Python | Main course

# Session 13

- Regex
- Python Regex
- HTTP requests
- Python requests module
- JSON
- API

by Mohammad Amin H.B. Tehrani - Reza Yazdani

www.maktabsharif.ir

# Regex

Regex

# Example

Write a python function that validates emails string.

Hint:
Email addresses only contains: words, digits, dots, periods
+ contains '@' character,
+ a valid domain name or IP

```
def email_validator(email) -> bool:
    # TODO: Code here
    ...

# example@email.co -> True
# exampleemail.co -> False
# akbar -> False
# asd @ gmail.com -> False
# akbar.babaii@yahoo.com -> True
```

Valid test cases:
- email@example.com
- firstname.lastname@example.com
- email@subdomain.example.com
- firstname+lastname@example.com
- email@123.123.123.123

Invalid test cases:
- plain address
- #@%^%#$@#$@#.com
- @example.com
- Joe Smith <email@example.com>
- email.example.com
- email@example@example.com

## Intro

A **RegEx**, or **Regular Expression**, is a sequence of characters that forms a search pattern.
RegEx can be used to check if a string contains the specified search pattern.

Examples:

- `(\d{1,3})(\.)(\d{1,3})(\.)(\d{1,3})(\.)(\d{1,3}): IP address`
  ```
  -> 11.2.1.2
  -> 127.0.0.1
  -> ...
  ```
- `(\d{4})[\.\-\/](\d{2})[\.\-\/](\d{2}): Date`
  ```
  -> 1922-02-02
  -> 1340/02/01
  -> ...
  ```
- `(www.)?([\w\-]+\.)?([\w\-]+)\.([\w\-]{2,})(\/.*)?: ?`

Regex

## Metacharacters

| Character | Description | Example |
|---|---|---|
| [] | A set of characters | "[a-m]" |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" |
| . | Any character (except newline character) | "he..o" |
| ^ | Starts with | "^hello" |
| $ | Ends with | "world$" |
| * | Zero or more occurrences | "aix*" |
| + | One or more occurrences | "aix+" |
| {} | Exactly the specified number of occurrences | "al{2}" |
| \| | Either or | "falls\|stays" |

Regex

# Special Sequences

| Character | Description | Example |
|---|---|---|
| \A | Returns a match if the specified characters are at the beginning of the string | "\AThe" |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word<br>(the "r" in the beginning is making sure that the string is being treated as a "raw string") | r"\bain"<br>r"ain\b" |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word<br>(the "r" in the beginning is making sure that the string is being treated as a "raw string") | r"\Bain"<br>r"ain\B" |
| \d | Returns a match where the string contains digits (numbers from 0-9) | "\d" |
| \D | Returns a match where the string DOES NOT contain digits | "\D" |
| \s | Returns a match where the string contains a white space character | "\s" |
| \S | Returns a match where the string DOES NOT contain a white space character | "\S" |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character) | "\w" |
| \W | Returns a match where the string DOES NOT contain any word characters | "\W" |

Regex

## Sets

| Set | Description |
|---|---|
| [arn] | Returns a match where one of the specified characters (a, r, or n) are present |
| [a-n] | Returns a match for any lower case character, alphabetically between a and n |
| [^arn] | Returns a match for any character EXCEPT a, r, and n |
| [0123] | Returns a match where any of the specified digits (0, 1, 2, or 3) are present |
| [0-9] | Returns a match for any digit between 0 and 9 |
| [0-5][0-9] | Returns a match for any two-digit numbers from 00 and 59 |
| [a-zA-Z] | Returns a match for any character alphabetically between a and z, lower case OR upper case |
| [+] | In sets, +, *, ., \|, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string |

## Some useful references...

- https://regexr.com/
  A editor, document,  reference, community for Regex.

- https://www.w3schools.com/python/python_regex.asp
  Python Regex reference.

# Python Regex

## Regex module

Python has a built-in package called **re**, which can be used to work with Regular Expressions.

Syntax:

```
import re
```

```python
import re

# Check if the string starts with "The" and ends with "Spain":
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)

if x:
   print("YES! We have a match!")
else:
   print("No match")
```

Regex

## findall() method

The **findall()** function returns a list containing all matches.

```
import re

txt = """Python was conceived in the late 1980s[38] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in
the Netherlands as a successor to ABC programming language, which was inspired by SETL,[39] capable of exception
handling and interfacing with the Amoeba operating system.[9] Its implementation began in December 1989.[40] """

references = re.findall("(\[[\d+\])", txt)
print(references)
```

```
['[38]', '[39]', '[9]', '[40]']
```

```
import re

txt = """Akbar's reign was chronicled extensively by his court historian Abul Fazl in the books Akbarnama and
Ain-i-akbari. Other contemporary sources of Akbar's reign include the works of Badayuni, Shaikhzada Rashidi and
Shaikh Ahmed Sirhindi."""

upper_cases = re.findall("([A-Z]\w*)", txt)
print(upper_cases)
```

```
['Akbar', 'Abul', 'Fazl', 'Akbarnama', 'Ain', 'Other', 'Akbar',
'Badayuni', 'Shaikhzada', 'Rashidi', 'Shaikh', 'Ahmed', 'Sirhindi']
```

Regex

# search() method

The **search()** function searches the string for a match, and returns a **Match** object if there is a match.
If there is more than one match, only the first occurrence of the match will be returned:

```
import re

txt = """Non tempora amet 1994-02-24 18:26:25.680292 est. Sed dolor labore ut labore velit porro tempora.
Quisquam
dolor non voluptatem. Numquam quiquia adipisci dolore eius numquam amet voluptatem.
14:39:40.982917 est. Ut tempora quisquam amet  1998-03-16 16:14:16.647591..."""

pattern = r"(\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}(.\d+)?)"
timestamp = re.search(pattern, txt)
print(timestamp)
```

```
<re.Match object; span=(17, 43), match='1994-02-24 18:26:25.680292'>
```

Regex

# finditer() method

Return an iterator yielding Match Object instances over all non-overlapping matches for the RE pattern in string.

```
import re

txt = """Non tempora amet 1994-02-24 18:26:25.680292 est. Sed dolor labore ut labore velit porro tempora.
Quisquam
dolor non voluptatem. Numquam quiquia adipisci dolore eius numquam amet voluptatem.
14:39:40.982917 est. Ut tempora quisquam amet  1998-03-16 16:14:16.647591..."""

pattern = r"(\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}(.\d+)?)"
for ts in re.finditer(pattern, txt):
   print(ts)
```

```
<re.Match object; span=(17, 43), match='1994-02-24 18:26:25.680292'>
<re.Match object; span=(295, 321), match='1998-03-16 16:14:16.647591'>
<re.Match object; span=(347, 373), match='2006-02-04 09:14:57.833855'>
...
```

Regex

## Match object

A **Match Object** is an object containing information about the search and the result.

The Match object has properties and methods used to retrieve information about the search, and the result:

- .span() returns a tuple containing the start-, and end positions of the match.
- .string returns the string passed into the function
- .group() returns the part of the string where there was a match
- .groups()  returns all groups tuple
- .groupdict()  returns all groups dict

## HTTP requests

HTTP requests

## Intro

### What is HTTP?

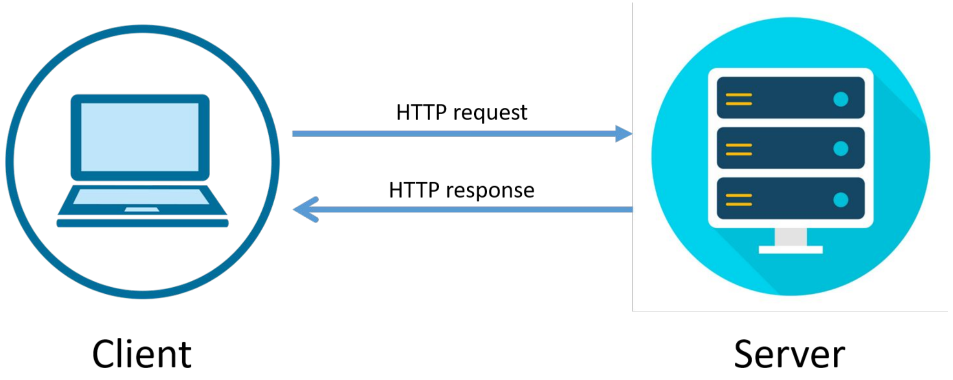
The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a **request-response protocol** between a client and server.

Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

### HTTP Methods

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS



HTTP requests

## GET request

**GET is used to request data from a specified resource.**

**GET is one of the most common HTTP methods.**

Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
http://google.com
```

```
http://github.com/maktab78
```

```
https://github.com/orgs/maktab78/repositories?q=&language=python
```

# POST request

**POST is used to send data to a server to create/update a resource.**

The data sent to the server with POST is stored in the request **body** of the HTTP request:

HOW TO SEND POST REQUESTS?

# Curl

**Use cURL to request a server**
cURL is a computer software project providing a library and command-line tool for transferring data using various network protocols.
cURL[edit]. cURL is a command-line tool for getting or sending data including files using URL syntax.

```
  ╭─yazdan@DarkBook in repo: Python78 via  v3.10.5 (venv) took 3ms
  ╰─λ curl https://icanhazip.com
123.231.123.321
```

# Python requests module

Python requests module

# Intro

The **requests** module allows you to send HTTP requests using Python.

The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

Install:

```
pip install requests
```

Import:

```
import requests
```

```python
import requests

url = 'https://icanhazip.com '
method = 'GET'
response = requests.request(method, url)

print(response.text)
```

```
123.321.123.321
```

Python requests module

## Methods

|  |  |
|---|---|
| ● **delete(url, args)** | Sends a DELETE request to the specified url |
| ● **get(url, params, args)** | Sends a GET request to the specified url |
| ● **head(url, args)** | Sends a HEAD request to the specified url |
| ● **patch(url, data, args)** | Sends a PATCH request to the specified url |
| ● **post(url, data, json, args)** | Sends a POST request to the specified url |
| ● **put(url, data, args)** | Sends a PUT request to the specified url |
| ● **request(method, url, args)** | Sends a request of the specified method to the specified url |

Python requests module

## Example

GET /maktab64/?name=akbar

```
import requests

url = 'http://ma-web.ir/maktab64/'
method = 'GET'
get_response = requests.request(method , url, params={'name': 'akbar'})   # = request.get(url,
...)
print(get_response.content)
```

```
<H1>GET</H1><p style='color:blue'>Hello akbar!</p>
```

POST /maktab64

```
import requests

url = 'http://ma-web.ir/maktab64/'
get_response = requests.post(url , data={'name': 'akbar'})   # = request.get(url, ...)
print(get_response.text)
```

```
<H1>POST</H1><p style='color:red'>Hello akbar!</p>
```

Python requests module

# BeautifulSoup

Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

Installation:
```
pip install beautifulsoup4
```

Import:
```
from bs4 import BeautifulSoup
```

```
from bs4 import BeautifulSoup

bs = BeautifulSoup('<h1>Hello world!</h1>')
print(bs.prettify())
print(bs.find(text='Hello world!'))
print(bs.find_all(name='h1'))
```

```
<html>
 <body>
  <h1>
   Hello world!
  </h1>
 </body>
</html>
Hello world!
[<h1>Hello world!</h1>]
```

## JSON

JSON

## Intro

### JavaScript Object Notation

- **JSON** stands for JavaScript Object Notation
- **JSON** is a lightweight format for storing and transporting data
- **JSON** is often used when data is sent from a server to a web page
- **JSON** is "self-describing" and easy to understand

### JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

### Syntax:

- **“Key”: “value”**
- {...} object notation (contains key-value pairs)
- [...] list notation (contains objects)

```
{
 "users": [
   {
     "firstName": "Akbar",
     "lastName": "Babaii",
     "marks": [20,20,19,18,20,19],
     "address": {
       "country": "iran",
       "city": "Isfahan",
       "street": "..."
     }
   },
   {
     "firstName": "Foo",
     "lastName": "Bar",
     "marks": [9,7,14,13,3,15],
     "address": {
       "country": "USA",
       "city": "New York",
       "street": "..."
     }
   }
 ]
}
```

# Python json module

Python has a built-in package called **json**, which can be used to work with JSON data.
Use can simply **load** json file (Deserialize python objects from strings)
Or **dump** json file (Serialize python objects into json string)

Methods:

- **loads():** Deserialize to a Python object.
- **dumps():** Serialize obj to a JSON formatted str.
- **load():** Deserialize a file to a Python object.
- **dump():** Serialize obj as a JSON formatted stream to file.

```
content = {
    'users': [
        {
            'first_name': 'akbar',
            'last_name': 'babaii',
            'phone': '09371237654',
            'is_admin': True
        }
    ]
}
```

```
import json

# Serialize content in to json file
with open('test.json', 'w') as f:
    json.dump(content, f)
```

```
import json

# Deserialize content from .json file
with open('test.json', 'r') as f:
    content = json.load(f)
```

## API

JSON

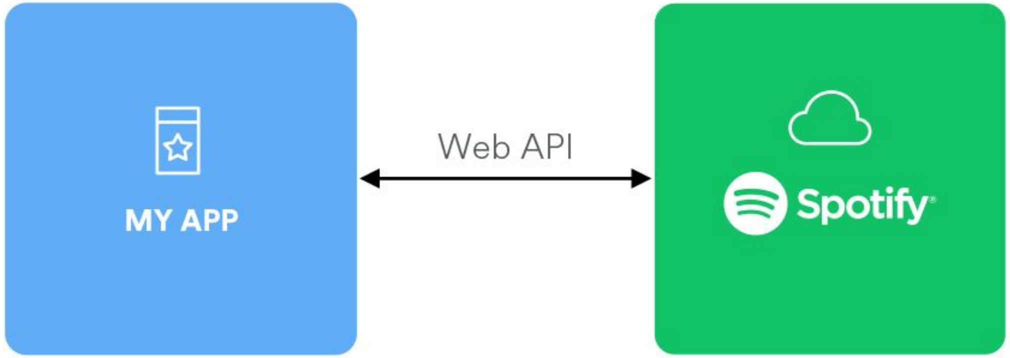
## API

**What is Web API?**

- API stands for Application Programming Interface.
- A Web API is an application programming interface for the Web.
- A Browser API can extend the functionality of a web browser.
- A Server API can extend the functionality of a web server  other.

How can application talk to each other??

**API** is the acronym for Application Programming Interface, which is a software intermediary **that allows two applications to talk to each other.** Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.


MY APP
Web API
Spotify