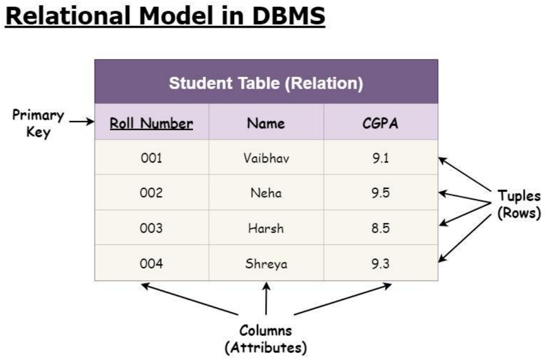
```
                                 0001 0004 0128
                                 0010 0000 0020
                                 0000 0000 0000
                                 0000 0000 0204
                                 4748 0048 e8e9
                                 2828 0028 fdfc
                                 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Structured Data

If your format is **consistent** across different files, **this format is structured**.

This is the most **convenient** format, and often what we **strive for** in our work.

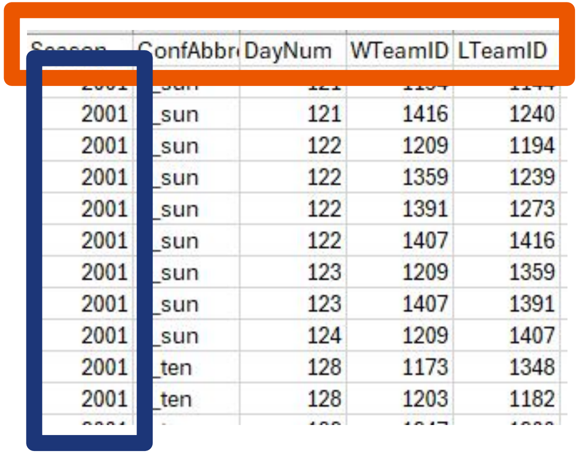
This includes **relational** databases, comma-separated-value (**CSV**) files, tab-separated-value (**TSV**) files



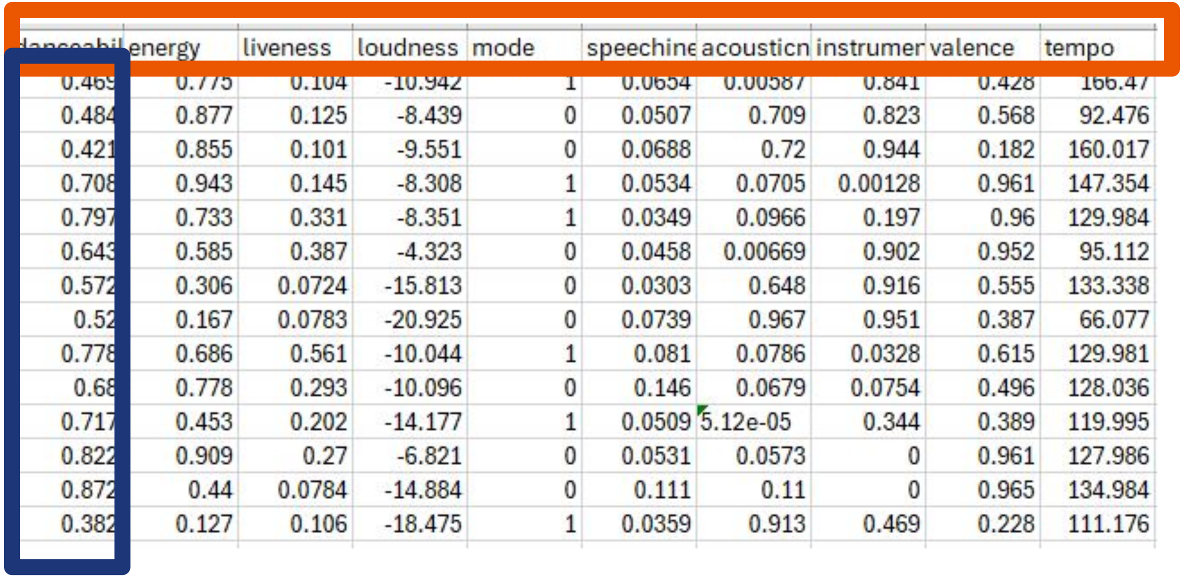
Relational Model in DBMS

Student Table (Relation)

Roll Number	Name	CGPA
001	Vaibhav	9.1
002	Neha	9.5
003	Harsh	8.5
004	Shreya	9.3



Season	ConfAbbr	DayNum	WTeamID	LTeamID
2001	_sun	121	1416	1240
2001	_sun	122	1209	1194
2001	_sun	122	1359	1239
2001	_sun	122	1391	1273
2001	_sun	122	1407	1416
2001	_sun	123	1209	1359
2001	_sun	123	1407	1391
2001	_sun	124	1209	1407
2001	_ten	128	1173	1348
2001	_ten	128	1203	1182



danceabil	energy	liveness	loudness	mode	speechine	acousticn	instrumer	valence	tempo
0.469	0.775	0.104	-10.942	1	0.0654	0.00587	0.841	0.428	166.47
0.484	0.877	0.125	-8.439	0	0.0507	0.709	0.823	0.568	92.476
0.421	0.855	0.101	-9.551	0	0.0688	0.72	0.944	0.182	160.017
0.708	0.943	0.145	-8.308	1	0.0534	0.0705	0.00128	0.961	147.354
0.797	0.733	0.331	-8.351	1	0.0349	0.0966	0.197	0.96	129.984
0.643	0.585	0.387	-4.323	0	0.0458	0.00669	0.902	0.952	95.112
0.572	0.306	0.0724	-15.813	0	0.0303	0.648	0.916	0.555	133.338
0.52	0.167	0.0783	-20.925	0	0.0739	0.967	0.951	0.387	66.077
0.778	0.686	0.561	-10.044	1	0.081	0.0786	0.0328	0.615	129.981
0.68	0.778	0.293	-10.096	0	0.146	0.0679	0.0754	0.496	128.036
0.717	0.453	0.202	-14.177	1	0.0509	5.12e-05	0.344	0.389	119.995
0.822	0.909	0.27	-6.821	0	0.0531	0.0573	0	0.961	127.986
0.872	0.44	0.0784	-14.884	0	0.111	0.11	0	0.965	134.984
0.382	0.127	0.106	-18.475	1	0.0359	0.913	0.469	0.228	111.176

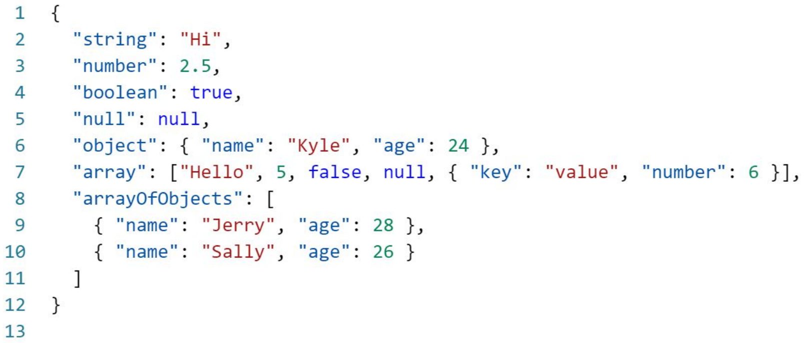
All csv files are **guaranteed** to have **structures (such as rows & columns), regardless of the data it captures.**

Semi-structured Data

If your format is **somewhat predictable**, but varies across files, this format is **semi-structured**.

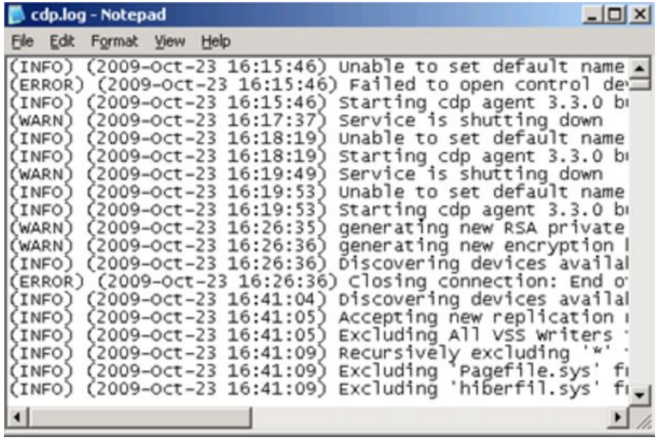
Web-data is often in this format.

This includes **JSON**, **Emails, XML**, and **log files.**



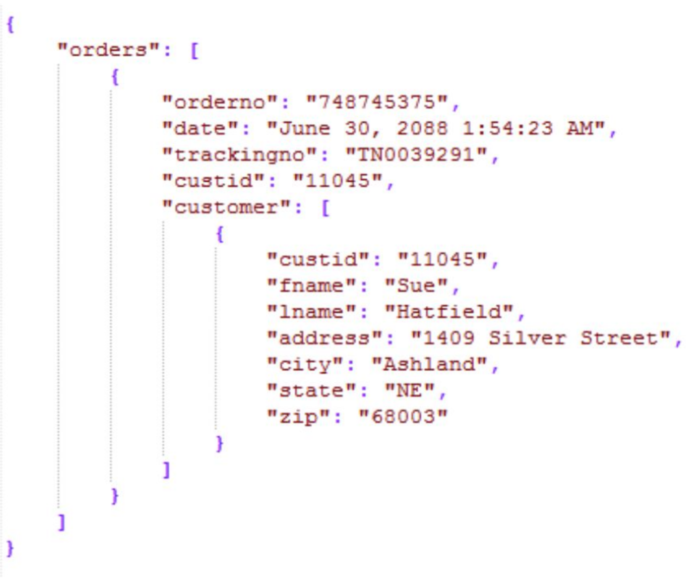
```
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}

```

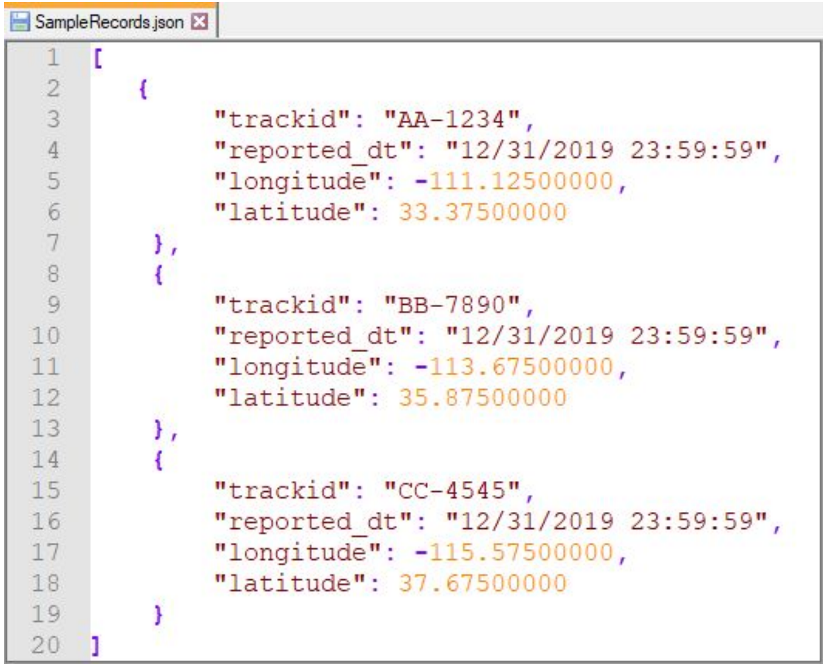


```
(INFO) (2009-Oct-23 16:15:46) Unable to set default name
(ERROR) (2009-Oct-23 16:15:46) Failed to open control de
(INFO) (2009-Oct-23 16:15:46) Starting cdp agent 3.3.0 b
(WARN) (2009-Oct-23 16:17:37) Service is shutting down
(INFO) (2009-Oct-23 16:18:19) Unable to set default name
(INFO) (2009-Oct-23 16:18:19) Starting cdp agent 3.3.0 b
(WARN) (2009-Oct-23 16:19:49) Service is shutting down
(INFO) (2009-Oct-23 16:19:53) Unable to set default name
(INFO) (2009-Oct-23 16:19:53) Starting cdp agent 3.3.0 b
(WARN) (2009-Oct-23 16:26:35) generating new RSA private
(WARN) (2009-Oct-23 16:26:36) generating new encryption
(INFO) (2009-Oct-23 16:26:36) Discovering devices availal
(ERROR) (2009-Oct-23 16:26:36) Closing connection: End o
(INFO) (2009-Oct-23 16:41:04) Discovering devices availal
(INFO) (2009-Oct-23 16:41:05) Accepting new replication
(INFO) (2009-Oct-23 16:41:05) Excluding All VSS writers
(INFO) (2009-Oct-23 16:41:09) Recursively excluding '*'
(INFO) (2009-Oct-23 16:41:09) Excluding 'Pagefile.sys' f
(INFO) (2009-Oct-23 16:41:09) Excluding 'hiberfil.sys' f
```

Notice the difference between these two files…



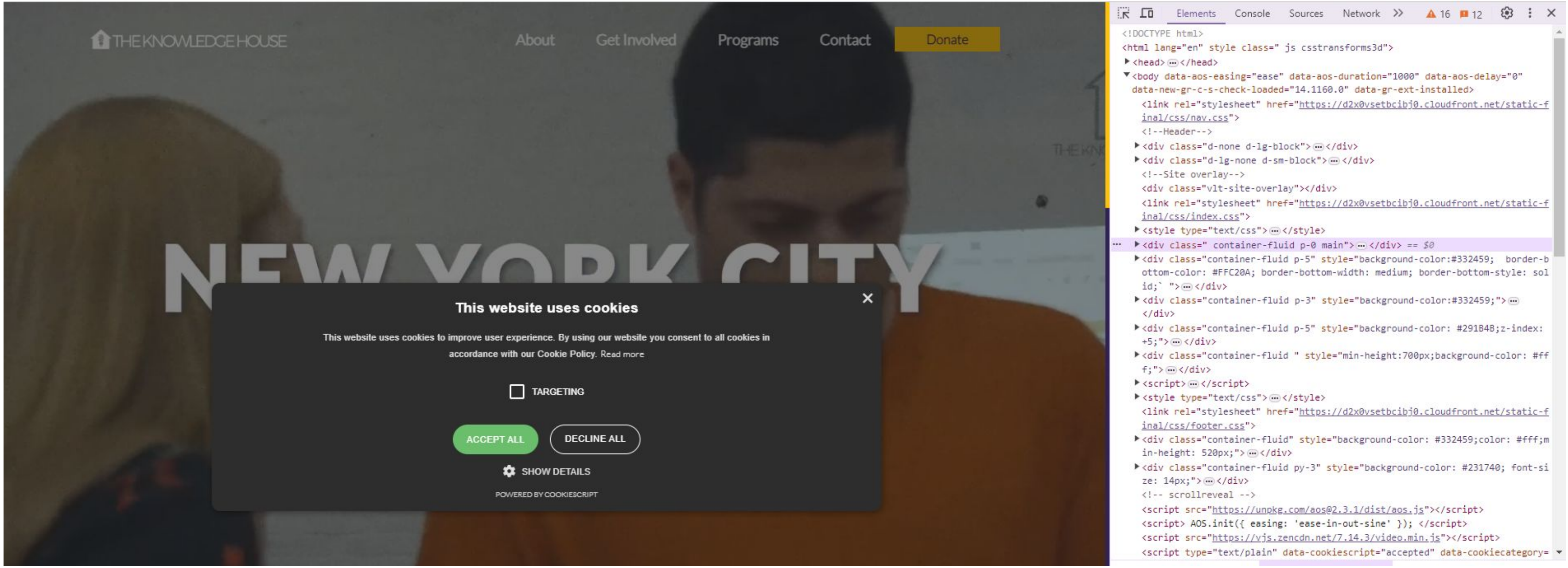
```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```



```
[
    {
        "trackid": "AA-1234",
        "reported_dt": "12/31/2019 23:59:59",
        "longitude": -111.12500000,
        "latitude": 33.37500000
     },
     {
        "trackid": "BB-7890",
        "reported_dt": "12/31/2019 23:59:59",
        "longitude": -113.67500000,
        "latitude": 35.87500000
     },
     {
        "trackid": "CC-4545",
        "reported_dt": "12/31/2019 23:59:59",
        "longitude": -115.57500000,
        "latitude": 37.67500000
     }
]
```

JSON files have a **predefined set of objects**, but there is no guarantee that all these objects **will appear in every file** or if they will appear **in the same order.**

Websites are just a stylized form of data!



This website uses cookies

This website uses cookies to improve user experience. By using our website you consent to all cookies in accordance with our Cookie Policy. Read more

□ TARGETING

ACCEPT ALL DECLINE ALL

SHOW DETAILS

POWERED BY COOKIESCRIPT

```
<!DOCTYPE html>
<html lang="en" style class=" js csstransforms3d">
 ▶<head>…</head>
 ▼<body data-aos-easing="ease" data-aos-duration="1000" data-aos-delay="0"
  data-new-gr-c-s-check-loaded="14.1160.0" data-gr-ext-installed>
    <link rel="stylesheet" href="https://d2x0vsetbcibj0.cloudfront.net/static-f
    inal/css/nav.css">
    <!--Header-->
   ▶<div class="d-none d-lg-block">…</div>
   ▶<div class="d-lg-none d-sm-block">…</div>
    <!--Site overlay-->
    <div class="vlt-site-overlay"></div>
    <link rel="stylesheet" href="https://d2x0vsetbcibj0.cloudfront.net/static-f
    inal/css/index.css">
   ▶<style type="text/css">…</style>
…  ▶<div class=" container-fluid p-0 main">…</div> == $0
   ▶<div class="container-fluid p-5" style="background-color:#332459;  border-b
    ottom-color: #FFC20A; border-bottom-width: medium; border-bottom-style: sol
    id;` ">…</div>
   ▶<div class="container-fluid p-3" style="background-color:#332459;">…
    </div>
   ▶<div class="container-fluid p-5" style="background-color: #291B4B;z-index:
    +5;">…</div>
   ▶<div class="container-fluid " style="min-height:700px;background-color: #ff
    f;">…</div>
   ▶<script>…</script>
   ▶<style type="text/css">…</style>
    <link rel="stylesheet" href="https://d2x0vsetbcibj0.cloudfront.net/static-f
    inal/css/footer.css">
   ▶<div class="container-fluid" style="background-color: #332459;color: #fff;m
    in-height: 520px;">…</div>
   ▶<div class="container-fluid py-3" style="background-color: #231740; font-si
    ze: 14px;">…</div>
    <!-- scrollreveal -->
    <script src="https://unpkg.com/aos@2.3.1/dist/aos.js"></script>
    <script> AOS.init({ easing: 'ease-in-out-sine' }); </script>
    <script src="https://vjs.zencdn.net/7.14.3/video.min.js"></script>
    <script type="text/plain" data-cookiescript="accepted" data-cookiecategory=
```

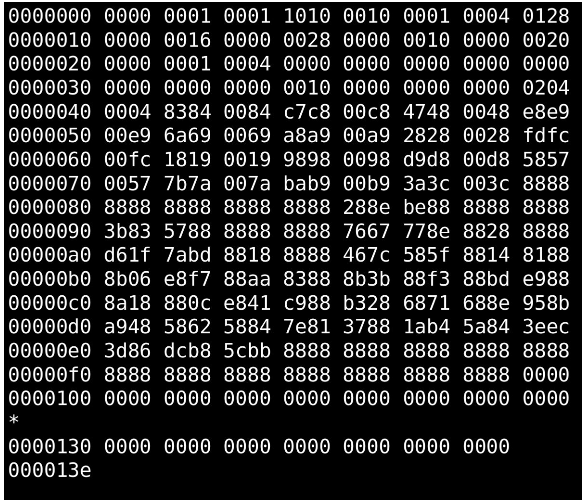
You might also be surprised to know that **websites** are a form of **semi-structured data**.

Unstructured Data

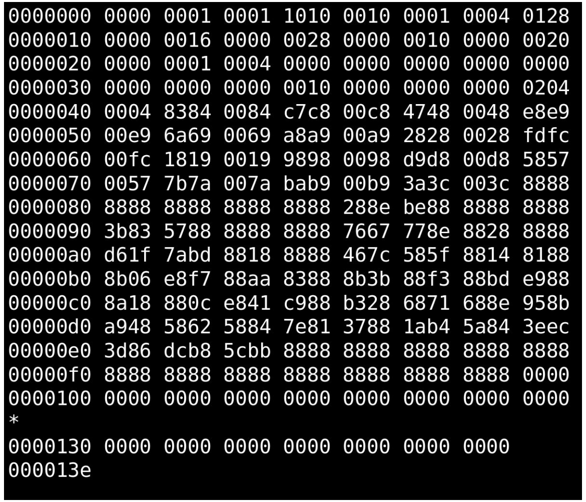
Lastly, if your format is **completely unpredictable**, and is expressed in **binary**, this format is **unstructured**.

Complex objects are often in this format.

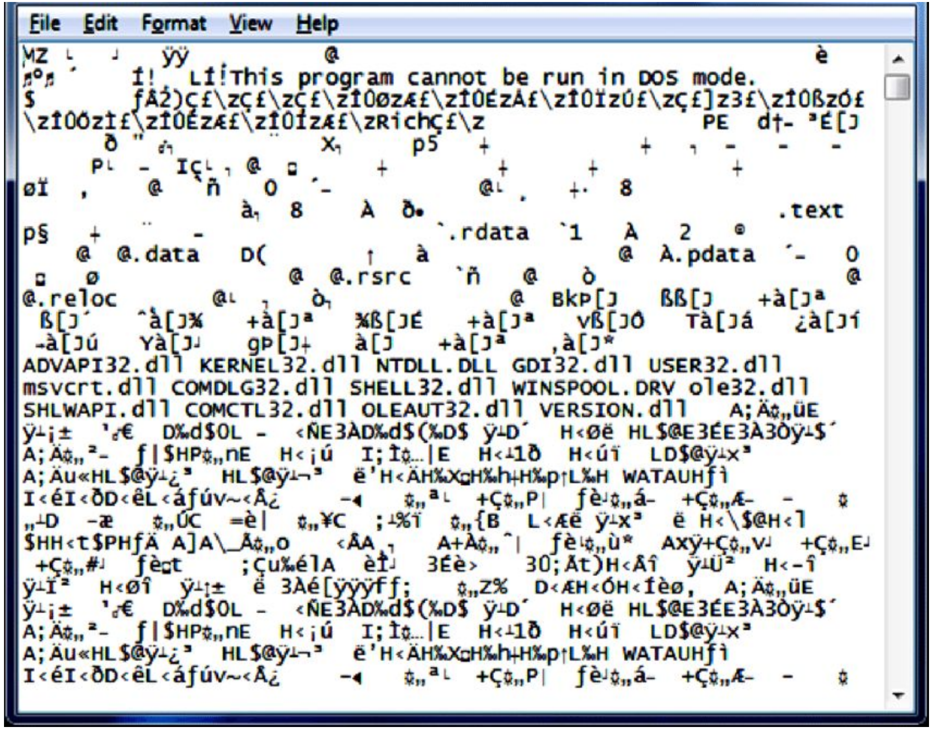
This includes **images, videos, BLOB files, etc**



```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

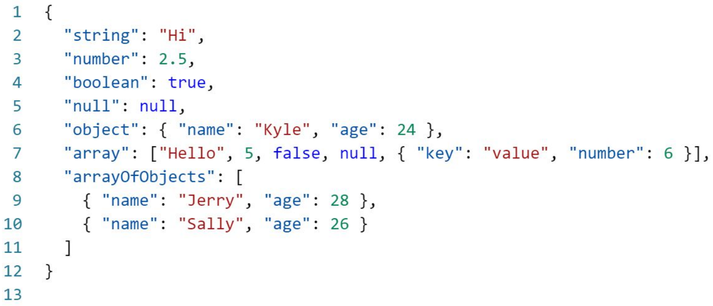


```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

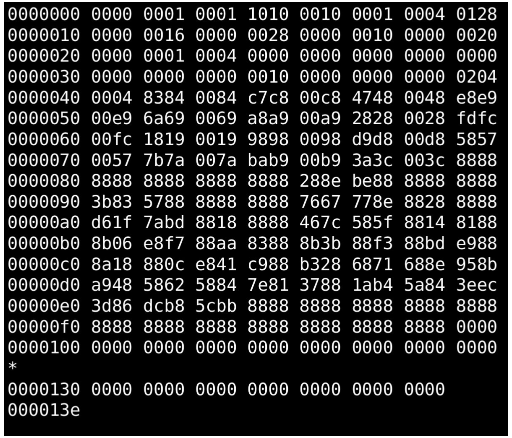


```
MZ └   ┘   ÿÿ          @                                    è
♬°♬ ´     ↨!   L↨!This program cannot be run in DOS mode.
$         fÅ2)Ç£\zÇ£\zÇ£\z↨0øzÆ£\z↨0ÉzÅ£\z↨0ÏzÚ£\zÇ£]z3£\z↨0ßzÓ£
\z↨0Òz↨£\z↨0ÉzÆ£\z↨0↨zÆ£\zRichÇ£\z                       PE  d†- ³É[J
      ð "                X┐      p5   +          +  ┐  -   -   -
     P└  -  IÇ└ ┐ @  ◘         +          +       +          +
øÏ   ,      @   ñ    0  ´-               @└  .   +·  8
                 à┐  8    À  ð•                                .text
p§   +                          `.rdata  `1    À   2   ©
    @  @.data   D(          ↑   à                 @  À.pdata  ´-   0
 ◘   ø                   @  @.rsrc  `ñ    @    ò                      @
@.reloc        @└  ┐   ò┐                  @  BkP[J   ßß[J   +à[Jª
 ß[J´    ˆà[J%   +à[Jª   %ß[JÉ    +à[Jª   vß[JÔ   Tà[Já    ¿à[Jí
 ¬à[Jú   Yà[J┘   gP[J┼    à[J    +à[Jª    ,à[J*
ADVAPI32.dll KERNEL32.dll NTDLL.DLL GDI32.dll USER32.dll
msvcrt.dll COMDLG32.dll SHELL32.dll WINSPOOL.DRV ole32.dll
SHLWAPI.dll COMCTL32.dll OLEAUT32.dll VERSION.dll    A;Ä¢„üE
ÿ┴¡±  '┌€  D%d$0L -  <ÑE3ÀD%d$(%D$ ÿ┴D´  H<Øë HL$@E3ÉE3À3Òÿ┴$´
A;Ä¢„²-  ƒ|$HP¢„nE   H<¡ú   I;Ì¢…|E   H<┴1ð   H<úï   LD$@ÿ┴x³
A;Äu«HL$@ÿ┴¿³   HL$@ÿ┴¬³   ë'H<ÄH%X◘H%h┼H%p↑L%H WATAUHƒì
I<éI<ðD<êL<áƒúv~<Å¿       -◄    ¢„ª└  +Ç¢„P|   ƒè┘¢„á-  +Ç¢„Æ-   -     ¢
„┴D   -æ    ¢„ÚC   =è|   ¢„¥C   ;┴%ï   ¢„{B   L<Æë ÿ┴x³   ë H<\$@H<]
$HH<t$PHƒÄ A]A\_Ä¢„o      <ÅA ┐    A+À¢„ˆ|   ƒè┘¢„ù*   Axÿ+Ç¢„V┘   +Ç¢„E┘
 +Ç¢„#┘   ƒè◘t       ;Çu%élA   è↨┘   3Éè>     30;Ät)H<Åî   ÿ┴Ü²   H<-î
ÿ┴Ï²   H<Øî   ÿ┴↕±   ë 3Àé[ÿÿÿff;     ¢„Z%   D<ÆH<ÓH<↨èø,   A;Ä¢„üE
ÿ┴¡±  '┌€  D%d$0L -  <ÑE3ÀD%d$(%D$ ÿ┴D´  H<Øë HL$@E3ÉE3À3Òÿ┴$´
A;Ä¢„²-  ƒ|$HP¢„nE   H<¡ú   I;Ì¢…|E   H<┴1ð   H<úï   LD$@ÿ┴x³
A;Äu«HL$@ÿ┴¿³   HL$@ÿ┴¬³   ë'H<ÄH%X◘H%h┼H%p↑L%H WATAUHƒì
I<éI<ðD<êL<áƒúv~<Å¿       -◄    ¢„ª└  +Ç¢„P|   ƒè┘¢„á-  +Ç¢„Æ-   -     ¢
```

Binary data that expresses some complex object is usually specific to some application. This data is **usually not designed to interface with humans!**



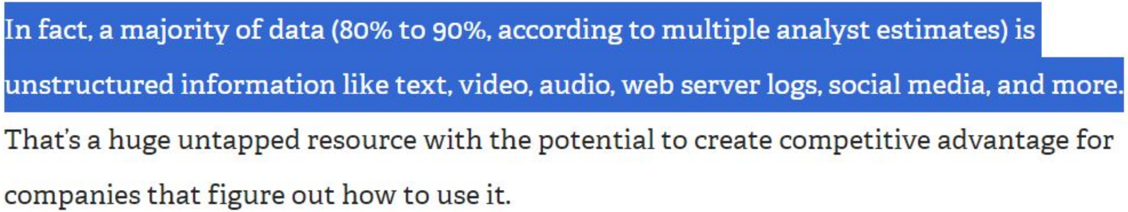
```
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}
```



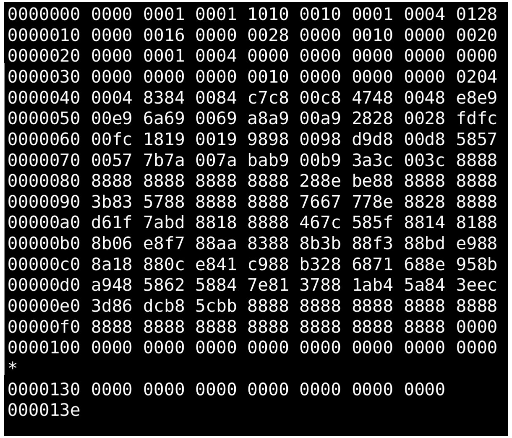
```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

You might be thinking: *“why do we need to bother figuring out how to interact with other forms of data if we can only do stats on structured data-files.”*

Guess what is the most common form of data format in the workplace: **structured**, **semi-structured**, or **unstructured** data?



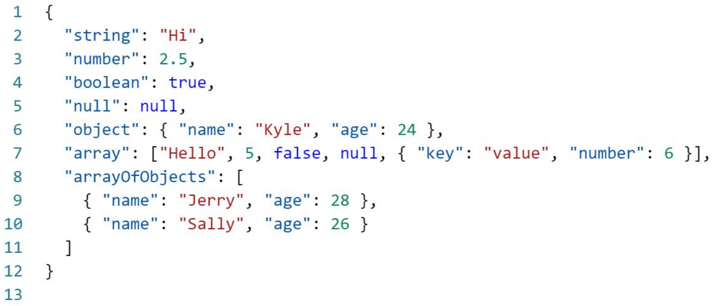
In fact, a majority of data (80% to 90%, according to multiple analyst estimates) is unstructured information like text, video, audio, web server logs, social media, and more. That's a huge untapped resource with the potential to create competitive advantage for companies that figure out how to use it.



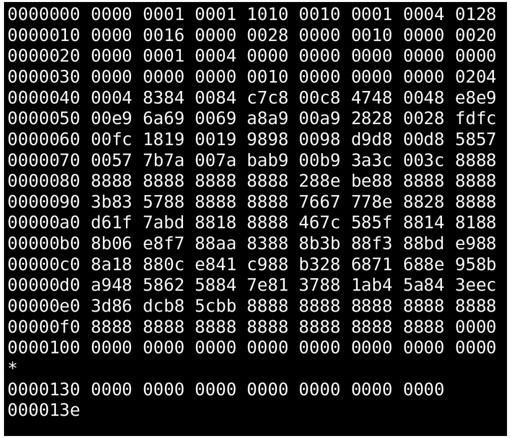
```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

The majority of data in the world is **unstructured**: https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data

Limiting yourself to CSV files **limits your ability to apply data science concepts**.



```json
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}
```



```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

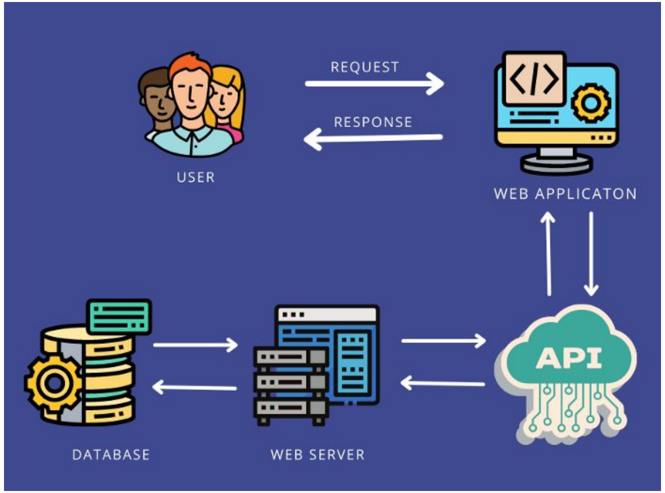
You should be able to operate (and learn how to operate) in multiple mediums

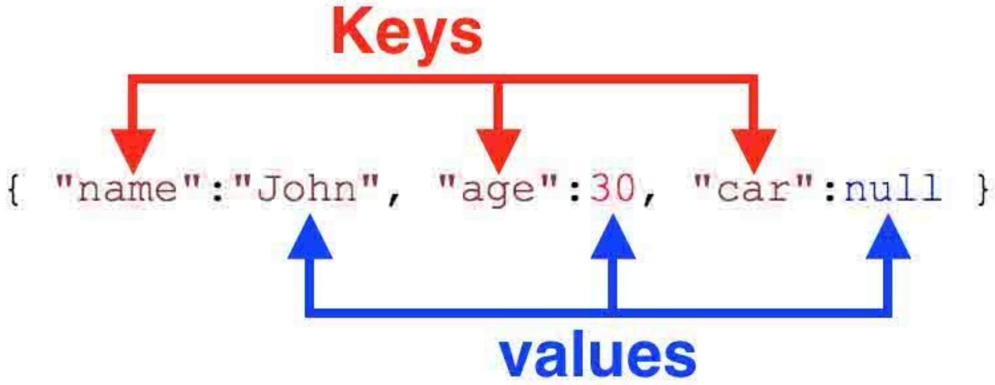
JSON

JSON Files

One popular data format that we often use when **requesting data over the web are JSON files**.

The **JavaScript-Object-Notation** file is a way to programmatically express information that can be easily interpreted by computers & humans.

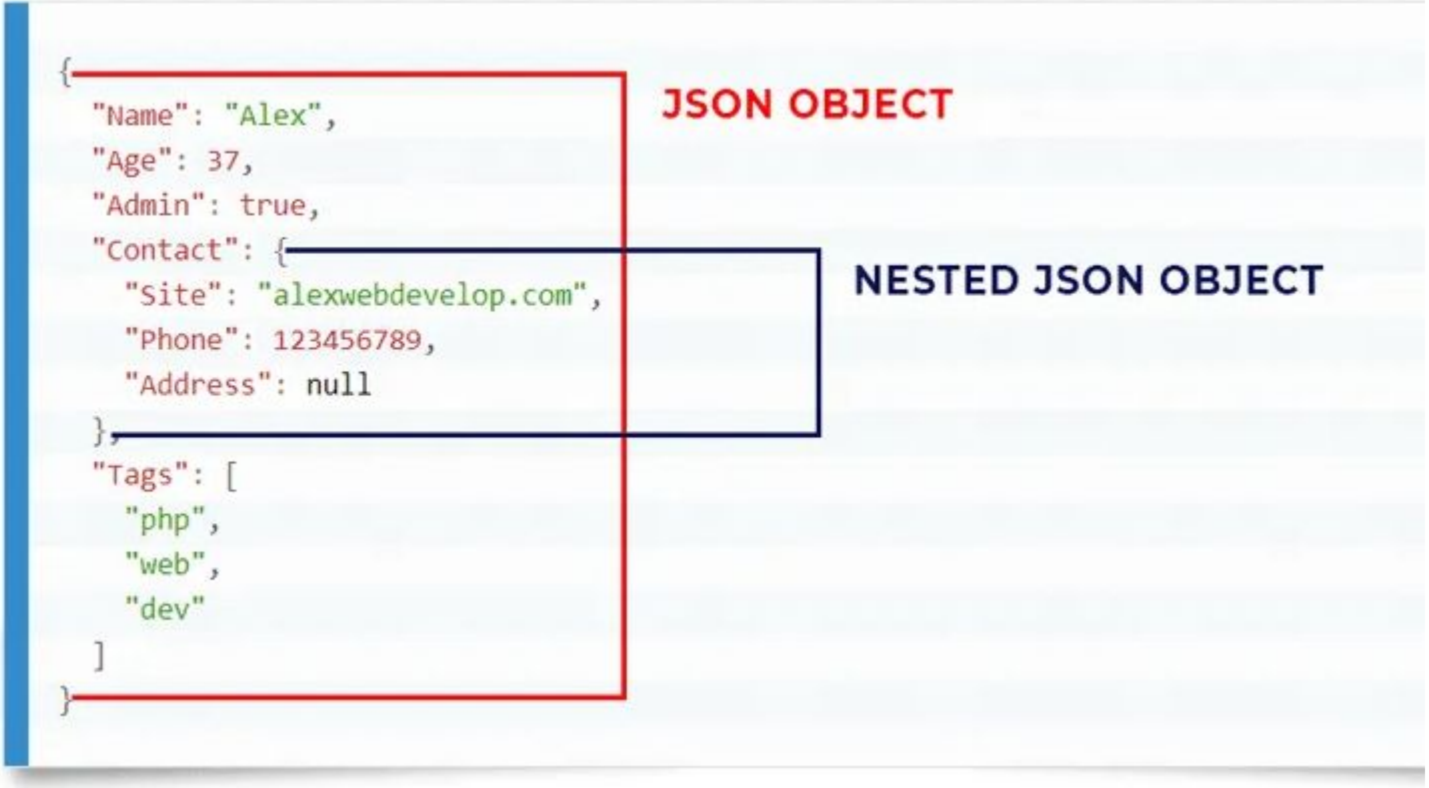




```
{ "name":"John", "age":30, "car":null }
```

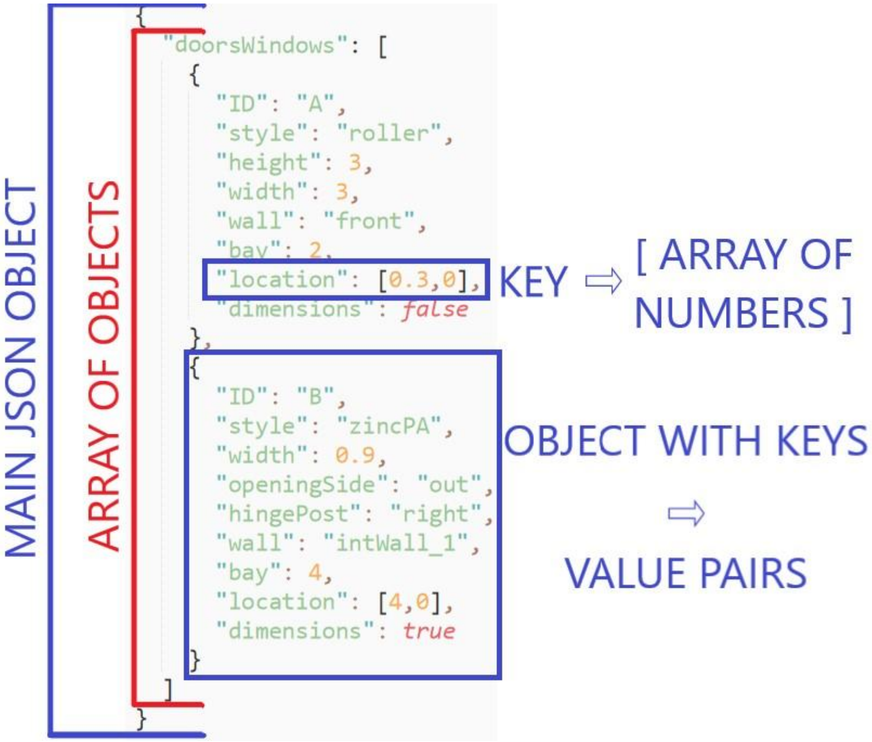
In the most basic form, JSON files are pairings of keys and values.

Which Python data-structure does this look like?



```
{
  "Name": "Alex",
  "Age": 37,
  "Admin": true,
  "Contact": {
    "Site": "alexwebdevelop.com",
    "Phone": 123456789,
    "Address": null
  },
  "Tags": [
    "php",
    "web",
    "dev"
  ]
}
```

JSON files are composed of **JSON objects** which are denoted via the curly brackets “**{}”. Note that objects can exist inside of keys!**



```
{
  "doorsWindows": [
    {
      "ID": "A",
      "style": "roller",
      "height": 3,
      "width": 3,
      "wall": "front",
      "bay": 2,
      "location": [0.3,0],
      "dimensions": false
    },
    {
      "ID": "B",
      "style": "zincPA",
      "width": 0.9,
      "openingSide": "out",
      "hingePost": "right",
      "wall": "intWall_1",
      "bay": 4,
      "location": [4,0],
      "dimensions": true
    }
  ]
}
```

As we've mentioned already, JSON objects are semi-structured. While there is some predictability regarding the file, structure can vary greatly.

Notice that we can also have arrays in our JSON files as well.

```
import json

with open('spotify-api.json') as j:
    d = json.load(j)

print(d)
```

Context manager.

Use this to replace:

```
j = open(“...”)
j.close()
```

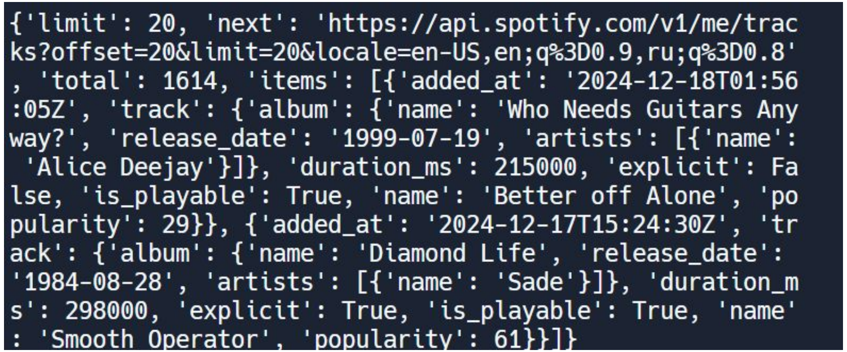
The easy part is loading in the file. Notice that this syntax is largely the same as your File I/O syntax.

We have an added structure called the “context manager”, however this does nothing new. Think of this as a convenient way to open your files without **needing to explicitly close the file after opening**.

```
import json

with open('spotify-api.json') as j:
   d = json.load(j)

print(d)
```



```
{'limit': 20, 'next': 'https://api.spotify.com/v1/me/trac
ks?offset=20&limit=20&locale=en-US,en;q%3D0.9,ru;q%3D0.8'
, 'total': 1614, 'items': [{'added_at': '2024-12-18T01:56
:05Z', 'track': {'album': {'name': 'Who Needs Guitars Any
way?', 'release_date': '1999-07-19', 'artists': [{'name':
 'Alice Deejay'}]}, 'duration_ms': 215000, 'explicit': Fa
lse, 'is_playable': True, 'name': 'Better off Alone', 'po
pularity': 29}}, {'added_at': '2024-12-17T15:24:30Z', 'tr
ack': {'album': {'name': 'Diamond Life', 'release_date':
'1984-08-28', 'artists': [{'name': 'Sade'}]}, 'duration_m
s': 298000, 'explicit': True, 'is_playable': True, 'name'
: 'Smooth Operator', 'popularity': 61}}]}
```

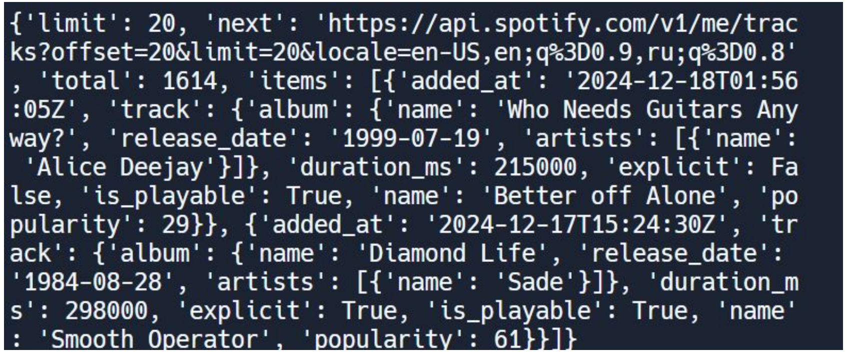
Once we load in our JSON object, this is simply expressed as a Python data-structure.

Look at the **outermost brackets**, which data structure does this appear to be?

```
import json

with open('spotify-api.json') as j:
  d = json.load(j)

print(d)
```



```
{'limit': 20, 'next': 'https://api.spotify.com/v1/me/trac
ks?offset=20&limit=20&locale=en-US,en;q%3D0.9,ru;q%3D0.8'
, 'total': 1614, 'items': [{'added_at': '2024-12-18T01:56
:05Z', 'track': {'album': {'name': 'Who Needs Guitars Any
way?', 'release_date': '1999-07-19', 'artists': [{'name':
 'Alice Deejay'}]}, 'duration_ms': 215000, 'explicit': Fa
lse, 'is_playable': True, 'name': 'Better off Alone', 'po
pularity': 29}}, {'added_at': '2024-12-17T15:24:30Z', 'tr
ack': {'album': {'name': 'Diamond Life', 'release_date':
'1984-08-28', 'artists': [{'name': 'Sade'}]}, 'duration_m
s': 298000, 'explicit': True, 'is_playable': True, 'name'
: 'Smooth Operator', 'popularity': 61}}]}
```

A **dictionary**!

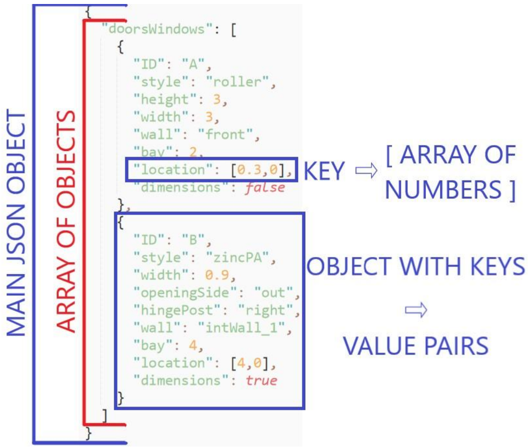
Accessing JSONs as dictionaries could prove challenging as they are usually multi-dimensional.

And since we can express this JSON object as a dictionary, we can also **access** it like a dictionary as well (*using named keys*). **Let’s practice doing this in the next few slides.**

JSON Files

However, it’s also important to note that JSON objects have slight differences from Python objects:

- No comments allowed
- No single-quoted strings
- No trailing commas



```
{
  "doorsWindows": [
    {
      "ID": "A",
      "style": "roller",
      "height": 3,
      "width": 3,
      "wall": "front",
      "bay": 2,
      "location": [0.3,0],
      "dimensions": false
    },
    {
      "ID": "B",
      "style": "zincPA",
      "width": 0.9,
      "openingSide": "out",
      "hingePost": "right",
      "wall": "intWall_1",
      "bay": 4,
      "location": [4,0],
      "dimensions": true
    }
  ]
}
```

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                 "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                 "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                 "id": "7G1GBhoKtEPnP86X2PvEYO",
                 "name": "Nina Simone",
                 "type": "artist",
                 "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
        ],
        "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
        }
    }
}
```

songs[...]

Let's say we have the following JSON object loaded in as a Python dictionary (in a variable called **songs**). How can we access the value of added_at?

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
          "album_type": "album",
          "artists": [
                   {
                    "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                    "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                    "id": "7G1GBhoKtEPnP86X2PvEYO",
                    "name": "Nina Simone",
                    "type": "artist",
                    "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                   }
       ],
       "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
       }
   }
}
```

songs[“*added_at*”]

We simply utilize the name of the key itself inside of our square brackets.
This gives us a value of **"2024-11-19T02:31:08Z"**

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                 "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                 "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                 "id": "7G1GBhoKtEPnP86X2PvEYO",
                 "name": "Nina Simone",
                 "type": "artist",
                 "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
     ],
     "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
     }
   }
}
```

songs[...]

What if we want to access the value of album_type? Can we simply use one key name to access this value, or do we need to write additional syntax?

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
        "album": {
                "album_type": "album",
                "artists": [
                        {
                         "external_urls": { "spotify":
                     "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                         "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                         "id": "7G1GBhoKtEPnP86X2PvEYO",
                         "name": "Nina Simone",
                         "type": "artist",
                         "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                        }
              ],
              "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
              }
        }
}
```

songs[...]

First off, let’s note, does the album_type value simply exist in the top-most key-value pairs of this dictionary, is it **nested**?

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                 "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                 "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                 "id": "7G1GBhoKtEPnP86X2PvEYO",
                 "name": "Nina Simone",
                 "type": "artist",
                 "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
     ],
     "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
     }
   }
}
```

songs["*track*"]

Notice that this value exists inside of a dictionary that is inside of our dictionary. This means that we must use multiple key accesses to get to this **nested value**!

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                  "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                  "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                  "id": "7G1GBhoKtEPnP86X2PvEYO",
                  "name": "Nina Simone",
                  "type": "artist",
                  "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
       ],
       "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
       }
    }
}
```

songs[“*track*”][“*album*”]

Next, we need to access the value of “album” which gives us the **nested dictionary.** However, how can we need to dive deeper. How can we access the keys of the album dictionary?

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
       "album_type": "album",
       "artists": [
               {
                "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                "id": "7G1GBhoKtEPnP86X2PvEYO",
                "name": "Nina Simone",
                "type": "artist",
                "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
               }
 ],
 "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
 }
   }
}
```

songs[“*track*”][“*album*”][...]

We can add another level of bracket notation to access the keys of the album dictionary. Which additional key should we use in order to get the value of the “album_type” key?

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                  "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                  "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                  "id": "7G1GBhoKtEPnP86X2PvEYO",
                  "name": "Nina Simone",
                  "type": "artist",
                  "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
        ],
        "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
        }
    }
}
```

songs[“*track*”][“*album*”][“*album_type*”]

Consider the workflow we just took:

- Note the **top-most** object we need to access
- Note the **next layers** we need to **access** to eventually get to our value (this might be more than 1!)

By specifying ***album_type*** in the next key level, we can now finally extract **“album”**

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                 "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                 "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                 "id": "7G1GBhoKtEPnP86X2PvEYO",
                 "name": "Nina Simone",
                 "type": "artist",
                 "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
    ],
    "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
    }
  }
}
```

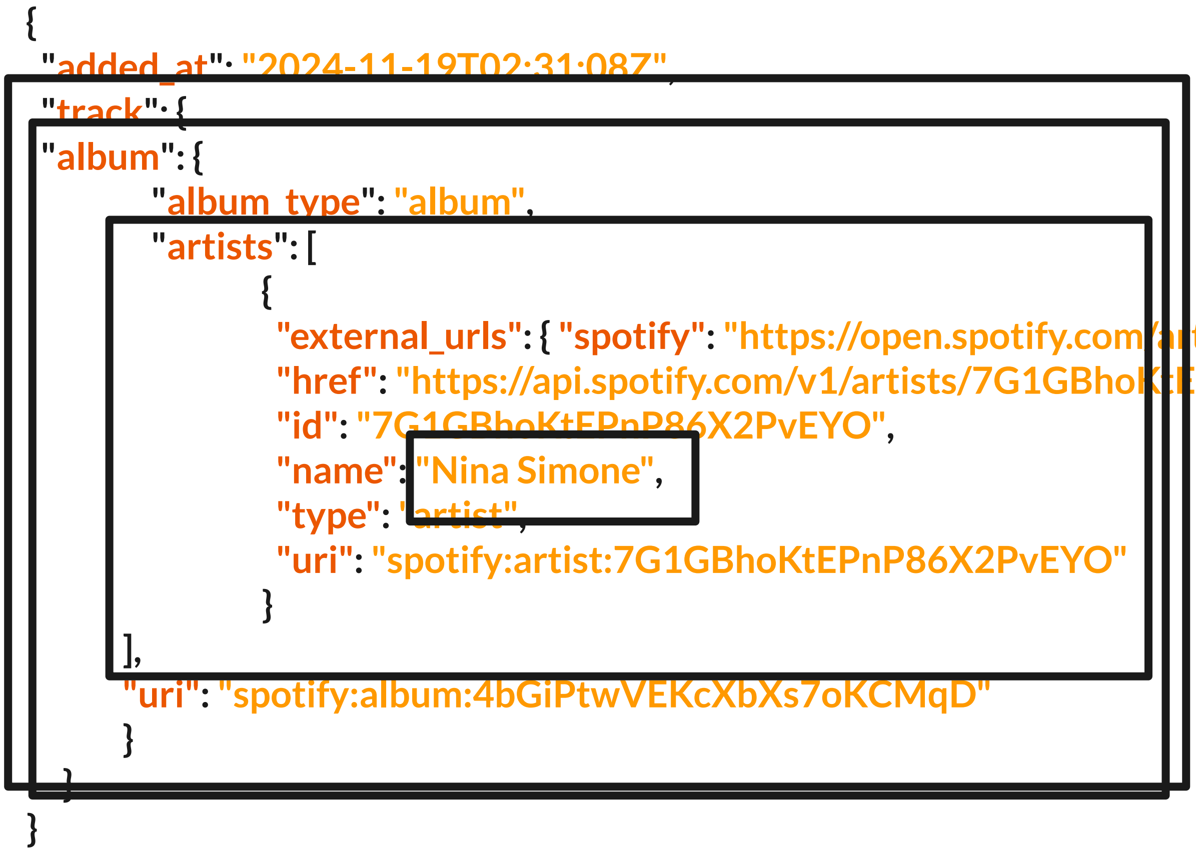
songs[...]

Let's see if we can work together to get the name value from this dictionary. First off, what is the top-most object we need to access to start this process?

```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
          "album_type": "album",
          "artists": [
                    {
                     "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                     "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                     "id": "7G1GBhoKtEPnP86X2PvEYO",
                     "name": "Nina Simone",
                     "type": "artist",
                     "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                    }
       ],
       "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
       }
  }
}
```

songs[“*track*”][“*album*”][...]

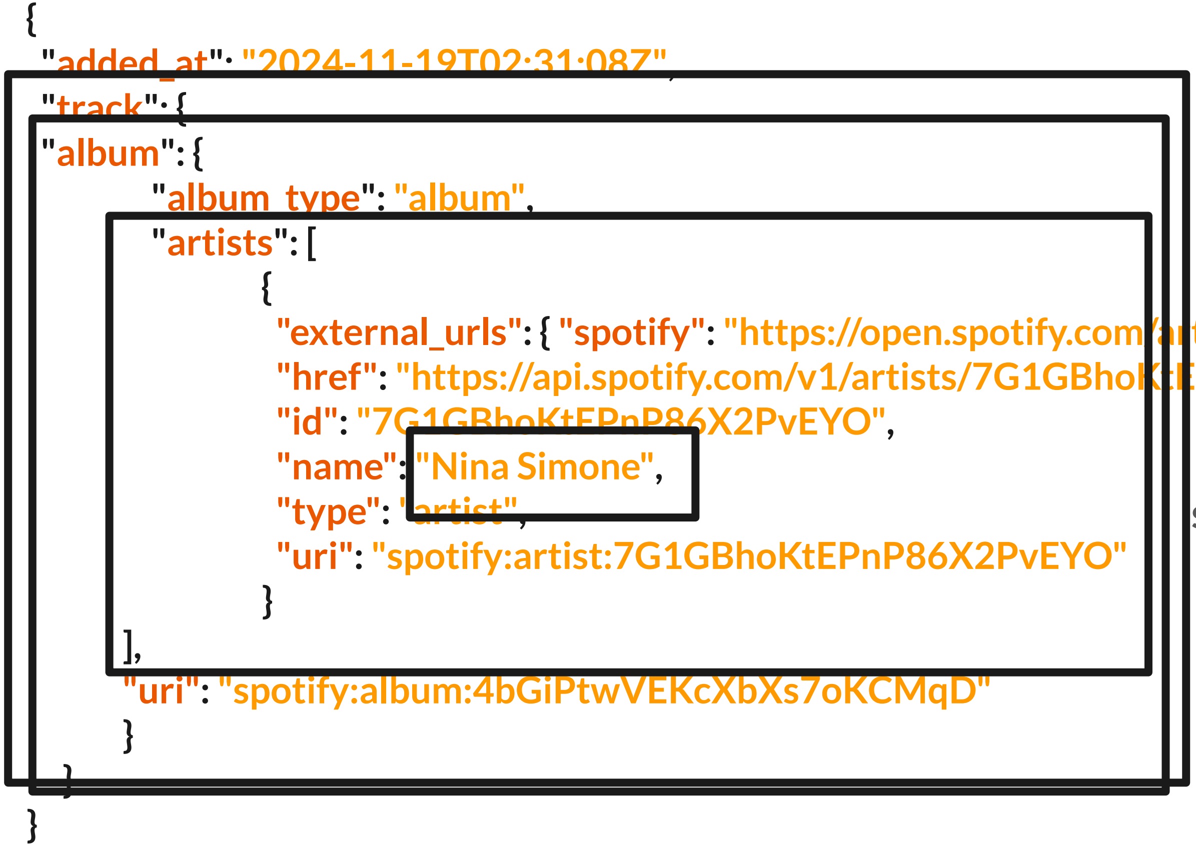
Just like before, we need to first access the ***track*** object and then ***album***.
What is the **next** object that contains our name value?



```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                 "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                 "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                 "id": "7G1GBhoKtEPnP86X2PvEYO",
                 "name": "Nina Simone",
                 "type": "artist",
                 "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
        ],
        "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
        }
    }
}
```

songs[“*track*”][“*album*”][“*artists*”]

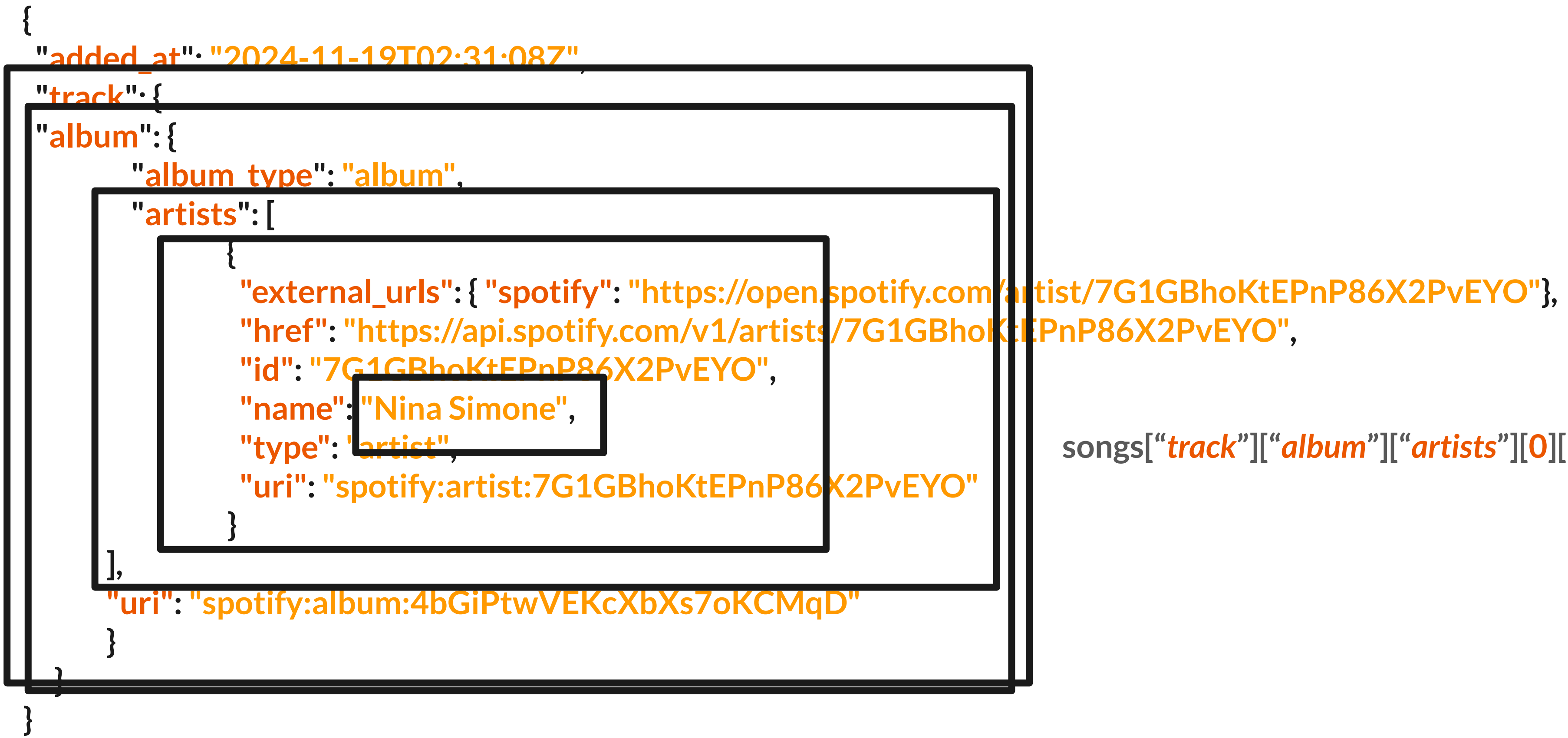
We must then access artists. However notice that artists is **not** a dictionary object. Take a look at the types of brackets that we’re using instead. What kind of data structure is this?



```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
                {
                  "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                  "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                  "id": "7G1GBhoKtEPnP86X2PvEYO",
                  "name": "Nina Simone",
                  "type": "artist",
                  "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
        ],
        "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
        }
    }
}
```

songs[“*track*”][“*album*”][“*artists*”][...]

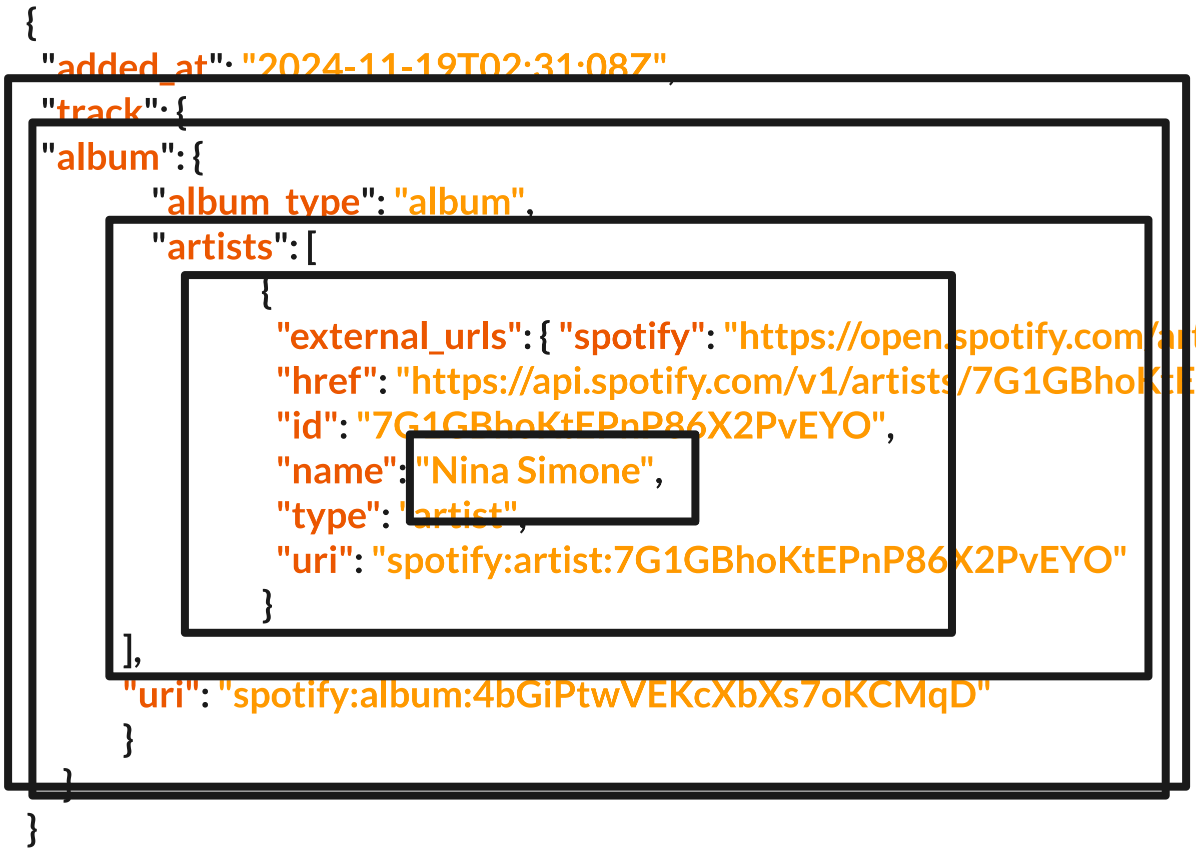
This is a **list** that could potentially store multiple artists. Since we have only one artist in this list, how do we access this one and only artist using bracket notation?



```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
        "album_type": "album",
        "artists": [
            {
             "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
             "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
             "id": "7G1GBhoKtEPnP86X2PvEYO",
             "name": "Nina Simone",
             "type": "artist",
             "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
            }
     ],
     "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
     }
  }
}
```

songs[“*track*”][“*album*”][“*artists*”][0][...]

We input the index value of 0 to access the **first and only dictionary** in this list. Now that we’ve finally accessed the last dictionary object, how can we get the name of this artist?



```
{
 "added_at": "2024-11-19T02:31:08Z",
 "track": {
 "album": {
         "album_type": "album",
         "artists": [
                {
                 "external_urls": { "spotify": "https://open.spotify.com/artist/7G1GBhoKtEPnP86X2PvEYO"},
                 "href": "https://api.spotify.com/v1/artists/7G1GBhoKtEPnP86X2PvEYO",
                 "id": "7G1GBhoKtEPnP86X2PvEYO",
                 "name": "Nina Simone",
                 "type": "artist",
                 "uri": "spotify:artist:7G1GBhoKtEPnP86X2PvEYO"
                }
        ],
        "uri": "spotify:album:4bGiPtwVEKcXbXs7oKCMqD"
        }
  }
}
```

songs[“*track*”][“*album*”][“*artists*”][0][“name”]

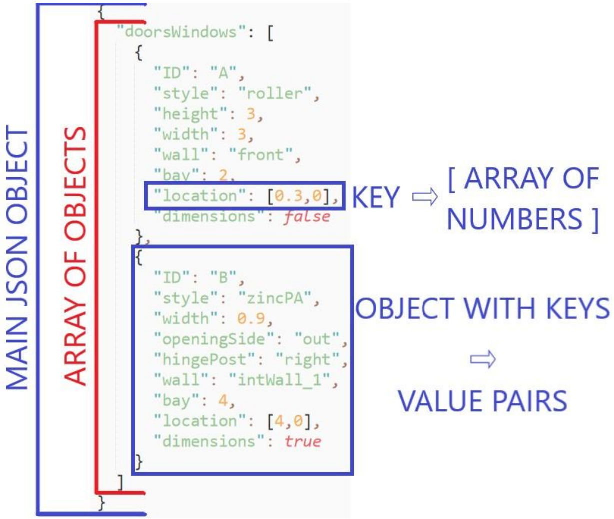
Now we can finally use the “***name***” key to get the value of **Nina Simone.**

JSON Files - Multidimensional Objects

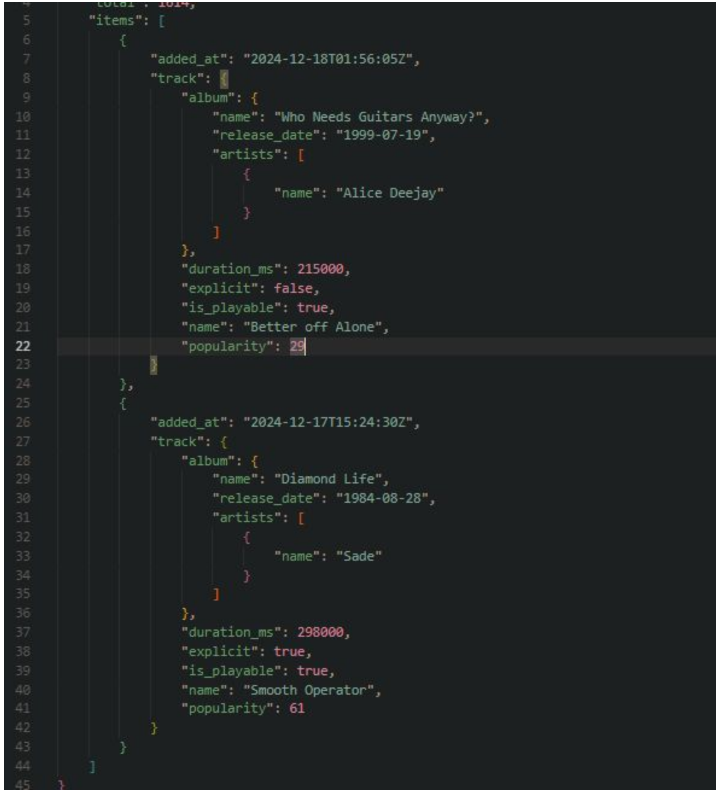
Remember not to get lost in the complexity of trying to **look at the entire object.**

Break the object down piece by piece and note which data-structure you are looking at by observing the syntax of your brackets.

- **Square-brackets:** *list, index position*
- **Curly-brackets:** *dictionary, key*



```
{
  "doorsWindows": [
    {
      "ID": "A",
      "style": "roller",
      "height": 3,
      "width": 3,
      "wall": "front",
      "bay": 2,
      "location": [0.3,0],
      "dimensions": false
    },
    {
      "ID": "B",
      "style": "zincPA",
      "width": 0.9,
      "openingSide": "out",
      "hingePost": "right",
      "wall": "intWall_1",
      "bay": 4,
      "location": [4,0],
      "dimensions": true
    }
  ]
}
```

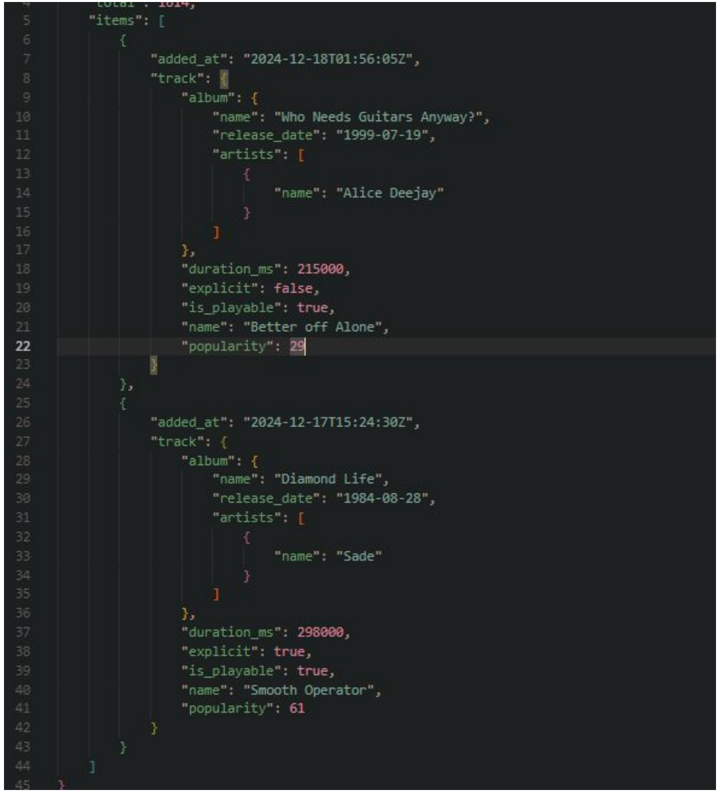


```
    "total": 1014,
    "items": [
        {
            "added_at": "2024-12-18T01:56:05Z",
            "track": {
                "album": {
                    "name": "Who Needs Guitars Anyway?",
                    "release_date": "1999-07-19",
                    "artists": [
                        {
                            "name": "Alice Deejay"
                        }
                    ]
                },
                "duration_ms": 215000,
                "explicit": false,
                "is_playable": true,
                "name": "Better off Alone",
                "popularity": 29
            }
        },
        {
            "added_at": "2024-12-17T15:24:30Z",
            "track": {
                "album": {
                    "name": "Diamond Life",
                    "release_date": "1984-08-28",
                    "artists": [
                        {
                            "name": "Sade"
                        }
                    ]
                },
                "duration_ms": 298000,
                "explicit": true,
                "is_playable": true,
                "name": "Smooth Operator",
                "popularity": 61
            }
        }
    ]
}
```

songs[“items”]

How will this for-loop look like?

In today’s lab you will be interacting with a JSON object with multiple nested objects inside of an array. Let’s assume the code above gives us a **list of dictionaries**. If I wanted to programmatically loop through this list, **which syntax should I write?**



```
    "total": 1014,
    "items": [
        {
            "added_at": "2024-12-18T01:56:05Z",
            "track": {
                "album": {
                    "name": "Who Needs Guitars Anyway?",
                    "release_date": "1999-07-19",
                    "artists": [
                        {
                            "name": "Alice Deejay"
                        }
                    ]
                },
                "duration_ms": 215000,
                "explicit": false,
                "is_playable": true,
                "name": "Better off Alone",
                "popularity": 29
            }
        },
        {
            "added_at": "2024-12-17T15:24:30Z",
            "track": {
                "album": {
                    "name": "Diamond Life",
                    "release_date": "1984-08-28",
                    "artists": [
                        {
                            "name": "Sade"
                        }
                    ]
                },
                "duration_ms": 298000,
                "explicit": true,
                "is_playable": true,
                "name": "Smooth Operator",
                "popularity": 61
            }
        }
    ]
}
```

```
for song in songs[“items”]:
        …
```

We could implement a for-loop! Each “song” iterator variable will be assigned to a dictionary object. **How could we access the nested keys inside of each dictionary we iterate through?**

Conda/JSON Lab

Lab - Conda

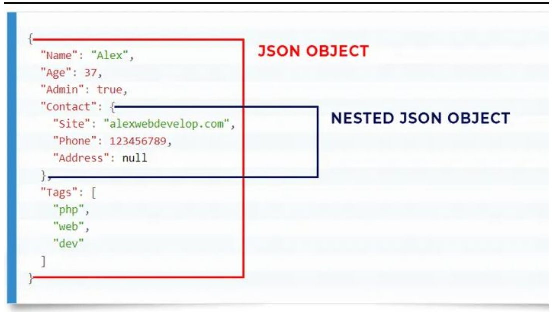
Complete your conda installation with the remaining lab time.

Thank you Dontaye & Lubna for pointing this out. There is a **1-line error** with the Conda environment: please delete the “json” package from the environment.yml file.

Lab - JSON

After completing the conda installation, complete the **JSON Lab.**

While you will not be submitting this, we will call on random groups to answer some of the most challenging questions.



```
{
  "Name": "Alex",
  "Age": 37,
  "Admin": true,
  "Contact": {
    "Site": "alexwebdevelop.com",
    "Phone": 123456789,
    "Address": null
  },
  "Tags": [
    "php",
    "web",
    "dev"
  ]
}
```

Wrap-Up

Lab (Due 03/28)

Taipei City, Taiwan

The company you work for, Seng-Links, aims to identify periods when a user sleeps or exercises using their varying recorded heart rates.

Your company has provided you a data folder (*data/*) of **4 files** that contain heart-rate samples from a participant. The participants device records heart rate data every 5 minutes (aka *sampling rate*).

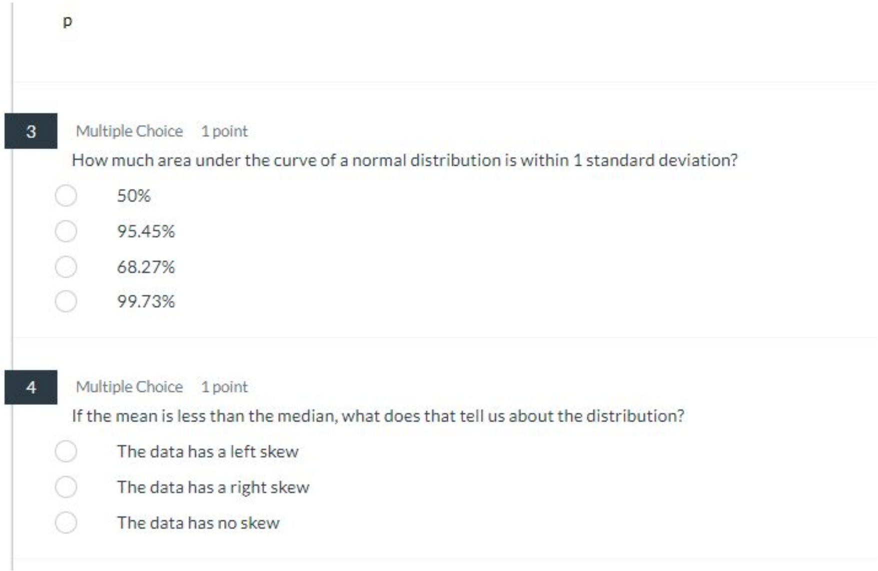
You are tasked with writing code that **processes each data file**. You will utilize test-driven development in order to complete this project.

Stats Quiz (Due 03/28)

Please complete this quiz by 03/28.

This is a 10-question quiz that will test your knowledge of statistics concepts.

2 attempts allowed.



p

3 Multiple Choice 1 point

How much area under the curve of a normal distribution is within 1 standard deviation?

○ 50%

○ 95.45%

○ 68.27%

○ 99.73%

4 Multiple Choice 1 point

If the mean is less than the median, what does that tell us about the distribution?

○ The data has a left skew

○ The data has a right skew

○ The data has no skew

Wednesday

Wednesday will entail:

- A review of visualizing data
- How to use Matplotlib
- TLAB Work

Jupyter: scratchpad of the data scientist

If you understand what you're doing, you're not learning anything. - Anonymous