

Cours Javascript

PSB 2020
Y. Stroppa

Bibliographie

- Javascript et JQuery La programmation web par la pratique : Frédéric Delobel -- Eni édition
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Language_Resources
- Si besoin de valider du code html
 - http://validator.w3.org/#validate_by_uri
- Pour les documents spécifiques au cours : voir site
 - <https://github.com/ystroppa/>

Conditions de notation

- DS à la fin de notre formation
- et projet de réalisation d'une application Javascript
 - projets proposés
 - Jeu de l'âne rouge
 - Jeu de rush hour
 - Jeu de puzzle
 - Jeu du taquin
 - Fiver
 - Et autres en fonction de vos motivations et envie

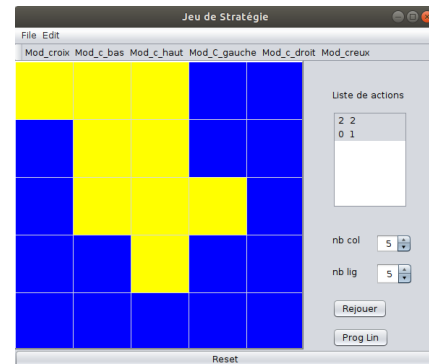
Présentation des projets



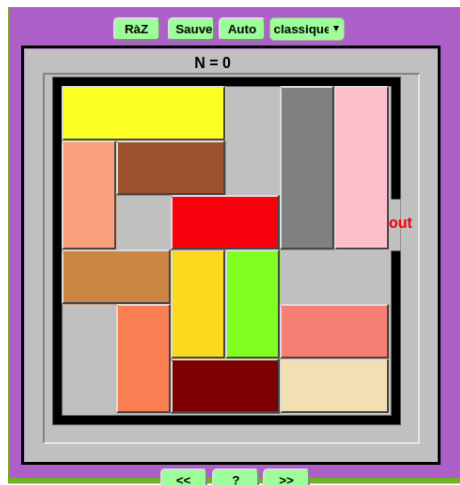
Taquin



Ane Rouge



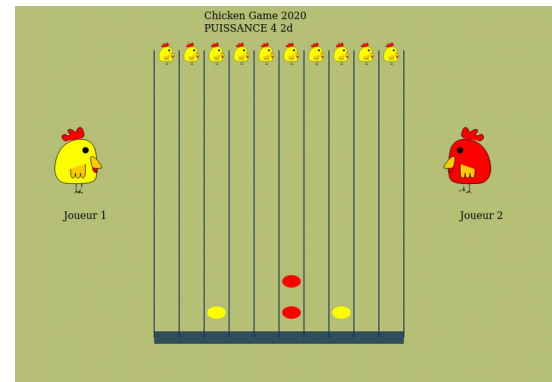
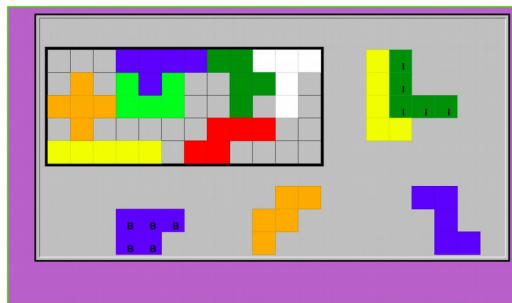
Fiever



Rush Hour

Puzzles à compléter

Jeu du KATAMINO 2020



Pouissance 4

Sommaire

- Javascript
- Fonctionnement de Javascript
- Les notions élémentaires
- les bases de Javascript
- Les structures de contrôle
- Javascript en profondeur
- Aller plus loin en Javascript

Exemple Mongodb

- Exemple d'extractions de données
 - La base de données NoSQL Mongodb est une base de type Document. Sa structure interne est composée :
 - de base de données
 - de collections
 - de documents
 - Un document est une structure de type json
- Le langage d'interrogation est Javascript

```

function (handle="") {
    var Resultats={},cursor={},var refs={};
    var cursor_personnels=db.personnels.findOne({"amf.person.handle":handle});
    if (cursor_personnels["amf"]["person"]["ispartof"]) {
        var ref_id="";
        if (cursor_personnels["amf"]["person"]["ispartof"]["organization"]) {
            if (cursor_personnels["amf"]["person"]["ispartof"]["organization"] instanceof Array){
                ref_id = cursor_personnels["amf"]["person"]["ispartof"]["organization"]["ref"][0];
            } else {
                ref_id= cursor_personnels["amf"]["person"]["ispartof"]["organization"]["ref"];
            }
        }
        if (ref_id != undefined )
            if (refs[ref_id]){
                refs[ref_id]["value"]+= 1;
            }
            else {
                refs[ref_id]={ "value":1};
            }
        }
    }
    for (var ref_ in refs) {
        // il faut rechercher les infos des etablisements cibles
        Resultats[ref_]= get_etablissement(ref_,false);
        Resultats[ref_]["value"]= refs[ref_]["value"];
    }
    return Resultats;
}

```

▼ Collections (15)

- System
- coll_citations
- coll_coauteurs
- etablissements
- index_PNR
- iscited
- personnels
- personnels_feminin_top10
- personnels_top5_all
- publications
- revuesV1
- revuesV3
- top5_allbis
- top_complet

▼ Functions (22)

- fx* get_auteur_etablissements
- fx* get_auteur_revue
- fx* get_citations_auteur
- fx* get_citations_pape...

▼ (3) ObjectId("5c8c07a4d4459842e11cd4... { 2 fields }

- _id ObjectId("5c8c07a4d4459842e11cd453")
- amf { 5 fields }
 - xsi:schemaLocation http://amf.openlib.org http://amf.openlib.org/20... String
 - xmlns:xsi http://www.w3.org/2001/XMLSchema-instance String
 - person { 8 fields } Object
 - ispartof { 1 field } Object
 - familyname ZÃ¡rate-Marco String
 - isauthorof azarate@unizar.es String
 - givenname [13 elements] Array
 - handle Anabel String
 - name pzar4 String
 - id Anabel ZÃ¡rate-Marco String
 - xmlns:repec RePEC:per:2013-05-07:anabel_zarate_marco String
 - xmlns http://repec.openlib.org String

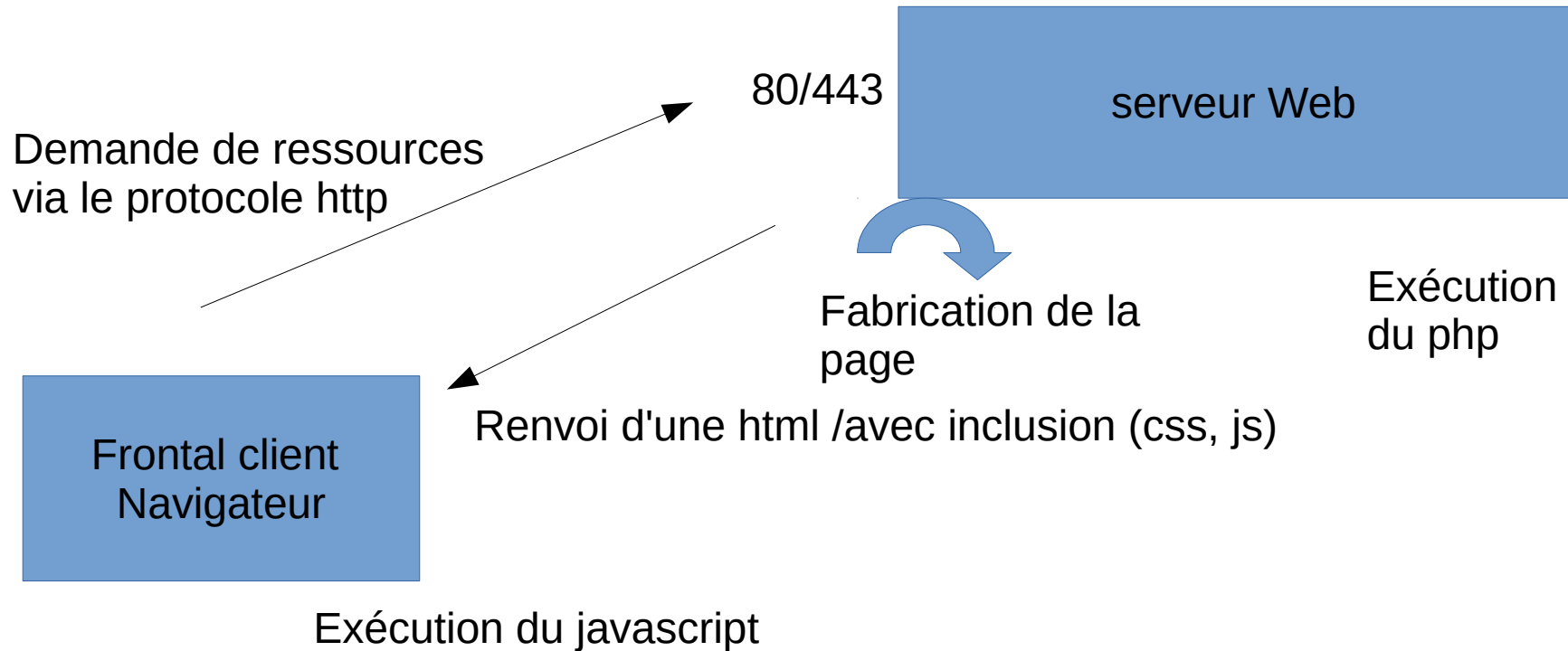
Introduction

- Né en 1995 --> Brendan Eich → programmeur chez Netscape
- En 97, succès → standard est créé -->normalisation par ECMA International ECMA-262
⇒ ECMAScript
 - Version actuelle : ECMAScript V8 en 2017
 - Pour plus de détail voir : https://developer.mozilla.org/fr/docs/Web/JavaScript/Language_Resources
- Javascript est un langage orienté Objet
- Javascript pourquoi faire ?
 - utilisation la plus connue à travers les navigateurs web
 - apporter une couche d'interaction entre les données affichées par la navigateur et l'utilisateur du site
 - il est possible de faire du Javascript sur un serveur en installant NodeJS
 - dans les bases de données NoSQL : ex mongodb

Fonctionnement serveur

- Les services Web sont principalement
 - Apache (serveur Web -- php,python ...)
 - IIS (serveurs d'applications Microsoft -- Assemblies C#, VB)
 - NodeJs (dédié Javascript)
- D'autres services existent : serveurs d'applications tomcat (Java)

Localisation des fichiers



Framework de développement Javascript

- AngularJS
- React
- Vue.js
- Ember.js
- Meteor
- Mithril
- NodeJS
- Polymer
- Aurelia
- Backbonejs

Environnements spécifiques pour les jeux

PhaserJS

Godot

Horde3D

Fonctionnement de Javascript

- On va pouvoir utiliser directement Javascript à partir d'un navigateur sans avoir nécessairement besoin de se connecter à une quelconque page Web.
 - Console du navigateur
- Intégrer du JS dans une page Web
 - `<script> </script>`
 - `<script src="nom_rep/nom_fichier.js"></script>`
 - `<script src="http://url/fichier.js"></script>`
- Organisation d'un projet
 - .html
 - /scripts
 - generalUtil.js
 - dateUtils.js
 - /styles

Les outils pour créer un script

- sublimeText
- Notepad++
- vcs
- Eclipse ...

Fonctionnement de Javascript

- Les notions élémentaires :
 - la casse
 - les commentaires
 - les expressions littérales
 - "chaîne"
 - 'autre type de cote "chaîne" suite '
 - [5,12,13,12]
 - {prix:12,3,monnaie='EUR'}
 - /[0-9]./
- Le caractère fin de ligne ;
- La syntaxe dite à points
 - accède aux attributs et méthodes à l'aide du .
 - ex : monObjet.attribut ou monObjet.methode()
- les valeurs spéciales
 - null, undefined, NaN, +Infinity, -Infinity
- Mode strict :
 - Pour indiquer au moteur Javascript de passer en mode Strict (contrôle plus important sur la syntaxe)

les bases de Javascript

- Pas de type de variable, tout est considéré comme objet sauf :
 - null, undefined, NaN,+Infinity,-Infinity
- De nombreux objets sont fournis
 - String, Number,Date,RegExp,Math,Boolean ...
- Il existe 3 instructions de déclaration de variable en JS
 - var,let,const
 - var a=32 ;

Les mots réservés

- break
- case
- catch
- class
- continue
- debugger
- in
- instance of
- near
- return
- super
- default
- delete
- do
- else
- enum
- export
- switch
- this
- throw
- try
- typeof
- extends
- finally
- for
- function
- if
- import
- var
- void
- while
- with
- Mode non strict
 - implements
 - interface
 - let
 - package
 - protected
 - public
 - static
 - private
 - yield

Expérimentations

- Sous la console de votre navigateur
 - 32+6
 - test
 - test=32
 - test/2
- Conversions implicites
 - test="valeur"
 - test+23
 - a='23'
 - b=16
 - a-b
 - a+b
- Variables numériques :
 - valeurs commençant par 0 est en base 8
 - 0123
 - 95+25
 - 95+025
- Utilisation des fonctions de conversions :
 - parseInt() : parseInt('12.54') parseInt('12',8)
'conversion d'un nombre octal en décimal
 - parseFloat() : parseFloat('2165,21')

Expérimentations

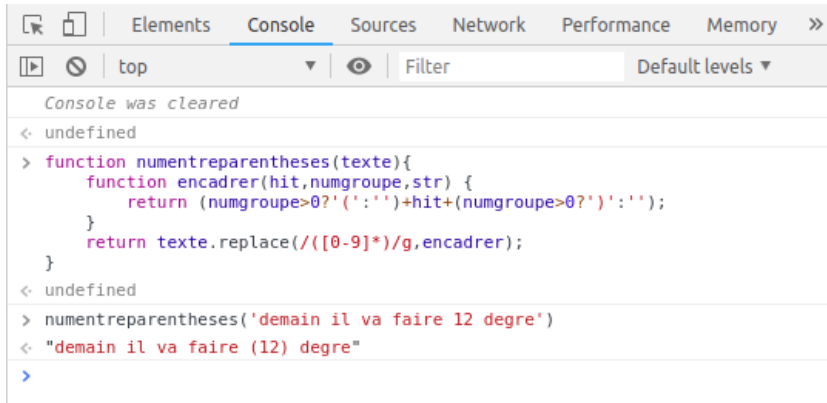
- EPSILON
 - Number.EPSILON
- toString()
 - 125.21,toString()
- isNaN()
 - pour vérifier si la valeur est NaN
- isInteger()
 - Number.isInteger(321)
- toFixed()
 - Nombre de chiffres après la virgule
 - 125.32.toFixed(3)
- toPrecision()
 - Nombre de chiffre en tout
 - 132.45,toPrecision(4)
 - 123.4
- Chaines de caract.
 - String(12345)
 - .length()
 - .repeat(nb de fois)
 - .charAt()
 - .endsWith()
 - .includes()
 - .indexOf()
 - .lastIndexOf()
- .replace()
- .slice()
- .split()
- .startsWith()
- .substr()
- .substring()
- .toLowerCase()
- .toUpperCase()
- .trim()

Les fonctions

- est une entité informatique dont le code ne figure qu'une seule fois dans le programme mais peut-être exécutée un grand nombre de fois. Un développeur peut créer ses propres fonctions.
 - `function maFonction() {console.log('premiere fonction') ;}`
- En Javascript tout est objet même les fonctions. Ce qui se traduit par la possibilité d'affecter une fonction à une variable.
 - `var mafonction=function(){console.log('premiere fonction') ;}`
 - fonction avec ou sans paramètre
- Les paramètres à l'intérieur d'une fonction sont passés en fonction de leur nature :
 - élément de base (String,Integer,Float) par valeur
 - Les autres : par référence

Exemple de fonction

```
function numentreparentheses(texte){  
  function encadrer(hit,numgroupe,str) {  
    return (numgroupe>0?(':'+hit+  
(numgroupe>0?')':''));  
  }  
  return texte.replace(/([0-9]*)/g,encadrer);  
}
```

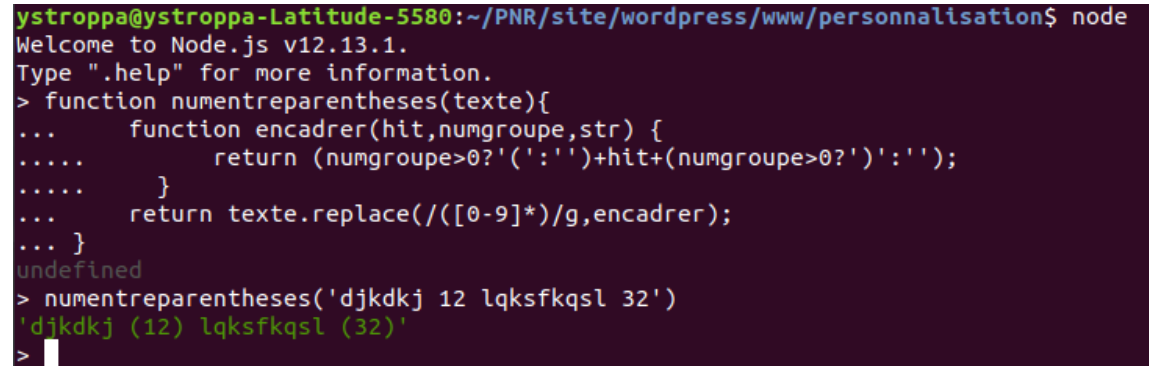


The screenshot shows a web browser's developer console with the 'Console' tab selected. The console output shows the function being defined and then called. The first call to `numentreparentheses('demain il va faire 12 degre')` returns `undefined`. The second call to `numentreparentheses('demain il va faire (12) degre')` returns the string `"demain il va faire (12) degre"`.

```
< undefined  
> function numentreparentheses(texte){  
  function encadrer(hit,numgroupe,str) {  
    return (numgroupe>0?(':'+hit+(numgroupe>0?')':''));  
  }  
  return texte.replace(/([0-9]*)/g,encadrer);  
}  
< undefined  
> numentreparentheses('demain il va faire 12 degre')  
< "demain il va faire (12) degre"  
>
```

sous node JS

```
> function numentreparentheses(texte){  
...   function encadrer(hit,numgroupe,str) {  
.....     return (numgroupe>0?(':'+hit+(numgroupe>0?')':''));  
.....   }  
...   return texte.replace(/([0-9]*)/g,encadrer);  
... }  
undefined  
> numentreparentheses('djkdj 12 lqsfkqsl 32')  
'djkdj (12) lqsfkqsl (32)'
```



The screenshot shows a terminal window with the command `node` executed. The output shows the function being defined and then called. The first call to `numentreparentheses('djkdj 12 lqsfkqsl 32')` returns `undefined`. The second call to `numentreparentheses('djkdj 12 lqsfkqsl 32')` returns the string `'djkdj (12) lqsfkqsl (32)'`.

```
ystroppa@ystroppa-Latitude-5580:~/PNR/site/wordpress/www/personnalisation$ node  
Welcome to Node.js v12.13.1.  
Type ".help" for more information.  
> function numentreparentheses(texte){  
...   function encadrer(hit,numgroupe,str) {  
.....     return (numgroupe>0?(':'+hit+(numgroupe>0?')':''));  
.....   }  
...   return texte.replace(/([0-9]*)/g,encadrer);  
... }  
undefined  
> numentreparentheses('djkdj 12 lqsfkqsl 32')  
'djkdj (12) lqsfkqsl (32)'  
>
```

Les variables tableaux

- `var monTableau=[1,2,3]`
- `var monTableauS=['chaine1','chaine2']`
- Extraire un élément du tableau
 - `monTableau[0]` `monTableauS[0]`
 - 1 chaine1
- On peut rajouter des valeurs directement dans un tableau
 - `monTableau['test']='Voiture' ;` valeurs associations
 - `monTableau[3]=21;`

Les variables tableaux

- `.length()`
 - indique le nombre d'éléments du tableau sans compter les valeurs/associations
- `.fill()`
 - remplit le tableau d'éléments à partir d'une position et un nombre de fois
 - `var monTableau=[12,51,10,9,8,7,6]`
 - `monTableau.fill(55,3,5)`
 - `[12,51,10,55,55,7,6]`
 - `monTableau.fill(99)`
 - `[99,99,99,99,99,99,99]`
- `.concat()`
 - concatène des éléments avec le tableau
 - `monTableau.concat(21)`
 - `[99,99,99,99,99,99,99,21]`

Les variables tableaux

- `.toString()`
 - forme une chaîne de caractères avec l'ensemble des éléments
- `.indexOf()`
 - retourne la position du caractère
- `.join()`
 - renvoie les éléments concaténés avec un séparateur
- `.includes()`
 - renvoie true si paramètre présent dans le tableau
- `.pop()`
 - supprime le dernier élément
- `.push()`
 - ajoute un ou plusieurs éléments dans le tableau
- `.reverse()`
 - inverse l'ordre des éléments dans le tableau
- `.shift()`
 - retire le premier élément
- `.slice()`
 - retourne une copie des éléments du tableau à partir de l'indice indiqué
- `.splice()`
 - permet de retirer et d'ajouter des éléments dans un tableau
 - Premier arg. : indice à partir duquel il faut supprimer
 - deuxième arg. : le nombre d'éléments à supprimer
 - Les autres arg. sont ajoutés à partir de l'indice de suppression
 - `monTableau.splice(2,2,'dhdh','djdj')`
 - `monTableau.splice(1)`
- `.unshift()`
 - insère au début des éléments
- `.from()`
 - `var tab=Array.from('dlmdml')`

Les variables Objets

- Différentes syntaxes pour créer des objets en Javascript

- var obj1={'nom' :'article','prix':12.3}
- Pour récupérer des infos de cet objet
 - obj1['nom'] ou obj1.nom ou obj1[expression]
 - person['age']; person.age; var x='age';person[x];

- Pour ajouter les éléments

- obj1.couleur='rouge' ; ou Object.defineProperty(person, "couleur","rouge") ;

- Pour supprimer

- delete obj1.nom ;

- pour créer un objet :

- var obj2=new Object() ;

- ensuite on lui rajoute les attributs supplémentaires
 - obj2.nom='article2' ;
 - obj2.prix=23.5 ;

// pour lister les attributs de person

```
for(variable in person)
{console.log(variable);}
```

ou

```
Object.getOwnPropertyNames(person) ;
```

Pour lister les keys etr les values

```
Object.Keys(person) ;
```

```
Object.Values(person) ;
```

Les variables Objets

```
var monObjet={  
    maprop:21,  
    mafonction : function() {  
        return 'la propriete maprop est de '+ this.maprop ;  
    }  
}
```

⇒

```
monObjet.mafonction()  
    'la propriete maprop est de 21
```


Introspection

- Plusieurs fonctions possibles
 - `typeof 'hello'`
 - String
 - indique la nature de l'élément
 - Vérifiez si une variable est instance de
 - `var maintenance=new Date() ;`
 - `maintenance instanceof Date ;`
 - true

```
var o=[1,2,3] ;  
  
var path='this' ;  
  
while(o) {  
  
    Object.getOwnPropertyNames(o).forEach(function(key) {  
  
        console.log(path+'.'+key) ;  
  
    }) ;  
  
    path+=".__proto__" ;  
  
    o=o.__proto__ ;  
  
}
```

liste l'ensemble des attributs et méthodes d'un objet Array

Les dates

- `new date()` ;
- Variable représentant des dates. Point de repère est le même que pour le système Linux 01 Janvier 1970. Ecart en millisecondes écoulé depuis cette date.
- L'objet Date est le seul qui n'offre pas de littéral pour représenter sa valeur, il faut passer par le constructeur :
 - `new Date()` ;
 - `new Date(998877654323)` ;
- La méthode accepte 7 paramètres optionnels :
 - année, mois, jour, heure, minute, secondes, les millièmes
 - `new Date(2020,0,1)`
 - Janvier =0
 - autre format : `new Date('2020-10-1')`

Les fonctions associées aux dates

- getDate()
- getDay()
- getYear()
- getFullYear()
- getHours()
- getMonth()
- getMinutes()
- getSeconds()
- getMilliseconds()
- getTime()
- getUTCDate()
- getUTCDay()
- getUTCYear()
- getUTCFullYear()
- getUTCHours()
- getUTCMonth()
- getUTCMinutes()
- getUTCSeconds()
- getUTCMilliseconds()
- getUTCTime()
- setDate()
- setDay()
- setYear()
- setFullYear()
- setHours()
- setMonth()
- setMinutes()
- setSeconds()
- setMilliseconds()
- setTime()

Fonctions de
conversion :
.toString()
.toLocaleString()

0 : Dimanche
1 : Lundi

Expressions régulières

- Une expression régulière est une formule utilisée pour reconnaître un modèle (Pattern , Motif) donné à l'intérieur d'un texte. S'appuie sur des mécanismes d'automate à états finis. S'utilise directement avec les méthodes `.exec()` et `.test()` de l'objet `RegExp` ou bien avec certaines méthodes des objets de type `String` :
 - `.replace()`
 - `.match()`
 - `.split()`
 - `.search()`

```
var maChaine='un exemple est ceci' ;  
var cExp=/(\w+)\s(\w+)\s(\w+)\s(\w+)/;  
maChaine.replace(cExp,'$4 $3 $1 $2') ;
```

L'objet Math

- Ne s'instancie pas, propose des méthodes dites statiques qui permettent d'effectuer les opérations mathématiques classiques et d'offrir les constantes de base.
 - `Math.tan(42)` ;
 - `Math.PI`

Les opérateurs

- Les opérateurs
 - ++ incrément
 - -- décrément
 - ** exponentiation
 - % module
 - +
 - -
- Opérateurs de comparaison
 - == ou ===
 - != ou !==
 - Attention aux comparaisons strictes
 - 3!= '3' false
 - 3!== '3' true
- Les opérateurs logiques
 - || ou
 - && et
 - ! not
- Les opérateurs binaires
 - & et
 - | ou
 - ^ ou exclusif
 - ~ Non binaire
 - << décalage à gauche
 - >> décalage à droite
- Opérateur binaire
 - ?:
 - maxvaleur<mimum ? 'dans la limite' : 'hors limite' ;

Les structures de contrôle

- La portée des variables et des fonctions
- Contexte global
 - à partir de la console d'un navigateur (this ou window et tous ses attributs et méthodes)
- Contexte privatif hérité
 - Lorsqu'on utilise l'instruction var pour déclarer une variable, celle-ci se retrouve créée dans le contexte où elle est jouée

Les structures de contrôle - exemple

```
function mafonctionGlobale() {  
  var maVariableLocale='je suis une var locale';  
  function maFonctionLocale(){  
    var maVariableLocale2='je suis une var locale 2';  
    console.log(1,maVariableGlobale) ;  
    console.log(2,maVariableLocale) ;  
    console.log(3,maVariableLocale2) ;  
  }  
  maFonctionLocale() ;  
  console.log(4,maVariableGlobale) ;  
  console.log(5,maVariableLocale) ;  
  console.log(6,maVariableLocale2) ;  
}
```

```
> var maVariableGlobale='je suis une variable globale';  
undefined  
> mafonctionGlobale()  
1 je suis une variable globale  
2 je suis une var locale  
3 je suis une var locale 2  
4 je suis une variable globale  
5 je suis une var locale  
Thrown:  
ReferenceError: maVariableLocale2 is not defined  
    at mafonctionGlobale (repl:12:16)  
>
```


Les structures de contrôle

- Les conditions :
 - if else if else
 - switch

```
switch() {  
    case 0:  
        instruction...  
        break ;  
    case 1 :  
        instruction...  
        break ;  
    default :  
        break ;  
}
```

Les structures de contrôle

- Les boucles :
 - `for(var i=0 ; i<21 ; i++){ }`
 - `do... while() ;`
 - `while() { }`

Parcours des éléments qui
possèdent l'attribut enumerable

- `for ... in ...`
- `for ... of ...`

```
var montableau=[1,2,3,41] ;  
for (var t in montableau) {  
    console.log(t) ;  
}  
for (var t of montableau) {  
    console.log(t) ;  
}
```

Gestion des erreurs : Error

- try
- catch ()
- finally
-
- L'instruction throw
 - Permet d'envoyer des erreurs personnalisées
 - throw new Error('problème de connexion') ;

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.every`
 - permet de savoir si tous les éléments d'un tableau remplissent une condition donnée
 - Par exemple si on veut vérifier que les éléments d'un tableau sont pairs

```
var monTableau=[2,4,6,8,10,120] ;  
var monTableau2=[2,4,6,8,10,120,13] ;  
var checkTabPaire=function(p) {return p%2==0;}  
monTableau.every(checkTabPaire) ;  
monTableau2.every(checkTabPaire) ;
```

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.filter`
 - ressemble à `every` mais renvoie un tableau de tous les éléments qui remplissent la condition
 - Par exemple si on veut récupérer un tableau avec tous les éléments pairs

```
var monTableau=[2,4,6,8,10,120,13] ;  
var checkTabPaire=function(p) {return p%2==0;}  
var tab=monTableau.filter(checkTabPaire) ;
```

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.find()`
 - Comme la précédente mais ne renvoie que le premier élément

```
var monTableau=[2,4,6,8,10,120,13] ;  
var checkTabPaire=function(p) {return p%2==0;}  
var tab=monTableau.find(checkTabPaire) ;
```

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.forEach()`
 - permet de parcourir tous les éléments du tableau et d'appliquer sur chacun d'eux un jeu d'instructions

```
var monTableau=[2,4,6,8,10,120,13] ;  
monTableau.forEach((e,i,a) => {a[i]=e*3;}) ;
```

e : élément
i : indice
a : array

monTableau
(7) [6, 12, 18, 24, 30, 360, 39]

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.map()`
 - identique à la précédente mais renvoie un tableau éventuellement transformé par la fonction callback

```
var monTableau=[2,4,6,8,10,120,13] ;  
monTableau.map(e => {return e*3;}) ;
```

(7) [6, 12, 18, 24, 30, 360, 39]

e : élément
i : indice
a : array

```
var monTableau=[2,4,6,8,10,120,13] ;  
monTableau  
  .filter(e => {return e%2==0;})  
  .map(e => {return e*3;}) ;
```

[6, 12, 18, 24, 30, 360]

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.reduce()`
 - permet de réduire à une seule valeur un tableau de données. la fonction callback reçoit pas 3 arguments mais 4. `p` valeur précédente. valeur retournée par la précédente itération et est égale à 0 par défaut.

```
var monTableau=[2,4,6,8,10,120,13] ;  
monTableau.reduce((p,e) => {return p+e;}) ;
```

163

- `.reduceRight()`
 - Le parcours du tableau est inversé. On commence par le dernier élément.

```
var monTableau=["1","2","3","4"] ;  
monTableau.reduce((p,e) => {return p+e;}) ;  
monTableau.reduceRight((p,e) => {return p+e;}) ;
```

"1234"
"4321"

Javascript en profondeur

Fonctions avancées pour les tableaux

- `.some()`
 - revoie true si au moins un des éléments dans le tableau vérifie la condition

```
var monTableau=[2,4,6,8,10,120,13] ;  
monTableau.some(e => {return e%2!=1;}) ;
```

true

- `.sort()`
 - permet de trier le tableau

["1", "14", "2", "4", "4"]

["1", "2", "4", "4", "14"]

```
var monTableau=["1","4","2","4","14"] ;  
monTableau.sort() ;  
monTableau.sort((a,b)=>{return Number(a)>Number(b)?1:Number(a)<Number(b) ?-1:0 }) ;
```

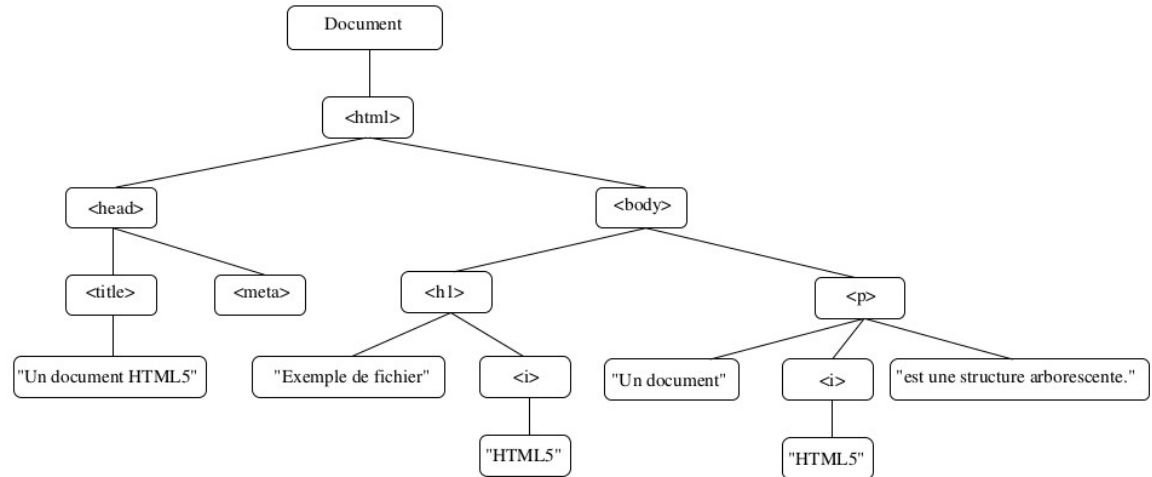
Explorer une page Web

- On travaille sur la DOM(Document Object Model) Voir https://www.w3schools.com/js/js_htmlDOM.asp
- Exemple parcourir la dom d'une page Web
 - La chargée dans votre navigateur, et à l'aide de l'instruction `window['document']` ou `window.document`
 - `window.innerHeight`
 - `window.innerWidth`
 - On a également la variable `navigator` qui représente la navigateur utilisé
 - `navigator.language`
 - `navigator.appName`
 - `navigator.userAgent`
 - Toutes les fonctions et variables déclarées directement dans la console auront une portée globale.

Exploration d'une DOM

Correspond à l'arborescence des imbrications des balises HTML d'un document

```
<!doctype HTML>
<html lang="fr">
<head>
  <meta charset="UTF-8" />
  <title>Un document HTML5</title>
</head>
<body>
  <h1>Exemple de fichier <i>HTML5</i></h1>
  <p>
    Un document <i>HTML5</i> est une structure
    arborescente .
  <p>
</body>
</html>
```



Explorer une page Web

les fonctions

- `getElementByName()`
 - retourne l'élément grâce à sa propriété HTML name
- `getElementById()`
 - retourne l'élément grâce à sa propriété HTML ID
- `getElementsByName()`
 - recherche et renvoie tous les éléments d'un type nommé. renvoi une `HTMLCollection` → qu'on peut convertir en tableau avec la fonction `array.from`
 - `getElementsByName('article')`
 - `getElementsByName('label')`
- `getElementByClassName`
 - renvoie les classes → `HTMLCollection`

Explorer une page Web

les fonctions

- `querySelector()`
 - reçoit une chaîne de caractères qui doit correspondre syntaxiquement à un sélecteur CSS → elle retourne le premier élément de la DOM qui correspond.
- `querySelectorAll()`
 - retourne tous les éléments de la DOM qui correspondent à la recherche

* : récupéré tous les éléments

: recherche par id

.x : recherche par classe CSS

x : recherche par balise HTML

[x] : recherche par attribut

[x : "y"] : éléments dont l'attribut x est égal à la valeur y

[x* = "y"] : éléments dont l'attribut x contient la valeur y

[x^ = "y"] : éléments dont l'attribut x commence par la valeur y

[x\$ = "y"] : éléments dont l'attribut x termine par la valeur y

x y : le premier élément y dans x

x > y : le premier élément y directement enfant de x

x + y : Tous les éléments x et tous les éléments y

x : not(y) : tous les éléments qui correspondent au sélecteur de x mais pas à y

Modifier les éléments de la page

- Une fois obtenu la référence vers l'élément ou les éléments, il est possible de modifier ses ou leurs caractéristiques

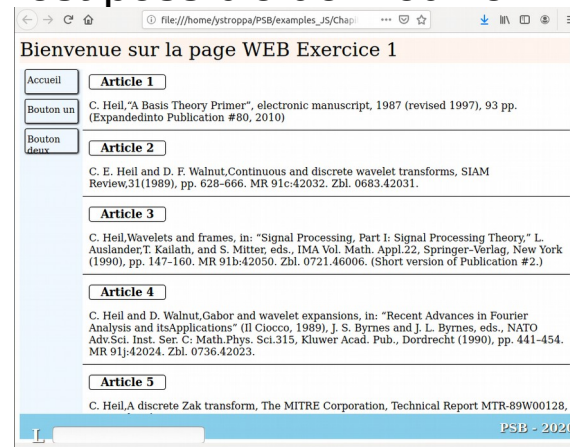
- contenu
- apparence
- position..

#Première modification :

```
var mesvals=["un","deux","trois","quatre","cinq"] ;  
var mesEtiqu=document.querySelectorAll("article > label") ;  
Array.from(mesEtiqu).forEach((e,i)=>{e.innerText='Article' + mesvals[i];});
```

#Deuxième modification : on veut également modifier le fond : couleur background

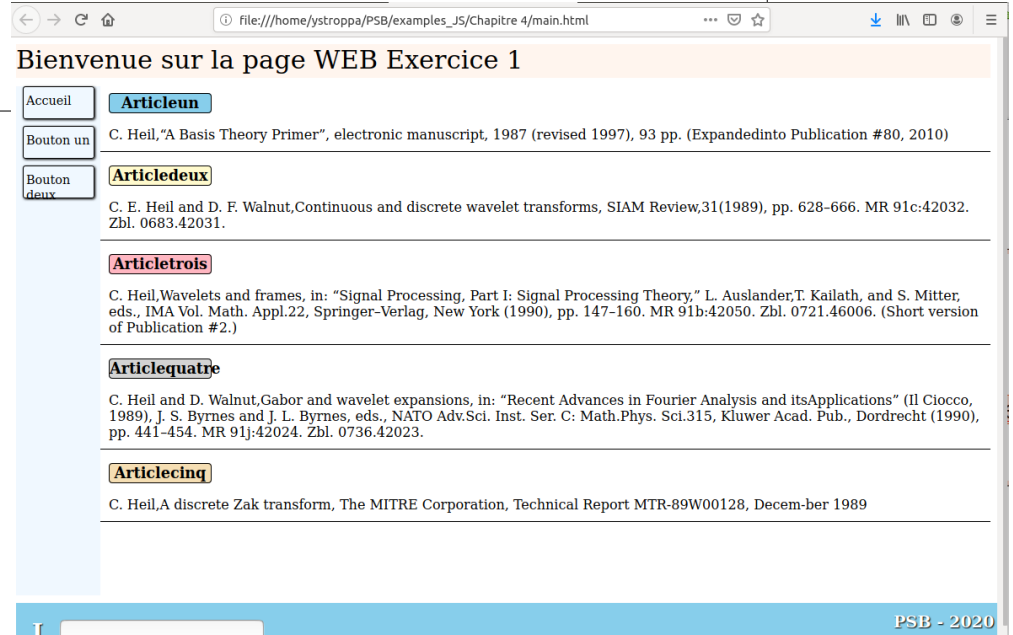
```
Array.from(mesEtiqu).forEach((e,i)=>  
    {e.innerText='Article' + mesvals[i];  
    e.style.background='skyblue' ;});
```



Modifier les éléments de la page

#troisième modification :

```
var mesEtiqu=document.querySelectorAll("article > label") ;  
Array.from(mesEtiqu).forEach((e,i)=>  
    {  
        e.innerText='Article' +["un","deux","trois","quatre","cinq"][i];  
        e.style.background=["skyblue",'lemonchiffon','lightpink','lightgrey','wheat'][i] ;  
    }) ;
```

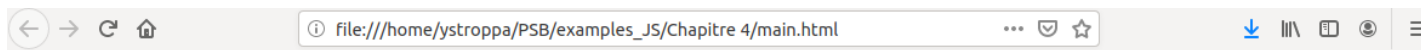


Injecter des éléments dans la DOM

- Pour injecter il faut sélectionner et le stocker en mémoire l'élément de la page dans lequel nous allons insérer le nouvel élément.
- Exemple pour rajouter un nouvel article dans notre page.

```
var monJournal=document.querySelector('.articleContainer') ;  
var monArticle=document.createElement('article') ;  
var monEtiquette=document.createElement('label') ;  
var monTexte = document.createElement('p') ;  
monEtiquette.innerText='Article 6' ;  
monTexte.innerText="J. J. Benedetto, C. Heil, and D. F. Walnut, Differentiation and the  
Balian–Low theorem, J. Fourier Anal. Appl., 1(1995), pp. 355–402. MR 96f:42002. Zbl.  
0887.42026." ;  
monArticle.setAttribute('data-order','6') ;  
monArticle.appendChild(monEtiquette) ;  
monArticle.appendChild(monTexte) ;  
monJournal.appendChild(monArticle);
```

Résultat



Bienvenue sur la page WEB Exercice 1

Accueil

Articleun

Bouton un

C. Heil, "A Basis Theory Primer", electronic manuscript, 1987 (revised 1997), 93 pp. (Expanded into Publication #80, 2010)

Bouton deux

Articledeux

C. E. Heil and D. F. Walnut, Continuous and discrete wavelet transforms, SIAM Review, 31(1989), pp. 628-666. MR 91c:42032. Zbl. 0683.42031.

Articletrois

C. Heil, Wavelets and frames, in: "Signal Processing, Part I: Signal Processing Theory," L. Auslander, T. Kailath, and S. Mitter, eds., IMA Vol. Math. Appl. 22, Springer-Verlag, New York (1990), pp. 147-160. MR 91b:42050. Zbl. 0721.46006. (Short version of Publication #2.)

Articlequatre

C. Heil and D. Walnut, Gabor and wavelet expansions, in: "Recent Advances in Fourier Analysis and its Applications" (Il Ciocco, 1989), J. S. Byrnes and J. L. Byrnes, eds., NATO Adv. Sci. Inst. Ser. C: Math. Phys. Sci. 315, Kluwer Acad. Pub., Dordrecht (1990), pp. 441-454. MR 91j:42024. Zbl. 0736.42023.

Articlecinq

C. Heil, A discrete Zak transform, The MITRE Corporation, Technical Report MTR-89W00128, December 1989

Article 6

J. J. Benedetto, C. Heil, and D. F. Walnut, Differentiation and the Balian-Low theorem, J. Fourier Anal. Appl., 1(1995), pp. 355-402. MR 96f:42002. Zbl. 0887.42026.

Événements en JavaScript

- Essayer de répondre de façon dynamique en fonction des événements.
- Qu'est ce qu'un événement
 - lorsque les caractéristiques de la page change un événement est déclenché. Ce changement de valeur de caractéristique peut-être par exemple :
 - une couleur
 - une forme
 - une position
 - le fait qu'un élément soit survolé par la souris ou cliqué
 - le fait qu'un élément obtienne le focus ou le perde
 - caractère entré par le clavier
- Lorsqu'un tel événement survient, un objet event est créé par JS. Cet objet contient toutes les propriétés permettant de décrire l'événement. Il fournit également des méthodes permettant d'effectuer certaines actions particulières sur les événements déclenchés.

Événements en JavaScript

- Propriétés et méthodes communes aux différents events
 - stopPropagation()
 - arrête le processus de traitement de l'événement, dire au gestionnaire d'événements d'arrêter de transmettre l'événement.
 - target et currentTarget
 - l'élément sur lequel est survenu l'événement.
 - preventDefault()
 - bloque l'exécution du comportement prévu par défaut.

blur	click	complete	copy	dblclick	dragend
change	close	contextmenu	cut	drag	dragstart
drop	keyup	mouseout	scroll	touchcancel	unload
error	load	mouseover	select	touchend	wheel
focus	mousedown	mouseup	show	touchenter	
keydown	mouseenter	reset	submit	touchmove	
keypress	mouseleave	resize	success	touchstart	

<https://developer.mozilla.org/fr/docs/Web/Events>

Evénements

- L'objet event ainsi créé est ensuite transmis aux fonctions callbacks déclarées comme devant être exécutées lorsque survient l'événement. Ces fonctions callbacks sont déclarées par des instructions spéciales que l'on nomme des écouteurs.

Evénements

- Poser des écouteurs sur des événements
 - 1^{ere} méthode est d'utiliser le mot on suivi du nom de l'événement à écouter

```
function maFonction(e) { console.log('clic');} ;  
var monElement=document.querySelector('Article[data-order="3"] > label') ;  
monElement.onclick=maFonction;
```

- Pour supprimer un écouter, il suffit de l'affecter à la valeur undefined :

```
monElement.onclick=undefined ;
```

Événements

- Poser des écouteurs sur des événements
 - 2ème méthode est d'utiliser addEventListener

```
function maFonction(e) { console.log('clic');} ;  
document.querySelector('Article[data-order="3"]>label').addEventListener('click',maFonction);
```

- Pour supprimer un écouter :

```
document.querySelector('Article[data-order="3"]>label').removeEventListener('click',maFonction);
```

Événements

Passage de la souris sur les articles

```
var lesarticles=document.querySelectorAll('article') ;
for(var articleEC of lesarticles) {
    articleEC.addEventListener('mouseenter',e=> {
        e.currentTarget.style.fontWeight='bold' ;
        e.currentTarget.style.fontStyle='italic' ;
        e.currentTarget.style.color='dodgerblue' ;
    }) ;
    articleEC.addEventListener('mouseleave',e=> {
        e.currentTarget.style.fontWeight='normal' ;
        e.currentTarget.style.fontStyle='normal' ;
        e.currentTarget.style.color='black' ;
    }) ;
}
```


Evénements

Passage de la souris sur les boutons

```
function changetaille(mode) {  
    var lesarticles=document.querySelectorAll('article > p ');  
    for(var articleEC of lesarticles) {  
        articleEC.style.fontSize=mode =='gros' ?'130%' :'100%';  
    }  
}  
document.querySelectorAll('nav > label')[1].addEventListener('click',e =>{changetaille('gros');});  
document.querySelectorAll('nav > label')[2].addEventListener('click',e =>{changetaille('normal');});
```

Le prototypage

- Comment fonctionne le modèle objet de Javascript.
- Les autres langages orientés objet c++, Java, sont fondés sur une distinction entre la définition d'un objet (classe) et sa réalité à l'exécution (instanciation)
- Pour créer un objet il est nécessaire d'écrire un fichier qui définit la classe puis d'utiliser cette définition pour créer véritablement l'objet.
- Les langages basés sur des prototypes tel Javascript ne font pas cette distinction. Ils ne possèdent que des objets.
- Les objets peuvent servir de modèles sur lesquels s'appuyer pour en créer d'autres. On peut aussi ajouter ou supprimer des propriétés ou des méthodes aux objets pendant l'exécution.

Le prototypage

- L'héritage est aussi très différent en javascript ;
- N'importe quel prototype peut être associé à un constructeur quel qu'il soit :

```
function Vehicule() {  
    this.type='non renseigné' ;  
    this.marque='non renseigné' ;  
}
```

```
function Voiture() {  
    this.type='voiture' ;  
    this.modele='non renseigné' ;  
    this.moteur='non renseigné' ;  
    this.puissance='non renseigné' ;  
}
```

```
function Renault() {  
    this.marque='Renault' ;  
}
```

```
Voiture.prototype=new Vehicule ;  
Renault.prototype=new Voiture ;  
var maVoiture=new Renault();
```

maVoiture
Object { marque: "Renault" }

```
var maVoiture2=new Renault();  
maVoiture2.kilometrage='124521' ;
```

maVoiture2
Object { marque: "Renault", kilometrage: "124521" }

Ajouter des nouvelles méthodes à un objet

- On souhaite modifier le prototype String de manière à lui ajouter une méthode nommée `initiale` qui restitue une chaîne de caractères en minuscules sauf le premier élément.

```
function initiale(txt) {  
    if (txt == undefined || txt=="") {return " " ;}  
    return txt.substr(0,1).toUpperCase()+txt.substr(1).toLowerCase() ;  
}  
pour l'adapter en tant que méthode prototype String  
String.prototype.initiale=function() {  
    if (this == undefined || this=="") {return " " ;}  
    return this.substr(0,1).toUpperCase()+this.substr(1).toLowerCase() ;  
}
```

Autre exemple pour de la conversion UTF8

```
String.prototype.encode = function() {  
    var result = "";  
    var s = this.replace(/\r\n/g, "\n");  
    for(var index = 0; index < s.length; index++) {  
        var c = s.charCodeAt(index);  
        if(c < 128) {  
            result += String.fromCharCode(c);  
        }  
        else if((c > 127) && (c < 2048)) {  
            result += String.fromCharCode((c >> 6) | 192);  
            result += String.fromCharCode((c & 63) | 128);  
        }  
        else {  
            result += String.fromCharCode((c >> 12) | 224);  
            result += String.fromCharCode(((c >> 6) & 63) | 128);  
            result += String.fromCharCode((c & 63) | 128);  
        }  
    }  
    return result;  
};
```

'première fois'.encode()
"premiÃre fois"

Autre exemple pour de la conversion UTF8

```
String.prototype.decode = function() {  
    var result = "";  
    var index = 0;  
    var c = c1 = c2 = 0;  
    while(index < this.length) {  
        c = this.charCodeAt(index);  
        if(c < 128) {  
            result += String.fromCharCode(c);  
            index++;  
        }  
        else if((c > 191) && (c < 224)) {  
            c2 = this.charCodeAt(index + 1);  
            result += String.fromCharCode(((c & 31) << 6) | (c2 & 63));  
            index += 2;  
        }  
        else {  
            c2 = this.charCodeAt(index + 1);  
            c3 = this.charCodeAt(index + 2);  
            result += String.fromCharCode(((c & 15) << 12) | ((c2 & 63) << 6) | (c3 & 63));  
            index += 3;  
        }  
    }  
    return result;  
};
```

```
var txt="premiÃ`re fois" ;  
txt.decode()
```

Format JSON

- Javascript Object Notation
 - consiste à stocker sous forme de chaînes de caractères ce que contient un objet dans la perspective de pouvoir le récréer à partir de cette chaîne. Permet également d'échanger entre plusieurs parties (serveurs) le contenu de variable Javascript.
 - Le principe est de créer des blocs d'accolades regroupant les objets en cascade.
 - Existe des sites de validations de format JSON
 -

Le prototype JSON

- Objectif est de sérialiser les objets.
- Pour des objets simples sans dépendance de type var
monobjet={"nom" : "stroppa" , "prenom" : "yvan"} assez facile à construire par contre lorsqu'on a des relations, héritages c'est un peu plus compliqué. D'où le prototype JSON qui propose des méthodes pour sérialiser et dé sérialiser les objets. La méthode de sérialisation : stringify() et la méthode de dé sérialisation parser().
- Exemple :
 - var voiture={"marque" : "citroen", "modele" : "c3"} ;
 - var obj1={"nom" : "Stroppa" , "prenom" : "Yvan", "voiture":voiture} ;
 - var obj2={"nom" : "Ly" , "prenom" : "a-phat"} ;
 - var tab=[obj1,obj2] ;
 - var tab1=JSON.parse(JSON.stringify(tab))

JSON : sérialisation avec fonction

- Détail de la fonction stringify :
 - JSON.stringify(value[, replacer[, space]])
 - // => value(required) — the object we want to serialize
 - // => replacer(optional) — a function that will be called for each // of the object's properties, or an array of strings and numbers // that serve as a whitelist for the property selection process
 - // => space(optional) — the number of spaces each key will receive // as indent

```
'use strict';
let person = {
  name: 'Susan',
  age: 24,
  sayHi: function() {
    console.log('Susan says hi!');
  }
};
```

```
let replacer = (key, value) => {
  // if we get a function, give us the code for that function
  if (typeof value === 'function') {
    return value.toString();
  }
  return value;
}
// get a stringified version of our object
// and indent the keys at 2 spaces
const serialized = JSON.stringify(person, replacer, 2);
```

JSON : désérialisation avec fonction

- Détail de la fonction parser :
 - JSON.parse(text[, reviver])
 - // => text(required) - the string we wish to de-serialize
 - // and convert back to a standard JavaScript object(JSON)
 - // => reviver(optional) - function used to pre-process keys and
 - // values in order to render a specific object structure

```
'use strict';
let person = {
  name: 'Susan',
  age: 24,
  sayHi: function() {
    console.log('Susan says hi!');
  }
};
```

```
// de_serialize.js
let reviver = (key, value) => {
  if (typeof value === 'string'
    && value.indexOf('function ') === 0) {
    let functionTemplate = `${value}`;
    return eval(functionTemplate);
  }
  return value;
}

const parsedObject = JSON.parse(serialized, reviver);
parsedObject.sayHi(); // Susan says hi!
```

Exercice de manipulation

Changement de couleur au bout d'un laps de temps pour les labels.

```
function parseelementengris(e) {  
    e.style.color='gainsboro';  
}  
function passegris() {  
    var lesarticles=document.querySelectorAll("article > p");  
    var cpt=0;  
    for (var article of lesarticles) {  
        setTimeout(function() {parseelementengris(article);},++cpt*1000);  
    }  
}  
  
document.querySelector('#pageUne').addEventListener('click',passegris);
```

Exercice de manipulation : Résultat

- Fonctionne pas comme prévu ???? mais que se passe t-il ---mais qu'est-ce qui se passe ?
- On ajoute un console log pour afficher la longueur des éléments

```
function parseelementengris(e) {  
    console.log(e.innerHTML.length);  
    e.style.color='gainsboro';  
}  
function passegris() {  
    var lesarticles=document.querySelectorAll("article > p");  
    var cpt=0;  
    for (var article of lesarticles) {  
        console.log(article.innerHTML.length);  
        setTimeout(function() {parseelementengris(article);},++cpt*1000);  
    }  
}  
document.querySelector('#pageUne').addEventListener('click',passegris);
```

Explication

- Il faut aménager la fonction pour permettre d'être traitée comme il le faut. Car quand la fonction est exécutée, la boucle est déjà finie. Donc il ne reste que la dernière valeur, et la fonction est exécutée 5 fois avec la même valeur. Pour pallier à ce problème, il faut donc recourir à ce que l'on appelle en javascript une Closure. Il s'agit d'exécuter la fonction dans un contexte dans lequel les variables sont stockées avec leur valeur du moment.

```
function passegris() {  
    var lesarticles=document.querySelectorAll("article > p");  
    var cpt=0;  
    for (var article of lesarticles) {  
        console.log(article.innerHTML.length);  
        (function (art) {  
            setTimeout(function() {parseelementengris(art);}, ++cpt*1000);  
        })(article);  
    }  
}  
  
document.querySelector('#pageUne').addEventListener('click',passegris);
```

AJAX

- Obtenir des données distantes en Javascript.
 - La plupart du temps une page web va en réaction aux actions de l'utilisateur avoir besoin d'échanger avec le serveur des images, données, fichiers sonores...
 - Au début d'Internet en naviguer de page en page.
 - Les technos actuelles permettent de charger que certaines parties de la page.
- Les principes généraux d'AJAX (Asynchronous JavaScript XML)
 - Les techno utilisées s'appellent la techno AJAX. Le principe est simple mais la disparité des navigateurs a rendu complexe son implémentation. Avec les navigateurs récents les disparités tendent à disparaître.
 - Le principe qu'instaure cette pratique est l'adresse des données soit située sur le même domaine que celui de la page dans laquelle réside et est exécuté le script.
 - Notion de : Same Domain Policy

AJAX

- Implémenter des appels AJAX consiste à utiliser les méthodes de l'objet: XMLHttpRequest()
 - <https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest>
- Que l'on instancie avec l'instruction new
 - `var serveur = new XMLHttpRequest();`
- Une fois l'instance déclarée, on place un écouteur sur la variable afin de lui indiquer la fonction callback à exécuter lorsqu'un changement de statut est détecté.
- Ensuite on prépare l'objet en lui indiquant l'adresse où trouver les données :
 - Méthode de passage (GET, POST ...)
 - Adresse
 - Mode de fonctionnement (Synchrone , asynchrone)

Exemple AJAX

```
serveur
    .onreadystatechange=function(){
        if (serveur.readyState== XMLHttpRequest.DONE) {
            if (serveur.status==200) {
                Array.from(document.querySelectorAll('article > p'))[0].innerHTML=serveur.responseText ;
            } else {
                Alert('Erreur'+ serveur.status) ;
            }
        }
    };
};
```

```
serveur.open('GET','data/texteArticle.txt',false) ;
```

On peut bien sur passer des paramètres

<http://monSiteWeb.mesTest.php?parama=1¶m2=12>

Pour la méthode post, un objet data est associé à l'adresse et passé au serveur sous format JSON.



- JQuery est une bibliothèque de Javascript la plus répandue au monde.
 - Gratuité, légère et assez simple à appréhender
- Historique:
 - John Resig qui lance la première version en 2006. Objectif fournir une couche d'abstraction aux développeurs permettant de simplifier le code en offrant des objets abstraits là où il y avait besoin de syntaxes différentes dépendantes des navigateurs.
- Apport de jQuery à Javascript
 - Offre des fonctionnalités simplifiées
 - pour accéder au DOM,
 - effectuer des demandes de données distantes,
 - ajouter des effets visuels et des animations dans les pages web,
 - manipuler les styles CSS appliqués aux éléments de la DOM

jQuery

- Intégrer jQuery dans une page Web
 - `<script src="./scripts/jquery-3.4.1.js"></script>`
 - La bibliothèque est disponible en version normale +ou- 280ko et en version minimal +ou-88ko.
- Les bases de jQuery
 - pour invoquer des méthodes de la bibliothèque de jQuery deux syntaxes :

- jQuery.
- \$

```
jQuery
  jQuery(selector, context) ⚡
  Animation: function Animation(elem, properties, options) ⚡
  Callbacks: function Callbacks(options) ⚡
  Deferred: function Deferred(func) ⚡
  Event: function Event(src, props) ⚡
  Tween: function Tween(elem, options, prop, end, easing) ⚡
  _data: function _data(elem, name, data) ⚡
  _evalUrl: function _evalUrl(url, options) ⚡
  _queueHooks: function _queueHooks(elem, type) ⚡
  _removeData: function _removeData(elem, name) ⚡
  active: 0
  ajax: function ajax(url, options) ⚡
  ajaxPrefilter: function addToPrefiltersOrTransports(dataTypeExpression, func) ⚡
  ajaxSettings: Object { url: "file:///home/ystroppa/PSB/examples_JS/rep_exercices/main.html", type: "GET", isLocal: true, ... }
  ajaxSetup: function ajaxSetup(target, settings) ⚡
  ajaxTransport: function addToPrefiltersOrTransports(dataTypeExpression, func) ⚡
  attr: function attr(elem, name, value) ⚡
  attrHooks: Object { type: {...} }
```

jQuery

- Le \$ n'étant qu'un alias sur jQuery. Cette fonction accepte en paramètres un objet ou une chaîne de caractères.
- Lorsque c'est une chaîne, elle doit être constituée d'un sélecteur jQuery valide pour renvoyer les éventuels éléments de la page web correspondants
- le symbole \$ sert aussi d'objet qui permet l'accès aux méthodes utilitaires proposées par jQuery.
 - Exemple : \$.ajax(.....)

jQuery

- Les fonctions callback
 - La plupart du temps, une méthode jQuery accepte en paramètre une fonction callback qui s'exécute avec pour contexte les éléments sur lesquels porte la sélection de la fonction \$.
 - `$(document).ready(function(){console.log("la page est chargée");});`
 - Dans cet exemple, la fonction callback anonyme passée en paramètre de la méthode `jQuery.ready()` se déclenche une fois tous les éléments de la page ont été chargés et dessinés et permet de préparer les variables au bon fonctionnement de l'application.

jQuery

- Prévenir les conflits avec d'autres biblio.
 - certains biblio. tierces utilisent le même alias \$
 - Solutions
 - `jQuery.noConflict()`
 - //l'alias \$ est disponible et ne référence plus la fonction jQuery
 - `var jq=jQuery.noConflict() ;`
 - //alias jq pourra être utilisé pour les fonctions de jQuery
 - Evolution de la bibliothèque
 - 10 Avril 2019 version 3.4

jQuery

- Interaction avec le DOM
 - les sélecteurs et pseudo-sélecteurs
 - Ils sont basés sur les sélecteurs CSS. La bibliothèque enrichit ces derniers de sélecteurs spécifiques qui permettent une sélection plus fine
 - 27 au total

jQuery -sélecteurs

- * :tous les éléments
- x :les éléments x
- #x:l'élément ayant l'id x
- .x : l'élément ayant la classe CSS x
- x > y : les éléments y enfants de x
- x y : les éléments y descendants de x
- x + y : le premier élément y précédé d'un élément x
- x ~ y : Les éléments y précédés de x
- [x] : les éléments ayant un attribut x
- [x="y"] : les éléments ayant un attribut x de valeur y
- [x*="y"] :les éléments ayant un attribut x dont la valeur contient y
- [x^="y"] :les éléments ayant un attribut x dont la valeur débute par y
- [x\$="y"] :les éléments ayant un attribut x dont la valeur s'achève par y

jQuery -sélecteurs

- [x ~="y"] :les éléments dont l'attribut x est une liste de valeurs séparées par des espaces et incluant y
- [x !="y"] :les éléments dont l'attribut x n'est pas de valeur y
- :eq(x):l'élément d'indice x dans la sélection
- :even:les éléments d'indice pair
- :odd : les éléments d'indice impair
- :first : le premier élément de la sélection
- :gt(x) : les éléments dont l'indice dans la sélection est supérieur à x
- :lt(x) : les éléments dont l'indice dans la sélection inférieur à x
- :last : le dernier de la sélection
- :checked : les éléments cochés de la sélection
- :selected : les éléments sélectionnés de la sélection
- :visible : les éléments visibles de la sélection
- :hidden() : les éléments cachés de la sélection
- :not(x) : les éléments de la sélection qui ne concordent pas avec le sélecteur x

jQuery vs Javascript

<code>var myElement=\$("#id01");</code>	ByID	<code>document.getElementById("#id01") ;</code>
<code>var myElement=\$("p");</code>	ByTag	<code>document.getElementsByTagName("p") ;</code>
<code>var myElement=\$(".intro");</code>	ClassName	<code>document.getElementsByClassName("intro")</code>
<code>var myElement=\$("p.intro");</code>	By CSS Selectors	<code>document.querySelectorAll("p.intro") ;</code>
<code>myElement.text("Hello") ;</code>	Set Text Content	<code>myElement.textContent="Hello" ;</code>
<code>var myText=myElement.text() ;</code>	Get Text Content	<code>var myText=myElement.textContent ou</code> <code>var myText=myElement.innerText ;</code>
<code>var myText.html("<p>Hello</p>");</code>	Set HTML Content	<code>myElement.innerHTML="<p>Hello</p>" ;</code>
<code>var Content=myElement.html();</code>	Get HTML Content	<code>var Content=myElement.innerHTML ;</code>
<code>myElement.hide();</code>	Hiding HTML element	<code>myElement.style.display="non" ;</code>
<code>myElement.show();</code>	Showing HTML element	<code>myElement.style.display="" ;</code>
<code>myElement.css("fontsize","35px");</code>	Styling HTML element	<code>myElement.style.dfontSize="35px" ;</code>

jQuery

- Modifier le DOM à partir d'un sélecteur jQuery
 - \$('article:odd').remove() ;
 - Permet de supprimer tous les articles impaires
- Manipuler les attributs des éléments de DOM

```
var mesArticles=document.querySelectorAll('article >p') ;  
for (var articles of mesArticles) {  
    articles.style.color="gainsboro" ;  
}
```

Avec jQuery==>

```
$('article > p').css('color','gainsboro') ;
```

jQuery

- Autres fonctions :

- .addClass()
- .removeClass()
- .toggleClass()
- .prop()
- .attr()
- .text()

- Autres fonctions :

- .empty()
- .replaceWith()
- .replaceAll()
- .wrap()
- .unwrap()
- .wrapAll()

- Autres fonctions :

- .wrapinner()
- .clone()
- .html()
- .val()
- .css()

- Les méthodes utilitaires :

- .each() `$([1,2,3,5]).each(function(i,e){console.log(i,e,this);});`
- .noConflict()

jQuery

- Intégrer de nouveaux éléments HTML dans le DOM
 - append()
 - prepend()
 - injecte en 1ère position tant que 1^{er} enfant
 - after() ou before():
 - permet l'insertion d'un nouvel élément HTML dans une page Web en spécifiant sa position relativement à un autre
 - prependTo(),appendTo(),insertBefore(),insertAfter()

jQuery

- les interactions utilisateurs en jQuery
 - Reagit à des événements
 - jQuery fournit une méthode dédiée pour gérer la pose des écouteurs d'événements sur une sélection d'éléments du DOM.
 - La méthode `.on()`
 - Accepte plusieurs paramètres et dispose de plusieurs signatures
 - On peut mettre plusieurs fonctions callback sur un même événement

jQuery

```
function callback1() { console.log("bouton de navigation clique") } ;  
$('nav > label').on('click',callback1) ;
```

```
function callback2() { console.log("bouton de navigation clique") } ;  
$('nav > label:gt(0)').on('click',callback2);
```

Pour désactiver .off

jQuery

- Déclencher des événements personnalisés
 - Lorsqu'on veut déclencher lors d'un délai écoulé
 - ou bien lorsque les données ont été calculées
 - ou formatées
 - et que l'algorithme de calcul ou de formatage prend un certain temps
- La méthode s'appelle **.trigger**

jQuery

Exemple de méthode .trigger

```
function verifierSiArticleNonLu(){
    return $('article:not(articleLu)').length>0;}

function marquerArticle(e){
    $(e.currentTarget).addClass('articleLu') ;
    if (!verifierSiArticleNonLu()){
        $('.articleContainer').trigger('ToutEstLu') ;
    }
}

$('.articleContainer').on('click','article',marquerArticle) ;
$('.articleContainer').on('ToutEstLu', e=>{
    $('article').css('color','tomato') ;
    console.log('vous avez tout lu') ;
}) ;
```


jQuery

- Les effets :

- .hide()
- .show()
- .toggle()
- .fadeOut()
- .fadeIn()
- .fadeToggle()

- Les effets :

- .fadeTo()
- .slideDown()
- .slideUp()
- .slideToggle()
- .animate()

jQuery

- Gérer la file d'attente des événements d'animation
- jQuery propose une gestion des files d'attente où sont stockées les animations devant survenir sur les éléments de la page web/
 - .queue()
 - permet d'accéder à la file d'attente des animations en cours
 - .dequeue()
 - .clearQueue()
 - .stop()
 - .finish()
 - .delay()

jQuery

- jQuery et les données
 - la méthode .data()
 - Avec cette méthode on peut stocker un objet contenant des informations que l'on souhaite, en l'associant aux éléments retournées par le sélecteur sur lequel elle est appliquée.
 - de la forme : key:value
 - \$('article:eq(1)').data("test","chaîne de caractères test") ;

jQuery

- Ajax et la mode asynchrone

- .load()

- jQuery.ajax() ou \$.ajax()

- async

- complete : une fonction callback exécutée lorsque la méthode est complètement terminée

- crossDomain : positionné à true si la requête est une demande intersite

- data: objet Javascript comportant les données

- error : fonction callback exécutée lorsqu'une erreur survient. Reçoit 2 à 3 paramètres

- objet reçu correspond à l'objet requête à proprement parler

- chaîne de caractères:un des valeurs descriptives : null, timeout,error,abond,parseerror

- statut de la page : 404

- method : GET, PUT ou POST

- password : dans le cas ou une authentification est demandée par la page

- success : fonction callback exécuter lorsque les données sont correctement reçus. 3 paramètres

- une variable contenant les données reçues

- une chaîne de caractères indiquant le statut de la requête

- objet requête à proprement parler

- url : indique l'url de la page web qui fournit les données

- password : password dans le cas d'une authentification

-

Autres méthodes :

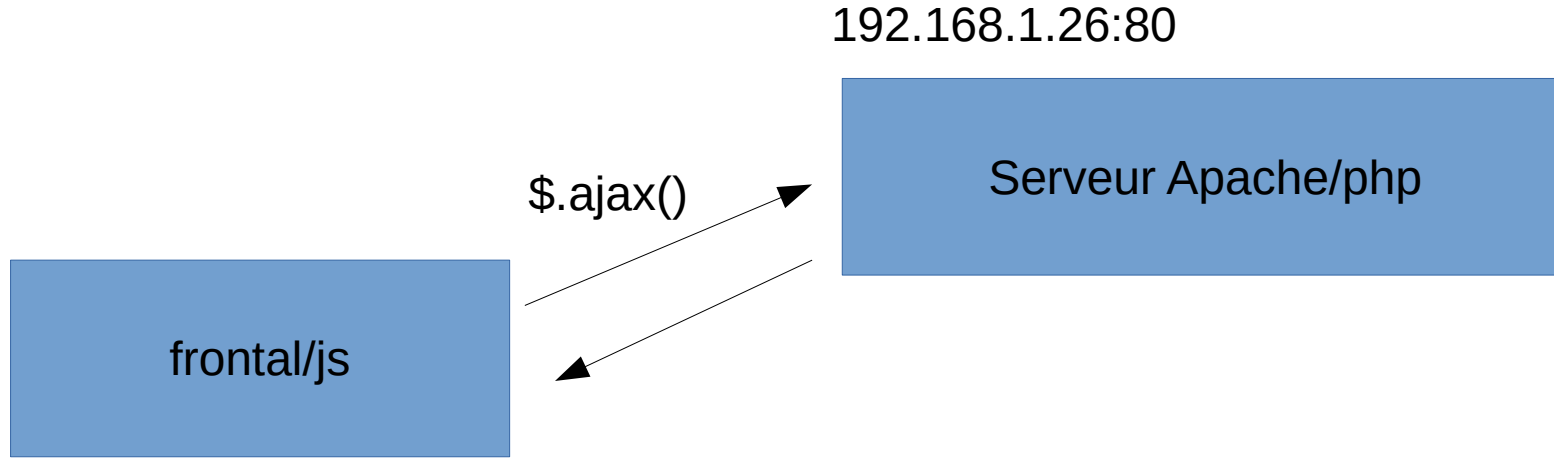
jQuery.get() ou \$.get()

jQuery.post() ou \$.post()

Exemple de requête \$.ajax

```
const Pays=pays.toLowerCase();
var values="pays="+Pays; //+token;
$.ajax({
  url: \"http://192.168.1.26:100/personnalisation/get_etab_pays.php\",
  type: "post",
  data: values,
  success: function(msg) {
    var inter = $.parseJSON(msg);
    if (inter.success === 'yes') {
      var inter_detail=$.parseJSON(inter.detail)
      rempli_institution(inter_detail.retval.etablissement);
      re_compose(inter_detail.retval.etablissement);
      return true;
    }else{
      alert(inter.detail);
      return false;
    }
  },error: function(resp) {
    alert(JSON.stringify(resp));
  }
}); .....
```

Exemple de requête \$.ajax



Plus loin avec jQuery

- jQuery est une bibliothèque ouverte, son code est accessible et donc de modifier, et qu'elle peut être étendue.
- Ecrire un plug-in basique
 - Il existe plusieurs façons les nouvelles fonctions qui enrichissent le prototype jQuery nommé \$.fn
 - \$.fn.maFonctionPSB=fonction(mesParam){moncode} ;
 - \$.extend(\$.fn,{maFonction : function(){code....};autrefunction : function(param){code autre fonction}}) ;
 - jQuery.extend(jQuery.fn,{maFonction : function(){code....};autrefunction : function(param){code autre fonction}}) ;
 - Quelque soit la solution utilisée, à l'intérieur de ces nouvelles méthodes, le contexte représenté par this correspond aux éléments retournés par le sélecteur.

Plus loin avec jQuery

exemple de définition de fonction

On souhaite faire une fonction qui change la couleur du fond de tous les éléments ramenés par le sélecteur.

```
function coloriser(couleur){this.each ((i,e)=>{$(e).css('background',couleur);});}
```

Pour étendre jQuery

```
$.fn.extend({coloriser:function(couleur){this.each((i,e)=>{$(e).css('background',couleur);}})}})
```

Pour utilise notre nouvelle fonction :

```
$('article').coloriser('yellow');  
$('article label').coloriser('red') ;
```

Si on essaye de chainer la fonction avec une autre du style

```
$('article').coloriser('red').css('color','white')
```

Une erreur survient ?????

Plus loin avec jQuery : Le chaînage

- Le chaînage fonctionne très bien quand coloriser est exécutée à la fin du chaînage Ceci est dû car lors de l'écriture de la nouvelle fonction nous n'avons pas respecté les recommandations de jQuery qui stipule d'une méthode appelée par un sélecteur doit transmettre en sortie les éléments qu'elle reçoit en entrée pour assurer le chaînage
- D'où la correction à faire :

```
$.fn.extend(  
  {  
    coloriser:function(couleur){  
      this.each((i,e)=>{$(e).css('background',couleur);});  
      return this ;  
    }  
  })  
$('article').coloriser('red').css('color','white')
```

Bienvenue sur la page WEB Exercice 1

Accueil	Article 1
Bouton un	C. Heil, "A Basis Theory Primer", electronic manuscript, 1987 (revised 1997), 93 pp. (Expanded into Publication #80, 2010)
Bouton deux	Article 2
	C. E. Heil and D. F. Walnut, Continuous and discrete wavelet transforms, SIAM Review, 31(1989), pp. 628-666. MR 91c:42032, Zbl. 0683.42031.
	Article 3
	C. Heil, Wavelets and frames, in: "Signal Processing, Part I: Signal Processing Theory," L. Auslander, T. Kailath, and S. Mitter, eds., IMA Vol. Math. Appl. 22, Springer-Verlag, New York (1990), pp. 147-160. MR 91b:42050, Zbl. 0721.46006. (Short version of Publication #2.)
	Article 4
	C. Heil and D. Walnut, Gabor and wavelet expansions, in: "Recent Advances in Fourier Analysis and its Applications" (Il Ciocco, 1989), J. S. Byrnes and J. L. Byrnes, eds., NATO Adv. Sci. Inst. Ser. C: Math. Phys. Sci. 315, Kluwer Acad. Pub., Dordrecht (1990), pp. 441-454. MR 91j:42024, Zbl. 0736.42023.
	Article 5
	C. Heil, A discrete Zak transform, The MITRE Corporation, Technical Report MTR-89W00128, Decem-ber 1989

Concevoir un plug-in plus complexe

- On souhaite pouvoir passer plus de paramètres à notre méthode. Si on garde le même principe il faut que l'on se souvienne de la position des arguments dans l'appel de la fonction. Une autre solution consiste à passer un objet

```
$.fn.extend(  
  {  
    coloriser:function(parametres){  
      this.each((i,e)=>{$(e).css('background',parametres.couleur);}) ;  
      return this ;  
    }  
  })  
$('article').coloriser({'couleur' :'yellow'})
```

Inconvénient : si objet n'a pas de propriété couleur ???

Concevoir un plug-in plus complexe

- Pour régler ce problème, on peut instancier un objet par défaut **var defaults={'couleur' : 'red'}** à l'intérieur de la fonction et on fusionnera l'objet reçu en paramètre et notre objet par défaut ce qui nous donnera dans tous les cas le paramètre couleur.

```
$.fn.extend(  
  {  
    coloriser:function(parametres){  
      var defaults={'couleur' : 'red'} ;  
      var options=$.extend({},defaults,parametres) ;  
      this.each((i,e)=>{$(e).css('background',options.couleur);});  
      return this ;  
    }  
  })  
$('article').coloriser({'couleur' : 'yellow'})
```

jQuery : Plug-ins

- Il existe plusieurs plug-ins pour la bibliothèque jQuery
 - jQuery.ui
 - <https://jqueryui.com>
 - <http://plugins.jquery.com>

Popular Tags

- [ui](#) (542)
- [jquery](#) (482)
- [form](#) (285)
- [animation](#) (273)
- [input](#) (252)
- [image](#) (210)
- [responsive](#) (184)
- [slider](#) (172)
- [ajax](#) (154)
- [scroll](#) (140)

Les bonnes pratiques

- Importance du nommage des variables et fonctions
- Favoriser les tests d'égalités et inégalités strictes
 - comprendre la différence entre les deux
- Utilisation de l'affectation : var
 - comprendre la portée des variables (scope)
- Délimiteur de fin de ligne
- Evitez de mettre du code partout dans les fichiers html par exemple. concentrer dans des fichiers js
 - Faciliter de lecture, réutilisabilité ..
- Essayez de garder vos fonctions atomiques

Les bonnes pratiques

- Essayez d'organiser vos fichiers de code en regroupant les fonctions qui traitent du même domaine métier
- Essayer d'organiser la structure de vos fichiers de code en plaçant en premier les fonctions et méthodes ayant un haut niveau d'abstraction
- n'utilisez pas trop des fonctions anonymes, réservez les par exemple pour des cas où vous êtes certain que leur exécution ne survient qu'une seule fois. Dès lors qu'une fonction ou une méthode est exécutée plusieurs fois mieux vaut avoir une fonction ou une méthode nommée.
- Commencez vos fonctions par un bloc de trois lignes maxi pour les commentaires
- Méfiez vous des dépendances entre objets, car il sera difficile d'utiliser ensuite un objet sans instancier l'autre dont il dépend.
- N'utilisez jamais une méthode ou une fonction ou un concept reconnu comme déprécié et surtout obsolète. Consultez fréquemment les sites web et les blogs spécialisés Javascript et jQuery.
- Pratiquez constamment la revue de code surtout lors d'un travail en équipe pour vérifier le respect des conventions fixées en début de projet.

NodeJS

Créé en 2009 par Ryan Dahl
codé en Javascript

- Introduction
 - Asynchrone
 - Non bloquant
 - Scalable
 - Déclenchement d'événements
 - Beaucoup de modules disponibles
 - Traitements du côté du serveur

Node JS

- Création d'un simple server sampleserver.js

```
var http = require('http');
function dealWithWebRequest(request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello PSB Node.js");
    response.end();
}
var webserver = http.createServer(dealWithWebRequest).listen(8124,"127.0.0.1");
webserver.once('listening', function() {
    console.log('Server running at http://127.0.0.1:8124/');
});
```

Exemple: <https://nodejs.org/docs/latest-v11.x/api/synopsis.html>

NodeJs - Modules

- http: serveur et client Web http
- net : serveur et client TCP
- cradle : base de données en cache
- Xml2js : XML vers Javascript
- Crud-file-serveur : CRUD de n'importe quel type de fichier
- Djangode : Framework utilisant les concepts de Django
- Mojito : MVC et librairies permettant le développement en HTML5
- Nodepress : MVC permettant le dev de blogs
- Compress : Compression de données en Gzip
- Session : gestionnaire de sessions
- Form2json : formulaire vers un format json
- + des librairies pour gérer tous les types de bases de données (postgres, mysql, oracle, mongodb ...)

NodeJs - Modules

- EveryAuth : connexion à plusieurs APIs
- emailjs : envois de mail
- Jsonjs : parser json
- chatio : simple chat
- Possibilité de créer vos propre module
- NPM : Node Packaged Modules
 - npm install everyauth

NPM

#npm --help

Usage: npm <command>

where <command> is one of:

access, adduser, audit, bin, bugs, c, cache, ci, cit,
clean-install, clean-install-test, completion, config,
create, ddp, dedupe, deprecate, dist-tag, docs, doctor,
edit, explore, get, help, help-search, hook, i, init,
install, install-ci-test, install-test, it, link, list, ln,
login, logout, ls, org, outdated, owner, pack, ping,
prefix,
profile, prune, publish, rb, rebuild, repo, restart, root,
run, run-script, s, se, search, set, shrinkwrap, star,
stars, start, stop, t, team, test, token, tst, un,
uninstall, unpublish, unstar, up, update, v, version,
view, whoami

npm <command> -h quick help on
<command>

npm -l display full usage info

npm help <term> search for help on
<term>

npm help npm involved overview

Specify configs in the ini-formatted file:

/home/ystroppa/.npmrc

or on the command line via: npm

<command> --key value

Config info can be viewed via: npm help
config

NPM --détail

#npm install -h

```
npm install (with no args, in package dir)
npm install [<@scope>/]<pkg>
npm install [<@scope>/]<pkg>@<tag>
npm install [<@scope>/]<pkg>@<version>
npm install [<@scope>/]<pkg>@<version range>
npm install <folder>
npm install <tarball file>
npm install <tarball url>
npm install <git:// url>
npm install <github username>/<github project>
```

aliases: i, isntall, add

common options: [--save-prod|--save-dev|--save-optional] [--save-exact] [--no-save]

#npm install

Lecture du fichier
packages json et
télécharge les éléments
indiqués avec leur
version.

NodeJs - Packages.json

- Des architectures riches pour permettre tout type de développement d'applications.
- Détails du fichier packages.json
 - fichier qui décrit le contenu de votre projet ainsi que ses dépendances en listant les modules et leur version.
 - du coup facile à déployer car on inscrit les éléments et les commandes

NodeJS - Express

- Micro framework
- Fournit des outils de base pour aller plus vite dans la conception d'applications Node.js
- Installation
 - `node install express`
- Permet de gérer plus facilement les routes et l'utilisation de templates.

NodeJs

- Organisation de votre projet :
 - package.json // indique les dépendances
 - /nodes-modules // répertoire de stockage des modules de votre projet
 - /bin/www
 - app.js
 - /routes/
 - /node_modules/

Socket.io

- Permet de construire des web sockets pour des applications temps réels entre plusieurs utilisateurs.
 - communication asynchrone
 - envois d'events entre le serveur et un ou plusieurs clients
 - Broadcast ou communication avec un seul client
 - Asynchrone donc pas de problème de concurrence entre les différents clients
 - utile pour les applications temps réels
- Exemples d'élaboration d'un chat simple :
 - <https://www.programwitherik.com/getting-started-with-socket-io-node-js-and-express/>
 - <https://socket.io/get-started/chat/>
-

Socket.io

- Exemple de mise ne oeuvre d'une application chat simple :
 - git clone <https://github.com/socketio/chat-exemple.git>
 - Analyse du contenu importé
 - |—— app.json
 - |—— index.html
 - |—— index.js
 - |—— package.json
 - |—— README.md

```
{
  "name": "socket-chat-example",
  "version": "0.0.1",
  "description": "my first socket.io app",
  "dependencies": {
    "express": "^4.15.2",
    "socket.io": "^1.7.3"
  },
  "scripts": {
    "start": "node index.js"
  }
}
```

```
// index.js
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);
var port = process.env.PORT || 3000;

app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', function(socket){
  socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });
});

http.listen(port, function(){
  console.log('listening on *:' + port);
});
```

```
<html>
<head>
  <title>Socket.IO chat</title>
  <style>
    * { margin: 0; padding: 0; box-sizing: border-box; }
    body { font: 13px Helvetica, Arial; }
    form { background: #000; padding: 3px; position: fixed; bottom: 0; width: 100%; }
    form input { border: 0; padding: 10px; width: 90%; margin-right: .5%; }
    form button { width: 9%; background: rgb(130, 224, 255); border: none; padding: 10px; }
    #messages { list-style-type: none; margin: 0; padding: 0; }
    #messages li { padding: 5px 10px; }
    #messages li:nth-child(odd) { background: #eee; }
    #messages { margin-bottom: 40px }
  </style>
</head>
<body>
  <ul id="messages"></ul>
  <form action="">
    <input id="m" autocomplete="off" /><button>Send</button>
  </form>
  <script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
  <script src="https://code.jquery.com/jquery-1.11.1.js"></script>
  <script>
    $(function () {
      var socket = io();
      $('#form').submit(function(){
        socket.emit('chat message', $('#m').val());
        $('#m').val("");
        return false;
      });
      socket.on('chat message', function(msg){
        $('#messages').append($('

```

Socket.io

- vérifiez si le port 3000 est disponible
- Exécutez la commande `npm install` sous le répertoire
 - l'utilitaire `npm` consulte le fichier `package.json` et vérifie si les modules demandés sont déjà présents sous le répertoire `node_modules` si pas présents créer le répertoire et les télécharge.
- `npm start`

Liste des fichiers téléchargés et installés

- app.json
- index.html
- index.js
- node_modules
 - accepts
 - after
 - arraybuffer.slice
 - array-flatten
 - backo2
 - base64-arraybuffer
 - base64id
 - better-assert
 - blob
 - body-parser
 - bytes
 - callsite
 - component-bind
 - component-emitter
 - component-inherit
 - content-disposition
 - content-type
 - cookie
 - cookie-signature
 - debug

- depd
- destroy
- ee-first
- encodeurl
- engine.io
- engine.io-client
- engine.io-parser
- escape-html
- etag
- express
- finalhandler
- forwarded
- fresh
- has-binary
- has-cors
- http-errors
- iconv-lite
- indexof
- inherits
- ipaddr.js
- isarray
- json3
- media-typer
- merge-descriptors

- media-typer
- merge-descriptors
- methods
- mime
- mime-db
- mime-types
- ms
- negotiator
- object-assign
- object-component
- on-finished
- options
- parsejson
- parseqs
- parseuri
- parseurl
- path-to-regexp
- proxy-addr
- qs

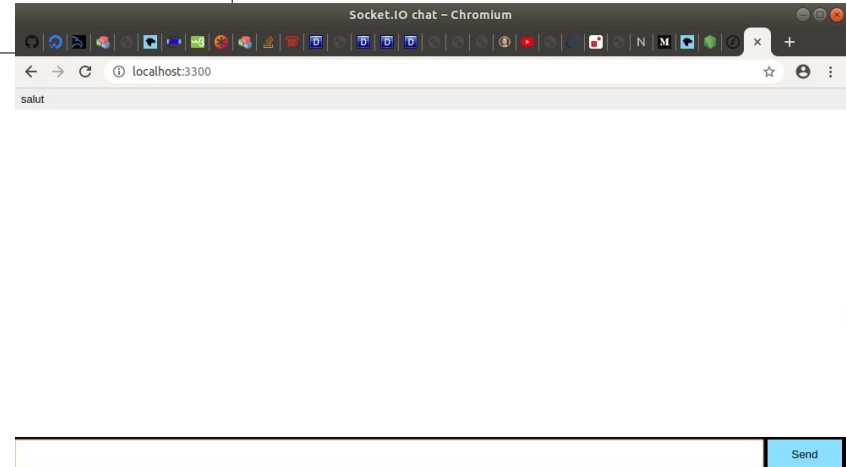
- range-parser
- raw-body
- safe-buffer
- safer-buffer
- send
- serve-static
- setprototypeof
- socket.io
- socket.io-adapter
- socket.io-client
- socket.io-parser
- statuses
- to-array
- toidentifier
- type-is
- ultron
- unpipe
- utils-merge
- vary
- ws
- wtf-8
- xmlhttprequest-ssl
- yeast
- package.json
- package-lock.json
- README.md

Socket.io

- Il nous suffit de démarrer l'application :
- `npm start` (lecture du fichier `package.json` pour savoir quel est le fichier à exécuter)

```
> socket-chat-example@0.0.1 start /home/ystroppa/PSB/socket-io/chat-example  
> node index.js
```

```
listening on *:3300
```



NodeJs

- Performances de NodeJs
 - avec une simple utilisation, les performances sont les mêmes que Apache/PHP
 - en mode multiserveurs les performances sont supérieures
- Evolution avec Socket IO très efficace, grâce à la scalabilité et au fonctionnement asynchrone.
- NodeJS casse les barrières du langage entre le client et le serveur

Jquery -- NodeJs

- Voir le site
 - <https://gist.github.com/diorahman/1520485>
-

Exemple d'appel jQuery avec Nodejs

main.js

```
var express = require('express');
var app = express.createServer();
app.use(express.bodyParser());
app.post('/endpoint', function(req, res){
    var obj = {};
    console.log('body: ' + JSON.stringify(req.body));
    res.send(req.body);
});
app.listen(3000);
```


Partie cliente

index.html

```
<html>
  <head>
    <title>jsonp test</title>
    <script src="http://code.jquery.com/jquery-1.6.2.min.js"></script>
    <script src=" ./scripts/fonctions.js"></script>>
  </head>
  <body>
    <div id="select_div"><a href="#" id="select_link">Test</a></div>
  </body>
</html>
```

fonctions.js

```
$(function(){
  $('#select_link').click(function(e){
    e.preventDefault();
    console.log('select_link clicked');
    var data = {};
    data.title = "title";
    data.message = "message";
    $.ajax({
      type: 'POST',
      data: JSON.stringify(data),
      contentType: 'application/json',
      url: 'http://localhost:3000/endpoint',

      success: function(data) {
        console.log('success');
      }

    });
  });
});
```

Solutions Graphiques

- Explications sur les librairies graphiques
 - highcharts
 - D3js

Exercice de conception d'une application de jeu

- On souhaite bâtir une application de jeu de type l'âne rouge/Rush Hour/ Katamino
- Positionnement du jeu dans la page
- Chargement des solutions
-

Partie 3D

- Utilisation de three.js pour faire des représentations en 3D et animation
-

Les promesses

https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses

- Une promesse est un objet (Promise) qui représente la complétion ou l'échec d'une opération asynchrone. Une promesse est un objet qui est renvoyé auquel on attache des callbacks.

```
function faireQqc() {  
  return new Promise((resolve, reject) => {  
    console.log("C'est fait");  
    // réussir une fois sur deux  
    if (Math.random() > .5) {  
      resolve("Réussite");  
    } else {  
      reject("Échec");  
    }  
  })  
}  
  
const promise = faireQqc();  
promise.then(successCallback, failureCallback);
```

```
// les deux fonctions callbacks passées  
function successCallback(résultat) {  
  console.log("L'opération a réussi avec le  
message : " + résultat);  
}
```

```
function failureCallback(erreur) {  
  console.error("L'opération a échoué avec le  
message : " + erreur);  
}
```

Les promesses

À la différence des imbrications de callbacks, une promesse apporte certaines garanties :

- Les callbacks ne seront jamais appelés avant la fin du parcours de la boucle d'évènements JavaScript courante
- Les callbacks ajoutés grâce à then seront appelés, y compris après le succès ou l'échec de l'opération asynchrone
- Plusieurs callbacks peuvent être ajoutés en appelant then plusieurs fois, ils seront alors exécutés l'un après l'autre selon l'ordre dans lequel ils ont été insérés.

```
// exemple de chainage des promesses
faireQqc()
  .then(result => faireAutreChose(result))
  .then(newResult => faireUnTroisiemeTruc(newResult))
  .then(finalResult => {
    console.log('Résultat final : ' + finalResult);
  })
  .catch(failureCallback);
```

Installation de Docker

```
FROM node:alpine
RUN apk add --no-cache curl
# Create app directory
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
# Bundle app source
COPY . .
EXPOSE 3000
CMD [ "npm", "start" ]
```

Commandes :

Construction de l'image
#docker build -t site .

Démarrage de l'image sous Docker
#docker run -p 3000:3000 site

Exercices pour vous entraîner

- https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_array_methods1
-
-