# Python for Text Analysis 2018-2019

Lecture 10: Data Formats part I [block 4]
29-11-2018

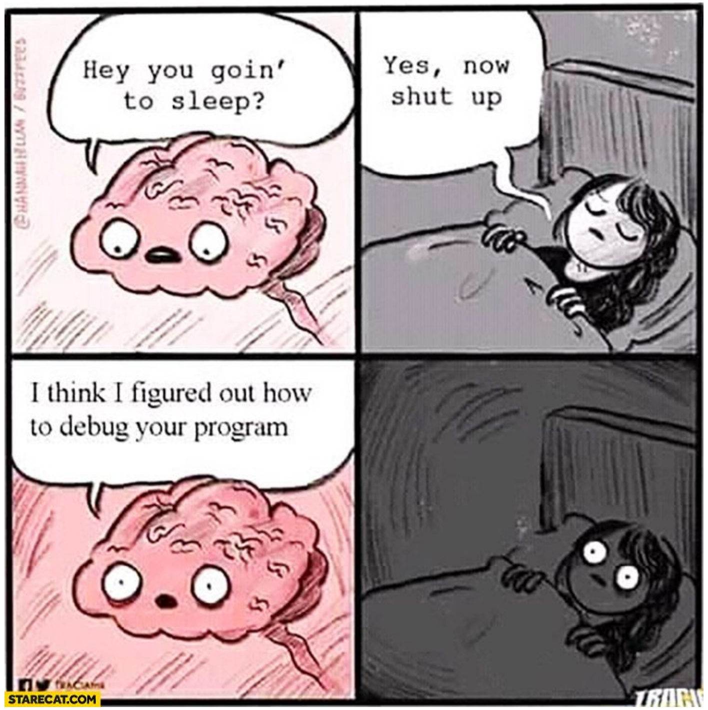
## Hooray, the last block!

**Thursday**

- ❖ Chapter 16: Data Formats I (CSV/TSV)
- ❖ Chapter 17: Data Formats II (JSON)

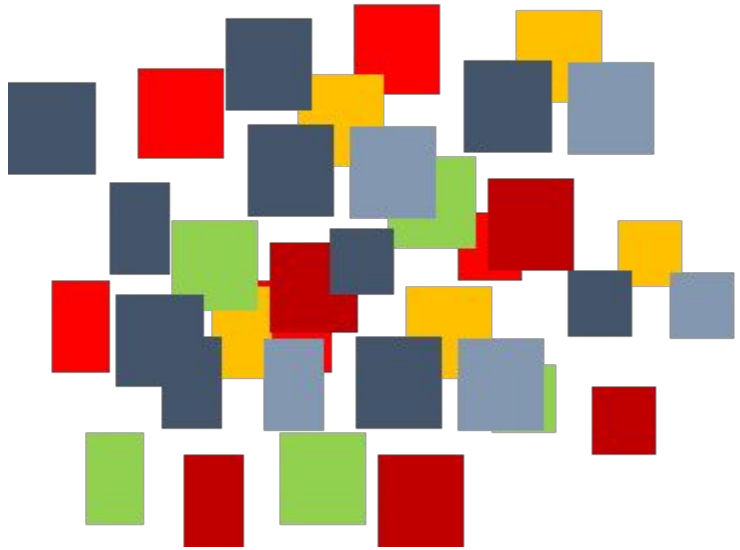
**Monday**

- ❖ Chapter 18: Data Formats III (XML)



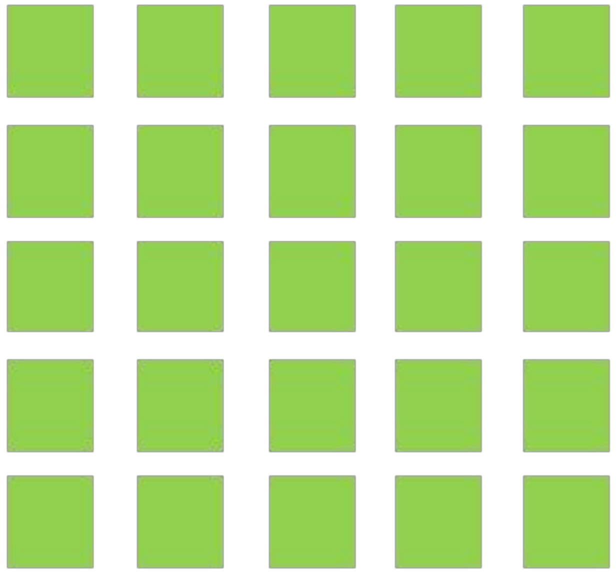
## Block 4: Data Formats

### UNSTRUCTURED DATA



(for example: plain text)

### (SEMI-)STRUCTURED DATA



(for example: CSV, JSON, XML)

## Goals for today

- Thinking of **CSV** and **JSON** as **(nested) Python objects**
  - Lists
  - Dictionaries
- **Reading** and **writing** CSV & JSON from/to files
- Maybe even more important: practicing with **nested lists and dicts**:
  - **accessing values** using indices and keys
  - **iterating over them**
  - Chapter 16, Exercise 2 (baby names CSV)
  - Chapter 17, Exercise 4 (Stranger Things JSON)

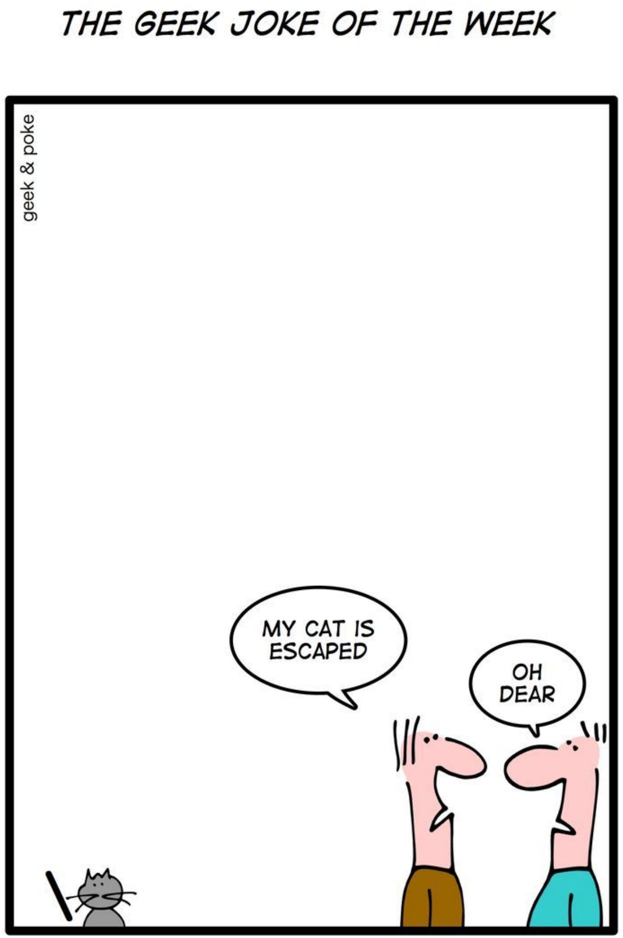
## SOCRATIVE

**Log in at:**

b.socrative.com/student

**Room:**

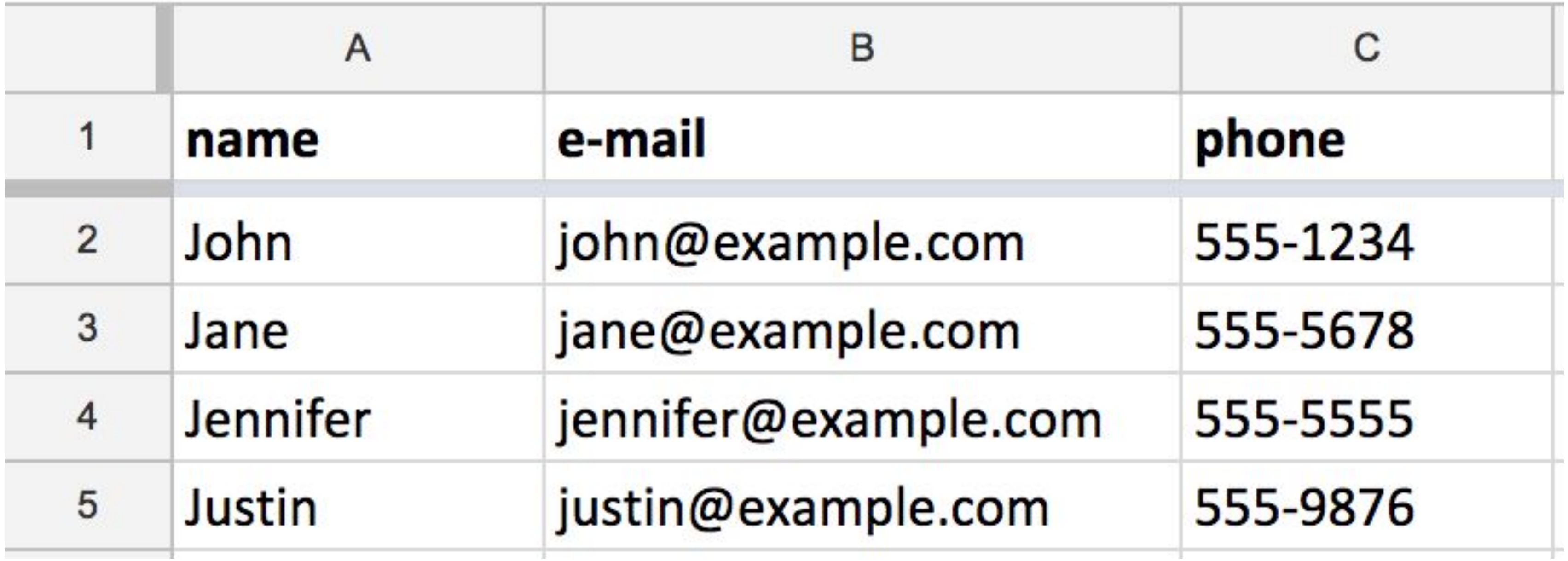
PYTHON1819VU



# **CHAPTER 16:** Data Formats I (CSV/TSV)

## About tables

A **table** stores data in a **structured format** consisting of **columns** and **rows**. Optionally, the first row represents the **header**.

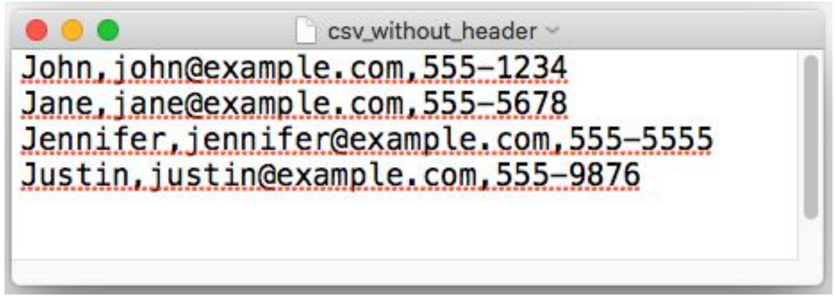


| | A | B | C |
|---|---|---|---|
| 1 | **name** | **e-mail** | **phone** |
| 2 | John | john@example.com | 555-1234 |
| 3 | Jane | jane@example.com | 555-5678 |
| 4 | Jennifer | jennifer@example.com | 555-5555 |
| 5 | Justin | justin@example.com | 555-9876 |

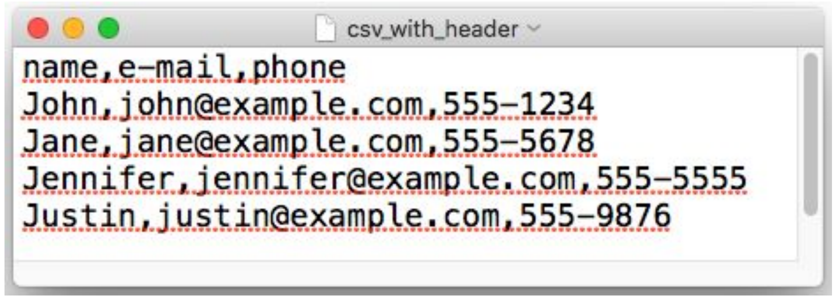
# About CSV

A **CSV file** (**C**omma-**S**eparated **V**alues file) is simply a **plain text file** representing tabular data, where:

- each **record (row)** is on a new line
- **commas** separate the values in a row (corresponding to the columns)



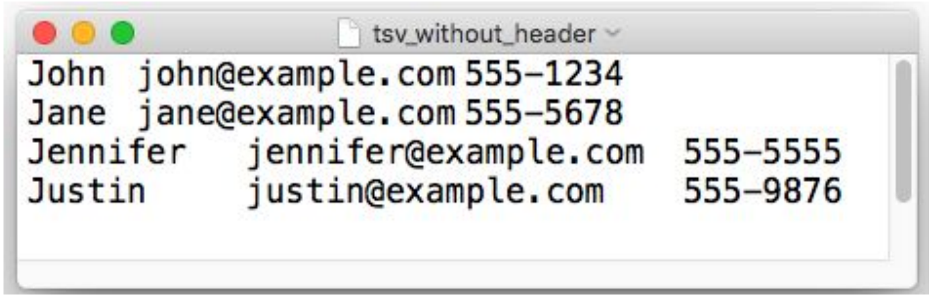
```
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```



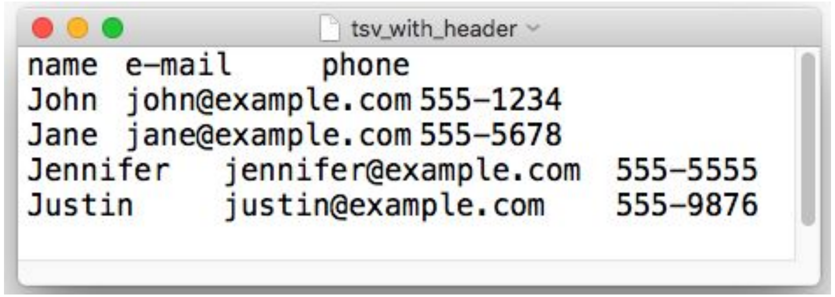
```
name,e-mail,phone
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

## About CSV

- In principle, **any character** can serve as the **delimiter**
- For example, **tabs** are also commonly used **(TSV)**



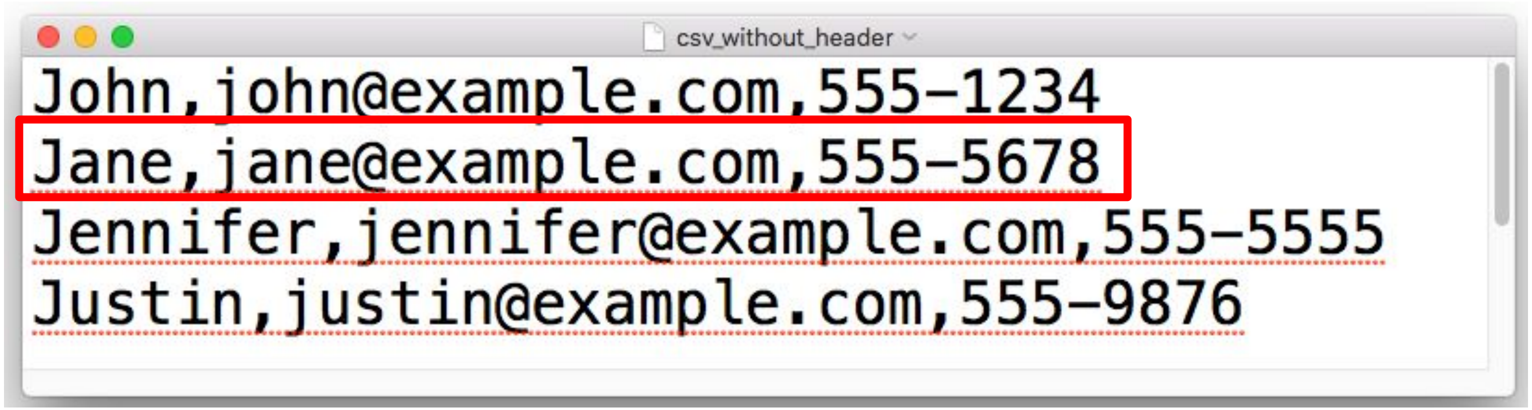
```
John	john@example.com	555-1234
Jane	jane@example.com	555-5678
Jennifer	jennifer@example.com	555-5555
Justin	justin@example.com	555-9876
```



```
name	e-mail	phone
John	john@example.com	555-1234
Jane	jane@example.com	555-5678
Jennifer	jennifer@example.com	555-5555
Justin	justin@example.com	555-9876
```

## Thinking about tabular data as Python objects

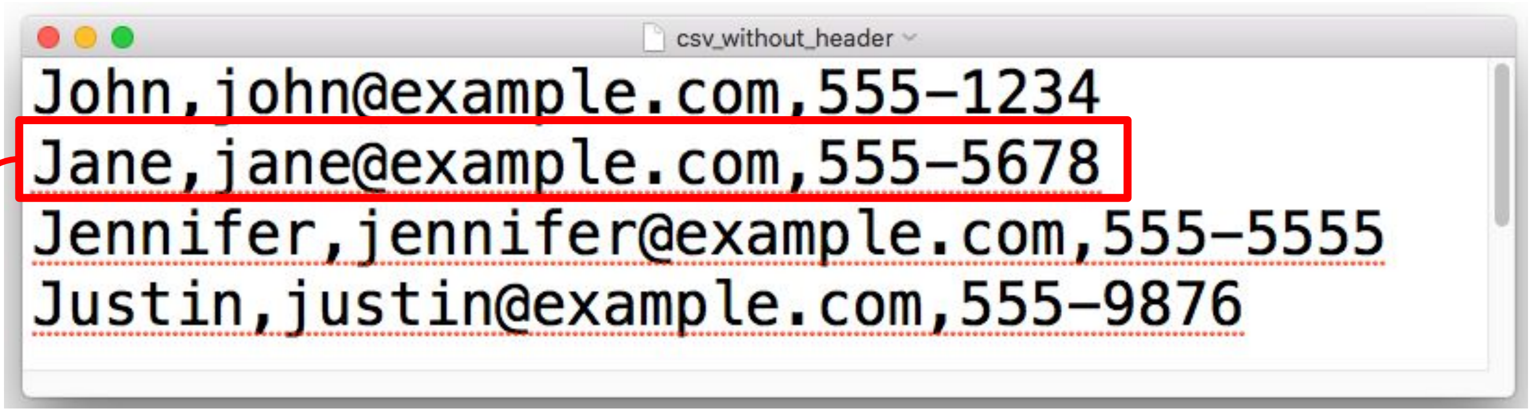
Does a **single row** remind you of a certain **Python data type**...?



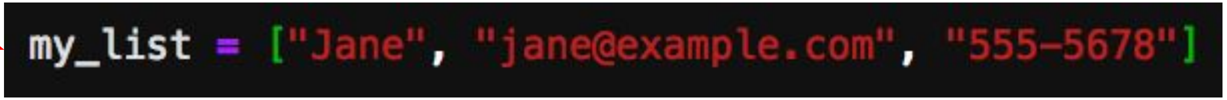
```
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

## Thinking about tabular data as Python objects

Does a **single row** remind you of a certain **Python data type**...?



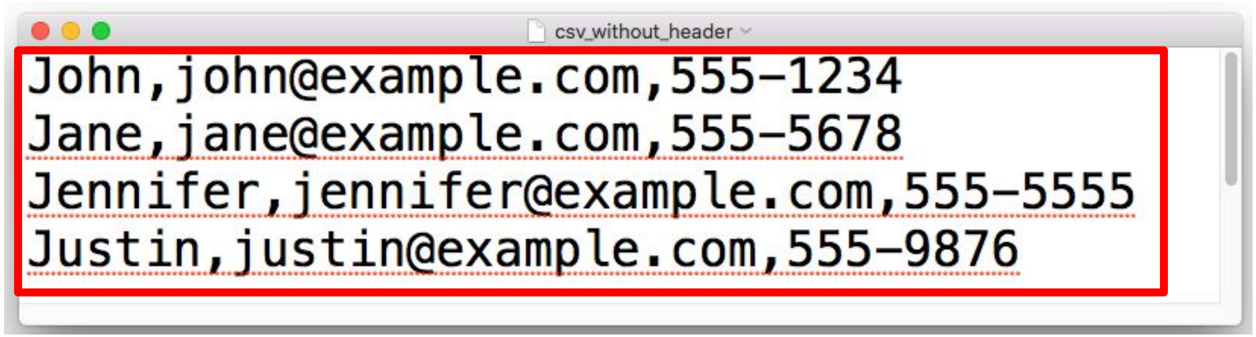
```
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```



```python
my_list = ["Jane", "jane@example.com", "555-5678"]
```

## Thinking about tabular data as Python objects

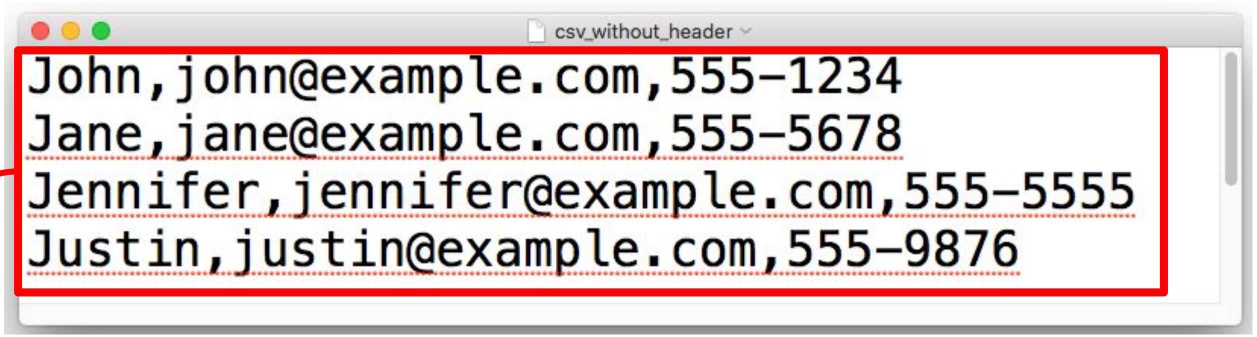
How could we represent **a collection of rows** in Python?



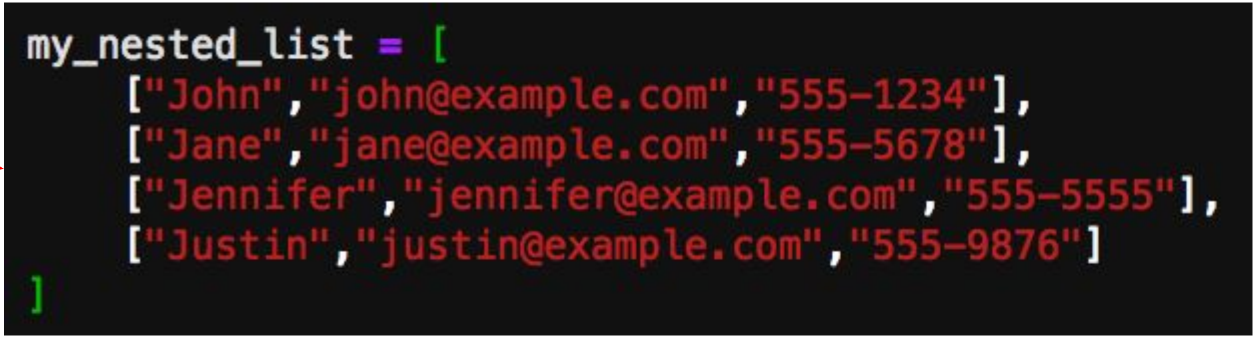
```
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

# Thinking about tabular data as Python objects

How could we represent **a collection of rows** in Python?



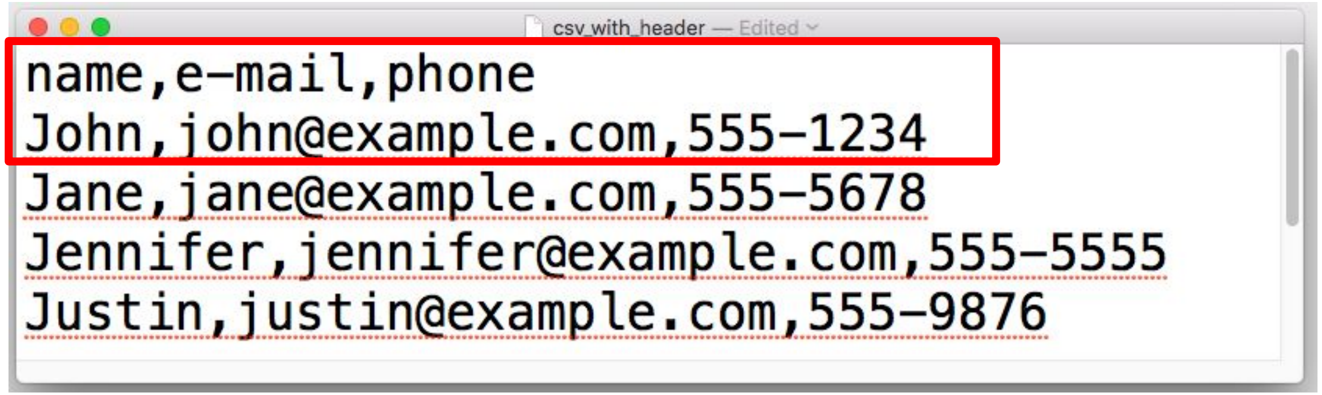
```
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```



```
my_nested_list = [
    ["John","john@example.com","555-1234"],
    ["Jane","jane@example.com","555-5678"],
    ["Jennifer","jennifer@example.com","555-5555"],
    ["Justin","justin@example.com","555-9876"]
]
```

## Thinking about tabular data as Python objects

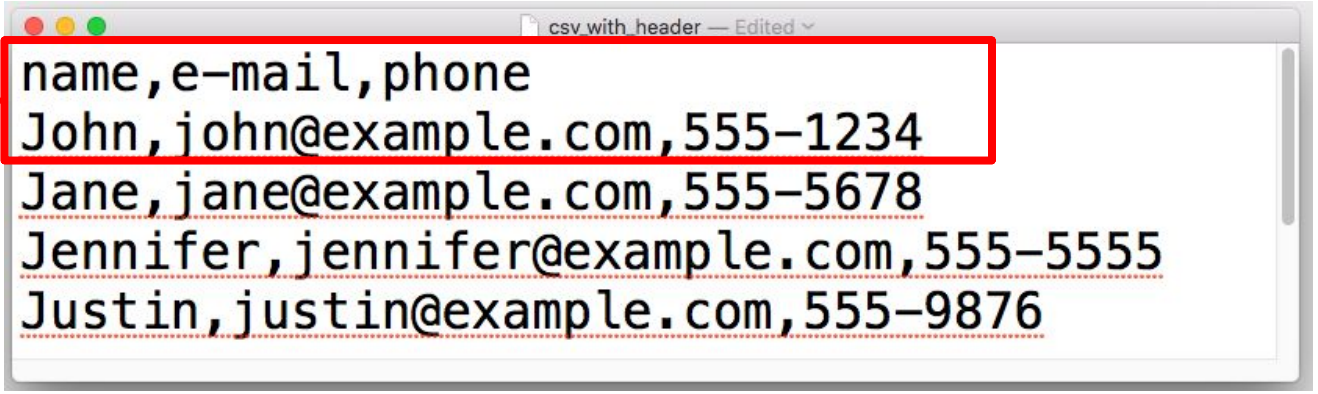
Does the **combination of the header with any of the other rows** remind you of a certain **Python data type**...?



```
name,e-mail,phone
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

## Thinking about tabular data as Python objects

Does the **combination of the header with any of the other rows** remind you of a certain **Python data type**...?

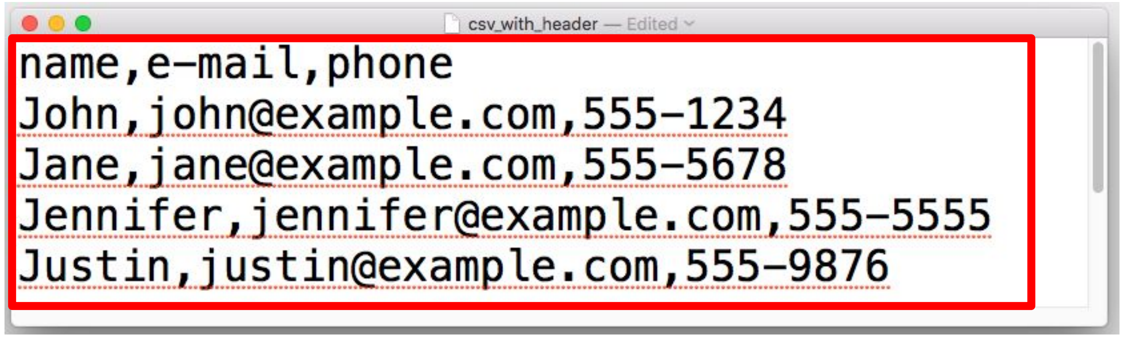


```
name,e-mail,phone
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

```
my_dict = {"name": "John", "e-mail": "john@example.com", "phone": "555-1234"}
```

## Thinking about tabular data as Python objects

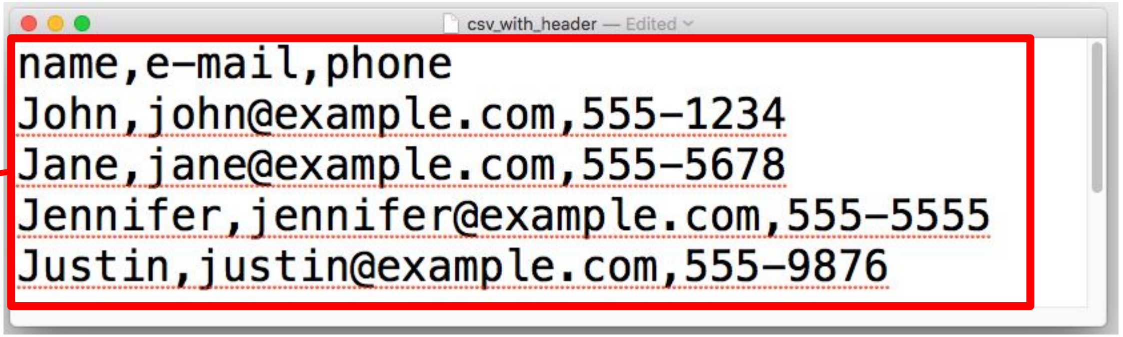
How could we represent **a collection of rows in combination with the header** in Python?



```
name,e-mail,phone
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

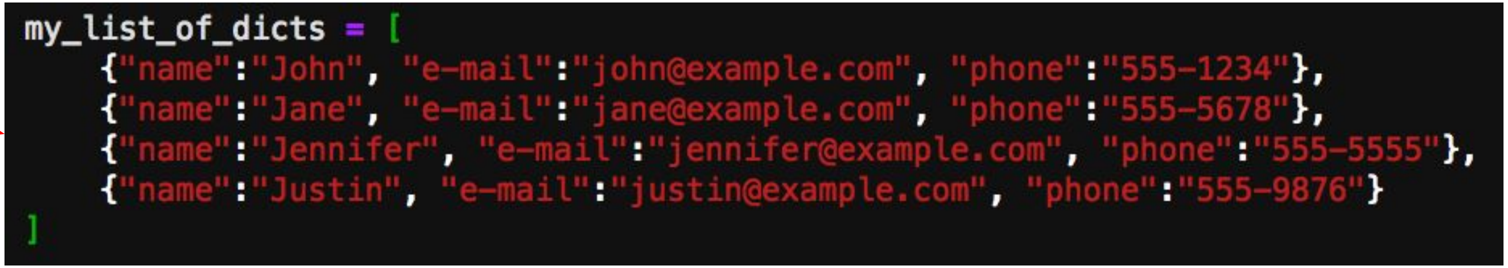
## Thinking about tabular data as Python objects

How could we represent **a collection of rows in combination with the header** in Python?



csv_with_header — Edited

```
name,e-mail,phone
John,john@example.com,555-1234
Jane,jane@example.com,555-5678
Jennifer,jennifer@example.com,555-5555
Justin,justin@example.com,555-9876
```

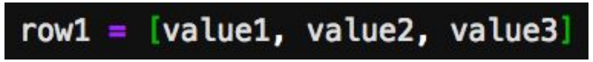


```
my_list_of_dicts = [
    {"name":"John", "e-mail":"john@example.com", "phone":"555-1234"},
    {"name":"Jane", "e-mail":"jane@example.com", "phone":"555-5678"},
    {"name":"Jennifer", "e-mail":"jennifer@example.com", "phone":"555-5555"},
    {"name":"Justin", "e-mail":"justin@example.com", "phone":"555-9876"}
]
```

## Thinking about tabular data as Python objects

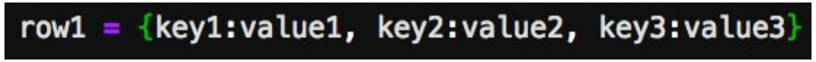
In sum:

❖ A **single row** is always represented as a **list** or as a **dictionary**



```
row1 = [value1, value2, value3]
```

OR

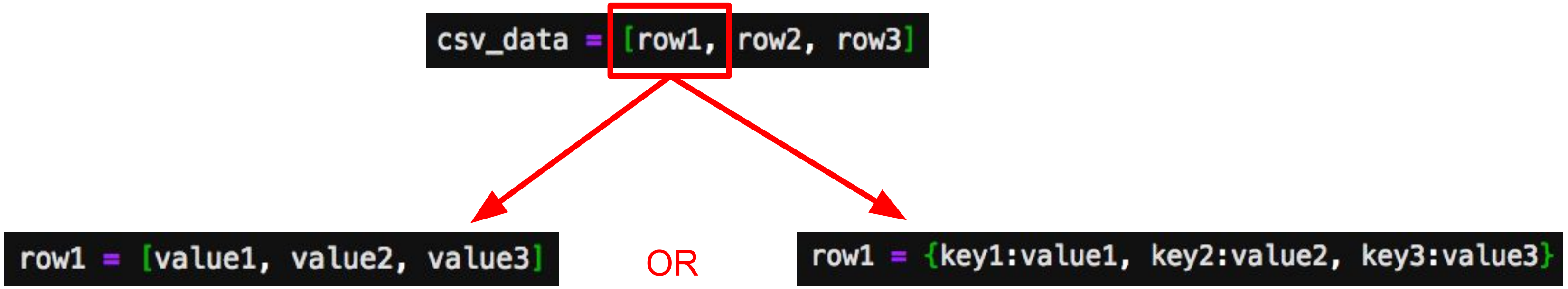


```
row1 = {key1:value1, key2:value2, key3:value3}
```

## Thinking about tabular data as Python objects

In sum:

- A **single row** is always represented as a **list** or as a **dictionary**
- The **collection of rows** is always represented as a **list**



```
csv_data = [row1, row2, row3]
```

```
row1 = [value1, value2, value3]
```

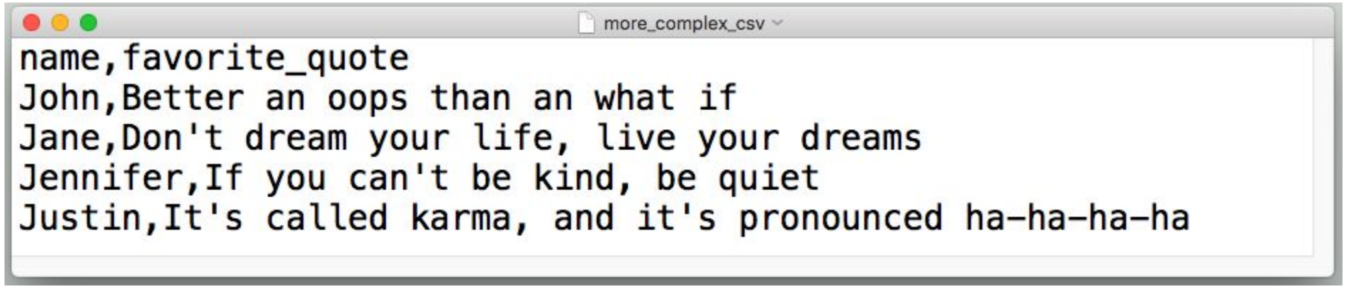
```
row1 = {key1:value1, key2:value2, key3:value3}
```

## Reading and Writing CSV in Python

**Let’s look at some code :-)**

# EXTRA: More complex CSV files

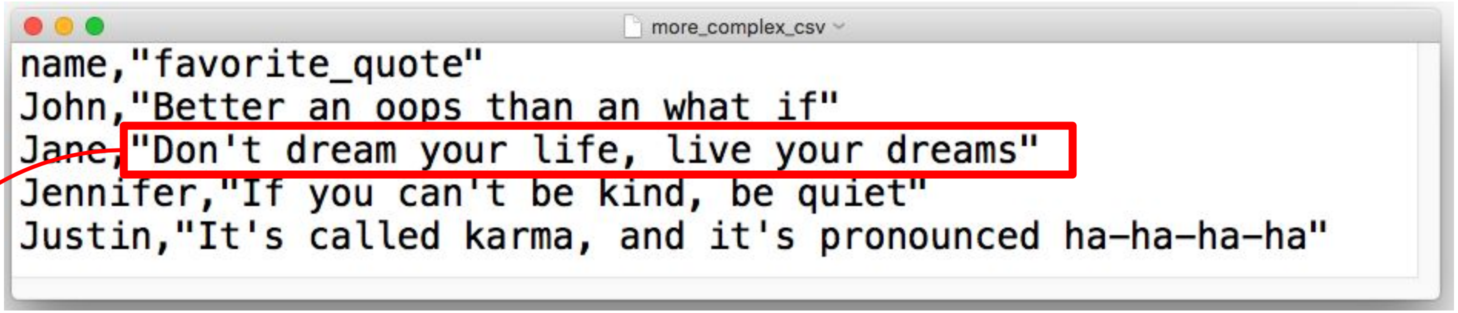
❖ What can be said about this CSV?



```
name,favorite_quote
John,Better an oops than an what if
Jane,Don't dream your life, live your dreams
Jennifer,If you can't be kind, be quiet
Justin,It's called karma, and it's pronounced ha-ha-ha-ha
```

## EXTRA: More complex CSV files

❖ Solution: use **quotation marks** around the fields

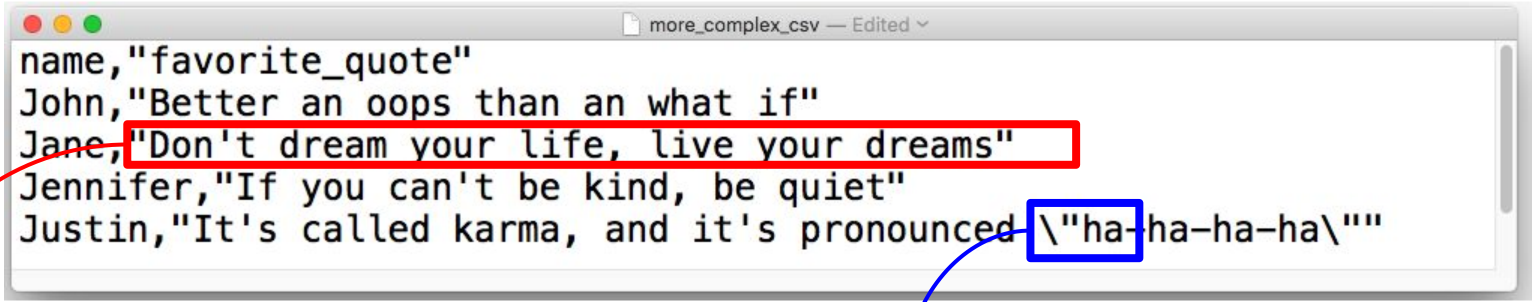


```
name,"favorite_quote"
John,"Better an oops than an what if"
Jane,"Don't dream your life, live your dreams"
Jennifer,"If you can't be kind, be quiet"
Justin,"It's called karma, and it's pronounced ha-ha-ha-ha"
```

"Treat any comma in between these quotation marks as **part of the string** instead of a separator."

## **EXTRA:** More complex CSV files

- Solution: use **quotation marks** around the fields
- However, sometimes the fields also contain quotation marks;
  in that case, you would use an **escape character** such as **\** (backslash)



```
name,"favorite_quote"
John,"Better an oops than an what if"
Jane,"Don't dream your life, live your dreams"
Jennifer,"If you can't be kind, be quiet"
Justin,"It's called karma, and it's pronounced \"ha-ha-ha-ha\""
```

"Treat any comma in between these quotation marks as **part of the string** instead of a separator."

"Treat the following quotation mark as **part of the string** instead of as the end of this string."

## **EXTRA:** More complex CSV files

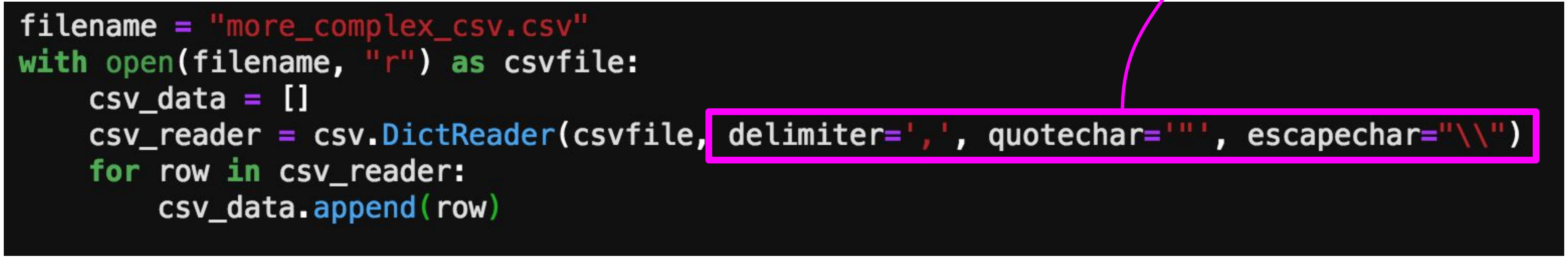
- The **csv module** is also extremely useful for reading/writing CSV, especially for more complex CSV files

- For **reading** CSV:
  - csv.reader() → for reading a file as a **list of lists**
  - csv.DictReader() → for reading a file as a **list of dicts**
- For **writing** CSV:
  - csv.writer() → for writing a **list of lists** to a file
  - csv.DictWriter() → for writing a **list of dicts** to a file

- These methods allow you, for example, to specify the **delimiter**, the **quotation marks** and the **escape character**

## **EXTRA:** csv module

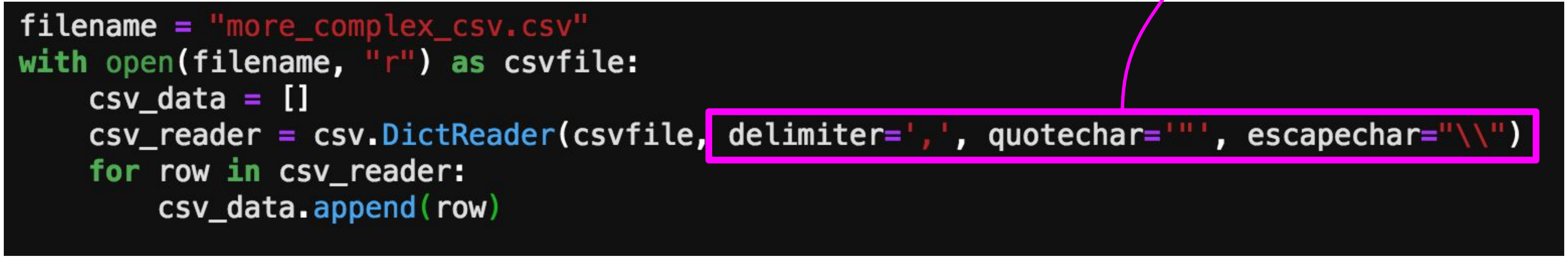
specify how the CSV looks like



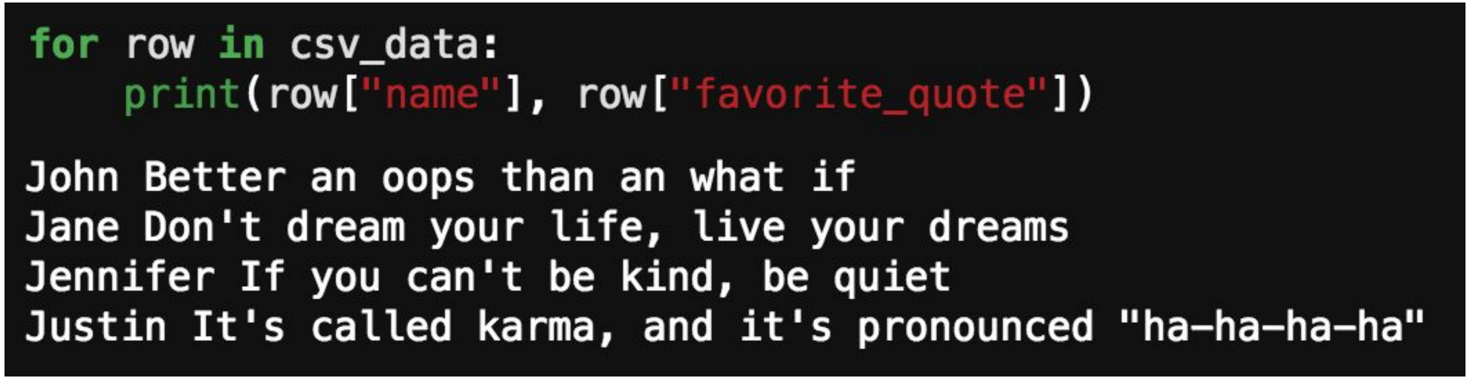
```
filename = "more_complex_csv.csv"
with open(filename, "r") as csvfile:
    csv_data = []
    csv_reader = csv.DictReader(csvfile, delimiter=',', quotechar='"', escapechar="\\")
    for row in csv_reader:
        csv_data.append(row)
```

## **EXTRA:** csv module

specify how the CSV looks like

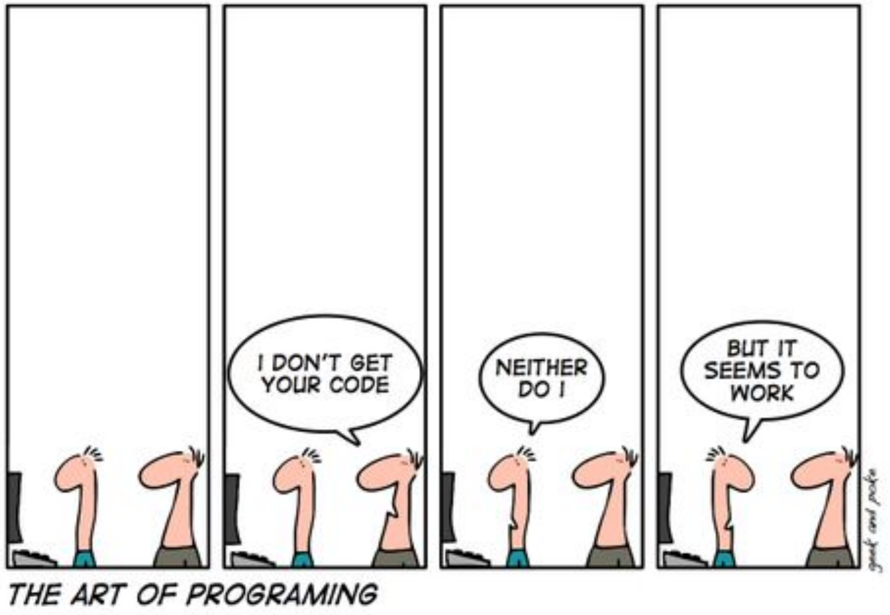


```python
filename = "more_complex_csv.csv"
with open(filename, "r") as csvfile:
    csv_data = []
    csv_reader = csv.DictReader(csvfile, delimiter=',', quotechar='"', escapechar="\\")
    for row in csv_reader:
        csv_data.append(row)
```



```python
for row in csv_data:
    print(row["name"], row["favorite_quote"])
```

```
John Better an oops than an what if
Jane Don't dream your life, live your dreams
Jennifer If you can't be kind, be quiet
Justin It's called karma, and it's pronounced "ha-ha-ha-ha"
```

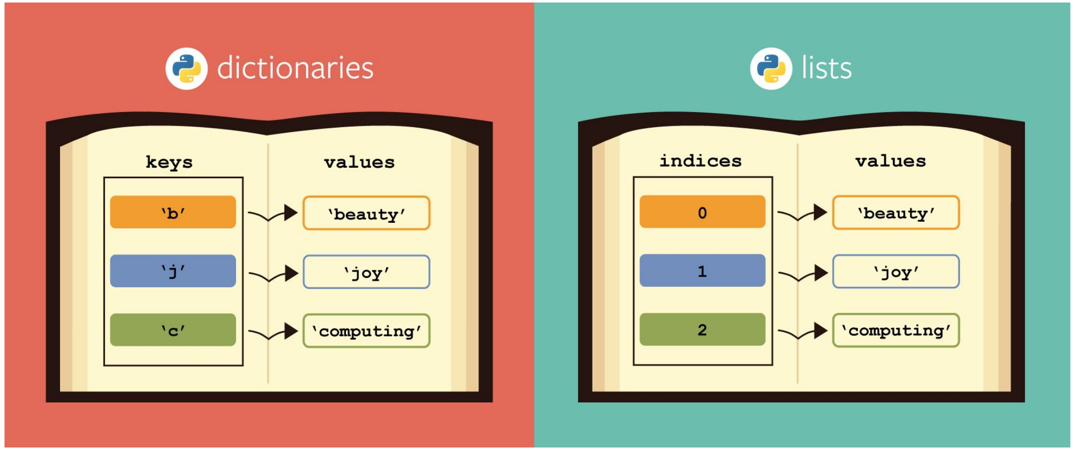


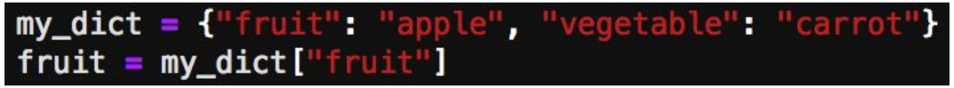
## PART I

Accessing data from (nested) lists and dicts

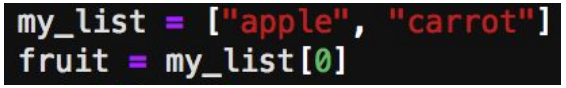
# Accessing data from lists and dicts

- Specific values in a **list** can be accessed by their **indices**
- Specific values in a **dictionary** can be accessed by their **keys**



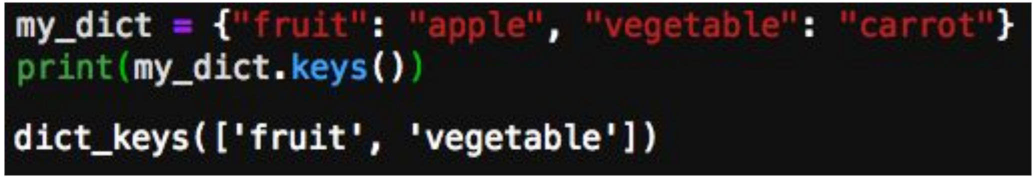


```
my_dict = {"fruit": "apple", "vegetable": "carrot"}
fruit = my_dict["fruit"]
```



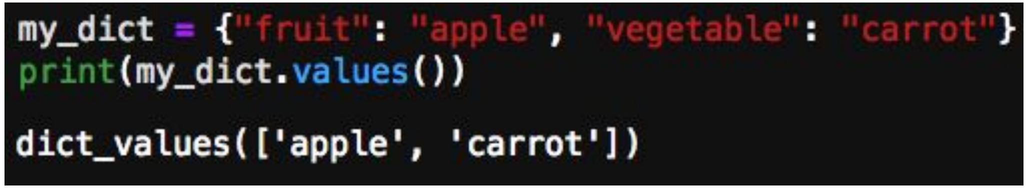
```
my_list = ["apple", "carrot"]
fruit = my_list[0]
```

## Accessing keys, values and key-value pairs in dicts



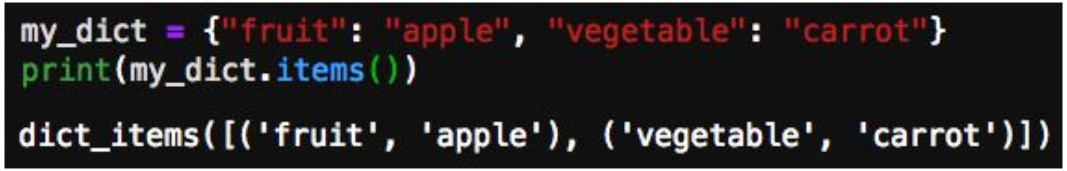
```
my_dict = {"fruit": "apple", "vegetable": "carrot"}
print(my_dict.keys())

dict_keys(['fruit', 'vegetable'])
```



```
my_dict = {"fruit": "apple", "vegetable": "carrot"}
print(my_dict.values())

dict_values(['apple', 'carrot'])
```

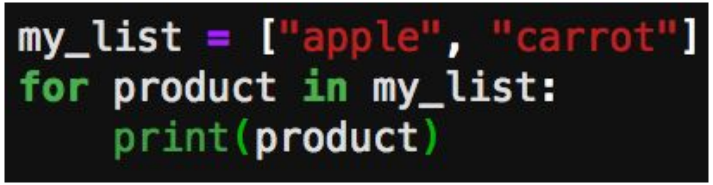


```
my_dict = {"fruit": "apple", "vegetable": "carrot"}
print(my_dict.items())

dict_items([('fruit', 'apple'), ('vegetable', 'carrot')])
```

# Iterating over lists and dicts

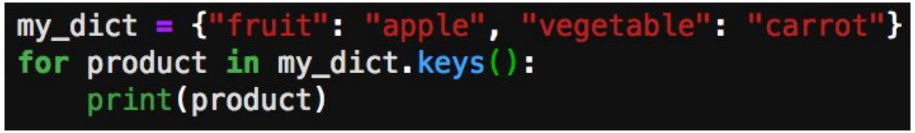
**List:** iterating over values



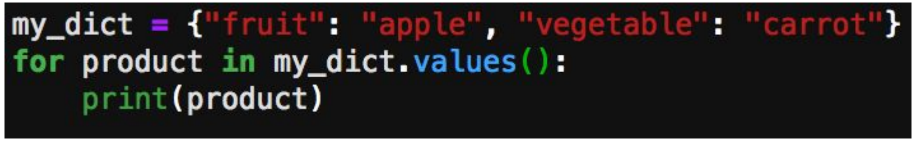
```python
my_list = ["apple", "carrot"]
for product in my_list:
    print(product)
```

**Dictionary:** iterating over either:

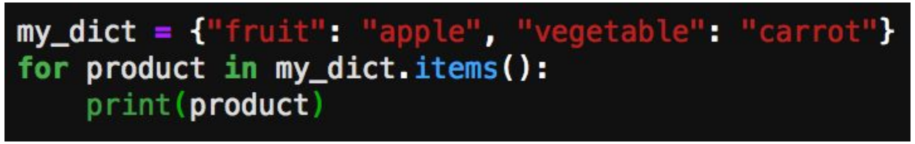
- ❖ Keys
- ❖ Values
- ❖ Key-value pairs (items)



```python
my_dict = {"fruit": "apple", "vegetable": "carrot"}
for product in my_dict.keys():
    print(product)
```



```python
my_dict = {"fruit": "apple", "vegetable": "carrot"}
for product in my_dict.values():
    print(product)
```



```python
my_dict = {"fruit": "apple", "vegetable": "carrot"}
for product in my_dict.items():
    print(product)
```

# SOCRATIVE

**Log in at:**

b.socrative.com/student

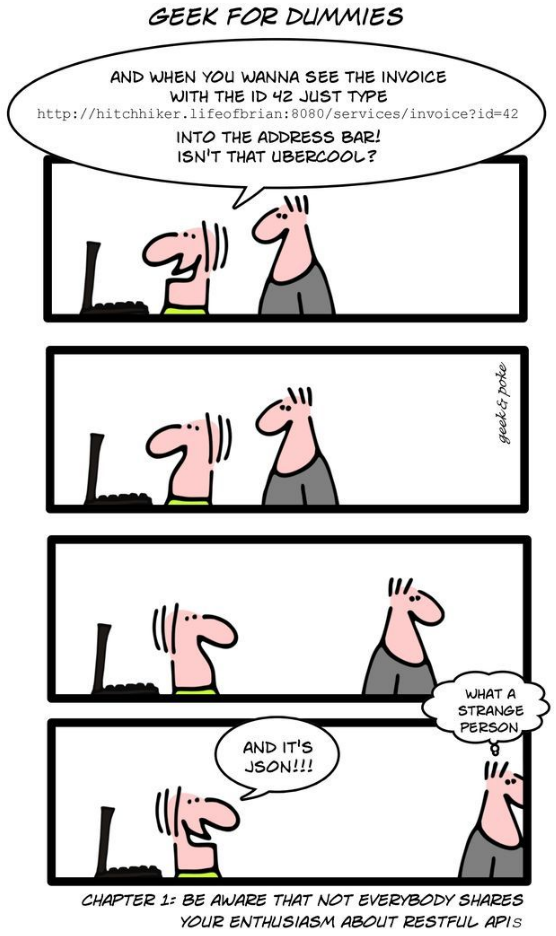
**Room:**

PYTHON1819VU

## Practicing a bit more

**Let’s look at some code :-)**

**Chapter 16, Exercise 2 (baby names)**


GEEK FOR DUMMIES
AND WHEN YOU WANNA SEE THE INVOICE WITH THE ID 42 JUST TYPE http://hitchhiker.lifeofbrian:8080/services/invoice?id=42 INTO THE ADDRESS BAR! ISN'T THAT UBERCOOL?
geek & poke
WHAT A STRANGE PERSON
AND IT'S JSON!!!
CHAPTER 1: BE AWARE THAT NOT EVERYBODY SHARES YOUR ENTHUSIASM ABOUT RESTFUL APIs

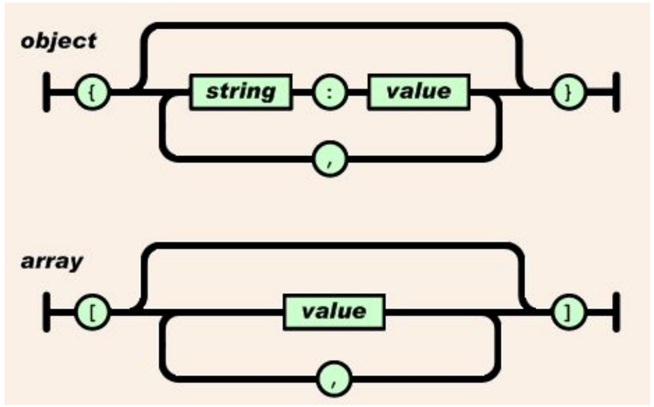
# **CHAPTER 2:** Data Formats II (JSON)

## About JSON

- **J**ava**S**cript **O**bject **N**otation
- Light-weight **data-interchange** format
- Many (public) **APIs** publish their data in JSON format:
  - Google Maps API
  - Twitter API
  - NASA API
  - GeoNames API
  - Pokémon API
  - Potter API
  - ....and many, many, many more

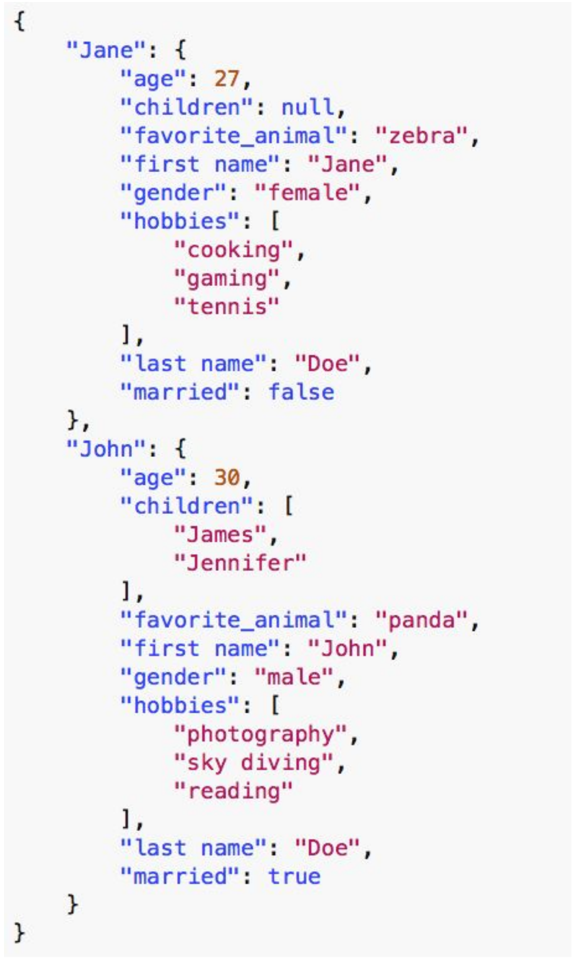
## About JSON

- It uses JavaScript *syntax*, but the format is **just text** and therefore **language-independent**
- The **terminology** might be different, but the **data structures** are the same
- JSON is built on two main structures:
  - **Object:** collection of key-value pairs (=dictionary)
    ➜ usually the top-level structure
  - **Array:** ordered collection of values (=list)



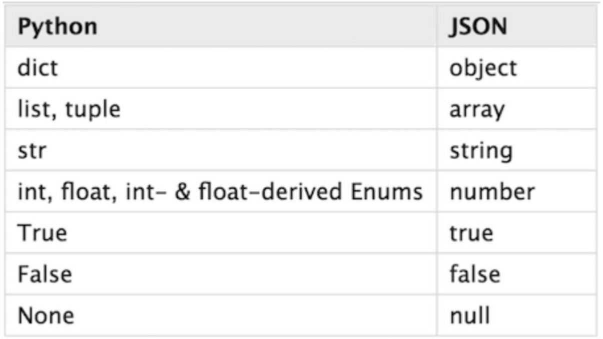
## Example of JSON

What do you recognize in this JSON?

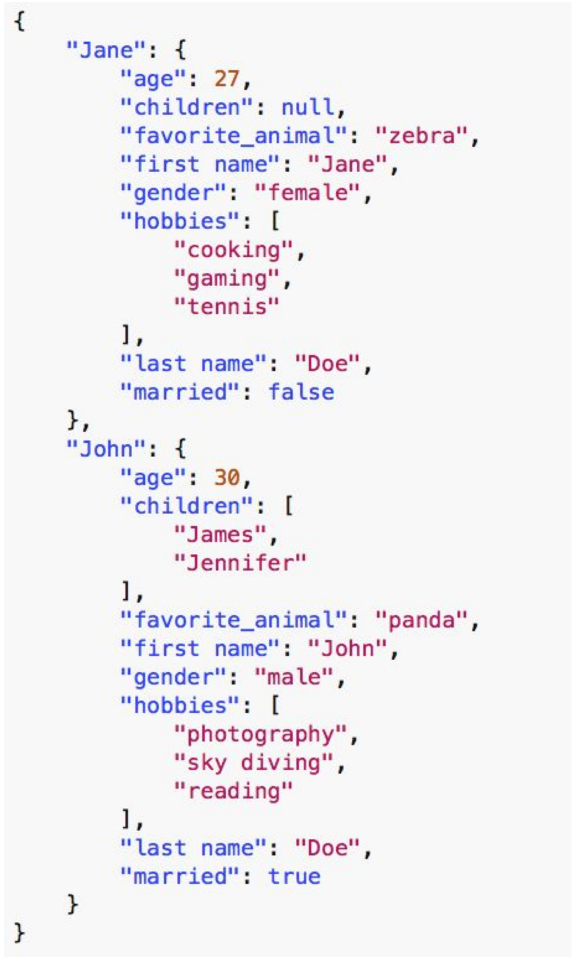


```json
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

## Example of JSON

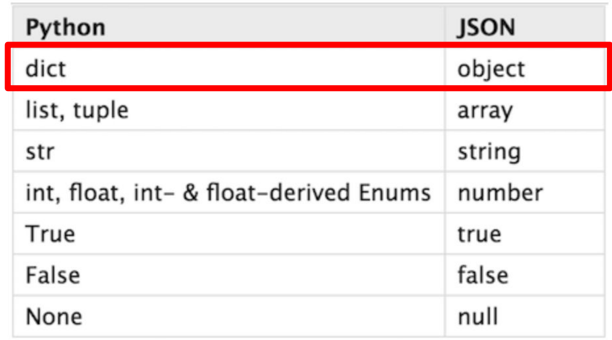


| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |

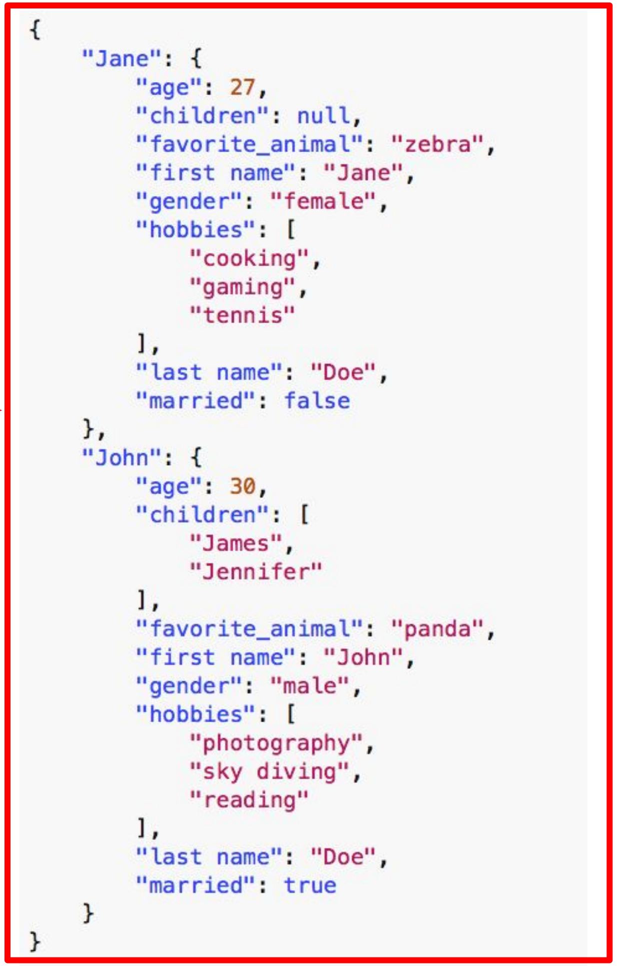


```json
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

# Example of JSON

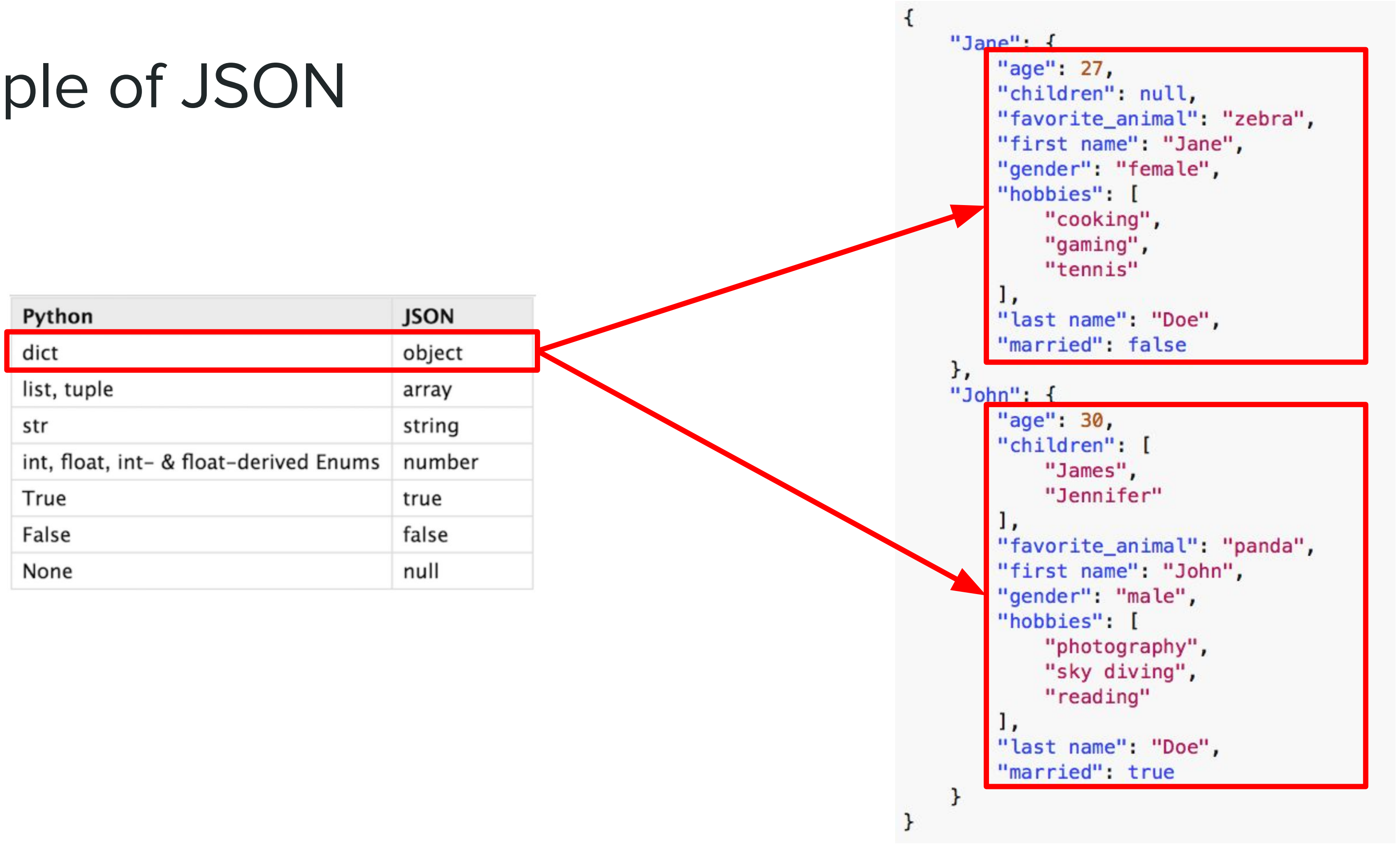


| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |



```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

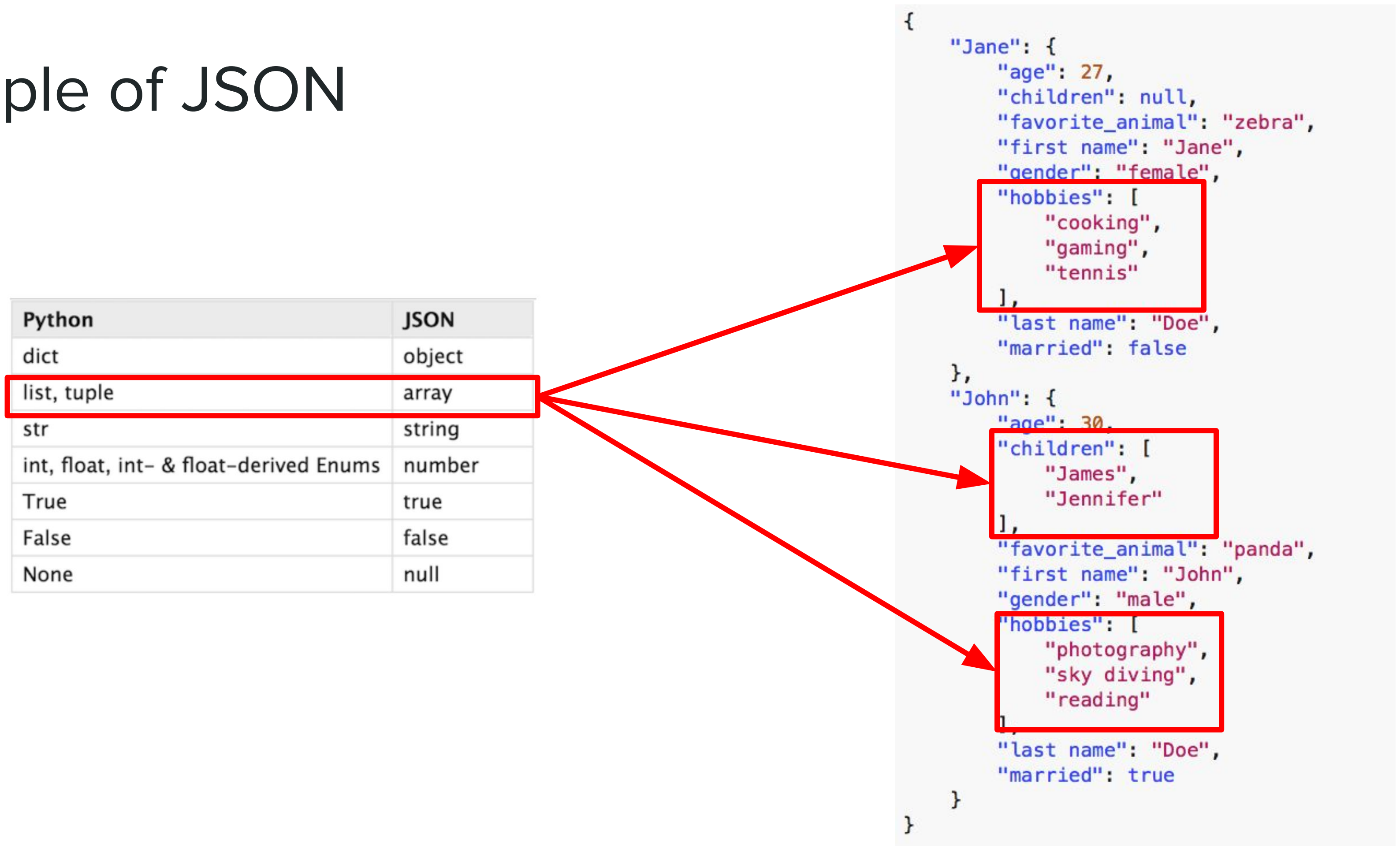
# Example of JSON



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

```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

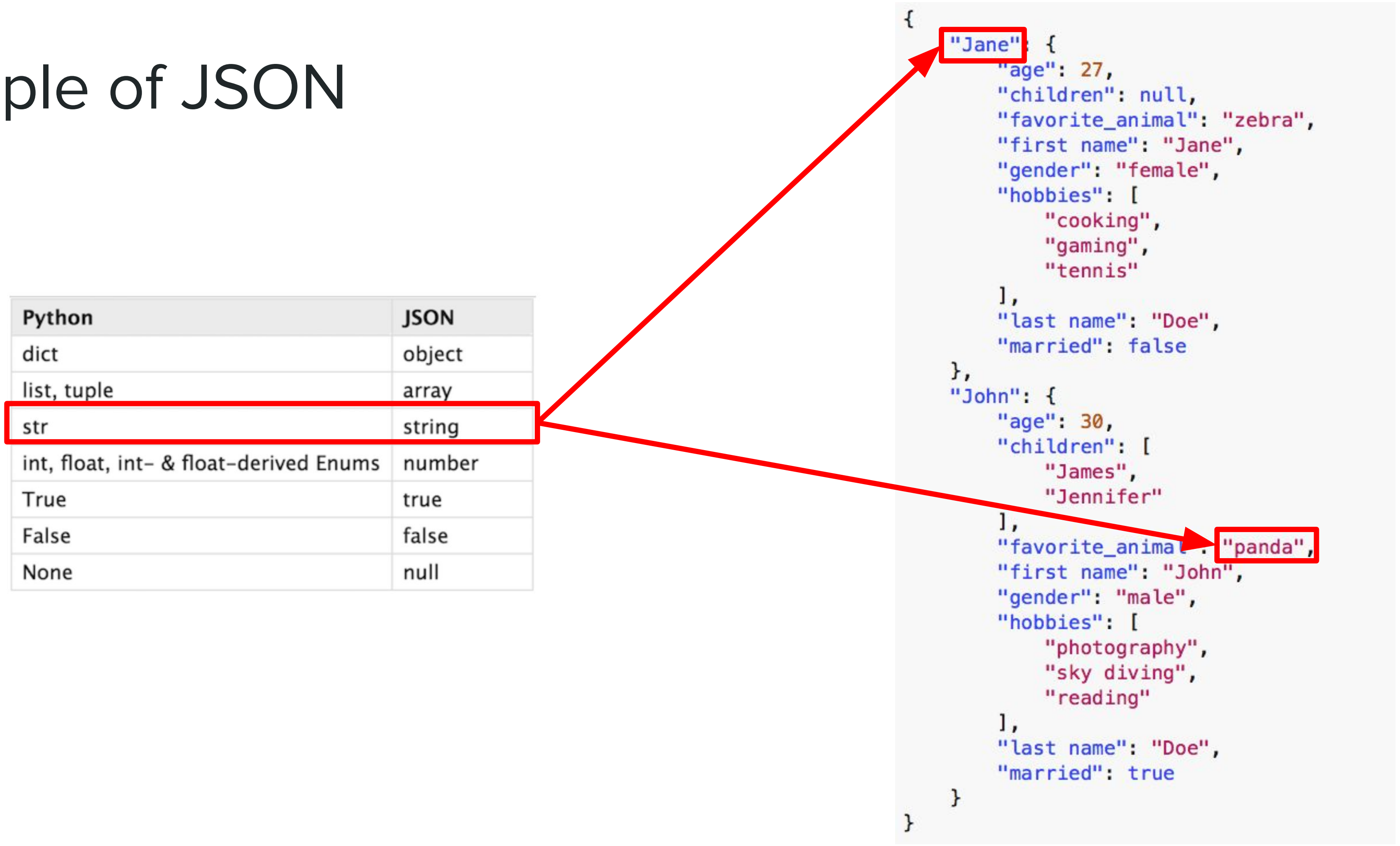
# Example of JSON



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

```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

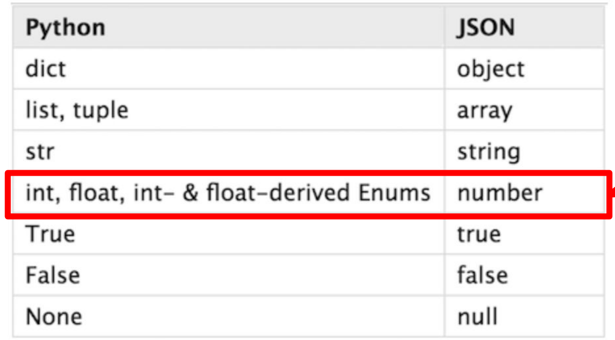
# Example of JSON



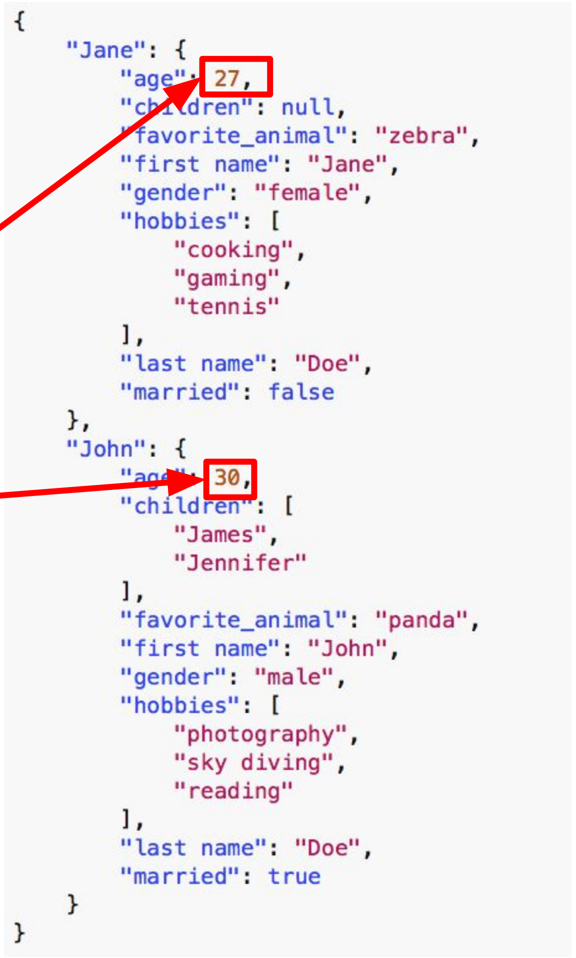
| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |

```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

# Example of JSON

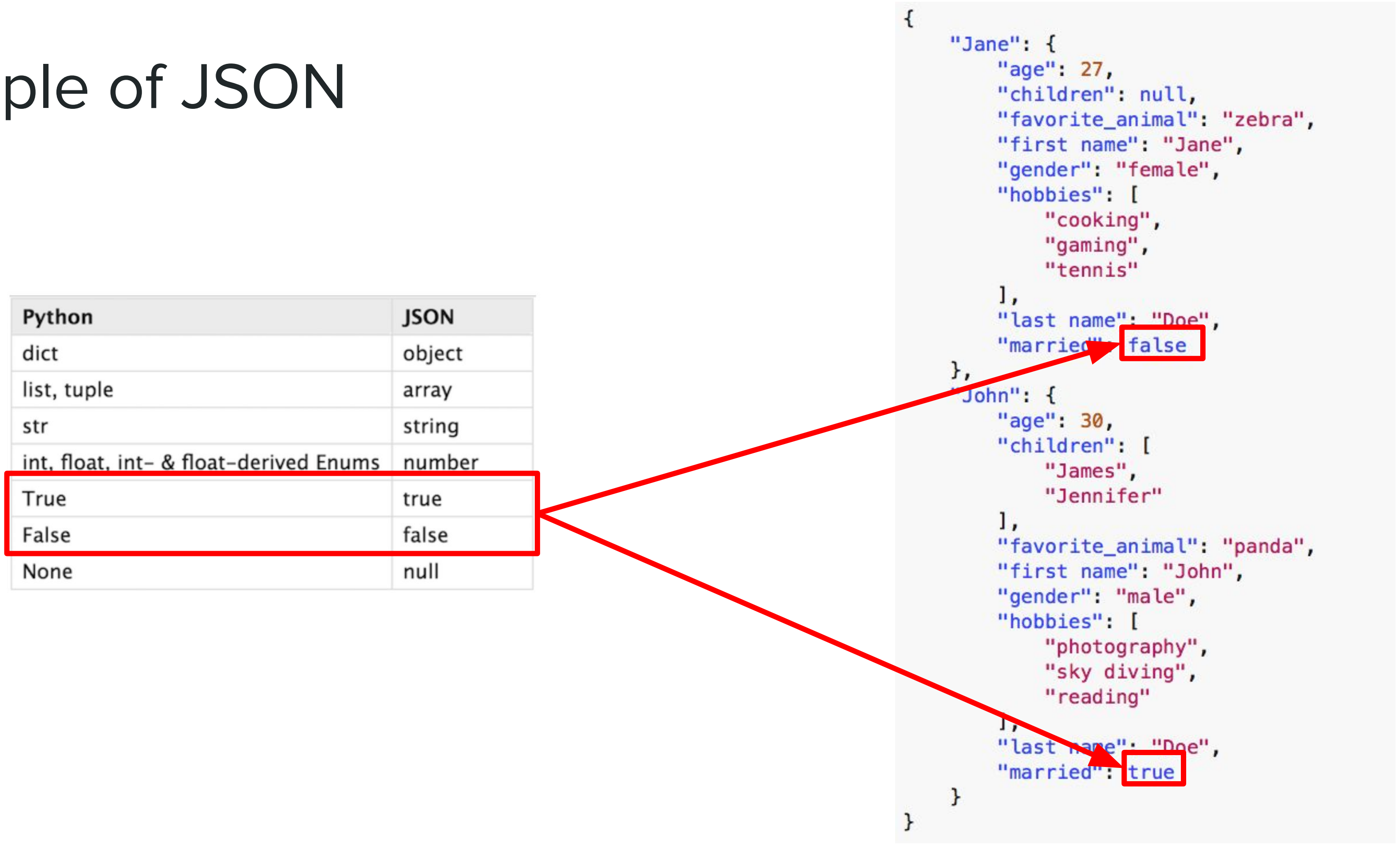


| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |



```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

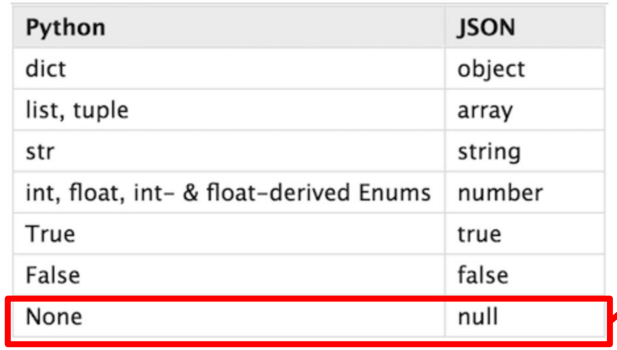
# Example of JSON



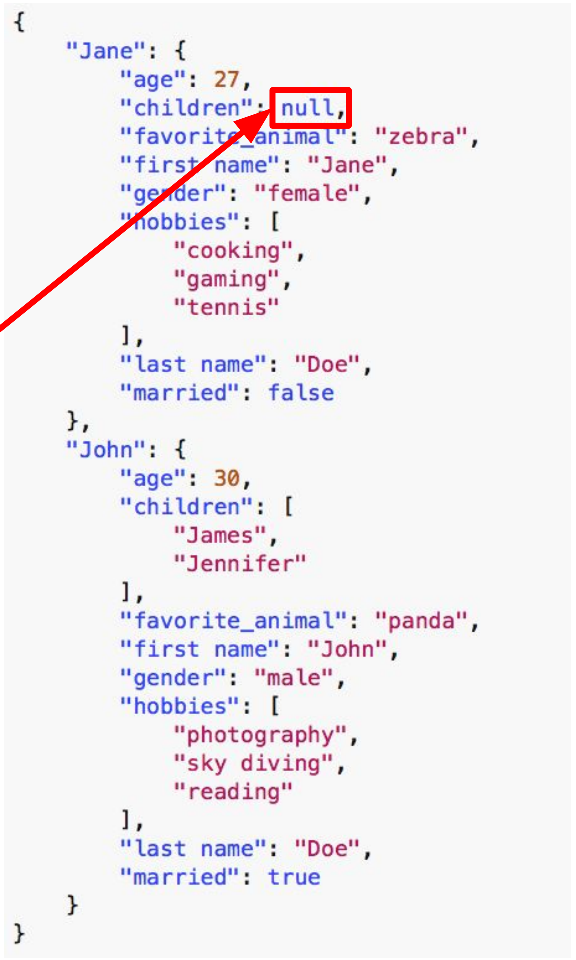
| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |

```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

# Example of JSON



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

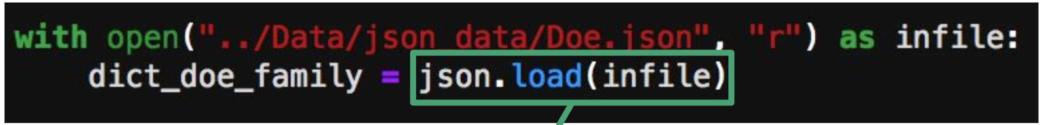


```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

## Reading and writing JSON in Python

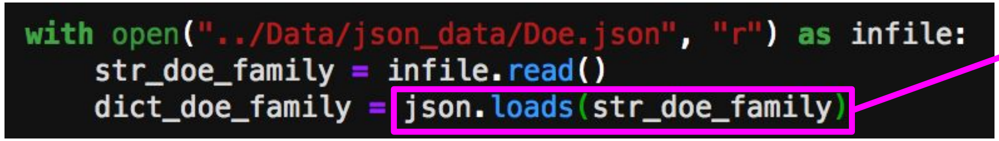
- We use the **json module**
- For **reading** JSON:
  - json.load() → load a JSON formatted **file** as a Python dictionary
  - json.load**s**() → load a JSON formatted **string** as a Python dictionary
- For **writing** JSON:
  - json.dump() → write a Python dictionary to a JSON formatted **file**
  - json.dump**s**() → write a Python dictionary to a JSON formatted **string**

## Reading JSON



```python
with open("../Data/json_data/Doe.json", "r") as infile:
    dict_doe_family = json.load(infile)
```

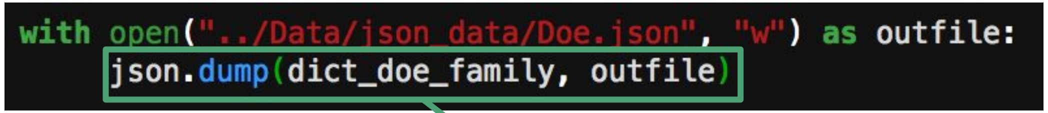
**load()** takes **file object**



```python
with open("../Data/json_data/Doe.json", "r") as infile:
    str_doe_family = infile.read()
    dict_doe_family = json.loads(str_doe_family)
```

**loads()** takes **string**

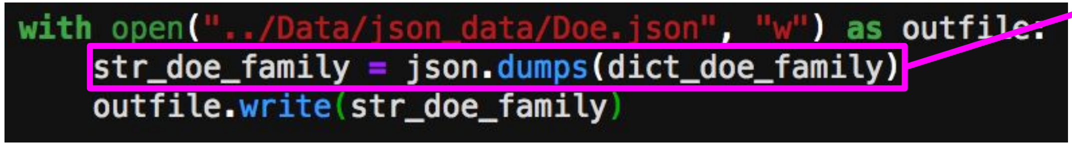
## Writing JSON



```
with open("../Data/json_data/Doe.json", "w") as outfile:
    json.dump(dict_doe_family, outfile)
```

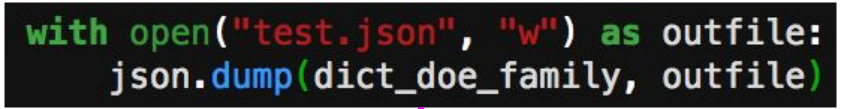
**dump()** writes directly to **file**

**dumps()** writes to **string**

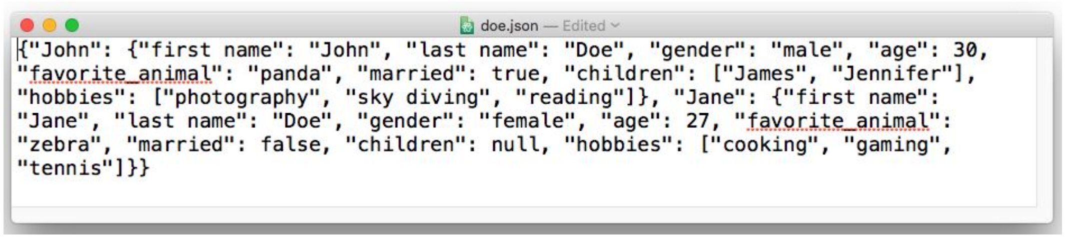


```
with open("../Data/json_data/Doe.json", "w") as outfile:
    str_doe_family = json.dumps(dict_doe_family)
    outfile.write(str_doe_family)
```

## Pretty-printing JSON

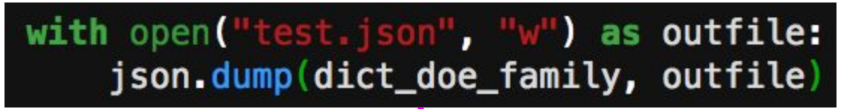


```python
with open("test.json", "w") as outfile:
    json.dump(dict_doe_family, outfile)
```

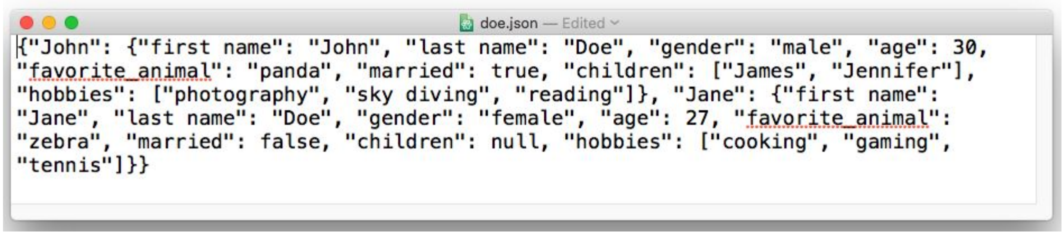


```
{"John": {"first name": "John", "last name": "Doe", "gender": "male", "age": 30,
"favorite_animal": "panda", "married": true, "children": ["James", "Jennifer"],
"hobbies": ["photography", "sky diving", "reading"]}, "Jane": {"first name":
"Jane", "last name": "Doe", "gender": "female", "age": 27, "favorite_animal":
"zebra", "married": false, "children": null, "hobbies": ["cooking", "gaming",
"tennis"]}}
```

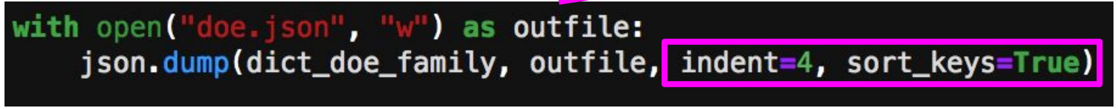
# Pretty-printing JSON



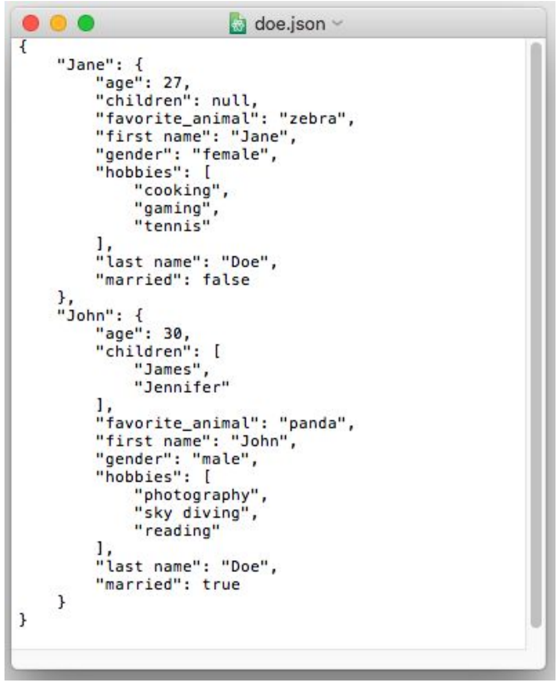
```python
with open("test.json", "w") as outfile:
    json.dump(dict_doe_family, outfile)
```



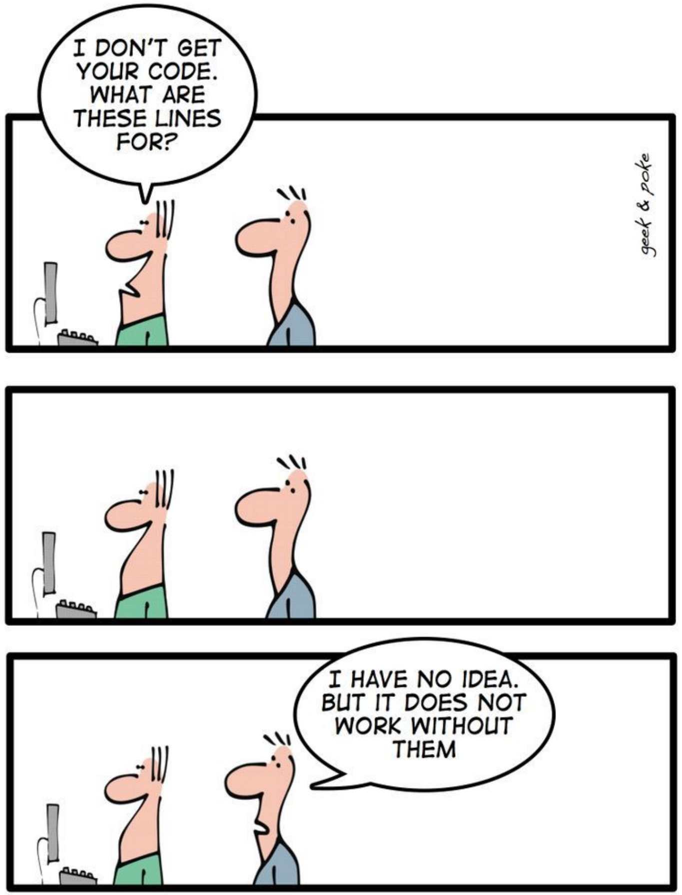
```
{"John": {"first name": "John", "last name": "Doe", "gender": "male", "age": 30,
"favorite_animal": "panda", "married": true, "children": ["James", "Jennifer"],
"hobbies": ["photography", "sky diving", "reading"]}, "Jane": {"first name":
"Jane", "last name": "Doe", "gender": "female", "age": 27, "favorite_animal":
"zebra", "married": false, "children": null, "hobbies": ["cooking", "gaming",
"tennis"]}}
```



```python
with open("doe.json", "w") as outfile:
    json.dump(dict_doe_family, outfile, indent=4, sort_keys=True)
```



```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

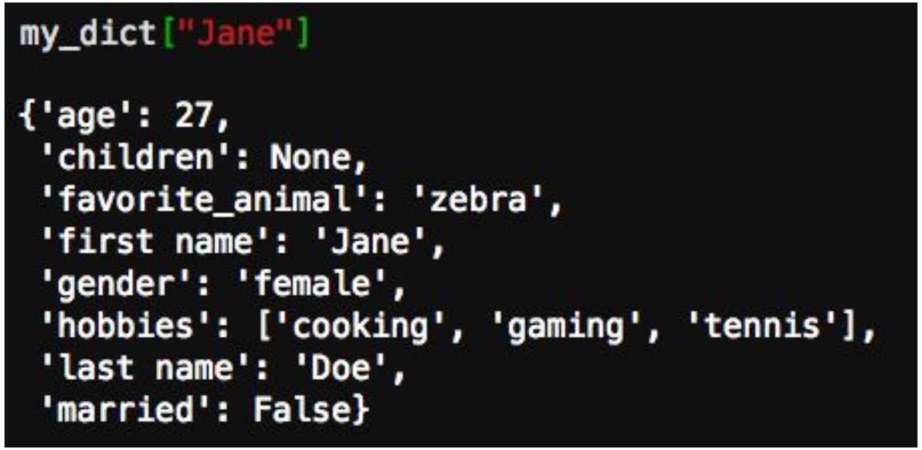

I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
geek & poke
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM

THE ART OF PROGRAMMING - PART 2: KISS

## PART II

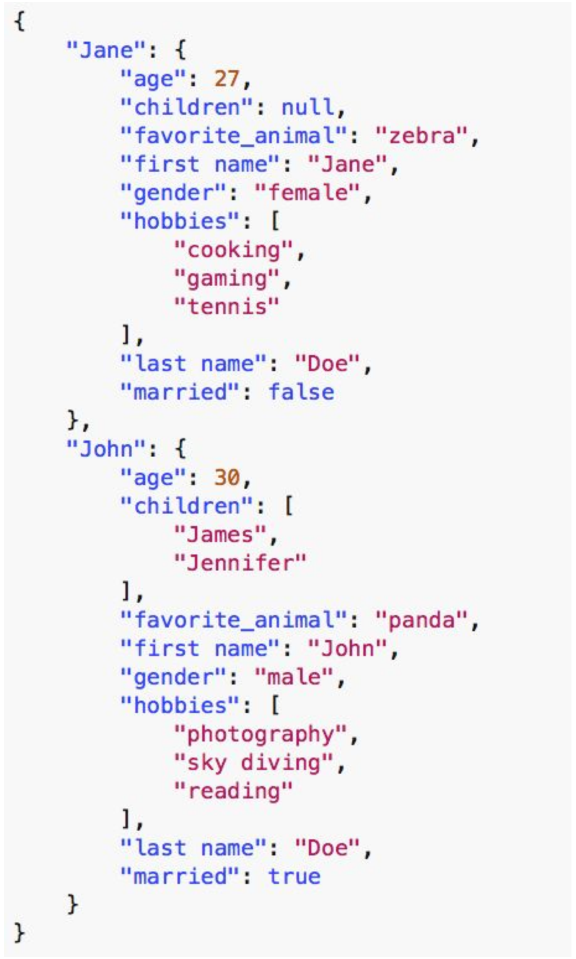
Accessing data from (nested) lists and dicts

## Nested lists and dicts



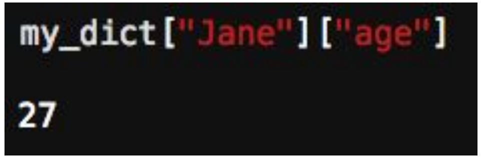
```
my_dict["Jane"]

{'age': 27,
 'children': None,
 'favorite_animal': 'zebra',
 'first name': 'Jane',
 'gender': 'female',
 'hobbies': ['cooking', 'gaming', 'tennis'],
 'last name': 'Doe',
 'married': False}
```



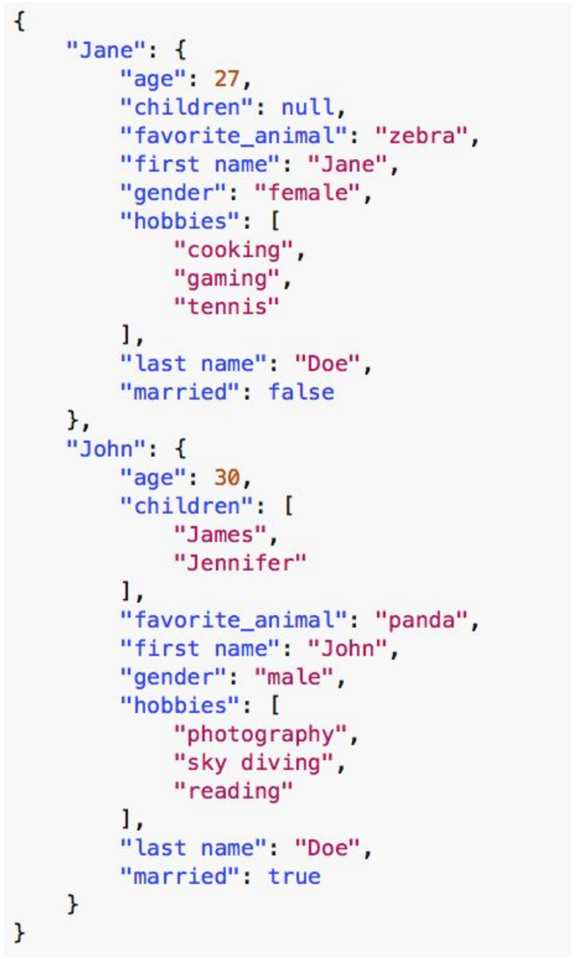
```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

## Nested lists and dicts



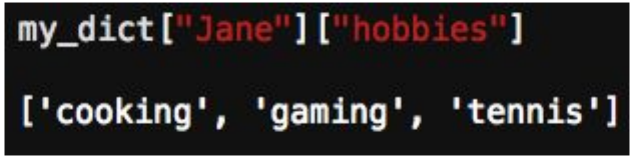
```
my_dict["Jane"]["age"]

27
```



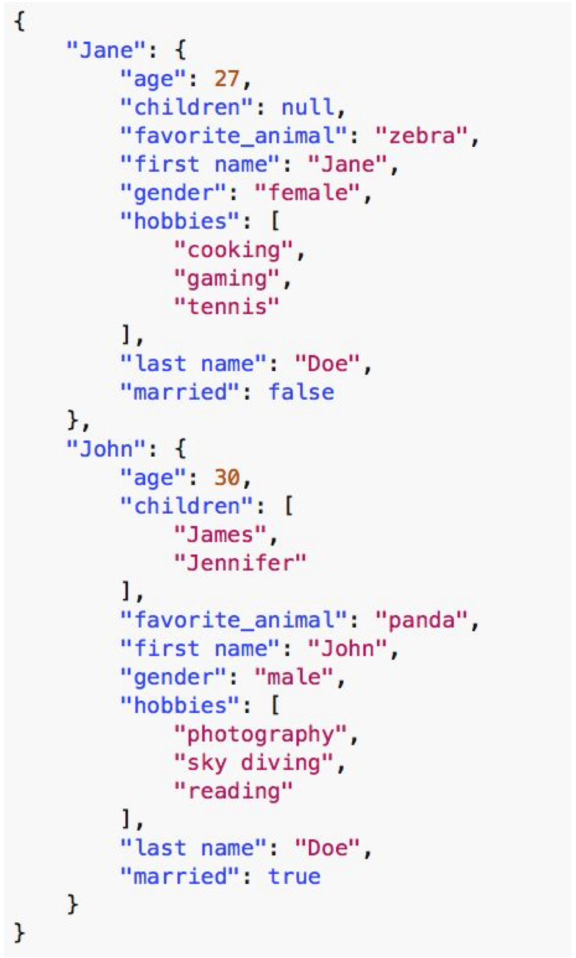
```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

# Nested lists and dicts



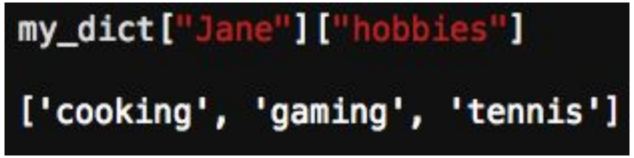
```
my_dict["Jane"]["hobbies"]

['cooking', 'gaming', 'tennis']
```



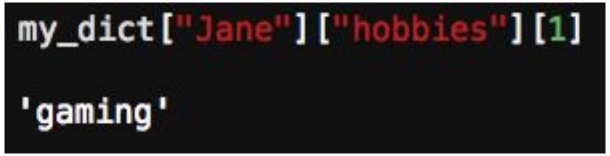
```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

## Nested lists and dicts



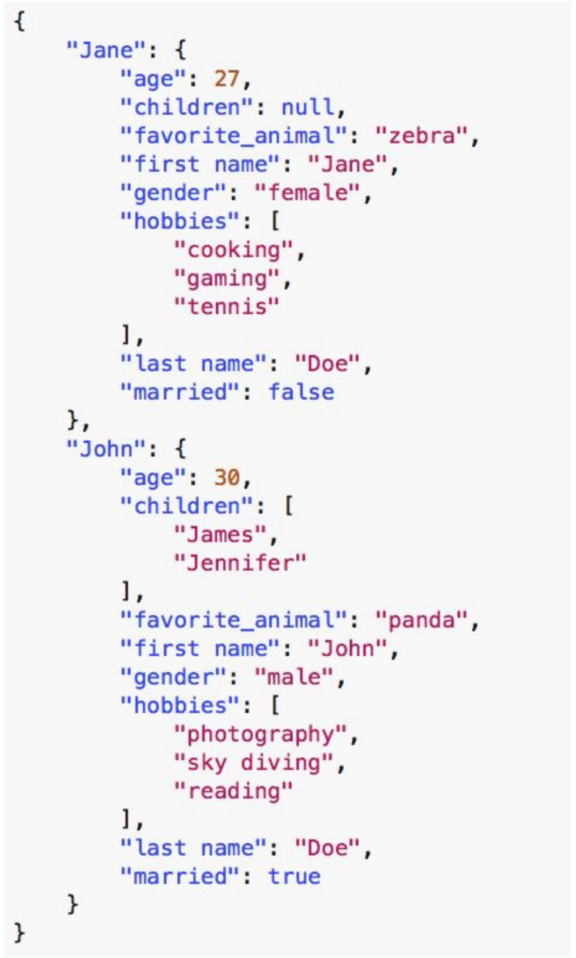
```
my_dict["Jane"]["hobbies"]

['cooking', 'gaming', 'tennis']
```



```
my_dict["Jane"]["hobbies"][1]

'gaming'
```



```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

## Practicing a bit more

**Let’s look at some code :-)**

**Chapter 17, Exercise 4 (Stranger Things)**

## Before Monday

- Finish Chapters 16-18!! (if you haven’t done so already)
- Make sure to be quite far already with Assignment 4a+4b, so that on Monday:
  - You already know a bit about XML
  - You can ask directed questions
  - We can help you where you get stuck on the assignment
  - ...and you’re not stressing out an hour before the deadline
- Deadline of Assignment 4: **Tuesday 4 December at 20:00**