Data processing technologies (TTD)

Lesson plan

Class 01

XML

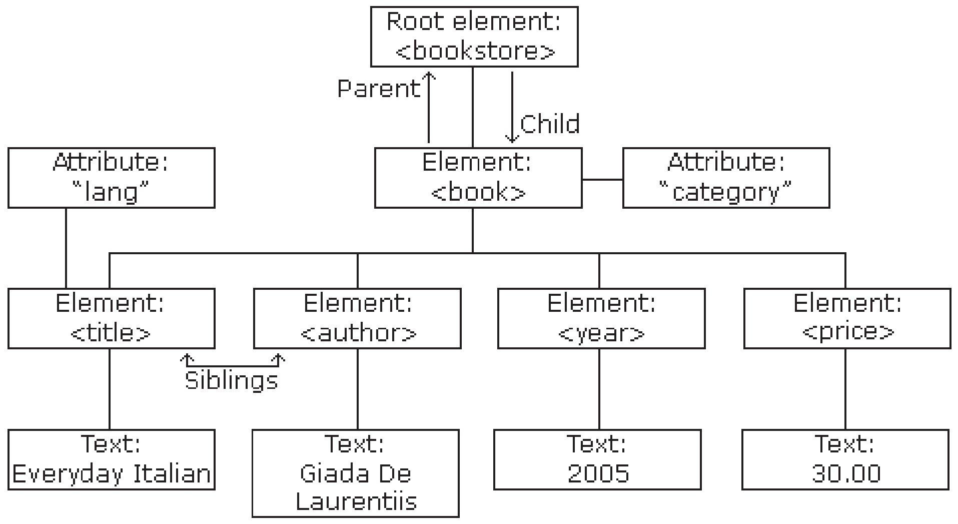
- Standing for *eXtensible Markup Language*
- Markup language used to store and transport data
- Designed to be both human and machine readable.
- Less popular in the last couple of years = being replaced with JSON)
- Still plays an important role in many different IT systems
  (used in many aspects of web development)
- Doesn’t depend on platform nor software nor programming language
  = it is possible to write a program in any language on any platform to send, receive or store data using XML
- Llike PHP includes = XML is a complement to HTML
  used to separate data from presentation
- XML doesn't actually do anything
  Simply used to structure data. = XML file can then be used in a program to display the data

XML structure

- XML resembles HTML = uses opening and endings tags
  BUT self-describing syntax = no predefined tags

```
<root>
        <child>
                <subchild>.....</subchild>
        </child>
</root>
```

- In XML = tags are created accordingly to the database needs
- Fields are grouped withing entries,
  entries are grouped within a table in a child-sibling relationship



```
<?xml version="1.0" encoding="UTF-8"?>          // XML prolog
<booklist>                                      // Table
        <book>                                  // Entry
                <title>Harry Potter</title>     // Field
                <author>J K. Rowling</author>
                <year>2005</year>
                <price>29.99</price>
        </book>
        <book>
                <title>The flowers of evil</title>
                <author>Charles Baudelaire</author>
                <year>1857</year>
                <price>39,95</price>
        </book>
</booklist>
```

Viewing XML files

- XML document must be saved using XML extension
- Can be opened in a browser (tree view)

Important notes regarding XML coding

XML prolog

- XML prolog is optional
- Must come first in the document + doesn't have a closing tag
- Good idea to use it though = may use international characters
- Encoding should be specified (or simply save your XML the file as UTF-8 (default character encoding for XML)

XML tags

- XML tags must have a closing tag (except the prolog which isn't a XML tag)
- XML tags = case sensitive
- Must start with a letter or an underscore
- Can use letters, digits, hyphens, underscores, and periods
- Cannot start with xml + no contain a space
- Must be properly nested
- Recommended : short descriptive names + prefer underscores to dashes

Empty element

- Opening and ending tags with no content = called an empty element
- The two tags can then be replaced with a self closing tag : <element />
- Empty elements can still have attributes.

XML attributes

- XML can use attributes, just like HTML
- Attributes = always be quoted
- Basic rule = data as elements and metadata as attributes.

Use of symbols

- Some symbols may cause errors
- Symbol smaller than ("< ") and ampersand ("&") are strictly illegal in XML (must be coded using predefined entities)
- Good idea : code all symbols
- Note : that the white spaces are not truncated in XML like they are in HTML

&lt;	**<**	less than
&gt;	**>**	greater than
&amp;	**&**	ampersand
&apos;	**'**	apostrophe
&quot;	**"**	quotation mark

CDATA

- Standing for *Character Data* = blocks of plain text
- Tells the parser a specific section of the document contains no markup (has to be treated as plain text)

```
<![CDATA[
        <message>Example...</message>
]]>
```

Online ressources

https://onlinexmltools.com/
https://www.freeformatter.com/
https://codebeautify.org/xmlviewer

Exercise 1 :

Create a XML database to be used for further exercises.
BOOKLIST containing BOOK containing TITLE, AUTHOR, YEAR and PRICE

Styling XML files using CSS

- NOT the most proper way to display XML data
  BUT possible to use a CSS file to style an XML document's content

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="mystyle.css"?>
<booklist>
        <book> ...
```

XML file :

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="mystyle.css"?>
<booklist>
        <book> ...
```

CSS file :

```
bookstore {
        display: block;
        width: 600px;
        margin: auto;
        column-count: 2;
}

book {
        display:block;
        margin-bottom: 20px;
        padding: 10px;
        background-color : rgba(0,0,0,.1);
        break-inside: avoid-column;
        border: solid 1px black;
        box-shadow: 3px 3px 6px rgba(0,0,0,.25);
}

title,author,year,price {display : block;}

 title {
    font-size : 25px;
    font-weight : bold;
}
```

Assignment 1 :

Use the database created today and use CSS to position and style the list of data.

Class 02

XML (suite)

XLS

- *eXtensible Stylesheet Language*
  = styling language for XML consisting of four parts :

  - **XSLT** : A language for transforming XML documents.
  - **XPath** : A language for navigating in XML documents.
  - **XSL-FO** : A language for formatting XML documents (**discontinued in 2013**).
  - **XQuery** : A language for querying XML documents.

XSLT

- XSL Transformations, XLTS = most important part of XLS

- Allowsto transform a XML document into another type of document readable by a browser (XML, HTML and XHTML...)

- It usually transform each XML element into an XHTML element

- To navigate within an XML document = XPath will be used.

- XSLT makes it possible to add or remove elements and attributes to or from the output file + rearrange and sort elements + perform tests and decide to hide or display specific elements, etc.

- Basically, **XSLT transforms an XML source-tree into an XML result-tree**.

Style Sheet Declaration

- FIRST : document type definition (DTD)
  SECOND : XLS declaration that will link the XML document to its XLSstylesheet :
  <xsl:stylesheet> or <xsl:transform> (both can be used).

Styling a XML document with XSL

XML document to be used :

```
<?xml version="1.0" encoding="UTF-8"?>                               // DTD
<?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>             // Link to XSL stylesheet

<bookstore>                                                          // Root
        <book>
                <title>Poèmes Français</title>
                <author>Réjean Thomas</author>
                <year>2006</year>
                <price>14.99</price>
        </book>
        <book>
                <title>Harry Potter</title>
                <author>J K. Rowling</author>
                <year>2005</year>
                <price>29.99</price>
        </book>
        <book>
                <title>Le petit prince</title>
                <author>Antoine de Saint-Exupéry</author>
                <year>1943</year>
                <price>19.99</price>
        </book>
        <book>
                <title>Learning XML</title>
                <author>Erik T. Ray</author>
                <year>2003</year>
                <price>39.95</price>
        </book>
        <book>
                <title>L'avalée des avalés</title>
                <author>Réjean Ducharme</author>
                <year>1966</year>
                <price>39.95</price>
        </book>
        <book>
                <title>Les fleurs du mal</title>
                <author>Charles Baudelaire</author>
                <year>1857</year>
                <price>49.95</price>
        </book>
</bookstore>
```

Creating the XSL Style sheet

- There are two parts in the stylesheet :
    - Styles (HEAD section)
    - Template (BODY section

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">

<html>
<head>
<style>
        Your styles here
</style>
</head>
<body>
        Your template here
</body>
</html>

</xsl:template>
</xsl:stylesheet>
```

- Styles : as usual
- **Template :**

```
<h2>Booklist</h2>
<table>
        <tr>
                <th>Title</th>
                <th>Author</th>
                <th>Year</th>
                <th>Price</th>
        </tr>

        <xsl:for-each select="bookstore/book">    // Loops the database to show all elements
        <xsl:sort select="year"/>                 // Sorts the results based on a value
        <tr>
                <td><xsl:value-of select="title"/></td>
                <td><xsl:value-of select="author"/></td>
                <td><xsl:value-of select="year"/></td>
                <td><xsl:value-of select="price"/></td>
        </tr>
        </xsl:for-each>
</table>
```

Filtering output

- In order to filter the results, we could have specified the select value of *xsl:for-each* :

Note

- Other XSLT options :*if*, *choose*, and *apply*.

- XSLT works fine BUT some browsers don't support it
  SO solution : using JavaScript for all the transformations

Assignment 2 :

Using the same XML database as for assignment 1, use XLS to position and style the list of data.

Class 03

Parsing XML

- We have seen = XML can be displayed using XSL (does a great job)
  BUT = XSL isn't always supported by all browsers
- JavaScript and PHP = can be used to extract and display data from XML (great solution)

Parsing XML with JavaScript

- To parse an external file, in jQuery = AJAX request must first be made
- find() method = external XML file is searched to find the <book> tags withing the root (<bookstore>)
- For every entries (books), 4 variables are used to find and store every <book> children

```
<div id="results"> </div>

<script language="JavaScript">
$(document).ready(function(){
$.ajax({                                                    // AJAX request
type: "GET",
url: "my_xml-02.xml",
dataType: "xml",
success: function(xml){
        var i = 0;
        $(xml).find('bookstore').children('book').each(function(){

                var sTitle = $(this).find('title').text();
                var sAuthor = $(this).find('author').text();
                var sYear = $(this).find('year').text();
                var sPrice = $(this).find('price').text();

                $("<p></p>").html("<b>" + sTitle + "</b>, " + sAuthor + ", " + sYear + ", " +
sPrice).appendTo("#results");
                i++;
        });
 var sTotalBooks = i;

$("<p></p>").html('<b>Total of books:</b> '+ sTotalBooks).prependTo("#results");
 },
error: function() {
        $("<p></p>").html('An error occurred while processing XML file.').prependTo("#re-
sults");
}
});
});
</script>
```

Using a jQuery shorthand for the AJAX request makes it even more simple :

```
<div id="results"> </div>
<script language="JavaScript">
$.get("my_xml-02.xml", function(data) {                    // AJAX request shorthand
            var i = 0;
            $(data).find('bookstore').children('book').each(function(){
                        var sTitle = $(this).find('title').text();
                        var sAuthor = $(this).find('author').text();
                        var sYear = $(this).find('year').text();
                        var sPrice = $(this).find('price').text();
                        $("<p></p>").html("<b>" + sTitle + "</b>, " + sAuthor + ", " + sYear + ", " +
sPrice).appendTo("#results");
            i++;
            });
var sTotalBooks = i;
$("<p></p>").html('<b>Total of books:</b> '+ sTotalBooks).prependTo("#results");
});
</script>
```

Parsing XML with PHP

- Parsing XML using PHP is a lot more simple (+ supported by all browsers + executed on server-side)

- All there is :
    - Store the content of the XML file in a variable using the function ***simplexml_load_file()***
    - Then, access the children of <book> tags by storing them in a variable
    - And use a ***foreach*** loop with keys to display them.

```
<h2>XML with PHP</h2>
<?php
$xml=simplexml_load_file("my_xml-02.xml") or die("Error: Cannot create object");

foreach($xml->children() as $books) {
            echo "<b>" . $books->title . "</b>, ";
            echo $books->author . ", ";
            echo $yearss->year . ", ";
            echo $books->price . "<br>";
}
?>
```

Assignment 3 :

Using the same XML database in an external file, use JavaScript or PHP to parse the data and CSS to style the results.

Class 04

JSON

- Standing for *JavaScript Object Notation*
- JSON is a human readable format for structuring data
- Primarily used to transmit data between a server and web application (alternative to XML)

Pros and cons

- Almost flawless = very easy to read, understand and use.

Pros

- Can be understood by humans and machines
- Doesn't require any real learning (except for the syntax using a specific punctuation)
- Doesn't depend on any other language (open data exchange)
- Is taken in charge by several languages : JavaScript, PHP, Perl, Python, Ruby, Java, etc.
- Allows to stock different types of data : strings (including base64 images), numbers, arrays, objects, booleans, null, etc.
- It's tree structure and simple syntax make it light and efficient.
- Widely used to integrate different types of contents to web pages, such as APIs.

Cons

- Just like for any database methods, security measures need to be put in place to protect confidential informations.

- The fact that the syntax is very simple may sometimes pause problems (e.g.: JSON use no tags such as XML so the developer needs to know the data structure)

- See Pros and Cons comparative table, course notes, page 3

Keys and Values

- JSON data consists in pairs of keys (equivalent of properties) and values separated by colons + placed between double quote marks in an object
  They make a ***key/value pair.***

Key: A key is always a string enclosed in quotation marks.
Value: A value can be a string, number, boolean expression, array, or object.

```
"car" : "Mazda"
```

Syntax and structure of JSON files

- Braces are used to define an object
- Objects may contain many key/value pairs
- NOTA : elements of JSON's objects can be : strings, numbers, arrays, objects, booleans, null, etc.

{...} Brace brackets are used to define an object.
[...] Brackets are used to define an array.
N.B. : Commas are used to separate elements of an object
(no comma after the last element or the element will not be valid).

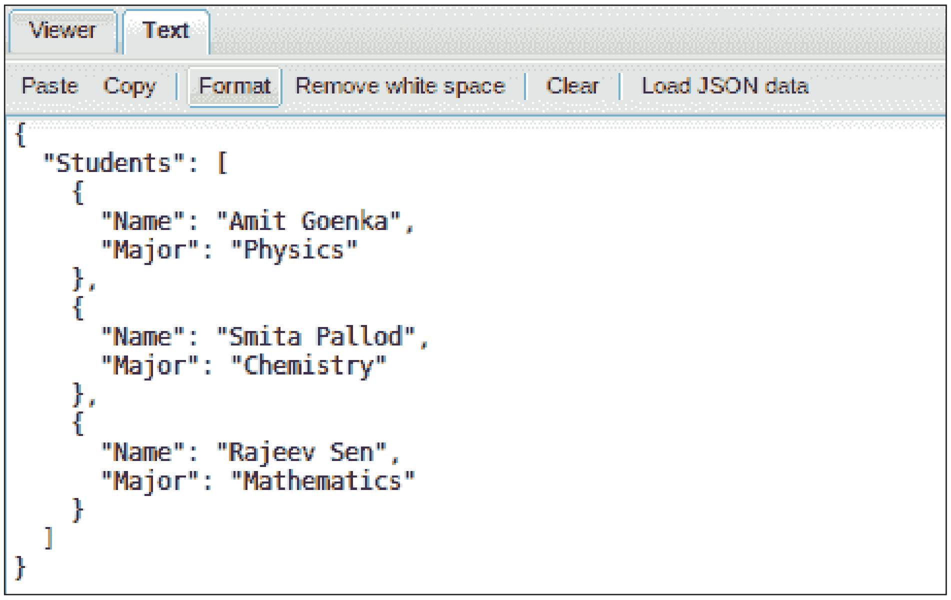
```
{                                                   // Start of the object
        "Course name" : "LS2",                      // String element
        "Topic" : "Script languages",
        "Students" : [                              // Start of array element
                {                                   // Start of array's object
                        "Last name" : "Norris",
                        "First name" : "Chuck",
                        "Age" : 75,                 // Numbers don't require double quotes
                        "Country" : "USA"
                },                                  // End of array object
                {
                        "Last name" : "Doe",
                        "First name" : "John",
                        "Age" : 44,
                        "Country" : "UK"
                },
                {
                        "Last name" : "The Poo",
                        "First name" : "Winnie",
                        "Age" : 10,
                        "Country" : "FRANCE"
                }
        ]                                           // End of array element
}                                                   // End of the object
```

Viewing and validating

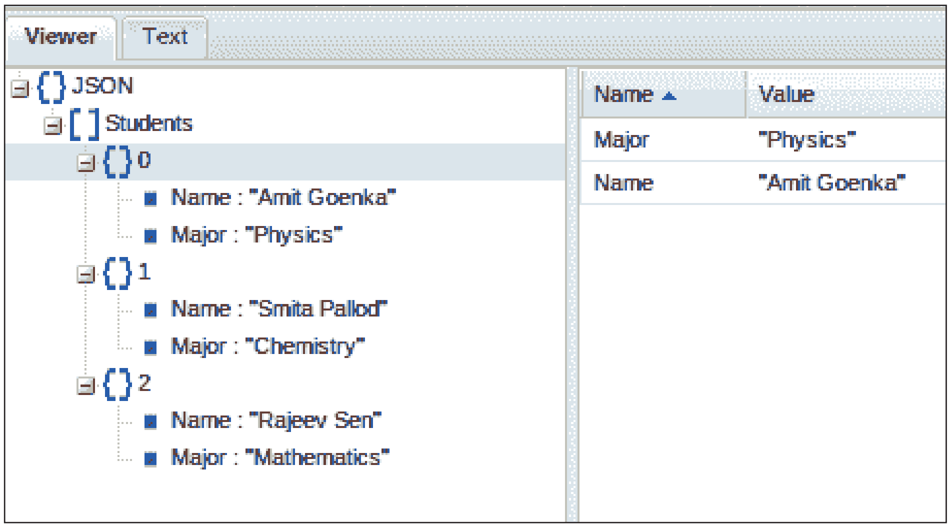
- Especially at the beginning = useful to be able to visualize and validate the source code

- Many online editors / validators :
  http://jsonviewer.stack.hu/
  https://jsonformatter.curiousconcept.com/
  https://jsoneditoronline.org/

- Simply copy and paste your code in the application's "Text" window.
  The validator the indicates if errors are found and the viewer window shows the tree structure of the code.



```
{
  "Students": [
    {
      "Name": "Amit Goenka",
      "Major": "Physics"
    },
    {
      "Name": "Smita Pallod",
      "Major": "Chemistry"
    },
    {
      "Name": "Rajeev Sen",
      "Major": "Mathematics"
    }
  ]
}
```



```
{} JSON
  [] Students
    {} 0
       Name : "Amit Goenka"
       Major : "Physics"
    {} 1
       Name : "Smita Pallod"
       Major : "Chemistry"
    {} 2
       Name : "Rajeev Sen"
       Major : "Mathematics"
```

Name ▲	Value
Major	"Physics"
Name	"Amit Goenka"

Retrieving data from a JSON files

- In JSON files = data consist in a string of variable length (can be enormous).

- Testing JSON with external JSON files
  = only works with hosted files (some virtual servers may allow testing locally).

Parsing the JSON data

- To be able to access the data of JSON files = needs to be parsed
- Can be done using the function ***JSON.parse()***.

Retrieving data (Example : 10/json-01.html)

- Once an object contain the parsed JSON data has been created
  it is possible to retrieve specific values (associated with the different keys)
  Syntax : *object.key*

```
<h2> </h2>

<script>
let myJson = '{"name":"John", "age":30, "city":"New York"}';
let myObject = JSON.parse(myJson);

$("h2").html(myObject.name + ", " + myObject.age + ", " + myObject.city);
</script>
```

(Example : 10/json-01.html)

Retrieving data from external JSON files

Online examples : http://www.monamijean.com/cdi/json-01.html
Retrieving data = basically the same as before (remember : external JSON files needs to be tested online)

Loading data from external JSON files (USE MAMP)

- for the data to be available in the current document
  = a http request needs to be made to store the data in an object (a variable)
  USING the function *XMLHttpRequest()* (complicated at first but we'll see another option)
- The open() and send() function will also be used

Using a http request

```
<h2></h2>

<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
                var obj = JSON.parse(this.responseText);
                $("h2").html(obj.prenom + " " + obj.nom + ", " + obj.age);
        }
};

xmlhttp.open("GET", "http://www.monamijean.com/cdi/mytest.json", true);
xmlhttp.send();
</script>
```

Using getJSON() function

```
<h1></h1>

<script type="text/javascript" language="javascript">
$.getJSON('mytest.json', function (data) {
        $("h1").append("You are " + data.age + " year old.");
});
</script>
```

Assignment 04

Using the external file supplied by the teacher (bookstore.json) create display a list using JSON, JavaScript, CSS and jQuery.

http://www.monamijean.com/cdi/bookstore.json

Class 05

Revision

Workshop

Class 06

Midterm exam

Class 07
APIs

- Standing for = *Application Programming Interface*
  => Set of rules and specifications that applications can follow to communicate with each other
  => They govern how applications can talk to each other + how data gets shared over the Internet.

- APIs = should be considered as tools made available
  Save a lot of time, solve various problems + allow create complex functionality more easily

Client-side APIs

- Many APIs are available to be used in JavaScript (Adds extra superpowers)

Browser APIs

- Built into the web browser
  = makes it possible to show data from the browser as well as from distant servers

  Example : *Web Audio API* makes it possible to manipulate audio in the browser

- Applications programmed in complex lower-level languages are used in the background to do the real job
  API acts like a plugin, a pipe that lets the results in

Third-party APIs

- Not built into the browser
  Generally necessary to retrieve their code and data from an external source on the web

  Example : Twitter API allows to do different things such as displaying your latest tweets

APIs possibilities

- Hundreds of APIs available online (doing all sorts of things)
  Some are better and more useful than others.

Common browser APIs

APIs for manipulating documents loaded into the browser.
Example : The DOM API makes it possible to manipulate HTML and CSS.

APIs that fetch data

Fetch data from the server to update small sections of a webpage automatically

Drawing and graphics manipulation APIS

- Mostly supported in browsers (Canvas and WebGL, for instance)
- To update the pixels information within a <canvas> element
  (2D and 3D views + apply different effects

Audio and Video APIs

Do things such as: Creating custom UI controls for playing audio and video (displaying captions and subtitles, grabbing video from your web camera to be manipulated via a canvas or displayed on someone else's browser in a web conference, etc.

Device APIs

Made for manipulating and retrieving data from modern device hardware (very useful for web apps.
Example: may notify user of an update is available for a specific app sending a notification.

Client-side storage APIs

To store data on the client-side (useful to save the state of an app between page loads, and perhaps even work when the device is offline)

Public APIs

- Also known as = *Open APIs*
  DATA made available to developers (usually free)

- Include = *Big Data*
  Very large amount of data made publicly available

- Very large number of public APIs + different ways to connect to them.

Your first API

- Get + test the endpoint
- Result = often JSON format
- For better viewing the data = JSONview extension (Chrome/Firefox) https://jsonview.com/
- Find and read the documentation
- Make an AJAX request = $getJSON

```
<h2> </h2>

<script type="text/javascript">
$.getJSON("https://ipapi.co/json/", function(data) {

        $("h2").html("You are now in the city of: " + data.city);
});
</script>
```

Exercise :

Extract and display data from https://ipapi.co/json/ and use CSS to create a nice looking presentation.

Redirecting users based on country name

```
<script type="text/javascript">
$.getJSON("https://ipapi.co/json/", function(data) {                // AJAX request

        let country = data.country;                                  // User's country name

        if (country == "CA"){                                        // For Canada
                location.replace("canada.html");
        } else if (country == "US") {
                location.replace("usa.html");                        // For USA
        }
        else {
                alert("You are not from CANADA nor USA")             // For all other countries
        }
});
</script>
```

Another geolocalization API:
https://api.ipgeolocationapi.com/geolocate

Exercise

Search the Internet for API not requiring an API key and build a good looking page using the APIs you have chosen.

Loading a map from mapquest in a <iframe> tag

- Go to mapquest.com
- A map is shown
- Use the Share button / embed = copy the code
- Paste it in a HTML document.

Assignment 05

using ipapi's API and Mapquest, creat a page that shows a map based on the user's geographical coordinates.

Class 08
API (suite)

Accessing third party APIs

APIs have their particularities, = generally offer common features + work more or less the same way

Find an API
Explore and find the proper API for your needs
Our example : https://openweathermap.org/.

Get a developer key
Most APIs require you to use an ID key for security reasons and accountability
May need to create an account first

Test the endpoint
End point = page containing the data you will be able to use (usually in JSON)
Test the end point in a browser
API's usually have different endpoints + different ways of reaching them
Our example : endpoints URL for each city.

Parsing content in your browser
For easier reading : extension https://jsonview.com/

Find and read the documentation
This is how you will find out about the API's features + how to use it

Loading JSON data into a web document

- Importing external data => AJAX request
- jQuery => *getJSON()*

Loading JSON data and displaying text content to a web document

Web document with the jQuery CDN OR external JavaScript file

Explanation :

- *getJSON() = AJAX request* to import the content
- Imported data stored in variable *data*
- *data* used as prefix in instructions
- Final instruction = writes the data in the tag containing *id="map"*

```
$.getJSON("http://api.openweathermap.org/data/2.5/weather?q=montreal,ca&units=metric&AP-
PID=c40b9d48e1ae53a86e25a3382957f4cf", function(data) {
        $("#map").html(data.name);
});
```

Displaying an icon to a web document

- In the JSON feed = second element *(**weather**)*
  => an array with only one object *(**weather[0]**)* containing four elements
  (id, main, description and *icon)*

```
let icon = "http://openweathermap.org/img/wn/" + data.weather[0].icon + "@2x.png";

$("#map").append('<img  src="' + icon + '" />');
```

If you code ***weather[0].icon*** = result will be a string, the name of the PNG icon : *04n*

- Icons stored at the URL *http://openweathermap.org/img/wn/ = S*tore the path to the icon in a variable
- Use concatenation to code the image in your web document

Assignment 06

Student use the openweather API to create a good looking responsive weather page.

Class 09

Using instagram API

- Create a Facebook developer account
  https://developers.facebook.com/

- From the main menu = create an app + name it
  Take note of the AP ID + secret code (in parameters/general from the lateral panel)

- Difficult to generate an access token from the developers site
  Use third party to do so : http://instagram.pixelunion.net/

- Test the endpoint
  https://api.instagram.com/v1/users/self/media/recent/?access_token=YOUR_ACCESS_TOKEN&count=10

- Make an AJAX request

- Create a loop + concatenations.

```
<div> </div>

<script type="text/javascript">
$.getJSON("https://api.instagram.com/v1/users/self/media/recent/?access_
token=YOUR_PERSONAL_ACCESS_TOKEN&count=10", function(data) {

let x = data.data[0].id;                    // Store the total number of elements

for(i=0; i<x.length; i++){

            $("div").append('<img src="' + data.data[i].images.standard_resolu-
tion.url + '">');

}

});
</script>
```

- Interesting tutorial : https://www.codeofaninja.com/2015/01/display-instagram-feed-website.html

Using feed from newsapi.org

- Sign in + login to https://newsapi.org
  + click on the «Get API key»

- Test the endpoint
  https://newsapi.org/v2/everything?q=canada&from=2020-02-01&sortBy=publishedAt&apiKey=YOUR_KEY

- Make an AJAX request and show data

```
<div> </div>
<script type="text/javascript">
$.getJSON("https://newsapi.org/v2/everything?q=can-
ada&from=2020-02-01&sortBy=publishedAt&apiKey=YOUR_API_KEY",
function(data) {

        let title = data.articles[0].title;        // Would display the title
        $("div").html(title);                      // of the first article
});
</script>
```

- Use a loop to display all content

```
<section> </section>
<script type="text/javascript">
$(document).ready(function() {

$.getJSON("https://newsapi.org/v2/everything?q=can-
ada&from=2020-02-01&sortBy=publishedAt&apiKey=YOUR_API_KEY",
function(data) {
let x = data.totalResults;              // Total number of articles available

for(i=0; i<x; i++){                     // Looping through articles
        let author = data.articles[i].author;
        let title = data.articles[i].title;
        $("section").html('<div class="title"><a target="_blank" href="' +
data.articles[i].url +'">' + title + '</a></div> <span class="author">(' + author
+ ')</span></article>');
}
});
});
</script>
```

Final project

Create a home page entirely fed buy APIs

Class 10

Revision + workshop

Class 11

Final exam