Javascript Chapter 1 : the basics of the programming language

Working Environment, Syntax, Data, Control Statements & algorithmic
Object Oriented JavaScript OOJS

Table of Contents

Basic approach

Working environment : Node.js & Visual Studio Code
Lexical grammar: Javascript syntax & console.log()
Declarations & types: Variables, Constants, Arrays
Expressions and Operators: assignment, arythmetic, equality & relational
Statements : control flow and iterations
Functions : User Defined Functions

Knowledge deepening & Object Oriented JavaScript

Variable scope and *hoisting*
OOJS, Built-in Objects: Number, Strings, Date, Properties & Methods
Data Format & Manipulation : JSON
Advance Control Statements
Arrow functions
Template literals

Basic approach

Working environment : Node.js & Visual Studio Code

Different working environments are available :

- Code editor : **Sublime Text**, Notepad ++, Atom, Bracket, … , with many extensions (packages, plug-in) to help you code

Refer to the manufacturer's site for installation, as well as extensions

https://www.sublimetext.com/ , https://packagecontrol.io/
https://notepad-plus-plus.org/ , https://github.com/notepad-plus-plus/nppPluginList/blob/master/doc/plugin_list_x64.md
https://atom.io/ , https://atom.io/packages
http://brackets.io/ , https://registry.brackets.io/

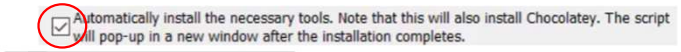
As an example, surch for Emmet, a powerfull tool for high-speed coding HTML and CSS, using abbreviations and snippet expansions.

- **Node.js & Visual Studio Code** as an IDE (Integrated Development Environment)

Install Node.js (Long Term Support), along with NPM (Node.js Package Manager), and probably Chocolatey, Python, ….

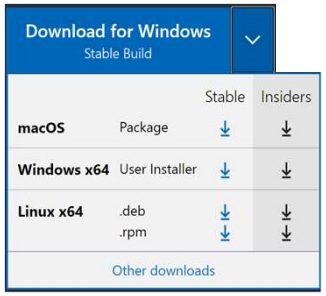
https://nodejs.org/en/download/

Optional



Node.js is a Javascript engine which allows to run JS scripts with Visual Studio Code, as an alternative to the Chrome browser. Node.js also allows to built Network Applications (Node.js web server as an alternative to Apache web server).

Install *VSCode* https://code.visualstudio.com/



Video help for installation:
fr https://youtu.be/ig9EOn17vFM
en https://youtu.be/JINE4D0Syqw

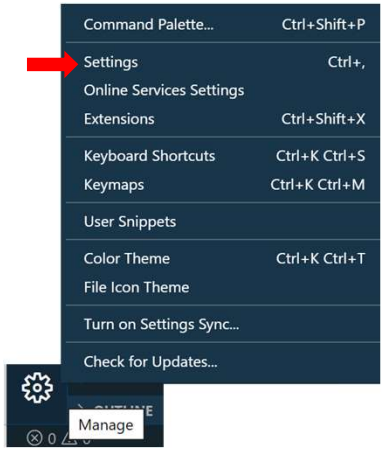
VSC First Steps Video:
https://code.visualstudio.com/docs/getstarted/introvideos

Université de Strasbourg

FRENCH-AZERBAIJANI UNIVERSITY

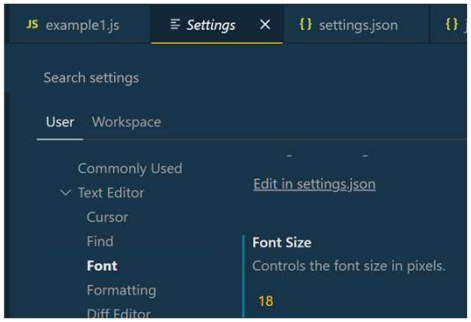
The purpose here is not to present Visual Studio Code in detail, many tutorials are available online. Look more specifically:

User and Workspace Settings



From the main menu : File – Preferences – Settings, or the Manage *gear button* :

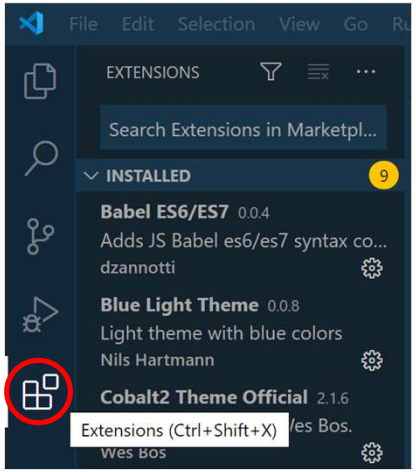
Example : change the font size of the text editor



More help :
https://code.visualstudio.com/docs/getstarted/settings

Extensions

Numerous extensions exist, see https://marketplace.visualstudio.com/vscode , or browse and install



More help :
https://code.visualstudio.com/docs/editor/extension-gallery

fr https://youtu.be/CTz1vpJG68E
en https://youtu.be/Fed01v3yYNE

➲Install, among others, the *Code Runner* extension

See also: Live Server, Prettier, Wrap Console Log, Live Share, …

Basic approach

Lexical grammar: Javascript syntax & console.log()

<script> element is used to embed executable code, usually in the head section, or in the body section, or to reference an external .js file

Lexical grammar:

Javascript is case-sensitive.

Instructions, called statements, are terminated with a semicolon **;**

Comments begin with **//** , or are delimited by **/* */**

JS script can be run in a browser, results are displayed in the web browser console, using **console.log()**

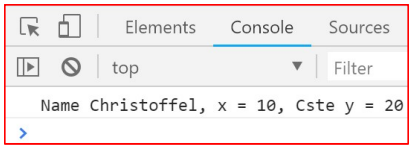
Embeded JS code

log() is a method of the **console** object to access to the browser's debugging console

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Syntax, variables, operators, statements</title>
6  <script src="my_code.js"></script>
7  <script>
8  var x=10;
9  const y=20;
10 name="Christoffel";
11 console.log('Name '+name+", x = "+x+', Cste y = '+y);
12 </script>
13 </head>
14 <body>
15
16 </body>
17 </html>
```

External JS file

Open the browser console

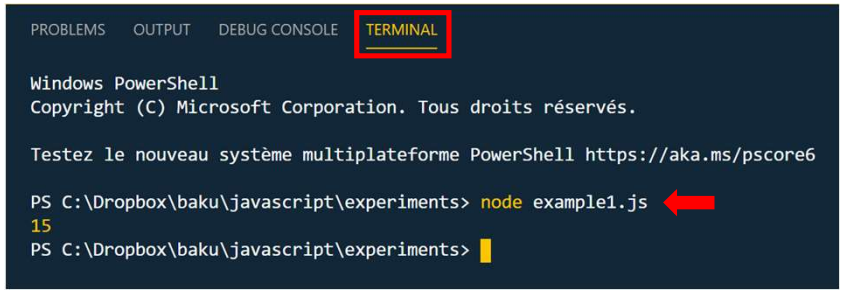


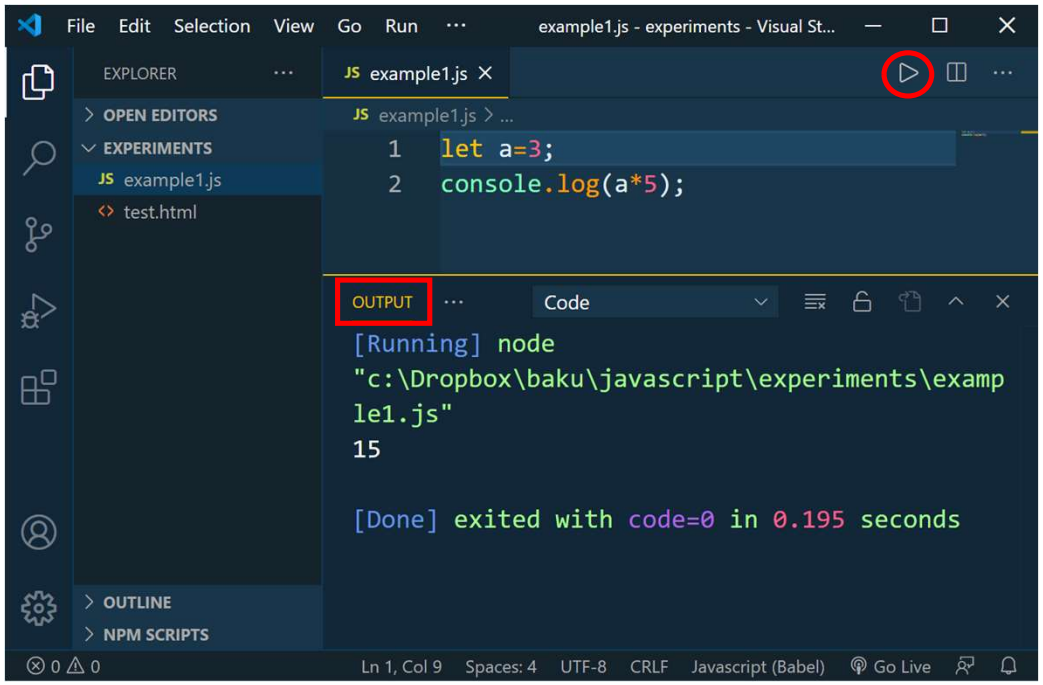
With **VSCode**, Node.js installed and the Code Runner extension installed :

- Use the *Run Code* button, the results are displayed in the Output Window

- From the Terminal window, type :

  node fileName.js





Basic approach

Declarations & types: Variables, Constants, Arrays

Variables, not typed, are used to store value : **integer** or **float** number, **string**, **boolean**, …
Variable declaration and value assignment (*assignment operator =*) :

var

```
8  var x=5;
9  var last_name="Christoffel";
10 y=3;//Possible, but to be avoided
11 console.log(last_name);
```

let

```
13 let z=10;
14 console.log(z);
```

const

```
16 const VAT=0.2;
17 console.log(VAT);
```

A constant value can't be changed!

Variable names (*identifiers*) **rules**:
Starts with : letter, _ , $
Subsequent characters : A-Z, a-z, 0-9, _
Case sensitive
Avoid accents (à,ü, ..) even if permitted

The declarations of the variables using the keywords *var* or *let*, are distinguished by the **scope of the variables** (see later)

global or **local** depends on the *execution context*

Arrays, are used to store a list of values : **integer** or **float** number, **string**, **boolean**, …
Each array element is indexed with an integer value *key* (there is no associative array like in PHP!)
Declaring an array and assigning values to array elements (*assignment operator* =) :

Array constructor

Instance of the **Array object** using the *new* keyword:

```
19 let studentList = new Array();
```

Assigning values to array elements using an index position:

```
21 studentList[0]="Glasscott";
22 studentList[1]="Willars";
```

Key indexes of an array : 0, 1, 2, … (integers only!)

```
24 let teacherList = new Array('Christoffel','Collet','Jeannin');
```

Accessing an array element using an index position:

```
26 console.log('Lastname : '+teacherList[0]);
```

```
Lastname : Christoffel
```

Literal constructor:

```
28 let subjectList=['HTML','CSS','Javascript','PHP'];
29 console.log(subjectList);
```

```
▼(4) ["HTML", "CSS", "Javascript", "PHP"]
   0: "HTML"
   1: "CSS"
   2: "Javascript"
   3: "PHP"
   length: 4
```

Basic approach

Expressions and Operators: assignment, arythmetic, equality & relational

- Assignment operators : =
- Arithmetic operators : +, -, *, /, % (modulus – remainder of the division)
- Arithmetic and assignment operators : += (add and assign), -=, *=, …
- Incrementing and decrementing operators : ++$i (pre-increment) , $i++ (post-increment), --$i, $i--
- String or concatenation operator : + , += (concate and assign)
- Comparison operators : ==, !=, >, >=, <, <=, === (strict equality with same data type)
- Logical operators : && (AND), || (OR), ! (Not)

```
10 //Assignment and arithmetic operators
11 let a=5;
12 let b=2;
13 let c=a*b;
14 let r=a%b;
15 b+=8;
16 console.log("c="+c+", r="+r+", b="+b);
17
18 //incrementing, decrementing
19 //postfix ++, a is assign to x,
20 //than incremented
21 console.log("Before: a = "+a);
22 let x=a++;
23 console.log("After: a="+a+", x="+x);
24 //prefix ++, a is incremented,
25 //then assigned to x
26 console.log("Before: a = "+a);
27 x=++a;
28 console.log("After: a="+a+", x="+x);
29
30 //string concatenation +
31 let lastname="Christoffel";
32 let firstname="Eric";
33 let fullname=lastname+' '+firstname;
34 console.log(fullname);
```

```
c = 10 , r = 1 , b = 10
Before: a = 5
After: a = 6 , x = 5
Before: a = 6
After: a = 7 , x = 7
Christoffel Eric
```

Basic approach

Statements : control flow and iterations

Conditional statement / Decision making

- The conditional statement allows a decision to be made and determines the progress of the program
- The various type of conditional statements :
  if… , **if**… **else** , **if**… **else if**… **else**
- **switch**… **{**case…**}** can simplified multiple if… else or nested if…
- Ternary operator can be used to minified an if.. else
  (condition) ? … *if true* **:** … *if false*

```
10 //if ... else if ... else
11 let a=51;
12 if(a<50){
13     console.log("less than 50");
14 } else if(a>50){
15     console.log("greater than 50");
16 } else {
17     console.log("equal to 50");
18 }
```

```
greater than 50
```

```
20 //switch case
21 let award="DI";
22 switch(award){
23     case "HD":
24     console.log("Hight Distinction");
25     break;
26     case "DI":
27     console.log("Distinction");
28     break;
29     case "CR":
30     console.log("Credit");
31     break;
32     case "PS":
33     console.log("Pass");
34     break;
35     case "FA":
36     console.log("Fail");
37     break;
38 }
```

```
Distinction
```

```
40 //ternary operator
41 let data=1;
42 let status= (data==1) ? "Active":"Inactive"
43 console.log(status);
```

```
Active
```

Loop / Doing repetitive task

- Loops are used to repeat a series of actions until a specified condition is fulfilled
- Different types of loops exist, with a specific conditional behavior

for – it is a counter, i.e. a variable that controls the number of iterations. The loop syntax manages the counter increment

while - iterations continue as long as the condition is evaluated at true. *The counter must be incremented inside the loop*. The loop iterations may never be executed

do… while – the condition is evaluated at the end of the loop, so the loop is executed at least once. *The counter must be incremented inside the loop.*

- **break** statement allows to exit the loop before it ends
- **continue** statement allows to skip an increment

```
47  //for loop as a counter
48  for(let i=0;i<10;i++){
49      console.log(i);
50  }
51  console.log('---------');
52  //while loop
53  i=1;
54  let counterMax=5;
55  while(i<=counterMax){
56      console.log(i);
57      i++;
58  }
59  console.log('---------');
60  //but may never be executed!
61  //i equals 6, previous while
62  while(i<=counterMax){
63      console.log(i);
64      i++;
65  }
66  console.log('---------');
67  //do... while
68  //executed at least once
69  do {
70      console.log(i);
71      i++;
72  } while(i<=counterMax);
```

```
0
1
2
3
4
5
6
7
8
9
---
1
2
3
4
5
---
--------
--------
6
```

Basic approach

Functions : User Defined Functions

Tips : choose a name function that reflects what it does!
The function name begins with a letter or an underscore.

Unlike PHP, which has many built-in functions, thousands, Javascript, which is primarily an object-oriented programming language, has very few built-in functions, a dozen: parseInt(), eval(), ….
A **user defined function** is a set of statements that can be run repeatedly.

Anatomy and Syntax of a function

Global scope variables, can be used everywhere, scrpit and functions (see later).

Function call, a return value can be expected, for future computing, or to display

Passing value ≡ Argument

Declared variables in a function exist only in that function : **Local scope**

Set of instructions

The function may return a single variable, but which can be an array, with the *return* keyword

The function can display messages, or can write dynamic HTML (see later).

Curly brackets are used to mark the function

```
9  var scriptVar=... ;
10 let inputData=.... ;
11
12 let returnValue=functionName(inputData1, [inputData2, ...]);
13
14 functionName(inputData1, [inputData2, ...]);
15
16
17 function functionName(passValue1, [passValue2, ...]){
18
19     var dataVar=..;
20     let dataLet=..;
21     var computedVar=....;
22
23 //statements to be executed;
24
25     console.log(".... with variable computed....");
26
27
28     return computedVar;
29
30
31 }
```

An example of a function to calculate the VAT (Value Added Taxes), from an initial amount.

```
16  const VATrate=20/100;
17  var amountExcludingVAT=1000;
18  var amountIncludingVAT=VAT();
19  console.log("Ammount excluding VAT = "+amountExcludingVAT+",
        Amount including VAT = "+amountIncludingVAT);
20  console.log("VAT amount = "+calculatedVAT);
21
22  function VAT(){
23      var calculatedVAT=amountExcludingVAT*VATrate;
24      return amountExcludingVAT+calculatedVAT;
25  }
```

Function call

Return calculated value

```
Ammount excluding VAT = 1000, Amount including VAT = 1200
▶ Uncaught ReferenceError: calculatedVAT is not defined
    at codeForChapter1BuiltInFunctions.html:20
```

Why ?

See later : variable scope

Variable scope and *hoisting*

var, **let** and **const** are the three types of variable declaration.
The **scope** of the variables, **their visibility**, depend on **1)** the *declaration type*, **2)** where they are declared: **global**, in the *main script*, **local**, within a *function* or a *block* of statements.

global scope

var, *let* and *const* keywords are used to declare variables and to assign a value, in the main script.
All these 3 variables are visible in the main script, in functions and in blocks (e.g. a for loop).
These 3 variables are said **global**.

```
main script: var a=a, let b=b, const c=c
function: var a=a, let b=b, const c=c
block: var a=a, let b=b, const c=c
```

```
6  <script type="text/javascript">
7  //global scope
8  var a="a";
9  let b="b";
10 const c="c";
11 //main script
12 console.log("main script: var a="+a+", let b="+b+", const c="+c);
13 //function
14 testGlobalScope();
15
16 function testGlobalScope(){
17 console.log("function: var a="+a+", let b="+b+", const c="+c);
18 }//end function
19 //block for
20 for(let i=0;i<1;i++){
21 console.log("block: var a="+a+", let b="+b+", const c="+c);
22 }//end for block
```

local scope : case of *var* in a *function*

var, *let* and *const* keywords are used to declare variables and to assign a value, inside a *function*.
All these 3 variables are visible inside the function only, but not in the main script.
var variable, declared inside a function is said **local inside the function**, i.e. its visibility is restricted to the function. The same applies to *let* and *const* variables.

```
6  <script type="text/javascript">
7  //local scope: function
8  testFunctionLocalScope();
9  function testFunctionLocalScope(){
10 var a="a";
11 let b="b";
12 const c="c";
13 console.log("function: var a="+a+", let b="+b+", const c="+c);
14 }//end function
15 console.log("main script: var a="+a);//not defined
16 console.log("main script: let b="+b);//not defined
17 console.log("main script: const c="+c);//not defined
```

```
function: var a=a, let b=b, const c=c
▸Uncaught ReferenceError: a is not defined
    at codeForChapter1Varia…tureExample.html:16
▸Uncaught ReferenceError: b is not defined
    at codeForChapter1Varia…tureExample.html:17
▸Uncaught ReferenceError: c is not defined
    at codeForChapter1Varia…tureExample.html:18
```

```
8  let a;
9  console.log("declared, but no value assigned: a="+a);
10 console.log("not declared, and no value: b="+b)
```

```
declared, but no value assigned: a=undefined
Uncaught ReferenceError: b is not defined
    at codeForChapter1notDe…d_undefined.html:10
```

Note: « not defined » means a variable that is not declared.
« undefined » means a declared variable, but with no value assigned.

local scope : case of *let* and *const* in a *block* of statements

var, *let* and *const* keywords are used to declare variables and to assign a value, inside a *block of statement*, e.g. a *for* loop.
All these 3 variables are visible inside the block, but *let* and *const* are no more visible in the main script.
let and *const* variables, declared inside a block are said **local within the block**, i.e. their visibility are restricted to the block.
On the other hand, the *var* variable, even if it has been declared inside a block, is still visible outside the block!

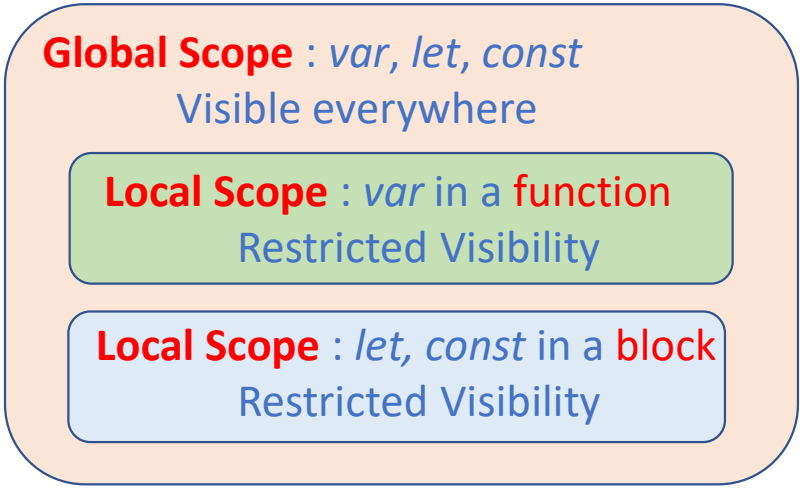
```
block (for): let i=1, let j=1, var k=1, const l=1
block (for): let i=2, let j=4, var k=8, const l=2
block (for): let i=3, let j=9, var k=27, const l=3
block (for): let i=4, let j=16, var k=64, const l=4
▸Uncaught ReferenceError: i is not defined
    at codeForChapter1Varia…tureExample.html:14
▸Uncaught ReferenceError: j is not defined
    at codeForChapter1Varia…tureExample.html:15
After block: var k=64
▸Uncaught ReferenceError: l is not defined
    at codeForChapter1Varia…tureExample.html:17
```

```
6  <script type="text/javascript">
7  //local scope: block
8  for(let i=1;i<5;i++){
9      let j=i*i;
10     var k=i*i*i;
11     const l=i;
12 console.log("block (for): let i="+i+", let j="+j+",
       var k="+k+", const l="+l);
13 }//end for block
14 console.log("After block: let i="+i);//not defined
15 console.log("After block: let j="+j);//not defined
16 console.log("After block: var k="+k);
17 console.log("After block: var l="+l);//not defined
```

Variable scope and *hoisting* : Summary

Restricting the availability of variables (*var* in functions, *let* and *const* in blocks of statements, prevents confusion between variables with the same name, when working with multiple functions and blocks, including the use of code from other sources.

Hoisting mechanism, to be continued…



Unlike PHP, very few built-in functions exist in Javascript, a dozen compare to thousands in PHP.
As an example, isNaN() returns false or true if the value of a variable is, or is not, a number.
This function can be compared with is_numeric() in PHP.

isNaN() means
is Not a Number

```
8  var a ='a';
9  var x=2;
10 console.log("isNaN => a:"+isNaN(a)+', x:'+isNaN(x));
```

```
isNaN => a:true, x:false
```

In fact, Javascript is an **Object Oriented Programming** language (OOP)!
Javascript is commonly referred to as **Object Oriented JavaScript** : **OOJS**.
Depending on the value assigned to a variable, the variable will actually be an **object**. These objects are of type: *Number*, *String*, *Boolean*, undefined,… for **Data Type**, *Object* for **Structural Type** (such as Date, Array, …).
The **typeof operator** determines the type of an object.

```
8  var a ='a';
9  var x=2;
10 console.log("typeof => a:"+typeof a+", x:"+typeof(x));
```

```
typeof => a:string, x:number
```

Object Oriented Programming : Syntax & Terminology

Each value of a variable is an object!

An object is made of **data**, and **behaviors** when seen from outside.

Objects have **properties** and **methods** to transform them :

Syntax

object.*property*

A property is a variable attached to the object.

object.*method()*

A method is a function applied to the object.

String Object : Data type object

Let's consider a string value for a variable. The *typeof* operator returns the objet type : **String**

By knowing the type of object, we can find the properties and methods.

Property :

length

Only 1 property : the number of characters

Methods :

toUpperCase()

toLowerCase()

charAt()

substring()

…

```
8  var lastname="Christoffel";
9  console.log(typeof lastname);
10 console.log(lastname.length);
11 console.log(lastname.toUpperCase());
12 console.log(lastname.charAt(1));
13 console.log(lastname.substring(1,4));
```

```
string
11
CHRISTOFFEL
h
hri
```

```
C h r i s t o f f e l
0 1 2 3 4 5 6 7 8 9 10
```

11 characters

⇐ Character position, from 0

It is necessary to document the numerous methods of the String object, to experiment with them, to learn them, to integrate them into your developments.

Date Object : Structural type object

A Javascript date is created and assigned to a variable using a **constructor** of the **Date()** object, with the keyword *new*

The Date object has no property, but numerous methods. Methods exist to return the hour, the minutes and the seconds for a given date

```
8  var today=new Date();
9  console.log(today);
10
11 var theHour=today.getHours();
12 var theMinutes=today.getMinutes();
13 var theSeconds=today.getSeconds();
14 if(theSeconds<10) theSeconds='0'+theSeconds;
15 var digitalClock=theHour+':'+theMinutes+':'+theSeconds;
16 console.log(digitalClock);
```

```
Mon Aug 09 2021 18:34:50 GMT+0200 (heure d'été d'Europe centrale)
18:34:50
```

It is necessary to document the numerous methods of the Date object, to experiment with them, to learn them, to integrate them into your developments.

Despite many methods, the manipulation of dates is not always easy, for example to calculate a difference in days between 2 dates, ... or simply to construct a precise date, other than the current one.

MomentJS API

Application Programming Interface : a JS library which facilitates the manipulation of dates, without having to use the native Date() object.

Run the API from an external JS source, a CDN (Content Delivery Network). Search the script moment.js , or a minified version, e.g. :

```
7  <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.1/moment.min.js"></script>
```

```
19  //API momentJS
20  var today=moment();
21  console.log(today.format('DD/MM/YYYY'));
22                                  10/08/2021
23
24
25  let nextWeek=today.add(1,'weeks');
26  console.log(nextWeek.format('YYYY-MM-DD'));
27                                  2021-08-17
28
29
30  let lastDayOfYear=moment().endOf("year").format('dddd DD/MM/YYYY');
31  console.log(lastDayOfYear);                          Friday 31/12/2021
32
33
34  var day1=moment('2021-08-11');
35  var day2=moment([2021,7,1]);
36  var dayDifference=day1.diff(day2,'days');
37  console.log(dayDifference);              10
```

No instance of an object (~~new~~), only a function call : moment(), which returns the object

See the documentation for the syntax of the string format

See the documentation for the argument of a given method

It is necessary to document the numerous methods of the momentJS object, to experiment with them, to learn them, to integrate them into your developments.

Array Object

An array object has only 1 property, the *length* of the array, i.e. the number of elements, and numerous method to manipulate, to transform the array.

```
28 let subjectList=['HTML','CSS','Javascript','PHP'];
29 let nbElements=subjectList.length;
30 console.log(subjectList);
31 console.log("Nb elements : "+nbElements);
```

```
▶ (4) ["HTML", "CSS", "Javascript", "PHP"]
Nb elements : 4
```

Here are some methods that are easy to understand, document and experiment with.

```
33 //Array.isArray()
34 if(Array.isArray(subjectList)){
35     console.log("Is subjectList an Array ? "+Array.isArray(subjectList));
36 } else {
37     console.log("Is subjectList an Array ? "+Array.isArray(subjectList));
38 }
```

```
Is subjectList an Array ? true
```

isArray() is a static method, so it is applied to the Array object, with a variable as argument.

```
40 //add elements after the last array element
41 subjectList.push('MySQL','Python');
42 nbElements=subjectList.length;
43 console.log(subjectList);
44 console.log("Nb elements : "+nbElements);
45 //remove the last array element
46 let elementRemoved=subjectList.pop();
47 console.log(subjectList);
48 console.log("Element removed : "+elementRemoved);
```

```
▶ (5) ["HTML", "CSS", "Javascript", "PHP", "MySQL"]
Element removed : Python
```

```
51 //sorting an array
52 subjectList.sort();
53 console.log(subjectList);
```

```
▶ (5) ["HTML", "CSS", "Javascript", "PHP", "MySQL"]
Element removed : Python
▶ (5) ["CSS", "HTML", "Javascript", "MySQL", "PHP"]
```

Before…

… After

```
55 //join() returns a String by concatenating the array elements
56 subjectListString=subjectList.join('-');
57 console.log('typeof : '+typeof subjectList+"- isArray ? "+
       Array.isArray(subjectList));
58 console.log('typeof : '+typeof subjectListString);
59 console.log(subjectListString);
```

```
typeof : object- isArray ? true
typeof : string
CSS-HTML-Javascript-MySQL-PHP
```

The split() function of the String object performs the reverse of the join() function of the Array object.

```
61 //split() returns an Array by dividing a string accoding
   a pattern
62 let htmlTagsString="<
       header>,<nav>,<section>,<article>,<footer>";
63 let htmlTagList=htmlTagsString.split(',');
64 console.log(htmlTagList);
```

```
▶ (5) ["<header>", "<nav>", "<section>", "<article>", "<footer>"]
```

Data Format & Manipulation : JSON

JSON (JavaScript Object Notation) is a text-based structured data used in JS
or to exchange data with a web server.
This text format is human readable, like XML, and machine readable.

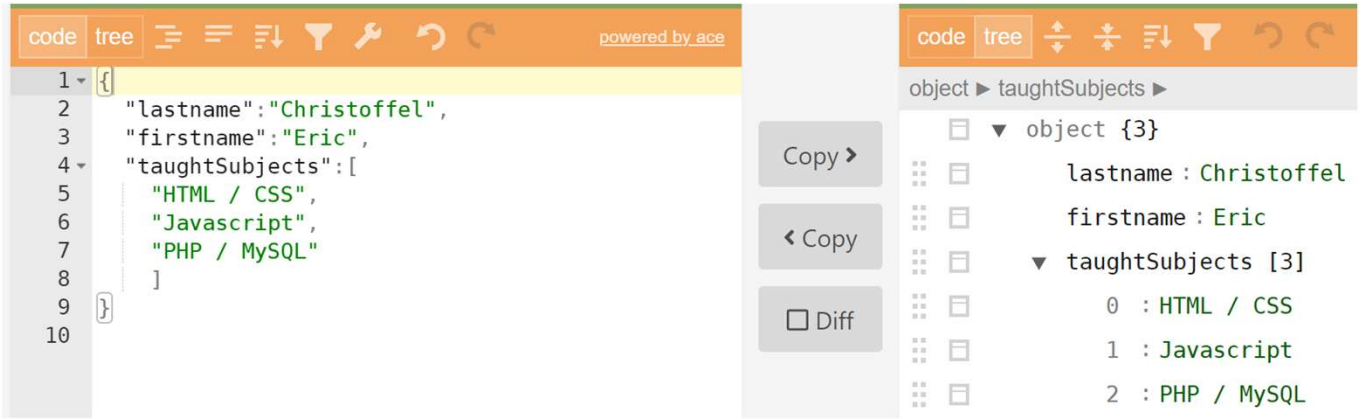
The 3 basic structures of JSON data are :

- An Object, enclosed with curly brackets *{ }*
- A *property : value* pair
- An Array, enclosed with squared brackets *[ ]*

```
1 {
2   "lastname":"Christoffel",
3   "firstname":"Eric",
4   "taughtSubjects":[
5     "HTML / CSS",
6     "Javascript",
7     "PHP / MySQL"
8     ]
9 }
```

An Object *{ }* is a collection of *property:value* pairs, seperated with a , comma (see lines 2, 3 and 4)
The *property* name is surrounded by double quotes, as well as the *value* for strings.
The *property* and its *value* are seperated with a *:* colon (see code line 2)
The value can be of type : Number, String, Boolean, Null, an Array (see code line 4 to 8), an Object
An Array element can be of type String (see lines 5, 6, 7), a Number, an Object.

Online tools can help you build your JSON structure : https://jsoneditoronline.org/
A *code* and a *tree* views are available. The tree view helps to understand the structure, which may be quite complex.



Copy the compact JSON structure (without indentation), and assign the value to a JS variable.


```
8  let data={"lastname":"Christoffel","firstname":"Eric",
       "taughtSubjects":["HTML / CSS","Javascript","PHP / MySQL"]};
9  console.log(data);
```

Note: data={ } means that the variable data is an object. It is not a String, so do not enclose it in double quotes!

```
11  console.log(typeof data);      object
```

To access a JSON *property*, use the . dot notation.
If the value of this property is of Array type, use the [ ] square brackets, for the corresponding *property*.

```
14 console.log("lastname is "+data.lastname);
15 console.log("first taught subject is "+data.taughtSubjects[0]);
```

```
lastname is Christoffel
first taught subject is HTML / CSS
```

Alternatively, the square bracket notation can also be used, with the *property* name as a String argument

```
16 console.log("lastname is "+data['lastname']);
```

```
lastname is Christoffel
```

The method JSON.stringify(), with a JSON object passed as an argument, converts a JSON Object to a String:

```
17 console.log(data);
18 console.log("Type of "+typeof data);
19 let dataString=JSON.stringify(data);
20 console.log(dataString);
21 console.log("Type of "+typeof dataString);
```

```
▼{lastname: "Christoffel", firstname: "Eric", taughtSubjects: Array(3)}
    firstname: "Eric"
    lastname: "Christoffel"
  ▶ taughtSubjects: (3) ["HTML / CSS", "Javascript", "PHP / MySQL"]
  ▶ [[Prototype]]: Object
Type of object
{"lastname":"Christoffel","firstname":"Eric","taughtSubjects":["HTML / CSS","Javascript","PHP / MySQL"]}
Type of string
```

Conversely, the method JSON.parse() with a String passed as an argument, converts a String, that respects the syntax and structure of JSON, to a JSON data:

```
23 let dataJSONfromString=JSON.parse(dataString);
24 console.log(dataString);
25 console.log(dataJSONfromString);
```

```
{"lastname":"Christoffel","firstname":"Eric","taughtSubjects":["HTML / CSS","Javascript","PHP / MySQL"]}
▼{lastname: "Christoffel", firstname: "Eric", taughtSubjects: Array(3)}
    firstname: "Eric"
    lastname: "Christoffel"
  ▶ taughtSubjects: (3) ["HTML / CSS", "Javascript", "PHP / MySQL"]
```

These methods are widely used when a web application exchanges date with a web server, or when working with the Web Storage API to store data locally in the user's browser.

Advance Control Statements

The **for** … **in** loop iterates over *enumerable properties* of an object. Example with a JSON object :

```
27  for(const property in data){
28      console.log('Property name : '+property+' ,
            value : '+data[property]);
29  }
```

```
Property name : lastname , value : Christoffel
Property name : firstname , value : Eric
Property name : taughtSubjects , value : HTML / CSS,Javascript,PHP / MySQL
```

Here, the constant *property* of the **for in** loop is used as an argument of the JSON object using the sqare bracket notation.

The **for** … **of** loop iterates over *iterable objects*, typically an Array, or a String :

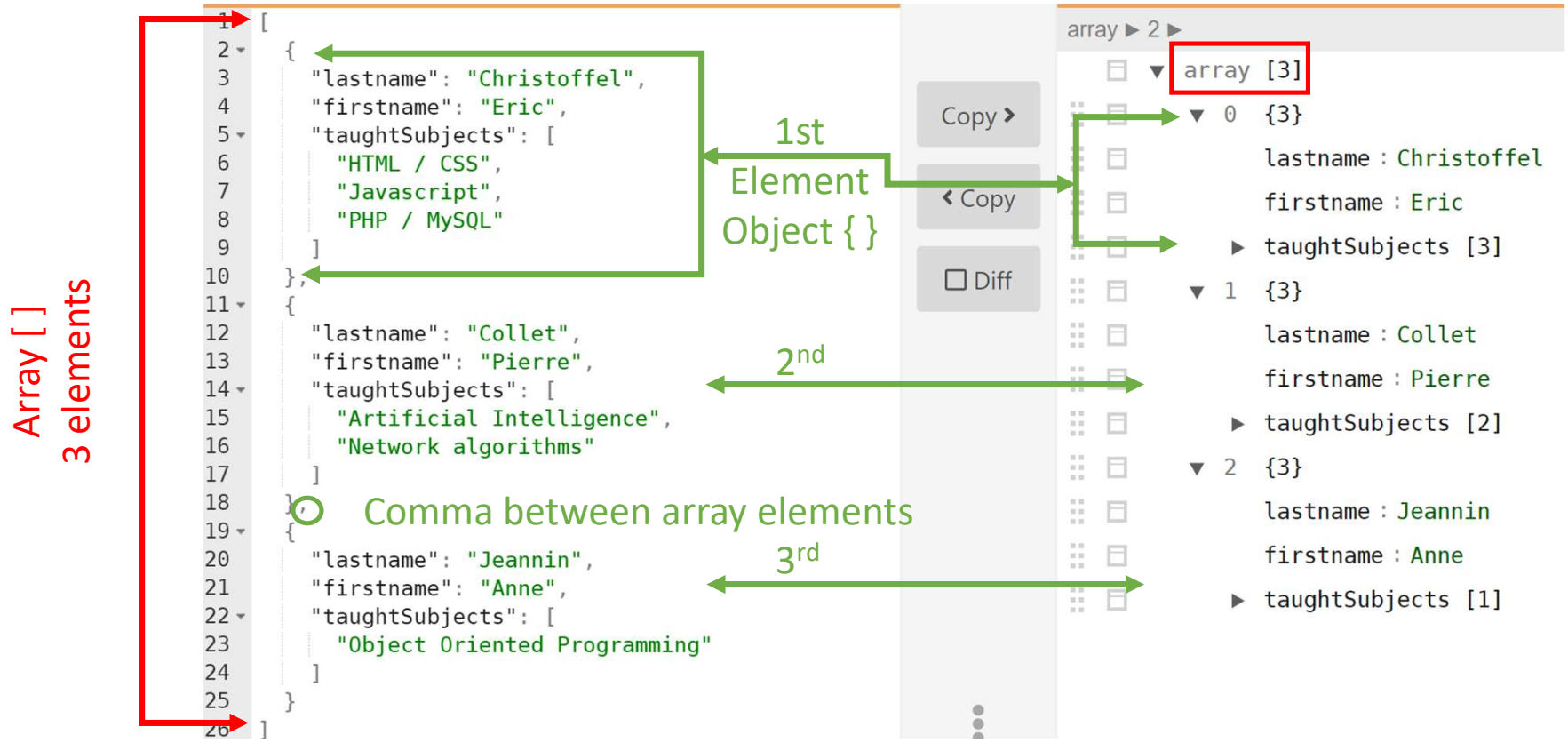
```
31  console.log("is array ? "+Array.isArray(data.taughtSubjects));
32  for(const subject of data.taughtSubjects){
33      console.log(subject);
34  }
```

```
is array ? true
HTML / CSS
Javascript
PHP / MySQL
```

```
35  console.log(typeof data.lastname);
36  for(const letter of data.lastname){
37      console.log(letter);
38  }
```

```
string
C
h
r
i
s
t
o
f
f
e
l
```

The **forEach()** method executes a function for each element of an array.
This is a method of the Array object, which is applied as a loop on each element of the array.
The JSON data :



```
8  let teacherList=[{"lastname":"Christoffel","firstname":"Eric","
       taughtSubjects":["HTML / CSS","Javascript","PHP / MySQL"]},{"
       lastname":"Collet","firstname":"Pierre","taughtSubjects":["
       Artificial Intelligence","Network algorithms"]},{"lastname":"
       Jeannin","firstname":"Anne","taughtSubjects":["Object Oriented
       Programming"]}];
9
10 console.log(teacherList);
11
12 teacherList.forEach(function(item){
13 console.log(item.lastname);
14 });
```

```
▼(3) [{…}, {…}, {…}]
  ▶0: {lastname: "Christoffel", firstname: "Eric", taughtSubjects: Array(3)}
  ▶1: {lastname: "Collet", firstname: "Pierre", taughtSubjects: Array(2)}
  ▶2: {lastname: "Jeannin", firstname: "Anne", taughtSubjects: Array(1)}
    length: 3
  ▶[[Prototype]]: Array(0)
Christoffel
Collet
Jeannin
```

Syntax

```
Array.forEach(function(item){

});
```

- Parentheses for the *forEach* method
- The argument is a *function*
- Parentheses for the function argument
- *item* returns each array element
- Curly brackets for function delimiters
- Semicolon ; at the end of the JS instruction

Arrow functions

Arrow functions are a compact writing code for functions, which should not be used systematically.
The example below compares a function and the corresponding arrow functions, written with different syntaxes.

```
9  const _VATrate=20/100;
10 console.log("0. Ammount excluding VAT = 1000 , Amount
       including VAT = "+amountIncludingVAT(1000));
11
12 console.log("0. Ammount excluding VAT = 2000 , Amount
       including VAT = "+amountIncludingVAT(2000));
13
14
15 //traditional function
16 function amountIncludingVAT(amount){
17     return amount*(1+_VATrate);
18 }
```

```
0. Ammount excluding VAT = 1000 , Amount including VAT = 1200
0. Ammount excluding VAT = 2000 , Amount including VAT = 2400
```

```
22 //arrow function, syntax param1 => expression
23 var amountIncludingVAT_V1 = amountExcludingVAT =>
       amountExcludingVAT*(1+_VATrate);
24
25 console.log("1. Ammount excluding VAT = 1000 , Amount
       including VAT = "+amountIncludingVAT_V1(1000));
26
27 console.log("1. Ammount excluding VAT = 2000 , Amount
       including VAT = "+amountIncludingVAT_V1(2000));
```

```
1. Ammount excluding VAT = 1000 , Amount including VAT = 1200
1. Ammount excluding VAT = 2000 , Amount including VAT = 2400
```

```
31 //arrow function, syntax (param1, paramN) => expression
32 var amountIncludingVAT_V2 = (amountExcludingVAT,
       VAT_rate) => amountExcludingVAT*(1+VAT_rate);
33
34 console.log("2. Ammount excluding VAT = 1000 , Amount
       including VAT = "+amountIncludingVAT_V2(1000,0.2));
35
36 console.log("2. Ammount excluding VAT = 2000 , Amount
       including VAT = "+amountIncludingVAT_V2(2000,0.2));
```

```
2. Ammount excluding VAT = 1000 , Amount including VAT = 1200
2. Ammount excluding VAT = 2000 , Amount including VAT = 2400
```

```
40 //arrow function, syntax (param1, paramN) =>
   {expressions;return}
41 var amountIncludingVAT_V3 = (amountExcludingVAT,
       VAT_rate) => {
42     let VAT=amountExcludingVAT*VAT_rate
43     return amountExcludingVAT+VAT;
44 }
45
46 console.log("3. Ammount excluding VAT = 1000 , Amount
       including VAT = "+amountIncludingVAT_V3(1000,0.2));
47
48 console.log("3. Ammount excluding VAT = 2000 , Amount
       including VAT = "+amountIncludingVAT_V3(2000,0.2));
```

```
3. Ammount excluding VAT = 1000 , Amount including VAT = 1200
3. Ammount excluding VAT = 2000 , Amount including VAT = 2400
```

Template literals

Up to now, when writing text and variables into a **String** variable, we used the concatenation operator : **+**
The *concat()* method of the **String** object can also be used.
Alternatively, ***template literals*** can be used, which are much more powerfull, and much more efficient for dynamic writing in JS (see later).

Syntax

The variables must be written between { } curly brackets, and preceded by the **$** character
The string, which consist of text and variables must must be enclosed by the backtick ` `

${variableName}
`text… **${variableName}** …text`

```
9  let lastname="Christoffel";
10 let firstname="Eric";
11
12 let fullname1="Fullname : "+lastname+' '+firstname;
13 console.log(fullname1);
14
15 let fullname2=lastname.concat(' ',firstname,' is the fullname');
16 console.log(fullname2);
17
18 let fullname3=`Fullname ${lastname} ${firstname}`;
19 console.log(fullname3);
```

```
Fullname : Christoffel Eric
Christoffel Eric is the fullname
Fullname Christoffel Eric
```