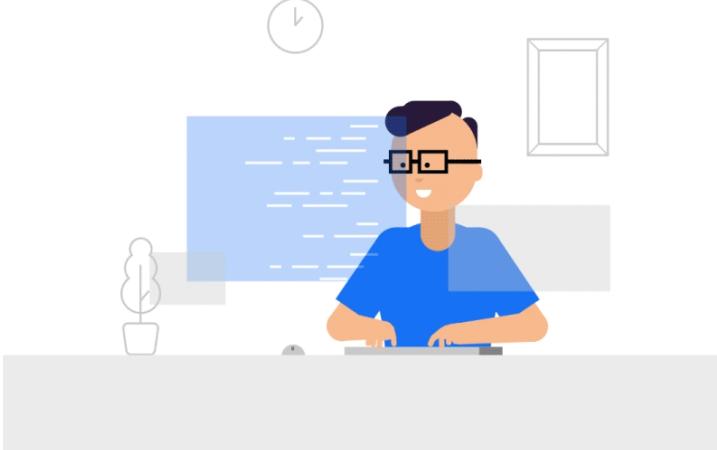


# 04 API

## *Scripting*

Elke Boonen, Dries Geysens & Tristan Vandevelde

# Learning objectives (ECTS)



- Learn the **syntax of Python**
- **Explore possibilities** regarding working with **data** (file, web API, web scraping...)
- Create **scripts**
- **Automate** processes

# Learning materials



- **Canvas** LMS <https://thomasmore.instructure.com/>
  - Presentations & Assignments (CodeGrade)
  - E-Book: Automate the boring stuff with Python (<https://automatetheboringstuff.com>)
- **Online** <https://www.python.org/> and <https://pypi.org/> (search packages)
- **Github** [Scripting](#) (code from slides) & [Scripting-Students](#) (code from class)
- **Software**
  - <https://www.python.org/downloads/> (IDLE Python)
  - <https://www.anaconda.com/>
  - Love to work with Visual Studio Code? Install Python extensions!

# Schedule

Before Easter break	After Easter break
01 Python basics	08 Scheduling
02 Python basics	09 Images
03 Python basics	10 Databases
04 Strings & regex	11 Packaging
05 Files (IO)	12 Conclusion/exam prep
06 API	<i>13 Extra recap</i>
07 Webscraping	

*Schedule is always subject to unexpected circumstances*

# Evaluation

- **1st term**

- Permanent Evaluation (30 %):
  - CodeGrade exercises (each week)
- Take-home exam (70 %)

- **2nd term**

- Computer Exam (100 %)



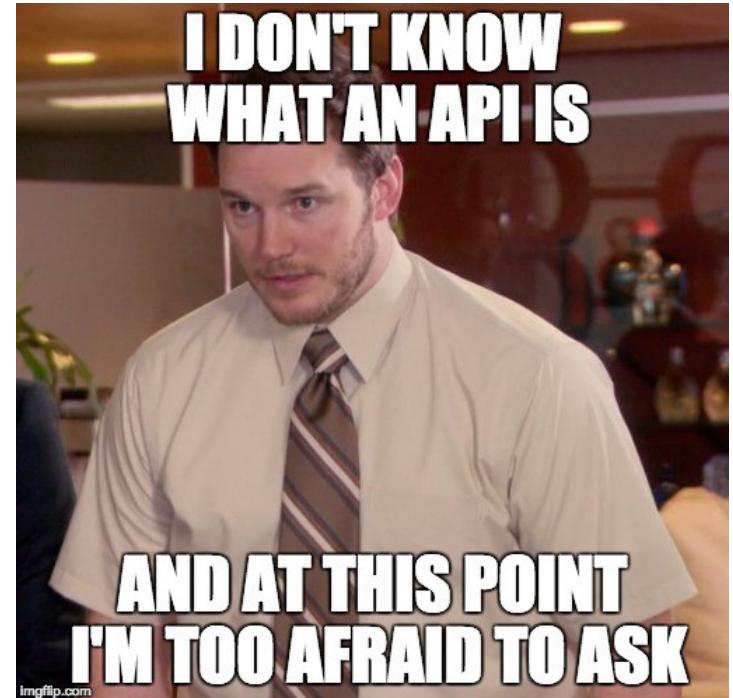
# 04 API

---

- What is a (web)API?
- Working with JSON & XML
- Doing the fun stuff

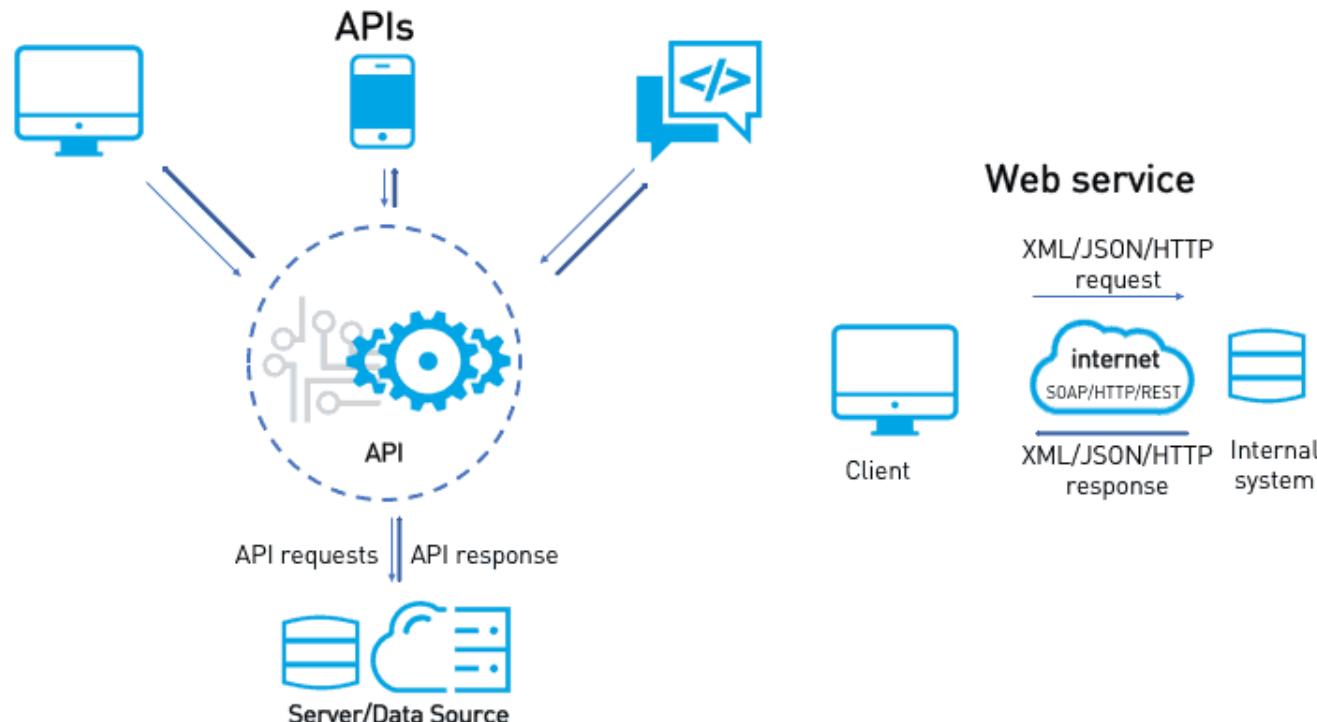
# What is an API?

- Application Programming Interface
- Interface that allows you to build on the data and functionality of another application
- Some examples:
  - We can print our artwork of Paint ;) but Paint itself doesn't need to know how to handle the printer itself, it uses the Print-API!
  - Skyscanner, cheaptickets, momondo.... all gather information about flights, but don't have to know squat about airports ;)



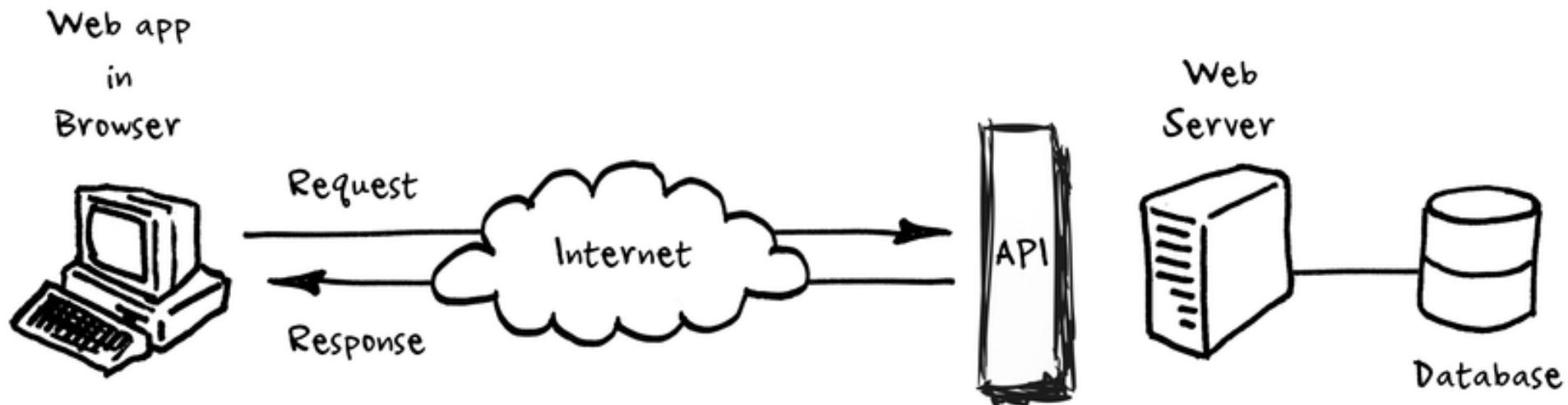
# What is a (web)API?

- A webservice = an API over the internet!
- All webservices are API's but now all API's are websercies!



# Send a request, get a response!

- HTTP request and response
- HTTP response body consists of XML/JSON or regular text



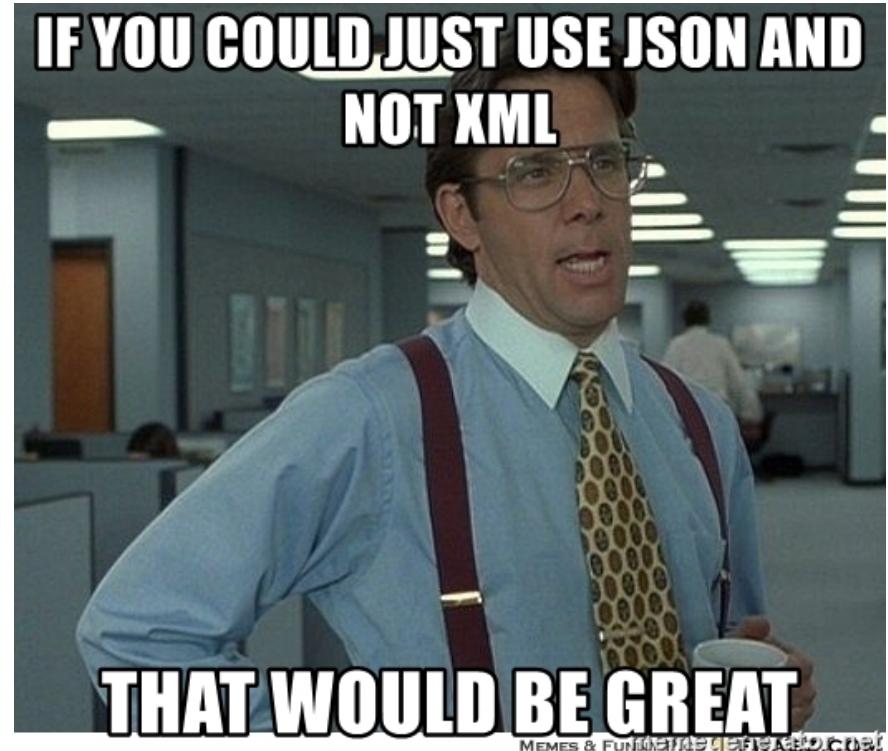
# 04 API

---

- ~~What is a (web)API?~~
- Working with JSON & XML
- Doing the fun stuff

# XML & JSON

- XML: eXtensible Markup Language
  - similar to HTML, works with tags
- JSON: JavaScript Object Notation
- Both are readable & self-describing
- JSON is the 'new' kid in town!



# XML

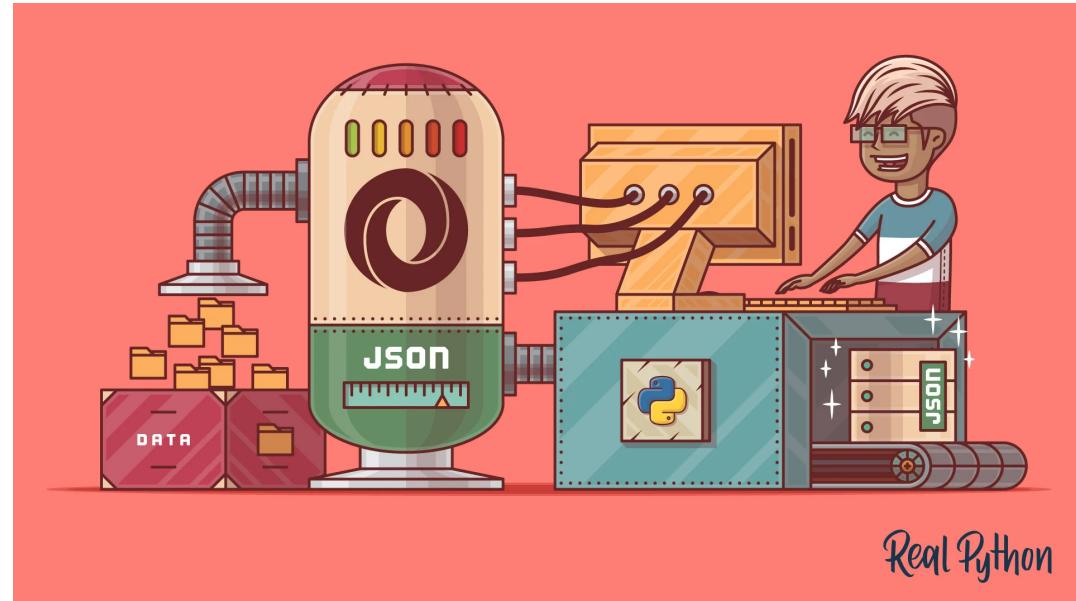
```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

# JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

# JSON module

- **Translation** between string with **JSON data and Python values**
- JSON can store/read the following types: strings, integers, floats, Booleans, lists, dictionaries, and `NoneType`, but no Python-specific objects, such as File objects, Regex objects etc...

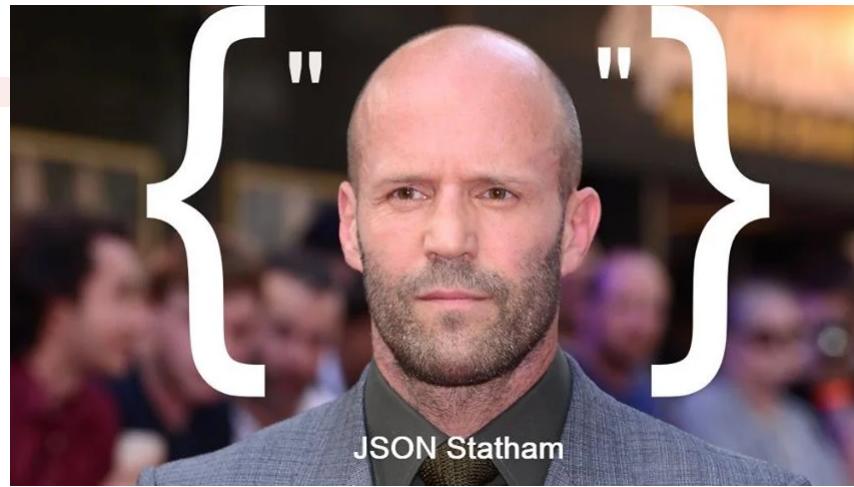


Real Python

# Read JSON

- Import module **json**
  - Use load() or loads()
    - load: file load
    - loads: string load
  - It returns the data as a Python dictionary
- Be aware: Python dictionaries are not ordered!*

```
1 import json
2
3 #string JSON example
4 json_data = '{ "name": "Jason Statham", "age": 54, "city": "Chesterfield"}'
5 y = json.loads(json_data)
6 print(y["age"]) # result is dictionary!
```



# Read JSON from a file

- First, always try to understand the structure of your JSON data
- <https://codebeautify.org/jsonviewer>

The screenshot shows the Code Beautify JSON Viewer interface. On the left, there is a code editor window titled "Code Beautify" containing a JSON file with over 90 lines of code. The JSON structure represents a collection of books. On the right, there is a detailed tree viewer showing the hierarchical structure of the JSON object. The root is an object containing a "books" array. The first item in the array is another object with properties like "isbn", "title", "subtitle", "author", "published", "pages", "description", "publisher", and "website". Below this, there are four additional items in the array, each represented by a triangle icon and a "(9)" label, indicating they are arrays of objects. A large black arrow points from the text "books [8]" in the JSON structure to the first item in the array.

```
object {1}
  books [8]
    0 {9}
      isbn : 9781593275846
      title : Eloquent JavaScript, Second Edition
      subtitle : A Modern Introduction to Programming
      author : Marijn Haverbeke
      published : 2014-12-14T00:00:00.000Z
      publisher : No Starch Press
      pages : 472
      description : JavaScript lies at the heart of almost every modern web application, from social apps to the newest browser-based games. Though simple to pick up and play with, JavaScript is a flexible, complex language you can use to build full-scale applications.
      website : http://eloquentjavascript.net/
    ▶ 1 {9}
    ▶ 2 {9}
    ▶ 3 {9}
    ▶ 4 {9}
```

JSON Viewer☆

Result mode: tree

Load Url

Browse

Tree Viewer

2 Tab Space

Beauty

Minify

Validate

JSON to XML

JSON to CSV

Download

object > books > 0 >

isbn : 9781593275846

title : Eloquent JavaScript, Second Edition

subtitle : A Modern Introduction to Programming

author : Marijn Haverbeke

published : 2014-12-14T00:00:00.000Z

publisher : No Starch Press

pages : 472

description : JavaScript lies at the heart of almost every modern web application, from social apps to the newest browser-based games. Though simple to pick up and play with, JavaScript is a flexible, complex language you can use to build full-scale applications.

website : <http://eloquentjavascript.net/>

▶ 1 {9}

▶ 2 {9}

▶ 3 {9}

▶ 4 {9}

# Read JSON from a file

- Open file
  - pass file-object to load()
  - read content and pass to loads()
- Do something with data!
  - print specific part
  - print all
  - loop dictionary
  - ...

Eloquent JavaScript, Second Edition  
Learning JavaScript Design Patterns  
Speaking JavaScript  
Programming JavaScript Applications  
Understanding ECMAScript 6  
You Don't Know JS  
Git Pocket Guide  
Designing Evolvable Web APIs with ASP.NET

```
1 import json
2
3
4 json_file = open("books.json")
5
6 json_data = json.load(json_file)          # option 1
7 json_data = json.loads(json_file.read())  # option 2
8
9 print(json_data["books"][0]["title"])      ↓
10
11 print(json_data)                         Eloquent JavaScript, Second Edition
12
13 [
14     {
15         "isbn": "9781593275846", "title": "Eloquent JavaScript, Second Edition to Programming", "author": "Marijn Haverbeke", "published": "2014 Press", "pages": 472, "description": "JavaScript lies at the heart of modern web development."}
16
17 for book in json_data["books"]:
18     print(book["title"])
19
20 # don't forget closing the file
21 json_file.close()
```

# JSON write

- Start from a dictionary (or a dictionary of a list of dictionaries :))
- Use dump(): convert python data to a string
  - see any difference between python\_data and json\_data?

```
1 import json
2
3 python_data = { "name": "Jason Statham", "age": 54, "city": "Chesterfield" }
4
5 json_data = json.dumps(python_data)
6 print(python_data)
7 print(json_data)
```

```
{'name': 'Jason Statham', 'age': 54, 'city': 'Chesterfield'}
{"name": "Jason Statham", "age": 54, "city": "Chesterfield"}
```

# JSON write

- Start from a dictionary (or a dictionary of a list of dictionaries :))
- Use dumps(): convert python data to a string and write it to JSON file

```
1 import json
2
3 #creating a dictionary of a list of dictionaries
4 python_data = {}
5 python_data[ 'people' ] = []
6 python_data[ 'people' ].append({
7     'name': 'Jason Statham',
8     'age': '54'
9 })
10 python_data[ 'people' ].append({
11     'name': 'Jason Momoa',
12     'age': '42'
13 })
14 python_data[ 'people' ].append({
15     'name': 'Jennifer Jason Leigh',
16     'age': '59'
17 })
18
19 json_file = open( 'jason.json' , 'w' )
20 json_data = json.dump(python_data, json_file)
21 json_file.close()
```

04 API > {} jason.json > ...

1 Statham", "age": "54"}, {"name": "Jason Momoa", "age":

object ► people ► 2 ►

▼ object {1}

    ▼ people [3]

        ▼ 0 {2}

            name : Jason Statham

            age : 54

        ▼ 1 {2}

            name : Jason Momoa

            age : 42

        ▼ 2 {2}

            name : Jennifer Jason Leigh

            age : 59

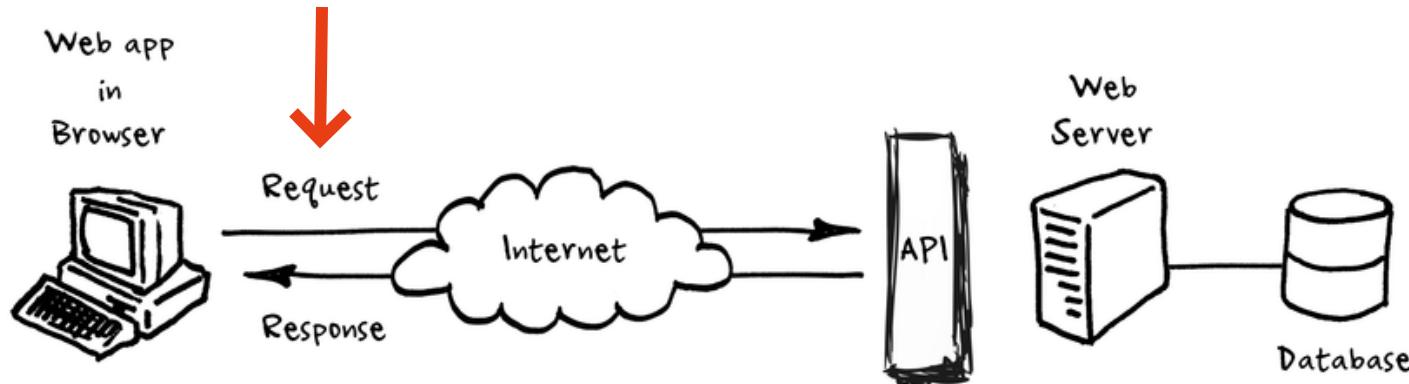
# 04 API

---

- ~~What is a (web)API?~~
- ~~Working with JSON & XML~~
- Doing the fun stuff

# We have a request!

- Asking for data from an API = making a request!
- Install Python package: **requests**
  - pip install requests
- Requests module sends HTTP requests (and gets responses back!)



# We have a request!

- 'Only' reading data from API = **get**!
- HTTP request
  - text-message to server
  - important! https, eg: 'scramble text'  
(s not important when reading data,  
unless it is FBI data ;))
- Check usage in API docs!

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

Diagram illustrating the structure of an HTTP request message:

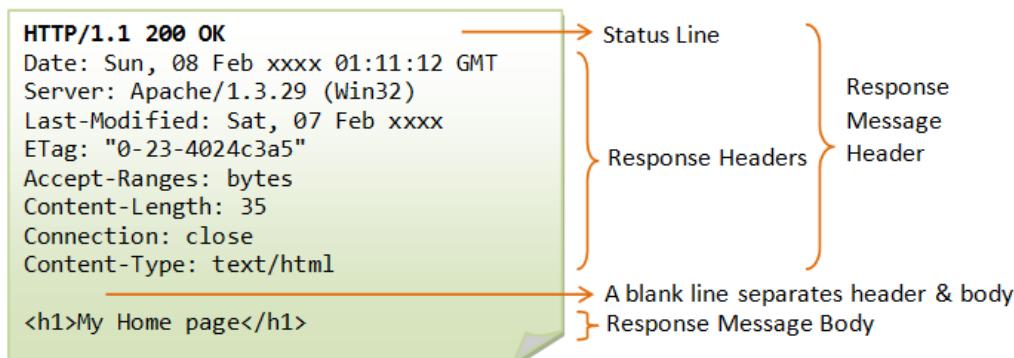
```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
bookId=12345&author=Tan+Ah+Teck
```

The message is divided into several parts:

- Request Line:** GET /doc/test.html HTTP/1.1
- Request Headers:** Host: www.test101.com, Accept: image/gif, image/jpeg, \*/\*, Accept-Language: en-us, Accept-Encoding: gzip, deflate, User-Agent: Mozilla/4.0, Content-Length: 35
- Request Message Header:** A blank line separates the headers from the body.
- Request Message Body:** bookId=12345&author=Tan+Ah+Teck

# We get a response

- Get response back from webserver
- HTTP response
  - status = number
  - message body --> this is where the magic happens with JSON/XML



STATUS CODE	EXPLANATION
200 - OK	The request succeeded.
204 - No Content	The document contains no data.
301 - Moved Permanently	The resource has permanently moved to a different URI.
401 - Not Authorized	The request needs user authentication.
403 - Forbidden	The server has refused to fulfill the request.
404 - Not Found	The requested resource does not exist on the server.
408 - Request Timeout	The client failed to send a request in the time allowed by the server.
500 - Server Error	Due to a malfunctioning script, server configuration error or similar.

# Kanye.rest

- Check <https://kanye.rest/> & check usage
- A useless free API for random Kanye West quotes ;)
- Load URL in JSON viewer to check JSON tree!

## API Usage

```
Select a node...
▼ object {1}
    quote : I feel calm but energized
```



```
GET https://api.kanye.rest  Tweet  Refresh

{
  "quote": "I honestly need all my Royeres to be museum quality... if I see a fake Royere
  Ima have to Rick James your couch"
}

Extra: https://api.kanye.rest?format=text
```



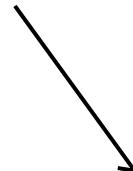
# Kanye.rest

- Import requests and json
- Use get-request!
- Check response!
  - what if URL is wrong?
  - exception handling!

```
1 import requests
2 import json
3
4 try:
5     r = requests.get("https://api.kanye.rest/")
6 except Exception as e:
7     print(e)
```

```
HTTPSConnectionPool(host='api.kanye.rest', port=443):
    connection.HTTPSConnection object at 0x000002AFBA7E65B0
```

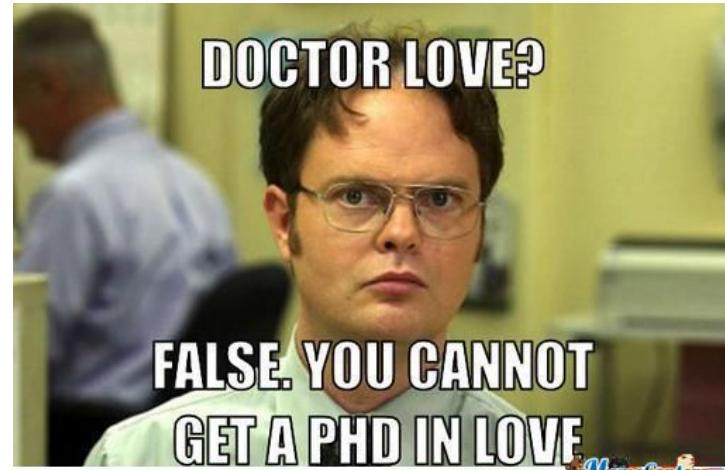
```
1 import requests
2 import json
3
4 response = requests.get("https://api.kanye.rest/")
5 print(response.status_code)
6 print(response.headers)
7 print(response.text)
8 print(response.json)
9
10 json_data = json.loads(response.text)
11 print(json_data["quote"])
```



```
200
{'Date': 'Mon, 22 Mar 2021 14:51:16 GMT',
 'Set-Cookie': '__cfduid=d65c3405cd644c921e143Ao9Gxah6bVIFmfThPTLj%2BaurBCh0w%2F7g%3D; __cf_bm=33f3333333333333333333333333333; __cf_cld=https; HttpOnly; SameSite=Lax', 'Access-Control-Allow-Methods': 'GET', 'cf-request-id': '000002AFBA7E65B0', 'cf-ray': '63403cb33f3333333333333333333333; ma=86400', 'Content-Type': 'application/json; charset=utf-8', 'Content-Length': '113', 'Content-Encoding': 'gzip', 'Connection': 'close', 'Server': 'cloudflare', 'CF-RAY': '63403cb33f3333333333333333333333; ma=86400'}
{"quote": "I'm the new Moses"}
I'm the new Moses
```

# Dr. Love aka the love calculator

- Sometimes you need an API-key!
- Create an account on <https://rapidapi.com/>
- The love calculator!
  - <https://rapidapi.com/ajith/api/love-calculator>
  - Check credentials & usage!



GET getPercentage

Using this to build something with a team? Organizations make collaboration possible. [Create Organization](#)

RapidAPI App default-application\_4441620 REQUIRED

Header Parameters

X-RapidAPI-Key [REDACTED] REQUIRED

X-RapidAPI-Host love-calculator.p.rapidapi.com REQUIRED

Code Snippets Example Responses Results

(Node.js) Unirest [Install SDK](#) [Copy Code](#)

```
var unirest = require("unirest");

var req = unirest("GET", "https://love-calculator.p.rapidapi.com/getPercentage");

req.query({
  "fname": "John",
  "sname": "Alice"
});

req.headers({
  "x-rapidapi-key": "[REDACTED]",
  "x-rapidapi-host": "love-calculator.p.rapidapi.com",
  "useQueryString": true
});

req.end(function (res) {
  if(res.error) throw new Error(res.error);

  console.log(res.body);
});
```

# Dr. Love aka the love calculator

- Build http-request with credentials!

- url
- headers with key
- query-data!

- Check the URL!

```
1 import requests
2 import json
3
4 api_url = "https://love-calculator.p.rapidapi.com/getPercentage"
5 api_headers = {
6     "x-rapidapi-key": "***",
7     "x-rapidapi-host": "love-calculator.p.rapidapi.com"}
8 api_query = {
9     "fname": "Elke",
10    "sname": "Christophe" }
11
12 response = requests.get(api_url, headers = api_headers, params=api_query)
13 print(response.url)
14 print(response.text)
```

<https://love-calculator.p.rapidapi.com/getPercentage?fname=Elke&sname=Christophe>

# Dr. Love aka the love calculator

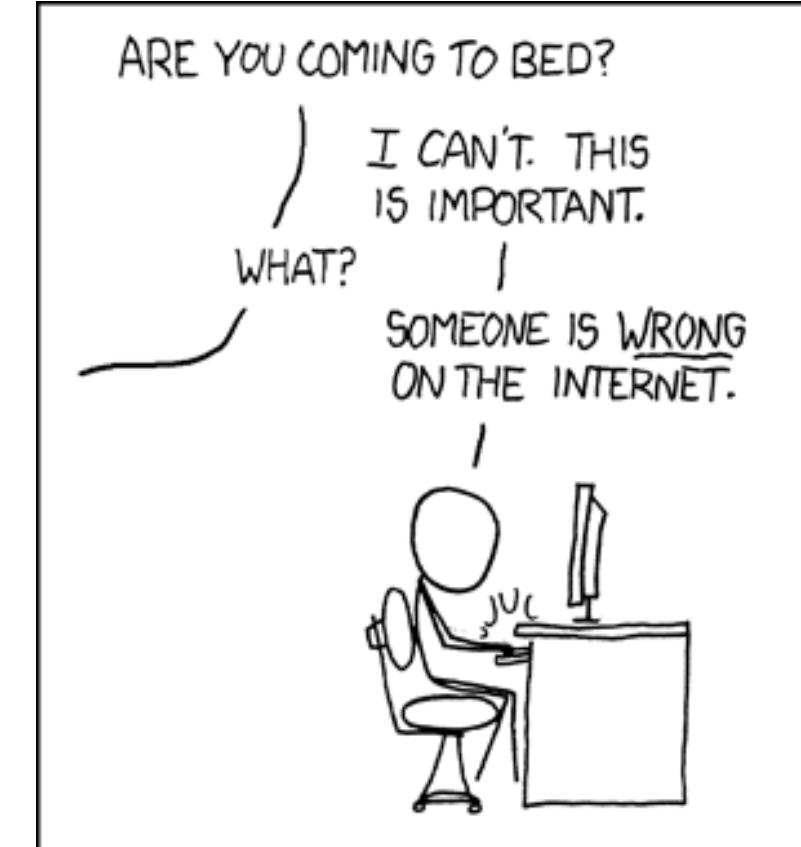
- Go wild ;)

congratulations! good choice, this is sample fancy text.

```
1 import requests
2 import json
3
4 api_url = "https://love-calculator.p.rapidapi.com/getPercentage"
5 api_headers = # your own credentials
6
7 fname = input("Enter first name")
8 sname = input("Enter second name")
9
10 api_query = {
11     "fname": fname,
12     "sname": sname }
13
14 response = requests.get(api_url, headers = api_headers, params=api_query)
15 json_data = json.loads(response.text)
16 result = json_data["result"]
17
18
19 api_url = "https://ajith-fancy-text-v1.p.rapidapi.com/text"
20 api_headers = # your own credentials
21
22 api_query = {
23     "text": result}
24
25 response = requests.get(api_url, headers = api_headers, params=api_query)
26 json_data = json.loads(response.text)
27 print(json_data["fancytext"])
```

# API's or packages?

- Some free APIs also have Python packages
- eg: XKCD (webcomics)
  - API : <https://xkcd.com/json.html>
  - Python package: pip install xkcd  
(<https://pypi.org/project/xkcd/>)
- Always go for the API (no installation needed!) because most of the time the package is just a wrapper!



# Practice makes perfect!

- Check out the extra (not graded) exercises in the notebooks! See [Github](#)
- Do your exercises, spend the hours!

*Say what? How many hours?*

$$6 \text{ SP} = 6 * 28 \text{ hours} = 168 \text{ hours}$$

$$\text{Lessons} = 12 * 3 \text{ hours} = 36 \text{ hours}$$

$$\text{Take-home exam} = 16 \text{ hours}$$

**Exercise = 168-36-10 = 116 hours**

