



2^{ème} partie

JavaScript

langage objet

Jean-Christophe DUBOIS
jean-christophe.dubois@univ-rennes.fr



Objet

- Définition

Un objet est une instantiation d'une Classe.

Élément, éventuellement complexe, défini par:

- un **constructeur** : création d'un nouvel objet.
Il définit toutes les caractéristiques initiales de l'objet
- des **propriétés** : variables rattachées à l'objet pour le définir
- des **méthodes** : fonctions utilisées pour effectuer des opérations sur les propriétés de l'objet

Un objet peut être abstrait : une date, une commande, un prix
ou concret : un produit, un client, une voiture

Les objets natifs

- JavaScript dispose d'objets prédéfinis appelés objets natifs
 - String : les chaînes de caractères
 - RegExp : les expressions régulières
 - Number : les nombres
 - Math : les constantes et fonctions mathématiques
 - Boolean : les booléens
 - Date : les dates
 - Array : les tableaux
 - Function : les fonctions
 - ...

Objets standards

Array

Propriétés

`Array.prototype.length`

`Array.prototype[@@unscopables]`

Méthodes

`Array.from()`

`Array.isArray()`

`Array.of()`

`Array.prototype.concat()`

`Array.prototype.copyWithin()`

Objets standards

String

Propriétés

`String.length`

Méthodes

`String.fromCharCode()`

`String.fromCodePoint()`

`String.prototype.anchor()`

`String.prototype.big()`

`String.prototype.blink()`

`String.prototype.bold()`

Différentes méthodes

- Méthode applicable par l'intermédiaire de la classe

```
Array.isArray([1,2,3]);           // true
Array.isArray(123);               // false
```

- Méthode applicable directement sur une instance d'objet :
Objet.prototype.methode()

```
[1,2].concat([4,5]);             // [1,2,4,5]
```

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

Objet global

- **Propriétés** non rattachées à un objet spécifique
 - **Infinity** : valeur numérique représentant l'infini positif ou négatif
 - **NaN** : "Not-a-Number" – valeur non numérique
 - **undefined** : variable pour laquelle aucune valeur n'a été assignée
- **Fonctions** non rattachées à un objet spécifique
 - **eval()** : évalue du code représenté sous forme d'une chaîne

```
var x = 2, y = 39;  
eval("x + y + 1");           // renvoie 42
```

MAIS utilisation à éviter (injection de code) !

Objet global

- **Fonctions** non rattachées à un objet spécifique (suite)
 - **isFinite()** : teste si la valeur est un nombre fini

```
isFinite(NaN);           // false
isFinite(0);             // true
isFinite("42");          // true
```

- **isNaN()** : teste si la valeur n'est pas un nombre

```
isNaN("texte");          // true
isNaN("42");             // true
isNaN(42);               // false
```

- **parseFloat()/parseInt()** : transforme une chaîne en un nombre flottant/un nombre entier

String

- **Propriété** de l'objet **String**
 - **length** : longueur de la chaîne de caractères
- Principales **méthodes** de l'objet **String**
 - *charAt(index)*
 - *concat(string)*
 - *indexOf(string)*
 - *includes(string, [indice])*
 - *lastIndexOf(string, [indice])*
 - *match(expReg)*
 - *replace(expReg)*
 - *search(expReg)*
 - *slice(indice1, [indice2])*
 - *substring(indice1, [indice2])*
 - *split(string)*
 - *toLowerCase(string)*
 - *toUpperCase(string)*
 - *trim()*

String

- *charAt(i)* : retourne le caractère à l'index *i* spécifié en paramètre
- *concat(s1,s2...)* : concatène 1 ou plusieurs chaînes et retourne la nouvelle chaîne
- *indexOf(s)* : retourne l'indice de la 1^{ère} occurrence d'une sous-chaîne *s*
- *includes(s, i)* : teste la présence d'une sous-chaîne *s* dans une chaîne
- *lastIndexOf(s, i)* : retourne l'indice de la dernière occurrence d'une sous-chaîne *s*
- *match(eR)* : teste la correspondance entre une chaîne et une exp. Régulière *eR*
- *search(eR)* : cherche une exp. régulière *eR* dans une chaîne et retourne sa position
- *replace(eR)* : remplacement d'après une exp. Régulière *eR*
- *slice(i1,i2)/substring(i1,i2)* : extrait une sous-chaîne d'après 1 ou 2 index donnés
- *split(s)* : divise la chaîne d'après le séparateur *s* et retourne un tableau des ss-chaînes
- *toLowerCase(s)/toUpperCase(s)* : convertit la chaîne *s* en minuscule/majuscule
- *trim()* : retire les espaces en début et fin de chaîne

Exercice

- **Larlépem-vous le louchébem ?** Parlez-vous l'argot des bouchers ?

Pour cela il faut :

- remplacer la première lettre par un « L »
- replacer la première lettre remplacée en fin de mot
- ajouter en toute fin une extension dépendante de la 1^{ère} lettre
 - « em » pour le « b »
 - « uche » pour le « m »
 - « ès » pour le « c »
 - « é » pour le « p »
 - « é » pour le « d »
 - « ès » pour les autres lettres
 - « oque » pour le « f »
- Pensez à mettre au propre le texte saisi par l'interne
 - supprimer les blancs en début et fin,
 - mettre en minuscule
- Testez les mots : boucher, merci, café, douce, fou
- Finalement, parliez-vous déjà le **louchébem** ?

Exemple

“ BOUcher ”
“BOUcher”
“boucher”
“loucher”
“loucherb”
“loucherbem”

RegExp

- Les **expressions régulières** (ou **exp. rationnelles**) : langage de description pour effectuer une recherche/remplacement de caractères, appelé **motif**, dans une chaîne de caractères. Ce langage est disponible aussi en Python, Perl, PHP...
- **Syntaxe** :
 /motif/drapeau ou **new RegExp(motif [, drapeau])**
avec **motif** l'élément recherché
et **drapeau** limite de l'opération à effectuer :
 - **i** : casse **i**gnorée // minuscule ou majuscule
 - **g** : recherche **g**lobale // toutes les occurrences
 - **m** : recherche **m**ulti-ligne // sur plusieurs lignes de texte

`/La/i` // recherche "la", "LA", "La" ou "lA"

`/li/gim` // retrouve "Li" et "li" dans "Lire deux \nlignes"

RegExp

- <https://www.regextester.com/>

The screenshot displays the 'RegEx Testing' website, which is part of 'Dan's Tools'. The navigation bar includes links for 'Web Dev', 'Conversion', and 'Encode/Decoder'. The main interface is divided into two primary sections: 'Regular Expression' and 'Test String'.

In the 'Regular Expression' section, the input field contains the regex `/li/ig`. To the right of this field, there is a dropdown menu set to 'Javascript' and a 'flags' button. A tooltip titled 'Expression Flags' is visible, showing three options: 'ignore case (i)' (checked), 'global (g)' (checked), and 'multiline (m)' (unchecked).

In the 'Test String' section, the input field contains the text 'Lire deux lignes'. The first 'l' in 'Lire' and the 'l' in 'lignes' are highlighted with blue boxes. A tooltip points to the 'l' in 'lignes', displaying the match results: 'match: li' and 'range: 10-11'.

RegExp

- Méthode de l'objet **RegExp**
 - **test()** : teste la correspondance entre un texte et une expression régulière.
Retourne **true** en cas de succès, **false** sinon

```
let chaine = "Bientôt le week-end";  
let motif1 = /t/;           // Recherche du caractère "t"  
console.log(motif1.test(chaine));           // true  
let motif2 = /T/;           // Recherche du caractère "T"  
console.log(motif2.test(chaine));           // false  
let motif3 = /T/i; // Recherche de "T" ou "t"  
console.log(motif3.test(chaine));           // true
```

RegExp

- Possibilité d'utiliser des méthodes de l'objet **String**
 - **match()** : retourne toutes les correspondances retrouvées **null** sinon

```
let chaine = "Bientôt le week-end";  
let motif1 = /t/;           // Recherche du caractère "t"  
let motif2 = /T/;           // Recherche du caractère "T"  
console.log(chaine.match(motif1));           // ["t", "t"]  
console.log(chaine.match(motif2));           // null
```

- **search()** : retourne l'indice de positionnement de la chaîne recherchée. **-1** si non trouvé.

```
console.log(chaine.search(/week/));           // 11  
console.log(chaine.search(/weak/));           // -1
```

RegExp

- Possibilité d'utiliser des méthodes de l'objet **String** (suite)
 - *replace()* : retourne une nouvelle chaîne de caractères dans laquelle le modèle indiqué est remplacé, chaque fois qu'il apparaît, par une nouvelle chaîne

```
let chaine = "Bientôt le week-end";  
console.log(chaine.replace(/Bientôt/g, "Vivement"));  
                // "Vivement le week-end"  
console.log(chaine.replace(/en/g, "in"));  
                // "Biintôt le week-ind"
```

RegExp

- Ensemble de caractères
 - (a|z) recherche l'un ou l'autre des caractères, a ou z
 - [aeiouy] recherche un des caractères précisé a, e, i, o, u ou y
 - [a-z] ou [A-Z] recherche une des lettres (minuscules ou majuscules) appartenant au groupe indiqué
 - [0-9] recherche un chiffre compris entre 0 et 9
 - [^0-9] recherche un caractère autre que ceux indiqués

```
/[^0-2]/g
```

```
/b[aiu]lle/g
```

RegExp

- Caractères spéciaux

- `.` recherche tout caractère excepté la fin de ligne `\n` et `\r`
- `\w` recherche une chaîne de caractères (`[a-zA-Z0-9_]`)
- `\W` recherche un caractère qui n'est **pas** une chaîne
- `\d` recherche un caractère numérique (idem `[0-9]`)
- `\D` recherche un caractère **non** numérique (idem `[^0-9]`)
- `\s` recherche un espace (espace, tabulation, retour chariot...)
- `\S` recherche un caractère **autre** qu'un espace

```
\D // retourne "% ?" parmi "100% ?"
```

```
\w // retourne "100Ahoui" parmi "100% ! Ah, oui ?"
```


RegExp

- **Caractères spéciaux (suite)**
 - `\t` caractère tabulation horizontale (`\v` si verticale)
 - `\n` caractère fin de ligne
 - `\r` caractère retour chariot
 - `\s` espace

- **Limites**

- Début `^` et fin de chaîne `$`

```
/^super/g // recherche une chaîne débutant par super  
/ment$/g // recherche une chaîne finissant par ment
```

- `\b` début ou fin de mot (espace et caractère de ponctuation)

```
/\bfi/gi // retourne "fi" dans "C'est fini, enfin !"
```

- `\B` chaîne contigüe à des caractères

```
/\Bfi/gi // retourne "fi" dans "Fini, enfin !"
```

RegExp

- Quantificateurs

- x^* : recherche 0, 1 ou plusieurs fois l'élément précédent
- x^+ : recherche 1 ou plusieurs fois l'élément précédent
- $x?$: recherche 0 ou 1 fois l'élément précédent
- $x(?:y)$: x si et seulement si x est suivi de y
- $x(?:!y)$: x si et seulement si x n'est pas suivi de y

- Intervalles

- $x\{n\}$: recherche le caractère x répété n fois
- $x\{n,\}$: recherche le caractère x répété au moins n fois
- $x\{n,m\}$: recherche le caractère x répété entre n et m fois

`/[a-z]{4,6}/g`

Number

- **Propriétés** de l'objet **Number**
 - EPSILON : plus petit intervalle en JavaScript
 - MIN/MAX_VALUE : plus petite/grande valeur
 - MIN//MAX_SAFE_INTEGER : plus petite/grande valeur entière
 - NaN : Not a Number – valeurs non numériques
 - NEGATIVE_INFINITY/POSITIVE_INFINITY : infini négatif/positif
- Principales **méthodes** de l'objet **Number**
 - **Number.isNaN()** : teste si la valeur n'est pas un nombre (NaN)
 - **Number.isFinite()** : teste si la valeur est un nombre fini
 - **Number.isInteger()** : teste si la valeur est un entier
 - **Number.isSafeInteger()** : teste si la valeur peut être représentée comme un entier, entre $-(2^{53} - 1)$ et $(2^{53} - 1)$
 - **toFixed()** : retourne une chaîne contenant un nombre arrondi à un nombre de décimales défini
 - **toString()** : retourne une chaîne représentant le nombre

Math

- Propriétés de l'objet Math

- PI : valeur approchée de Pi (3.141592653589793)
- LN2 : valeur du Logarithme Népérien de 2 (~ 0.693)
- LN10 : valeur du Logarithme Népérien de 10 (~ 2.302)
- mais aussi E, LOG2E, LOG10E, SQRT1_2 et SQRT2

- Principales méthodes de l'objet Math

- | | | |
|-------------------|-----------------------|-----------------|
| • Math.cos(x) | • Math.round(x) | • Math.log(x) |
| • Math.acos(x) | • Math.ceil(x) | • Math.exp(x) |
| • Math.sin(x) | • Math.floor(x) | • Math.pow(x,y) |
| • Math.asin(x) | • Math.max(x,y,...,n) | • Math.sqrt(x) |
| • Math.tan(x) | • Math.min(x,y,...,n) | |
| • Math.atan(x) | • Math.random() | |
| • Math.atan2(y,x) | • Math.abs(x) | |

Math

- Spread ... avec appel de fonction

- La notation **spread ...** est particulièrement intéressante pour l'appel de fonction avec un grand nombre d'arguments
- Soit les tableaux de valeurs monTab1 et monTab2 :

```
let monTab1 = [12, 23, 1, 43, 54, 102, 203];  
let monTab2 = [-12, 23, 51, 234, 81, 11, 0, 1024];
```

Il est possible d'appeler la fonction en passant en argument la totalité des tableaux

```
alert(Math.max(...monTab1));  
alert(Math.min(...monTab1, ...monTab2));
```

Boolean

- Méthode de l'objet Boolean
 - *Boolean()* : teste si une expression est vraie
 - *Boolean(val1 > val2)* ou plus simplement (*val1 > val2*)
- Quelques valeurs remarquables
 - *Boolean(0);* // false
 - *Boolean(-0);* // false
 - *Boolean("");* // false
 - *Boolean(null);* // false
 - *var x;* // x undefined
 - *Boolean(x);* // false
 - *Boolean(true);* // true

Date

- **Définition** : Une **date** est définie en précisant une année, un mois et de façon optionnelle un jour, une heure, une minute, une seconde et une milliseconde
- **Constructeur**
 - **new Date()** : constructeur
 - Initialisations :
 - let maDate = **new Date**(1704063600000);
// 31/12/23 à 0h00 soit 1.704.063.600.000 ms depuis le 01/01/1970
 - let maDate = **new Date**("1 janvier 2020 00:00:00");
 - let maDate = **new Date**("2005, 11, 14"); // 14/11/2005
- Sans argument, la date créée est initialisée avec la date du jour
- Sans constructeur **new**, la date du jour sera donnée sous la forme d'une chaîne de caractère

Date

- **Propriété** de l'objet **Date**
 - **length** : nombre d'arguments du constructeur (7)
- **Méthodes** de l'objet **Date**
 - **Date.now()** : retourne la valeur numérique correspondant au temps courant. Le nombre de millisecondes depuis le 01/01/1970
 - **Date.parse()** : analyse une représentation textuelle d'une date ("14 Nov 2005"), et retourne le nombre de millisecondes depuis le 01/01/1970
 - **Date.UTC()** : accepte une date dans sa version longue (2005, 11, 14, 0, 0, 0), et retourne le nombre de millisecondes depuis le 01/01/1970

Date

- Méthodes de l'objet Date

- *getFullYear()* : renvoie l'année (sur 4 chiffres) d'une date
- *getMonth()* : renvoie le mois (0-11) d'une date
- *getDay()* : renvoie le jour de la semaine (0-6) d'une date
- *getDate()* : renvoie le jour du mois (1-31) d'une date
- *getHours()* : renvoie l'heure (0-23) d'une date
- *getMinutes()* : renvoie les minutes (0-59) d'une date
- *getSeconds()* : renvoie les secondes (0-59) d'une date
- *getMilliseconds()* : renvoie les millièmes de secondes (0-999) d'une date
- *getTime()* : nombre de millisecondes écoulées entre la date spécifiée et le 01/01/1970 (pour les dates antérieures, les valeurs sont négatives)

Array

- **Propriété** de l'objet **Array**
 - **length** : nombre d'éléments dans le tableau
- **Méthodes** de l'objet **Array**
 - *Array.isArray()*
 - *concat()*
 - *includes()*
 - *indexOf()*
 - *lastIndexOf()*
 - *join()*
 - *pop()*
 - *push()*
 - *reverse()*
 - *shift()*
 - *unshift()*
 - *splice()*
 - *slice()*
 - *toString()*
 - *sort()*

Array

- *Array.isArray(obj)* : vérifie si un objet est un tableau
- *concat(t1, t2...)* : concaténation de plusieurs tableaux *t1, t2...*
- *includes(elt)* : vérifie si un tableau contient l'*élément* spécifié
- *indexOf(elt, i)* : recherche un *élément* dans le tableau à partir de *i* et renvoie son indice
- *lastIndexOf(elt)* : recherche un *élément* dans le tableau, depuis la fin, et renvoie son indice
- *join(sép)* : joint tous les éléments du tableau en une chaîne de caractères (avec *sép*)
- *pop()* : supprime le dernier élément du tableau et retourne l'élément supprimé
- *push(elts)* : ajoute des *éléments* à la fin du tableau et renvoie la nouvelle longueur
- *shift()* : supprime le premier élément du tableau, et retourne l'élément supprimé
- *unshift(elts)* : ajoute des *éléments* au début du tableau et renvoie la nouvelle longueur
- *splice(i, n, elts)* : ajoute/supprime *n éléments* dans un tableau à l'indice *i*
- *slice(i_déb, i_fin)* : sélectionne une partie d'un tableau et retourne le nouveau tableau
- *reverse()* : inverse l'ordre des éléments du tableau
- *toString()* : convertit un tableau en une chaîne de caractères et renvoie le résultat

Les tableaux

- Un **tableau** est une structure dans laquelle les **données** sont accessibles grâce à un **indice**

Indice	0	1	2	3
Donnée	val1	val2	val3	val4

- Deux modes de **déclaration** sont utilisables :

```
let monTab = new array ("val1", "val2", "val3", "val4");  
let monTab = ["val1", "val2", "val3", "val4"]; // à favoriser  
let monTab = [ ]; // tableau vide
```

- Accès** au contenu d'une cellule :

```
let cellule = monTab[0]; // cellule vaut "val1"
```

- Affectation/Modification** du contenu d'une cellule :

```
monTab[0] = "nouvelleva1" ;
```

Les tableaux

- Ajout d'une cellule
 - en fin de tableau avec la méthode `push()`
`monTab.push("val5", "val6");`

Indice	0	1	2	3	4	5
Donnée	val1	val2	val3	val4	val5	val6

- en début de tableau avec la méthode `unshift()`
`monTab.unshift("val7");` // décalage

Indice	0	1	2	3	4	5	6
Donnée	val7	val1	val2	val3	val4	val5	val6

Les tableaux

- **Suppression** d'une cellule
 - en **fin** de tableau avec la méthode **pop()**
`monTab.pop();`

Indice	0	1	2	3	4	5	6
Donnée	val7	val1	val2	val3	val4	val5	val6

- en **début** de tableau avec la méthode **shift()**
`monTab.shift();`

Indice	0	1	2	3	4	5
Donnée	val7	val1	val2	val3	val4	val5

Indice	0	1	2	3	4
Donnée	val1	val2	val3	val4	val5

Les tableaux

- Découpage d'une chaîne de caractère dans un tableau
 - `String.split()` découpe une chaîne de caractères.
Le séparateur est passé en paramètre.
Chaque sous-chaîne est affectée à une cellule du tableau.

```
let planete = "mercure, venus, terre, mars";  
let tabPlanete;  
tabPlanete = planete.split(", ");  
alert(tabPlanete[0]); // tabPlanete[0] contient mercure
```

Indice	0	1	2	3
Donnée	mercure	venus	terre	mars

- `join()` permet l'opération inverse en transformant un tableau en chaînes de caractères séparées par un caractère passé en paramètre

Les tableaux

- Duplication d'un tableau

- Lors de la duplication d'un tableau, la référence est dupliquée mais les 2 variables pointent sur le même tableau

```
let tabPlanete1 = ["mercure", "venus", "mars"];  
let tabPlanete2 = tabPlanete1;  
tabPlanete1[2] = "terre";  
console.log(tabPlanete2); // ["mercure", "venus", "terre"];
```

- tabPlanete2 est donc affecté par la modification effectuée au niveau de tabPlanete1 !

```
let tabPlanete3 = [].concat(tabPlanete1);  
tabPlanete1[2] = "jupiter";  
console.log(tabPlanete3); // ["mercure", "venus", "terre"];
```

- **concat()** crée un nouveau tableau tabPlanete3 indépendant

Les tableaux

- Parcourir un tableau avec une boucle for
 - Une boucle **for** permet de parcourir chaque cellule du tableau.
La propriété **length** d'un tableau (comme pour une chaîne de caractères) indique le nombre total de cellules.

```
let tabPlanete = ["mercure", "venus", "terre", "mars"];  
for (let i=0; i<tabPlanete.length; i++) {    // length vaut 4  
    alert(tabPlanete[i]);  
}    // les 4 planètes seront successivement affichées  
tabPlanete [7] = "uranus";                    // length vaut 8
```

- La valeur de **length** peut être augmentée (sans modification du tableau) ou réduite (sup. de cellules)

Les tableaux

- Parcourir un tableau avec une boucle For ... of

```
let tabPlanete = ["mercure", "venus", "terre", "mars"];  
for (let planete of tabPlanete) {  
    alert(planete);  
} // les 4 planètes seront successivement affichées
```

- For ... of peut être utilisé sur n'importe quel objet itérable, tel qu'un tableau, une chaîne de caractères, ...

Décomposition d'un tableau

- Transformation des données d'un tableau en variables

```
let tabDateAlunissage = [ 20, 07, 1969];  
let jour = tabDateAlunissage[0];           // syntaxe  
let mois = tabDateAlunissage[1];           // très  
let annee = tabDateAlunissage[2];          // fastidieuse
```

Simplification de la syntaxe en encadrant avec des crochets [] les futures variables à créer

```
let [ jour, mois, annee ] = tabDateAlunissage;  
alert(jour);           // 20  
alert(mois);           // 07  
alert(annee);          // 1969
```

Le tableau original n'est pas modifié

Décomposition d'un tableau

- Choix des données du tableau à conserver

```
let tabDateAlunissage = [ 20, 07, 1969];  
let jour = tabDateAlunissage [0];           // syntaxe  
let mois = tabDateAlunissage [1];           // très  
let annee = tabDateAlunissage [2];          // fastidieuse
```

Si aucun nom de variable n'est donnée, les valeurs ne sont pas utilisées

```
let [ jour, , annee ] = tabDateAlunissage  
alert(jour);           // 20  
alert(annee);          // 1969
```

La syntaxe *spread* d'un tableau

- Manipulation de tableaux

Soit les 2 tableaux suivants :

```
let tabTellurique = [ "Vénus", "Terre", "Mars" ];  
let tabGazeuse = [ "Jupiter", "Neptune", "Saturne" ];
```

Constitution d'un 3^{ème} tableau à partir des 2 autres :

```
let tabPlanete = [...tabTellurique, ...tabGazeuse];
```

Ajouts d'éléments supplémentaires

```
let tabSystSol = [ "Mercure", ...tabPlanete, "Uranus" ];
```

- Spread ...** est utile pour décomposer les éléments d'un tableau. Il facilite également la copie ou la fusion de tableaux

La syntaxe *spread* d'un tableau

- Manipulation de chaîne de caractères

Soit la chaîne suivante :

```
let maChaine = "Terre";
```

Affectation de chaque caractère de la chaîne dans une cellule d'un tableau :

```
let tabTerre= [...maChaine];  
// tabTerre vaut ["T", "e", "r", "r", "e"]
```