JS - ES6 à ES10

Gérard Rozsavolgyi
roza@univ-orleans.fr

Les bases du développement avec JS
Objets usuels JS
Manipulation du DOM
Arrays
Objets et JSON
Classes

JS

Langage "client" par excellence

- Calculs
- Aide ergonomique / Menus
- Animations
- Saisie et vérification de données
- Galleries photo
- Langage pivot en

Propriétés de JS

- extension de HTML : `onmouseover()` ou `onclick()`
- extension pour la validation de formulaires
- Menus - Animations
- Ajax (Interactivité et fluidité)

Universalité

- Langage interprété par tout navigateur
- Langage Pivot en Cross-Dev : Ionic, Unity3D
- Existe côté serveur : NodeJS

Origines

- LiveScript développé par Netscape
- Repris par Sun fin 1995 sous le nom de JavaScript
- Normalisation européenne sous le nom de ECMA Script 6 en 2015

Attention

JavaScript, ECMAScript != JAVA

Seulement quelques ressemblances très limitées

HTML5 et JS

- Dessins sur Canvas
- GeoLocalisation
- Recherche d'éléments dans le DOM: `document.getElementsById('xx')`
- Sauvegarde hors-ligne

Fonctions en JS

```
function carre(i){
  return i*i;
}
```

- paramètres non typés
- return non obligatoire

HTML et JS

```
<script>
  let s = "Bonjour à vous !";
  console.log(s);
</script>
<!-- ou -->
<scriptn src="myscript.js"></script>
```

Initiation JS

Travail sur la console Web

Inutile d'avoir même un editeur de code !

Windows	macOS	Linux
`Ctrl` + `Maj` + `K`	`Cmd` + `Opt` + `K`	`Ctrl` + `Maj` + `K`

Essayez =>

Recharger une page

(Après modification)

On peut recharger une page Web (au moins sous Firefox) en désactivant le cache :

- En tenant appuyée la touche "Shift" tout en pressant l'icône de rechargement.
- "Ctrl + F5" ou "Ctrl + Shift + R" (Windows,Linux)
- "Command + Shift + R" (Mac)

Déclaration de variables en JS

- var a = 3
- let s = "bonjour"
- const f = 3.0

VOUS AVEZ QUELQUE CHOSE À DÉCLARER ?

JavaScript dispose de 3 manières de déclarer des variables.

- *var* est la méthode historique mais correspond à une portée (scope) très particulière (cf exos)
- *let* est la façon standard de déclarer une variable avec une portée "normale"
- *const* permet de déclarer une variable constante de type primitif ou une variable qui *ne sera pas réallouée ultérieurement* (un tableau peut-être const par exemple)

Objets usuels

- `String, Number, Date, Math, Array, RegExp`
- Objets liés à la fenêtre : `Window, document, parent, top, frames[ ], screen`…
- Objets liés au navigateur : `Navigator, location, history`
- Objets liés au document HTML : `HTMLElement : Form, Checkbox, input, select, submit`, etc.

Lecture de propriétés système

- navigator.appCodeName
- navigator.appName
- navigator.language
- navigator.platform
- navigator
- navigator.userAgent
- screen.availHeight
- screen.availWidth

Méthodes de l'objet window

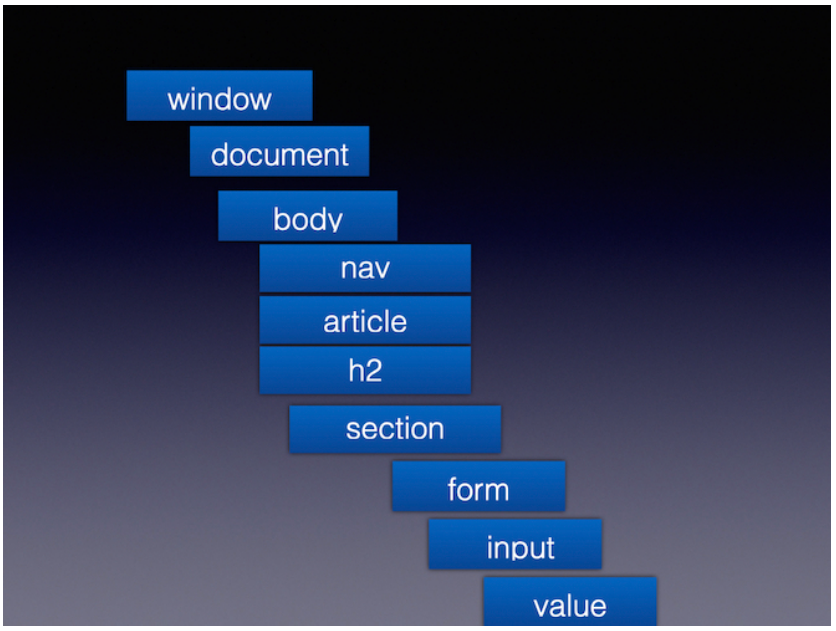
- open
- prompt
- alert
- confirm

Ouvrir un lien vers une autre fenêtre

```
<a href="autre-doc.html" onClick="new_win('autre_doc.html')">
Texte du lien</a>
<script>
function new_win(url) {
    window.open(url,'MaNouvelleFenetre','menubar,location,resizabl
}
//ou :
function new_window(url) {
window.open(url,'new','width=300,height=200,toolbar=no,location=nc
}
</script>
```

DOM

- Document Object model
- Structure arborescente d'une page



Exploration du DOM en JS

```
- document.getElementById()
- document.getElements ByTagName()
- document.getElementsByClassName()
- document.querySelector()
- document.querySelectorAll()
```

Exemple lecture DOM

```
<img id=«monimage» src="toto.png" width="50px" />
<script>
  let image = document.getElementById("monimage");
  let imgurl = image.src;
  let width = parseInt(image.getAttribute("width"));
  image.setAttribute("class", "mini");
</script>
```

Modification du DOM en JS

```
titre = document.createElement("<h1>");
texte = Document.createTextNode("Mon titre")
titre.appendChild(texte)
```

insertion à un emplacement spécifique dans le DOM

```
function insere_dom(parent, child, n){
  if (n<0 || n > parent.childNodes.length)
      throw new Error("index invalide");
  else
    if (n == parent.childNodes.length)
           parent.appendChild(child);
    else
      parent.insertBefore(child, parent.childNodes[n]);
}
```

Objets littéraux JS - JSON

```
const vide = {};
let point = {x:0, y:0};
let etudiant = {
    'nom': "Jean-Claude",
    "numero secu": "13141592653",
    'naissance': "27/09/2000",
    formation: {
      intitule: "M1 Info",
      annee:"2021-2022",
      groupe:"XYZ"
    }
}
let secu = etudiant["numero secu"]
```

Parcours des propriétés d'un Objet JS

Grâce à la boucle for/in

```
for(let p in etudiant){
  console.log(p + ' : ' + etudiant[p] )
}
```

Arrays JS

Parcours classique ou for/of

```
let langages = [ "C", "C++", "C#", "JavaScript" ];
for(let i=0;i<langages.length;i++)
   console.log(langage[i])
//ou
for (language of languages) {
    console.log(language);
}
```

Arrays

Tris: sort

```
etudiants = ['Jules', 'agatha', 'asterix',
 'Idefix', 'Zazou', 'Robert'  ];
etudiants.sort()
```

Si le tri par défaut ne nous plait pas:

```
etudiants.sort(function(a,b){
    x=a.toLowerCase();
    y=b.toLowerCase();
    if (x<y) return -1;
    else if (x>y) return 1;
    else return 0;
});
```

On redéfinit le tri souhaité avec une `fonction anonyme`

Arrays

Tris: sort avec 'fat arrow'

```
etudiants.sort(
    (a,b) => (b.toLowerCase() < a.toLowerCase() )
);
```

String: ajout d'une méthode reverse au Prototype de String qui renvoie la chaîne inversée

```
    String.prototype.reverse=function(){
        return this.split("").reverse().join("");
}
```

Arrays
Utilisation de map
pour inverser tous les prénoms

```
invetuds = etudiants.map(
    x =>  x.toLowerCase().reverse()
    );
console.log(invetuds);
```

Arrays : Différences de 2 tableaux
Utilisation de map

```
const a = [3, 4, 7]
const b = [3, 9, 10]
const c = b.map((e, i) => e - a[i])
```

// Donne [0, 5, 3]

Arrays

Filtrage des étudiants

dont le nom commence par une lettre < 'k'

```
etuds = etudiants.etudiants.filter(
            x=> (x.toLowerCase() < "k")
        );
console.log(etuds);
```

Arrays
Chainage

```
[1, 2, 3, 4]
    .map(value => value * 2)
    .filter(value => value > 2)
    .forEach(value => console.log(value))
// 4, 6, 8
```

> NB: map renvoie un nouvel objet tandis que forEach opère *sur place*

Arrays
reduce

```
let prod = [1, 2, 3, 4, 5, 6]
            .reduce( (x,y) => x*y, 1);
console.log(prod); // 720
```

ES6

- let et const pour la portée des variables
- boucles for of
- Mot clef class
- Générateurs: yield
- Paramètres variables dans les appels de fonction
- Templates String
- Objets littéraux
- Arrow function ("lambda")
- modules
- Promises

Classes JS

```
class Personne{
  constructor(id, prenom, nom){
      this._id = id;
      this._nom = nom;
      this._prenom = prenom;
  }
  toString(){return `${this._nom} ${this._prenom}`;}
  get nom() { return this._nom.toLowerCase();}
  set nom(newNom){
      if(newNom){
          this._nom = newNom;
      }
  }
};
let p = new Personne(3,'John', 'Doe');
console.log(p);
p.nom='Azerty';
console.log(p.nom);
```

Arrays (avancé)

Destructuration

Exemple pour échange

```
a = 65;
b = 99;
[a, b] =  [b, a];
console.log(a,b);
```

Arrays

Destructuration

Exemples dans des tableaux

```
/// from https://developer.mozilla.org/fr/docs/Web/JavaScript/Refe
let articulations = ['épaules', 'genoux'];
let corps = ['têtes', ...articulations, 'bras', 'pieds'];
// ["têtes", "épaules", "genoux", "bras", "pieds"]
```

Arrays

Destructuration

Exemples de concaténations de tableaux

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];
arr1 = [...arr1, ...arr2];
// arr1 vaut désormais [0, 1, 2, 3, 4, 5]
```

Arrays
Zip "à la Python"
grâce à la restructuration ...

```
let t1 = ['A', 'B', 'C', 'D']
let t2 = [1,2,3]
const zip = ([x,...xs], [y,...ys]) => {
  if (x === undefined || y === undefined)
    return []
  else
    return [[x,y], ...zip(xs, ys)]
}
zip(t1, t2)
```

Donne : [['A', 1], ['B', 2], ['C', 3]]

Quicksort

```
// Quicksort avec destructuration, filter et fat arrow
const quicksort = array => {
    if (array.length === 0) return [];
    let [x, ...xs] = array;
    return [
        ...quicksort(xs.filter(y => y < x)),
        x,
        ...quicksort(xs.filter(y => y >= x))
    ];
};

let tab = [17,3,5,8, -1,12];
console.log(quicksort(tab))
```

Arrays

Produit cartésien

Pour le fun !

```
const
values = [['a', 'b'], ['c', 'd', 'e'], ['f', 'g', 'h']],
result = values.reduce(
    (acc, curr) => acc.reduce((r, v)
    => r.concat(curr.map(w => [].concat(v, w))), []));
result.map(a => console.log(...a));
```

Donne : a c f, a c g, a c h etc.

Arrays

Destructuration

Memoization

```
/// from https://medium.com/front-end-hacking/today-i-learned-memo
let memoize = fn => {
  let cache = {};
  return (...args) => {
    let stringifiedArgs = JSON.stringify(args);
    let result = cache[stringifiedArgs] = cache[stringifiedArgs] |
    return result;
  };
};
```

Arrays

Memoization exemple

```
function facto(n){
    if (n <=1 ) return 1 ; else return n*facto(n-1);
}
let memoizedFacto = memoize(facto);
memoizedFacto(17);
memoizedFacto(17);
```

Générateurs avec function* et yield

```
// Une fonction fibo génératrice
function* fibonacci() {
    let x = 0, y = 1;
    while(true) {
        yield y;
        [x,y] = [y,x+y];
    }
};
// Obtenir un générateur
let f = fibonacci();
// Puis employer le générateur comme itérateur
console.log("fibo");
for(let i = 0; i < 10; i++) console.log(f.next());
```

Fonction range comme en Python

```
function* range(min, max) {
  for(let i = Math.ceil(min); i <= max; i++) yield i;
};
```

Connexion à une API depuis JS

2 Manières incontournables d'interroger une API en JS

- JQuery
- fetch/then

Avec JQuery

```
$.ajax(
    {
    url: "http://localhost:3000/tasks",
    type: "GET",
    dataType : "json",
    success: function(tasks) {
              console.log(JSON.stringify(tasks));
              for(let i=0;i<tasks.length;i++){
                 console.log(tasks[i]);
            }
        },
    error: function(req, status, err) {
           console.warn("Erreur: "+err+" "+status)
        }
     });
```

Idem avec fetch/then

```
requete = "http://localhost:3000/tasks";
fetch(requete)
.then( response => {
       if (response.ok) return response.json();
       else throw new Error('Problème ajax: '+response.status);
       })
 .then(listerTaches)
 .catch(onerror);
```

Que des promesses

- *fetch/then* utilise des *Promesses* ou *Promises*
- Les promesses sont une manière de *réifier* l'attente d'un résultat dans un appel *asynchrone* très courant en JS
- On ne sait pas quand ce résultat sera disponible alors on fait la requête puis on attend que le résultat soit disponible grâce au *then*
- On peut également utiliser la combinaison *await/async* provenant de Python comme on le verra ensuite

Requêter une API en JS

- fetch/then
- await/async
- await / for

fetch/then simple avec querystring

```
// si node
require('isomorphic-fetch');
let querystring = require('querystring')
let data = { q:'Orleans', APPID:API_KEY, units: 'metric'}
url = "http://api.openweathermap.org/data/2.5/weather?"+querystrir
console.log(url)
fetch(url)
      .then(x => x.json())
      .then(xx => {
              console.log(xx);
          });
```

fetch/then avec Objet URL et module node

```
const { URL } = require('url');
require('isomorphic-fetch');
class OpenWeatherMap{

    constructor(APPID, units) {
        this._APPID = APPID;
        this._units = units;
        this.weather = {};
  }
getWeather(city){
    const urlow = new URL("http://api.openweathermap.org/data/2.5/
        params = {q:city, APPID: this._APPID, units: this._units};
    Object.keys(params).forEach(key => urlow.searchParams.append(k
    return fetch(urlow.toString())
            .then(response => response.json());
    }
};
module.exports.OpenWeatherMap = OpenWeatherMap;
```

Requête de traduction en node

```
require('isomorphic-fetch');
let querystring = require('querystring');
const source = "fr";
const cible = "en";
const phrase = "il fait très beau aujourd'hui les amis";
let data = {sl:"fr", tl:"en",dt:"t",q:phrase};
// on fabrique la requête
const query = querystring.stringify(data);
console.log(query);
url = "https://translate.googleapis.com/translate_a/single?client=
console.log(url);
fetch(url)
      .then(x => x.json())
      .then(xx => {
            console.log("la traduction de \""+xx[0][0][1]+"\" est
            console.log(xx[0][0][0]);
          });
```

Traduction avec Objet URL (possible dans un navigateur)

```
const source = "fr";
const cible = "en";
const phrase = "il fait très beau aujourd'hui les amis";
// on fabrique la requête
const url = new URL("https://translate.googleapis.com/translate_a/
params = {sl:"fr", tl:"en",dt:"t",q:phrase};
Object.keys(params).forEach(key => url.searchParams.append(key, pa
fetch(url)
.then(x => x.json())
.then(xx => {
    console.log("la traduction de \""+xx[0][0][1]+"\" est : ");
    console.log(xx[0][0][0]);
    });
```

Avec await/async : Méthode 1

```
// si nodejs
require('isomorphic-fetch');
let Url = "https://ipinfo.io/json";
async function getInfo(url) {
  const response = await fetch(url);
  return response.json();
}

(async function main() {
 let ipInfo = await getInfo(Url);
 console.log(ipInfo);
})(); // code auto-exécutable :IIFE
```

Avec async/await: Méthode 2

```
// nodejs
require('isomorphic-fetch');
let Url = "https://ipinfo.io/json";
async function getInfo(url) {
  const response = await fetch(url);
  return await response.json();
}

getInfo(Url)
  .then(ipInfo => console.log(ipInfo));
```

Avec GoogleMaps:

```
// si nodeJS
const fetch = require("isomorphic-fetch");
const API_KEY= 'votre clef API'
const url = "https://maps.googleapis.com/maps/api/geocode/json?adc
const getLocation = async url => {
  try {
    const response = await fetch(url);
    const json = await response.json();
    console.log(
      `City: ${json.results[0].formatted_address} -`,
      `Latitude: ${json.results[0].geometry.location.lat} -`,
      `Longitude: ${json.results[0].geometry.location.lng}`
    );
  } catch (error) {
    console.log(error);
  }
};
location = getLocation(url);
console.warn(location);
```

async/await download files

```
// pour node
require('isomorphic-fetch');
const downloadFile = (async (url, path) => {
      const res = await fetch(url);
      const fileStream = fs.createWriteStream(path);
      await new Promise((resolve, reject) => {
          res.body.pipe(fileStream);
          res.body.on("error", (err) => {
            reject(err);
          });
          fileStream.on("finish", function() {
            resolve();
          });
       });
    });
```

Gérer plusieurs promesses

```
// https://developer.mozilla.org/en-US/docs/Web/JavaScript/Referer
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise(function(resolve, reject) {
  setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3]).then(function(values)
  console.log(values);
});
// Donne : Array [3, 42, "foo"]
```

Promise.resolve

envoie un objet *Promise* qui est résolu avec la valeur donnée.

- si cette valeur est une promesse, la promesse est renvoyée,
- si la valeur possède une méthode then, la promesse renvoyée « suivra » cette méthode et prendra son état
- sinon, la promesse renvoyée sera tenue avec la valeur

Promise.resolve exemple

```
// Resolving a thenable object
let p1 = Promise.resolve({
  then: function(onFulfill, onReject) {
       onFulfill('succes !');
       }
});
console.log(p1 instanceof Promise) // Vrai
p1.then(function(v) {
    console.log(v); // "succes !"
  }, function(e) {
    // En cas d'erreur
});
```

Asynchrone

PROGRAMMATION ASYNCHRONE

La programmation *Asynchrone*, en particulier basée sur les *Promesses* est délicate à mettre en oeuvre, demande beaucoup de mise au point et de *tests* mais peut s'avérer très stimulante et très puissante ...