JavaScript Object Notation

- The JavaScript Object Notation (JSON) is a text format used for storing and transporting data.
  - Other text formats: .csv (tabular)

- JSON is often used when data is sent from a server to a webpage. (so modern browsers use JSON a lot!)

Benefits of JSON

- **Programming language independent:** There are libraries in Python, Java, C++, etc, for working with JSON strings and files.
- **Programmer familiarity:** The notation/syntax is a close match for how programmers define data.
- **Readability:** The format is text-based so the data is readable and editable directly in a text editor.
- **Lightweight but powerful:** The notation is simply but can handle complex situations.
- **Straightforward interface:** Writing a JSON file only requires a data structure and an appropriate library.

JSON Data Structure Example

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

- This looks almost identical to a Python ________!

Python Supports JSON Natively

- Python comes with a built-in package called `json` for encoding and decoding JSON data.

```
import json
```

  - Encoding means **writing** data to disk; decoding means **reading** data into memory.

Serializing JSON

- Python objects to JSON translation:

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

Serialization Example

```
import json
data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}

with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
```

Deserializing JSON

```
with open("data_file.json", "r") as read_file:
    data = json.load(read_file)
```

- The result of the `load(.)` call is able to return any of the allowed data types from the conversion table.

- We will explore JSON more later in the course.
- Let’s now talk about Tabular Model.

Tabular Model: Structure and Formats

Def: A ***data model***

- the *structure* of the data,
- the *operations* to obtain and update data within the model,
- *constraints* that, within the model, limit the data in various ways.

- **Tabular data** is data that is structured into rows, each of which contains information about some thing. Each row contains the same number of cells (although some of these cells may be empty), which provide values of the properties of the thing described by the row.

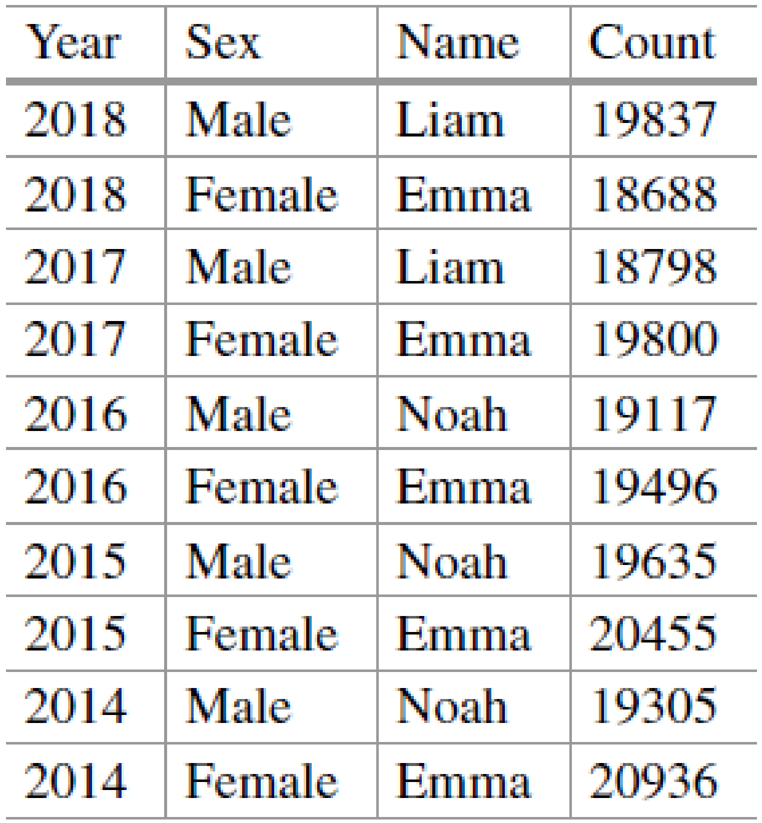
- In tabular data, cells within the same column provide values for the same property of the things described by each row. This is what differentiates tabular data from other line-oriented formats.

Tidy Data

Def: A data set in the tabular data model is said to be ***tidy data*** if it conforms to the following.

1. Each column represents exactly one variable of the data set. (*TidyData*1)
2. Each row represents exactly one unique (relational) mapping, that maps from a set of givens (the values of the independent variables) to the values of the dependent variables. (*TidyData2*)
3. Exactly one table is used for each set of mappings involving the same independent variables. (*TidyData3*)

Top Baby Names

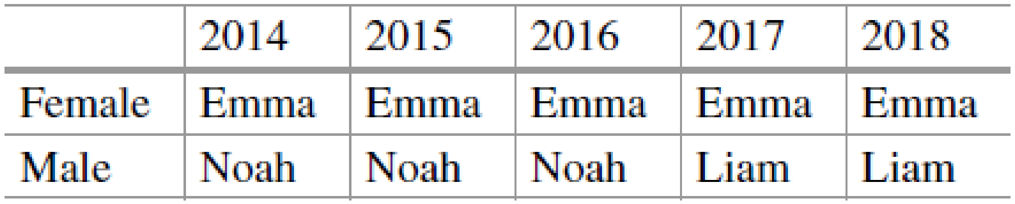


Year	Sex	Name	Count
2018	Male	Liam	19837
2018	Female	Emma	18688
2017	Male	Liam	18798
2017	Female	Emma	19800
2016	Male	Noah	19117
2016	Female	Emma	19496
2015	Male	Noah	19635
2015	Female	Emma	20455
2014	Male	Noah	19305
2014	Female	Emma	20936

- There are alternate ways that this data set could be represented in a tabular format.

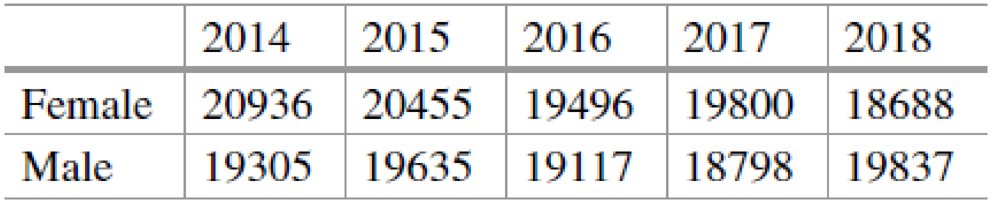
Tidy Data?

Table 6.4 Top baby names by year



	2014	2015	2016	2017	2018
Female	Emma	Emma	Emma	Emma	Emma
Male	Noah	Noah	Noah	Liam	Liam

Table 6.5 Top baby name application counts by year



	2014	2015	2016	2017	2018
Female	20936	20455	19496	19800	18688
Male	19305	19635	19117	18798	19837

- We can see columns headed up by year *values*, so that we are violating TidyData1—the columns here are not a variable.

Tidy Data?

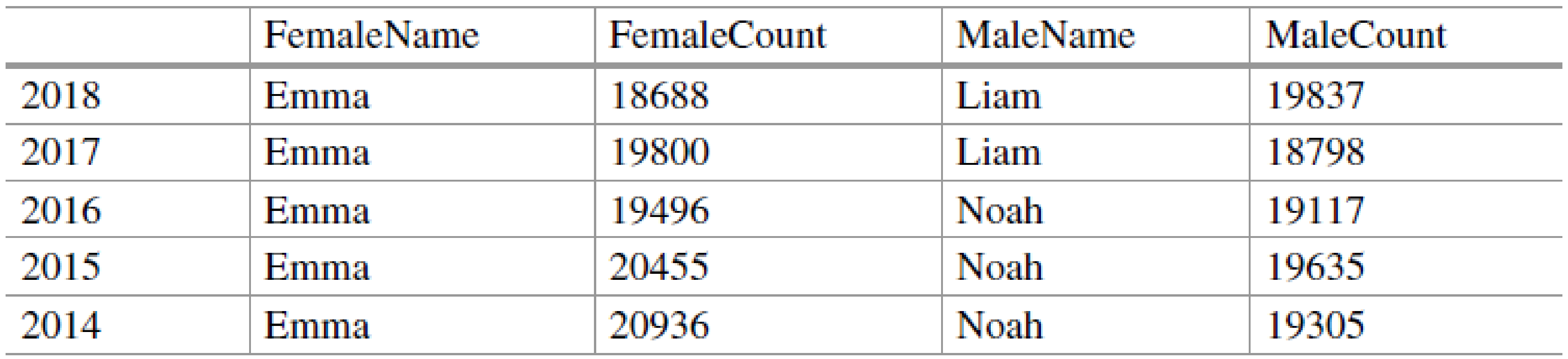


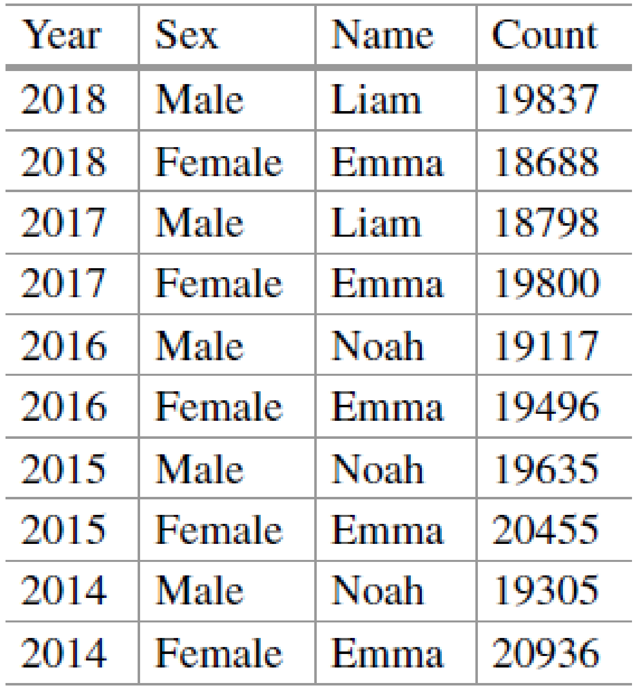
Table 6.7 Top baby names with female/male columns

	FemaleName	FemaleCount	MaleName	MaleCount
2018	Emma	18688	Liam	19837
2017	Emma	19800	Liam	18798
2016	Emma	19496	Noah	19117
2015	Emma	20455	Noah	19635
2014	Emma	20936	Noah	19305

- Note that the columns like `FemaleName` do not represent exactly one variable.
- The name of the column itself shows two parts, one of which is a value of a categorical variable, and the other part is a variable that is then repeated in `MaleName`.
- This is a violation of TidyData1.

Tidy Data?

Table 6.1 Top baby names



Year	Sex	Name	Count
2018	Male	Liam	19837
2018	Female	Emma	18688
2017	Male	Liam	18798
2017	Female	Emma	19800
2016	Male	Noah	19117
2016	Female	Emma	19496
2015	Male	Noah	19635
2015	Female	Emma	20455
2014	Male	Noah	19305
2014	Female	Emma	20936

- Only the original table, Table 6.1, satisfies all three requirements of tidy data.

Format

Def: A ***format*** (*file* format or document *format*) is a specification for the encoding of information for a data structure into a sequence of characters or bytes, in a file or across a stream, to be communicated between a creator/writer/provider and a corresponding consumer/reader/client.

Examples: `jpg`, `mp3`, `docx`, `mov`

- The most common format <u>for tabular data</u> is **the *comma-separated value*** **(CSV)** format.

CSV Format

- CSV strikes a delicate balance between being readable by humans and readable by computers.

- CSV is a tabular format consisting of rows of data, each row containing multiple cells. Rows are (usually) separated by line terminators, so each row corresponds to one line.

- The CSV format assumes exactly one tabular structure per separate file.
  - This is more constrained than a spreadsheet model, in which a single file can contain multiple “worksheets” that could each hold its own tabular structure.

- CSV is a “text-based” format, i.e., a CSV file is a text file.

Typical Structure of CSV

- One “header row,” which gives the names of the columns.
- May “rows” of data.
  - One row of data is encoded in <u>one line</u> of the file.
  - Values are separated by commas.
- Sometimes a ‘#’ is used for comment lines. (although rarely)

CSV Example

- Header row:

```
LastName, FirstName, Initial, Course, Time
```

- Data (subsequent rows):

```
Lauer, Hugh, C, CS-1004, 0800
Wills, Craig, E, CS-3013, 1200
Hamel, Glynis, M, CS-1101, 1000
```

Tabular Data in Python

- We could represent tabular data in Python with native built-in data structures using a *dictionary of lists* (DoL) or a *lists of lists* (LoL).

- As we will see, these approaches have distinct disadvantages.

Dictionary of Lists (DoL)

- We could represent tabular data in Python using a *dictionary of lists* (DoL):

Year	Sex	Name	Count
2018	Male	Liam	19837
2018	Female	Emma	18688
2017	Male	Liam	18798
2017	Female	Emma	19800
2016	Male	Noah	19117
2016	Female	Emma	19496

- This is effective for accessing and manipulating entire columns but working with rows requires additional code.

```
topnames = {’year’: [2018, 2018, 2017, 2017, 2016, 2016],
            ’sex’: [’Male’, ’Female’, ’Male’,
                        ’Female’, ’Male’, ’Female’],
            ’name’: [’Liam’, ’Emma’, ’Liam’, ’Emma’,
                         ’Noah’, ’Emma’],
            ’count’: [19837, 18688, 18798, 19800, 19117, 19496]}
```

Dictionary of Lists (DoL)

```
topnames = {'year': [2018, 2018, 2017, 2017, 2016, 2016],
          'sex': ['Male', 'Female', 'Male',
                    'Female', 'Male', 'Female'],
          'name': ['Liam', 'Emma', 'Liam', 'Emma',
                     'Noah', 'Emma'],
          'count': [19837, 18688, 18798, 19800, 19117, 19496]}
```

With this representation, it is easy to get data for an entire column:

```
print(topnames['year'])
```

```
| [2018, 2018, 2017, 2017, 2016, 2016]
```

But getting data for a row is more involved. Consider the row with index 3:

```
row_index = 3
print(topnames['year'][row_index], topnames['sex'][row_index],
    topnames['name'][row_index], topnames['count'][row_index])
```

```
| 2017 Female Emma 19800
```

List of Lists (LoL)

- We could also represent tabular data in Python using a *list of lists* (LoL):

Year	Sex	Name	Count
2018	Male	Liam	19837
2018	Female	Emma	18688
2017	Male	Liam	18798
2017	Female	Emma	19800
2016	Male	Noah	19117
2016	Female	Emma	19496

- While effective for dealing by rows, there is more work for manipulating columns.

```
topnames = [[2018, ’Male’, ’Liam’, 19837],
            [2018, ’Female’, ’Emma’, 18688],
            [2017, ’Male’, ’Liam’, 18798],
            [2017, ’Female’, ’Emma’, 19800],
            [2016, ’Male’, ’Noah’, 19117],
            [2016, ’Female’, ’Emma’, 19496]]
columns = [’year’, ’sex’, ’name’, ’count’]
```

List of Lists (LoL)

```
topnames = [[2018, 'Male', 'Liam', 19837],
            [2018, 'Female', 'Emma', 18688],
            [2017, 'Male', 'Liam', 18798],
            [2017, 'Female', 'Emma', 19800],
            [2016, 'Male', 'Noah', 19117],
            [2016, 'Female', 'Emma', 19496]]
columns = ['year', 'sex', 'name', 'count']
```

In this representation, we can easily access and refer to a particular row, as long as we know its index:

```
row_index = 5
print(topnames[row_index])
```

```
| [2016, 'Female', 'Emma', 19496]
```

List of Lists (LoL)

```
topnames = [[2018, 'Male', 'Liam', 19837],
            [2018, 'Female', 'Emma', 18688],
            [2017, 'Male', 'Liam', 18798],
            [2017, 'Female', 'Emma', 19800],
            [2016, 'Male', 'Noah', 19117],
            [2016, 'Female', 'Emma', 19496]]
columns = ['year', 'sex', 'name', 'count']
```

To access a column, we need to know its index. In this example, we may need to know that the count values are at index 3 within the list in each row, and to access all the values in a particular column, we must refer to all of the individual rows:

```
col_index = 3
print(topnames[0][col_index], topnames[1][col_index],
      topnames[2][col_index], topnames[3][col_index],
      topnames[4][col_index], topnames[5][col_index])
```

```
| 19837 18688 18798 19800 19117 19496
```

Pandas

- Pandas Dataframe is the solution!!
  - The need for Python data structures to handle tabular data has led to the development of the `pandas` module.

- Let’s begin learning Pandas!