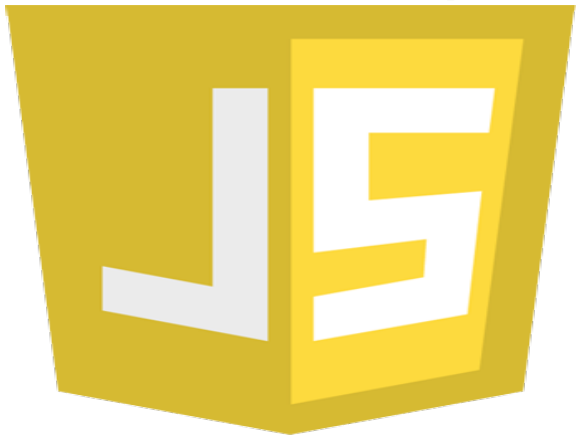
JavaScript : types complexes

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

Plan

JavaScript

Tableau (array)

- une variable pouvant avoir simultanément plusieurs valeurs
- l'accès à chaque valeur s'effectue via son indice et avec l'opérateur `[] : [ indice ]`
- la première valeur (on dit aussi élément) du tableau est d'indice 0.
- les valeurs peuvent avoir plusieurs types différents

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

Le raccourci

```
var tab = [value1, value2,... valueN];
```

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

Le raccourci

```
var tab = [value1, value2,... valueN];
```

Accès à l'élément du tableau d'indice `i`

```
tab[i]
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

Pour connaître la taille du tableau (nombre d'élément)

```
console.log(tab.length);
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

Pour connaître la taille du tableau (nombre d'élément)

```
console.log(tab.length);
```

Pour afficher le premier élément du tableau

```
tab[0]
```

JavaScript

Exemple

```
var tab = [2, 3, 5];
```

Pour connaître la taille du tableau (nombre d'élément)

```
console.log(tab.length);
```

Pour afficher le premier élément du tableau

```
tab[0]
```

Pour afficher le dernier élément du tableau

```
tab[tab.length - 1]
```

JavaScript

Exemple avec indice négatif

```
console.log(tab[-1]);
// affiche undefined
```

JavaScript

Exemple avec indice négatif

```
console.log(tab[-1]);
// affiche undefined
```

Exemple avec indice inexistant (dépassant la taille du tableau)

```
console.log(tab[100]);
// affiche undefined
```

JavaScript

Pour ajouter une nouvelle valeur (par exemple 7) au tableau

```
tab[tab.length] = 7;
```

JavaScript

Pour ajouter une nouvelle valeur (par exemple 7) au tableau

```
tab[tab.length] = 7;
```

Ou tout simplement

```
tab.push(7);
```

Pour afficher un tableau

```
console.log(tab);
```

Pour afficher un tableau

```
console.log(tab);
```

Pour parcourir et afficher un tableau

```
for (var i = 0; i < tab.length; i++) {
  console.log(tab[i]);
}
```

Pour afficher un tableau

```
console.log(tab);
```

Pour parcourir et afficher un tableau

```
for (var i = 0; i < tab.length; i++) {
  console.log(tab[i]);
}
```

On peut aussi utiliser la version simplifiée de `for`

```
for(var elt of tab) {
  console.log(elt);
}
```

Pour afficher un tableau

```
console.log(tab);
```

Pour parcourir et afficher un tableau

```
for (var i = 0; i < tab.length; i++) {
  console.log(tab[i]);
}
```

On peut aussi utiliser la version simplifiée de `for`

```
for(var elt of tab) {
  console.log(elt);
}
```

Et pour récupérer l'indice

```
for (const [index, value] of tab.entries()) {
    console.log(index, value);
}
```

JavaScript

Opérations sur les tableaux

- `push()` : ajoute un élément passé en paramètre au tableau
- `pop()` : supprime le dernier élément du tableau
- `shift()` : supprime le premier élément du tableau
- `indexOf()` : retourne la position de l'élément, passé en paramètre, dans le tableau, `-1` sinon.
- `reverse()` : inverse l'ordre des éléments du tableau
- `sort()` : trie un tableau
- `splice()` : permet d'extraire, ajouter ou supprimer un ou plusieurs éléments (selon les paramètres, voir slide suivante)
- `includes()` : retourne `true` si la valeur passée en paramètre est dans un tableau, `false` sinon.
- `...`

JavaScript

Utilisons `splice` pour supprimer un seul élément à partir de la position 2

```
var sports = [ "foot", "tennis", "basket", "volley" ];
tab = sports.splice(2, 1);

for (var elt of sports) {
  console.log(elt);
}
// affiche foot, tennis, volley

console.log(tab);
// affiche basket
```

JavaScript

Utilisons `splice` pour ajouter les deux sports `rugby` et `natation` à la position 2

```
var sports = [ "foot", "tennis", "basket", "volley" ];
tab = sports.splice(2, 0, "rugby", "natation");

for (var elt of sports) {
  console.log(elt);
}
// affiche foot, tennis, rugby, natation, basket, volley

console.log(tab);
// n'affiche rien
```

JavaScript

Utilisons `splice` pour ajouter les deux sports `rugby` et `natation` à la position 2

```
var sports = [ "foot", "tennis", "basket", "volley" ];
tab = sports.splice(2, 0, "rugby", "natation");

for (var elt of sports) {
  console.log(elt);
}
// affiche foot, tennis, rugby, natation, basket, volley

console.log(tab);
// n'affiche rien
```

Aucun élément supprimé car le deuxième paramètre = 0

JavaScript

`splice` **peut être utilisée pour ajouter et supprimer à la fois**

```
var sports = [ "foot", "tennis", "basket", "volley" ];
tab = sports.splice(2, 1, "rugby", "natation");

for (var elt of sports) {
  console.log(elt);
}
  // affiche foot, tennis, rugby, natation, volley

console.log(tab);
// affiche basket
```

JavaScript

`splice` **peut être utilisée pour ajouter et supprimer à la fois**

```
var sports = [ "foot", "tennis", "basket", "volley" ];
tab = sports.splice(2, 1, "rugby", "natation");

for (var elt of sports) {
  console.log(elt);
}
  // affiche foot, tennis, rugby, natation, volley

console.log(tab);
// affiche basket
```

Un seul élément supprimé car le deuxième paramètre = 1.

JavaScript

À ne pas confondre avec `slice` qui permet d'extraire un sous-tableau sans modifier le tableau d'origine

```
var sports = [ "foot", "tennis", "basket", "volley"
   ];
tab = sports.slice(1, 3);

for(var elt of sports) {
  console.log(elt);
}
// affiche foot, tennis, basket, volley

console.log(tab);
// affiche tennis, basket
```

JavaScript

Exercice

Écrire un code **JavaScript** qui

1. permet à l'utilisateur de saisir un sport
2. supprime le sport saisi s'il est déjà dans le tableau `sports`
3. ajoute le sport saisi s'il n'est pas dans le tableau `sports`

JavaScript

Il est possible de préciser la taille d'un tableau et d'initialiser ses valeurs avec la méthode `fill`

```
var tab = new Array(3).fill(0);
console.log(tab);
// affiche [ 0, 0, 0 ]
```

JavaScript

Il est possible de préciser la taille d'un tableau et d'initialiser ses valeurs avec la méthode `fill`

```
var tab = new Array(3).fill(0);
console.log(tab);
// affiche [ 0, 0, 0 ]
```

On peut aussi utiliser `fill` pour modifier les valeurs d'une partie du tableau

```
var tab = [0, 1, 2, 3, 4, 5, 6, 7, 8];
tab.fill(9, 2, 5);
console.log(tab);
// affiche [ 0, 1, 9, 9, 9, 5, 6, 7, 8 ]
```

JavaScript

Une fonction en **JavaScript**

- Un bloc d'instructions réalisant une fonctionnalité bien déterminée.
- Pouvant retourner une seule valeur
- Pouvant prendre 0 ou plusieurs paramètres
- Pouvant appeler d'autres fonctions
- Acceptant les valeurs par défaut pour les paramètres
- Pouvant accepter un nombre variable de paramètres

JavaScript

Déclarer une fonction

```
function nomFonction ([les arguments]){
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction

```
function nomFonction ([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
function somme (a, b) {
  return a + b;
}
```

JavaScript

Déclarer une fonction

```
function nomFonction ([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
function somme (a, b) {
  return a + b;
}
```

Appeler une fonction

```
var resultat = somme (1, 3);
```

JavaScript

La fonction suivante retourne `undefined`

```
function somme(a, b) {
    return
    a + b
}

console.log(somme(2, 3));
// affiche undefined
```

JavaScript

En revanche, le suivant retourne le bon résultat

```
function somme(a, b) {
    return (
        a + b
    )
}

console.log(somme(2, 3));
// affiche 5
```

JavaScript

Le `console.log` dans la fonction `somme` ne sera jamais atteint

```
function somme(a, b) {
    return a + b
    console.log('fin')
}

console.log(somme(2, 3));
// affiche undefined
```

JavaScript

Exercice 1

Écrire une fonction `maximum2(a, b)` qui retourne le maximum entre `a` et `b`.

JavaScript

Exercice 1

Écrire une fonction `maximum2(a, b)` qui retourne le maximum entre `a` et `b`.

Exercice 2

Écrire une fonction `maximum3(a, b, c)` qui retourne le maximum entre `a`, `b` et `c` (vous pouvez utiliser la fonction `maximum2`) .

JavaScript

Exercice 3

Écrire une fonction **JavaScript** qui retourne la factorielle d'un nombre passé en paramètre.

Factorielle : rappel

- $0! = 1$
- $1! = 1$
- $n! = n * (n - 1)!$

JavaScript

Solution

```
function factorielle(n) {
    let result = 1
    for (let i = 2; i <= n; i++) {
        result *= i
    }
    return result
}


console.log(factorielle(0))
// affiche 1

console.log(factorielle(1))
// affiche 1

console.log(factorielle(3))
// affiche 6
```

JavaScript

Une version récursive de `factorielle` (une fonction récursive est une fonction qui s'appelle elle-même)

```
function factorielle(n) {
    if (n == 0 || n == 1) {
        return 1
    }
    return n * factorielle(n - 1)
}


console.log(factorielle(0))
// affiche 1

console.log(factorielle(1))
// affiche 1

console.log(factorielle(3))
// affiche 6
```

JavaScript

Exercice

Écrire une fonction récursive

- acceptant comme paramètre un entier positif `n`
- retournant le `nième` terme de la suite de **Fibonacci**

Suite de **Fibonacci**

- $U_0 = 0$
- $U_1 = 1$
- $U_n = U_{n-1} + U_{n-2}$

JavaScript

Fonction récursive : avantages

- Clarté du code
- Élégance mathématique

Fonction itérative : avantages

- Plus performant et moins coûteux en mémoire
- Pouvant souvent être optimisée par le compilateur

JavaScript

Fonction récursive : avantages

- Clarté du code
- Élégance mathématique

Fonction itérative : avantages

- Plus performant et moins coûteux en mémoire
- Pouvant souvent être optimisée par le compilateur

Fonction récursive : inconvénients

- Moins performant et plus coûteux en mémoire
- Difficulté de débogage

Fonction itérative : inconvénients

- Complexité du code
- Moins compréhensible

JavaScript

Déclarer une fonction anonyme et l'affecter a‘ une variable

```
var nomVariable = function ([les arguments]) {
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction anonyme et l'affecter a‘ une variable

```
var nomVariable = function ([les arguments]) {
  les instructions de la fonction
}
```

Exemple

```
var somme2 = function (a, b){
  return a + b;
}
```

JavaScript

Déclarer une fonction anonyme et l'affecter a‘ une variable

```
var nomVariable = function ([les arguments]) {
  les instructions de la fonction
}
```

Exemple

```
var somme2 = function (a, b){
  return a + b;
}
```

Appeler une fonction anonyme

```
var resultat2 = somme2 (1, 3);
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
paramN",] "instructions");
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
paramN",] "instructions");
```

Exemple

```
var somme4 = new Function ("a", "b", "return a + b");
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
paramN",] "instructions");
```

Exemple

```
var somme4 = new Function ("a", "b", "return a + b");
```

Appeler une fonction anonyme

```
var resultat4 = somme4 (1, 3);
```

JavaScript

Et si on appelle la fonction somme sans passer de paramètres

```
console.log(somme());
```

JavaScript

Et si on appelle la fonction somme sans passer de paramètres

```
console.log(somme());
```

Le resultat

```
NaN
```

JavaScript

Et si on appelle la fonction somme sans passer de paramètres

```
console.log(somme());
```

Le resultat

```
NaN
```


On peut affecter une valeur par défaut aux paramètres

```
function somme (a = 0, b = 0) {
  return a + b;
}
```

JavaScript

Et si on veut que la fonction somme retourne la somme quel que soit le nombre de paramètres

```
function somme () {
  result = 0;
  for (var i = 0; i < arguments.length ; i++) {
    result += arguments[i];
  }
  return result;
}
```

JavaScript

Et si on veut que la fonction somme retourne la somme quel que soit le nombre de paramètres

```
function somme () {
  result = 0;
  for (var i = 0; i < arguments.length ; i++) {
    result += arguments[i];
  }
  return result;
}
```


Tous ces appels sont corrects

```
console.log(somme(2, 3, 5));
console.log(somme(2, 3));
console.log(somme(2, 3, 5, 7));
console.log(somme(2));
```

JavaScript

Exercice

Écrire une fonction **JavaScript** qui permet de déterminer si le premier paramètre est divisible par tous les autres paramètres. Le nombre de paramètres est variable et la fonction retourne un booléen.

Exemple

```
console.log(estDivisiblePar(10, 2, 3, 5));
// affiche false

console.log(estDivisiblePar(10, 2,  5));
// affiche true
```

JavaScript

Remarque

Les paramètres optionnels n'existent pas en **JavaScript** mais existent en **TypeScript**.

JavaScript

Fonctions fléchée

- fonction anonyme
- pouvant prendre des paramètres
- utilisant => pour séparer la partie paramètres et la partie traitement

JavaScript

Syntaxe

```
(arguments) => {
  traitement
}
```

JavaScript

Syntaxe

```
(arguments) => {
  traitement
}
```

Exemple : une seule instruction $\Rightarrow$ pas besoin de `return` ni { }

```
var somme = (a, b) =>  a + b
```

JavaScript

Syntaxe

```
(arguments) => {
  traitement
}
```

Exemple : une seule instruction ⇒ pas besoin de `return` ni { }

```
var somme = (a, b) =>  a + b
```

Appel d'une fonction fléchée

```
var resultat = somme (1, 3)

console.log(resultat)
// affiche 4
```

JavaScript

Les parenthèses peuvent être supprimées pour les fonctions mono-paramètre

```
var carre = a => a * a
```

JavaScript

Les parenthèses peuvent être supprimées pour les fonctions mono-paramètre

```
var carre = a => a * a
```

L'appel ne change pas

```
var resultat = carre (2)

console.log(resultat)
// affiche 4
```

JavaScript

Callback

- traduite souvent en fonction de rappel ou fonction de retour
- fonction appelée comme un paramètre d'une deuxième fonction
- très utilisée en **JavaScript** (**jQuery** et **nodeJS**) avec les fonctions asynchrones

JavaScript

Callback

- traduite souvent en fonction de rappel ou fonction de retour
- fonction appelée comme un paramètre d'une deuxième fonction
- très utilisée en **JavaScript** (**jQuery** et **nodeJS**) avec les fonctions asynchrones

Considérons les deux fonctions suivantes

```
function somme(a, b) {
  return a + b;
}

function produit(a, b) {
  return a * b;
}
```

JavaScript

Utilisons les fonctions précédentes comme callback d'une fonction `operation()`

```
function operation(a, b, fonction) {
  console.log (fonction(a, b));
}

// appeler la fonction opération
operation (3, 5, somme);
// affiche 8
```

JavaScript

Exercice

- Modifier la fonction opération pour qu'elle puisse accepter deux fonctions callback ensuite retourner le résultat de la composition des deux.
- Par exemple :
  `operation (2, 3 , 6, somme, produit)` retourne 30 = (2 + 3) * 6

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

Déclarons une fonction qui affiche bonjour

```
function direBonjour() {
  alert("Bonjour");
}
```

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

Déclarons une fonction qui affiche bonjour

```
function direBonjour() {
  alert("Bonjour");
}
```

Utilisons cette fonction comme callback dans `setTimeout`

```
function direBonjourApresXSecondes(x) {
  setTimeout(direBonjour, x * 1000);
}
```

La fonction `setTimeout` accepte deux paramètres

- le premier : une fonction callback
- le deuxième : une durée (en millisecondes) qui précède l'exécution de la fonction callback

Déclarons une fonction qui affiche bonjour

```
function direBonjour() {
  alert("Bonjour");
}
```

Utilisons cette fonction comme callback dans `setTimeout`

```
function direBonjourApresXSecondes(x) {
  setTimeout(direBonjour, x * 1000);
}
```

Appelons `direBonjourApresXSecondes` et bonjour sera affiché après

```
direBonjourApresXSecondes(5);
```

JavaScript

Il est aussi possible d'utiliser une fonction anonyme

```
function direBonjourApresXSecondes(x) {
  setTimeout(function() {
    alert("Bonjour");
  },
  x * 1000);
}
direBonjourApresXSecondes(5);
```

JavaScript

Il est aussi possible d'utiliser une fonction anonyme

```
function direBonjourApresXSecondes(x) {
  setTimeout(function() {
    alert("Bonjour");
  },
  x * 1000);
}
direBonjourApresXSecondes(5);
```

`setTimeout()` ne permet pas de répéter l'affichage toutes les `x` secondes.

JavaScript

Il est aussi d'annuler l'exécution de la fonction callback de `setTimeout` avec `clearTimeout` si cette dernière est appelée avant la fin de la durée de `setTimeout`

```
var timeout;
function direBonjourApresXSecondes(x) {
  timeout = setTimeout(function(){
    alert("Bonjour");
  },
  x * 1000);
}

direBonjourApresXSecondes(5);

clearTimeout(timeout);
// Bonjour ne sera jamais affiché
```

JavaScript

Pour afficher un message toutes les `X` secondes, on peut utiliser `setInterval` qui a la même signature que `setTimeout`

```
function direBonjourToutesXSecondes(x) {
  setInterval(direBonjour, x * 1000);
}

function direBonjour() {
  alert("Bonjour");
}

direBonjourToutesXSecondes(5);
```

JavaScript

Pour afficher un message toutes les `X` secondes, on peut utiliser `setInterval` qui a la même signature que `setTimeout`

```
function direBonjourToutesXSecondes(x) {
  setInterval(direBonjour, x * 1000);
}

function direBonjour() {
  alert("Bonjour");
}

direBonjourToutesXSecondes(5);
```

Comme `clearTimeout()`, il existe une fonction `clearInterval()`.

JavaScript

Remarque

Si la fonction callback nécessite de paramètres, alors il est possible de les préciser après la durée dans `setInterval` et `setTimeout`.

JavaScript

Méthodes sur les tableaux utilisant fonctions fléchées et callback (ES5)

- `forEach()` : pour parcourir un tableau
- `map()` : pour appliquer une fonction sur les éléments d'un tableau
- `filter()` : pour filtrer les éléments d'un tableau selon un critère défini sous forme d'une fonction anonyme ou fléchée
- `reduce()` : pour réduire tous les éléments d'un tableau en un seul selon une règle définie dans une fonction anonyme ou fléchée
- `...`

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab = [2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a, b){
            console.log(a - 3);
        }
    );
// affiche 3 et 5
```

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab = [2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a, b){
            console.log(a - 3);
        }
    );
// affiche 3 et 5
```

On peut permuter `map` et `filter` selon le besoin.

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab = [2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a, b){
            console.log(a - 3);
        }
    );
// affiche 3 et 5
```

On peut permuter `map` et `filter` selon le besoin.
On peut aussi remplacer les fonctions fléchées par des fonctions anonymes

JavaScript

Exemple avec `reduce` : permet de réduire les éléments d'un tableau en une seule valeur

```
var tab =[2, 3, 5];
var somme = tab.map(x => x + 3)
     .filter(x => x > 5)
     .reduce(function(sum, elem){
         return sum + elem;
     });

console.log(somme);
// affiche 14
```

JavaScript

Tri lexicographique (ordre alphabétique pour des chaînes de caractères)

```
let fruits = ['Banane', 'Orange', 'Pomme', 'Mangue'];
fruits.sort();

console.log(fruits);
// affiche  ['Banane', 'Mangue', 'Orange', 'Pomme']
```

JavaScript

Tri numérique (croissant)

```
let nombres = [10, 5, 2, 8, 1];
nombres.sort((a, b) => a - b);

console.log(nombres);
// affiche [1, 2, 5, 8, 10]
```

JavaScript

Tri numérique (croissant)

```
let nombres = [10, 5, 2, 8, 1];
nombres.sort((a, b) => a - b);

console.log(nombres);
// affiche [1, 2, 5, 8, 10]
```

Tri numérique (décroissant)

```
let nombres = [10, 5, 2, 8, 1];
nombres.sort((a, b) => b - a);

console.log(nombres);
// affiche [10, 8, 5, 2, 1]
```

JavaScript

Explication

La fonction de comparaison qu'on passe à `sort()` doit renvoyer une valeur qui indique l'ordre relatif des éléments comparés :

- une valeur inférieure à 0 $\Rightarrow$ le premier élément est rangé avant le deuxième.
- une valeur supérieure à 0 $\Rightarrow$ le deuxième élément est rangé avant le premier
- 0 $\Rightarrow$ l'ordre des éléments est inchangé.

JavaScript

Considérons les deux fonctions suivantes

```
var i = 0;
function f() {
  i++;
}
f();
console.log(i); // affiche 1
function g() {
  var j = 5;
}
g();
console.log(j); // génère une erreur
```

JavaScript

Considérons les deux fonctions suivantes

```
var i = 0;
function f() {
  i++;
}
f();
console.log(i); // affiche 1
function g() {
  var j = 5;
}
g();
console.log(j); // génère une erreur
```

Remarque

Une variable déclarée dans une fonction avec le mot-clé `var` n'a pas une portée globale. Donc, elle est locale.

JavaScript

La closure : déclarer une variable locale dans une fonction qui existe en globale

```
var i = 0;
function f() {
var i = 5;
i++;
console.log(i);
}
f(); // affiche 6
console.log(i); // affiche 0
```

JavaScript

La closure : déclarer une variable locale dans une fonction qui existe en globale

```
var i = 0;
function f() {
var i = 5;
i++;
console.log(i);
}
f(); // affiche 6
console.log(i); // affiche 0
```

Remarque

La variable `i` affichée à la dernière instruction est la variable déclarée à la première ligne.

JavaScript

Comment sauvegarder toutes ces donnees dans une seule variable ?

nom	wick
prénom	john
⋮	⋮

JavaScript

Comment sauvegarder toutes ces donnees dans une seule variable ?

nom	wick
prénom	john
⋮	⋮

Comment faire ?

- Les tableaux indexés ne le permettent pas
- Les tableaux associatifs ou les objets

JavaScript

Objet ?

un ensemble de

- propriétés (attributs, variables, champs) : clé + valeur[s]
- méthodes : fonctions

Création d'un objet

```
var obj = {
  nom: "wick",
  prenom: "john"
};
```

JavaScript

Création d'un objet

```
var obj = {
  nom: "wick",
  prenom: "john"
};
```


Accès à une propriété de l'objet

```
console.log(obj.nom);
console.log(obj["prenom"]);
```

JavaScript

Un objet est non-itérable avec `for ... of`

```
for(var elt of obj) {
    console.log(elt);
}
```

JavaScript

Un objet est non-itérable avec `for ... of`

```
for(var elt of obj) {
    console.log(elt);
}
```

Il est itérable avec `for ... in`

```
for(var key in obj) {
    console.log(key + " : " + obj[key]);
}
```

JavaScript

Un objet est non-itérable avec `for ... of`

```
for(var elt of obj) {
    console.log(elt);
}
```

Il est itérable avec `for ... in`

```
for(var key in obj) {
    console.log(key + " : " + obj[key]);
}
```

Un objet peut aussi être créé ainsi

```
var obj = new Object();
obj.nom = "wick";
obj.prenom = "john";
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un ⇒ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un ⇒ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

Ajouter un nouvel attribut à un objet existant

```
obj2.age = 35;
```

JavaScript

Pour cloner un objet

```
function clone(obj){
  var obj2 = {};
  for (key in obj)
    obj2[key] = obj[key];
  return obj2;
}

var obj = {nom: "wick", prenom: "john"};
var obj2 = clone(obj);
obj2.nom = "abruzzi";

console.log(obj);
// affiche wick

console.log(obj2);
// affiche abruzzi
```

JavaScript

Ou tout simplement

```
var obj2 = Object.assign({}, obj);
console.log(obj);
console.log(obj2);
```

JavaScript

Ou tout simplement

```
var obj2 = Object.assign({}, obj);
console.log(obj);
console.log(obj2);
```

La méthode `Object.create()` permet de copier un objet.

JavaScript

Ou tout simplement

```
var obj2 = Object.assign({}, obj);
console.log(obj);
console.log(obj2);
```

La méthode `Object.create()` permet de copier un objet.

Depuis ES6, nous pouvons utiliser l'écriture suivante

```
var obj2 = { ...obj };
console.log(obj);
console.log(obj2);
```

JavaScript

Pour récupérer l'ensemble des valeurs d'un objet dans un tableau

```
var obj = { nom: "wick", prenom: "john" };
var values = Object.values(obj);

for (value of values)
  console.log(value);

// affiche wick ensuite john
```

JavaScript

Pour récupérer l'ensemble des clés d'un objet dans un tableau

```
var keys = Object.keys(obj);

for (key of keys)
  console.log(key + " : " + obj[key]);

// affiche nom : wick
// prenom : john
```

JavaScript

Considérons toujours l'objet `obj`

```
const obj = { nom: 'wick', prenom: 'john' };
```

JavaScript

Considérons toujours l'objet `obj`

```
const obj = { nom: 'wick', prenom: 'john' };
```

Déclarons une variable `name` qui reçoit sa valeur d'un attribut d'`obj`

```
var name = obj.nom;
```

JavaScript

Considérons toujours l'objet `obj`

```
const obj = { nom: 'wick', prenom: 'john' };
```

Déclarons une variable `name` qui reçoit sa valeur d'un attribut d'`obj`

```
var name = obj.nom;
```

Modifions la valeur d'un n'affecte pas l'autre

```
name = "travolta";
console.log(obj.nom);
// affiche wick

obj.nom = "abruzzi";
console.log(name);
// affiche travolta
```

JavaScript

Pour transformer un objet en chaîne de caractère

```
var perso = { nom: 'wick', prenom: 'john' };
var str = JSON.stringify(perso);
console.log(str);
// affiche {"nom":"wick","prenom":"john"}
```

JavaScript

Pour transformer un objet en chaîne de caractère

```
var perso = { nom: 'wick', prenom: 'john' };
var str = JSON.stringify(perso);
console.log(str);
// affiche {"nom":"wick","prenom":"john"}
```

Pour transformer une chaîne de caractère en objet

```
var p = JSON.parse(str);
console.log(p.nom + " " + p.prenom);
// affiche wick john
```

Considérons l'objet suivant

```
let marquePrix = {
    peugeot: 15000,
    ford: 30000,
    citroen: 20000,
    jeep: 50000,
    fiat: 10000
}
```

Considérons l'objet suivant

```
let marquePrix = {
    peugeot: 15000,
    ford: 30000,
    citroen: 20000,
    jeep: 50000,
    fiat: 10000
}
```

Exercice 1

Écrire un script JS qui demande à l'utilisateur de saisir une marque puis il affiche son prix. Si la marque n'est pas dans l'objet, on affiche "marque non disponible pour le moment.

Considérons l'objet suivant

```
let marquePrix = {
    peugeot: 15000,
    ford: 30000,
    citroen: 20000,
    jeep: 50000,
    fiat: 10000
}
```

Exercice 1

Écrire un script JS qui demande à l'utilisateur de saisir une marque puis il affiche son prix. Si la marque n'est pas dans l'objet, on affiche "marque non disponible pour le moment.

Exercice 2

Écrire un script JS qui demande à l'utilisateur de saisir un budget puis il affiche toutes les marques dont le prix est inférieur ou égale à la valeur saisie.

JavaScript

Considérons la variable `key` suivante

```
const key = 'nom';
```

JavaScript

Considérons la variable `key` suivante

```
const key = 'nom';
```

Pour créer un objet avec une propriété calculée (définie dans une autre variable)

```
const objet = {
  prenom: 'john',
  [key]: 'wick'
};
```

JavaScript

Considérons la variable `key` suivante

```
const key = 'nom';
```

Pour créer un objet avec une propriété calculée (définie dans une autre variable)

```
const objet = {
  prenom: 'john',
  [key]: 'wick'
};
```

L'objet final ressemble donc à ceci

```
{
  prenom: 'john',
  nom: 'wick'
}
```

JavaScript

Nous pouvons également utiliser une expression plus complexe dans les crochets

```
const objet = {
  prenom: 'john',
  [key + 'DeFamille']: 'wick'
};
```

JavaScript

Nous pouvons également utiliser une expression plus complexe dans les crochets

```
const objet = {
  prenom: 'john',
  [key + 'DeFamille']: 'wick'
};
```

L'objet final ressemble à

```
{
  prenom: 'john',
  nomDeFamille: 'wick'
}
```

JavaScript

Il est possible de construire un objet à partir d'un autre

```
const objet = {
  prenom: 'john',
  [key + 'DeFamille']: 'wick'
}
const objet2 = {
  ...objet,
  prenom: 'jack',
  age: 45
}
console.log(objet2);
```

JavaScript

Il est possible de construire un objet à partir d'un autre

```
const objet = {
  prenom: 'john',
  [key + 'DeFamille']: 'wick'
}
const objet2 = {
  ...objet,
  prenom: 'jack',
  age: 45
}
console.log(objet2);
```


L'objet final ressemble à

```
{
  prenom: 'jack',
  nomDeFamille: 'wick',
  age: 45
}
```

JavaScript

Un objet avec propriétés et méthodes

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour");
  }
};
```

JavaScript

Un objet avec propriétés et méthodes

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour");
  }
};
```


Pour appeler la méthode `direBonjour`

```
obj.direBonjour();
```

JavaScript

Un objet avec propriétés et méthodes

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour");
  }
};
```

Pour appeler la méthode `direBonjour`

```
obj.direBonjour();
```

Comment afficher le nom dans la méthode ?

JavaScript

Ceci génère une erreur

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour " + nom);
  }
};
obj.direBonjour();
```

JavaScript

Ceci génère une erreur

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour " + nom);
  }
};
obj.direBonjour();
```

Il faut utiliser l'opérateur `this`

```
var obj = {
  nom: "wick",
  prenom: "john",
  direBonjour: function (){
    console.log("bonjour " + this.nom);
  }
};
obj.direBonjour();
```

JavaScript

Constructeurs ?

- Besoin d'un moule pour créer des objets (comme les classes en **Java**, **C#**, **PHP**...)
- Tous les objets **JavaScript** sont de type `Object`
- Il est possible de créer ou de cloner un objet a‘ partir d'un autre en utilisant des méthodes d'`Object`
- Et si on veut créer un modèle d'objet (comme la classe), on peut utiliser les constructeurs

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom, prenom){
  this.nom = nom;
  this.prenom = prenom;
}
```

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom, prenom){
  this.nom = nom;
  this.prenom = prenom;
}
```

Explication

- Un constructeur d'objet permettant d'initialiser la valeur de deux attributs `nom` et `prenom`
- L'appel de ce constructeur avec l'opérateur `new` permet de créer plusieurs objets ayant tous les attributs `nom` et `prenom`
- C'est comme une classe dans un LOO à classes

JavaScript

Créer un objet en utilisant l'opérateur `new`

```
perso = new Personne("wick","john");
```

JavaScript

Créer un objet en utilisant l'opérateur `new`

```
perso = new Personne("wick","john");
```

Afficher les valeurs d'un objet

```
console.log(perso);
```

JavaScript

Créer un objet en utilisant l'opérateur `new`

```
perso = new Personne("wick","john");
```

Afficher les valeurs d'un objet

```
console.log(perso);
```


Pour vérifier qu'il s'agit bien d'un objet

```
console.log(typeof perso);
```

JavaScript

Déclarer un constructeur contenant une méthode

```
var Personne = function(nom,prenom){
  this.nom = nom;
  this.prenom = prenom;
  this.direBonjour = function (){
    console.log("bonjour " + this.nom);
  }
}
```

JavaScript

Déclarer un constructeur contenant une méthode

```
var Personne = function(nom,prenom){
  this.nom = nom;
  this.prenom = prenom;
  this.direBonjour = function (){
    console.log("bonjour " + this.nom);
  }
}
```


Pour appeler cette méthode

```
var perso = new Personne("wick","john");
perso.direBonjour();
```

JavaScript

Une fois le constructeur déclaré, impossible de lui ajouter de nouvel attribut de cette manière

```
Personne.age = 0;
```

JavaScript

Une fois le constructeur déclaré, impossible de lui ajouter de nouvel attribut de cette manière

```
Personne.age = 0;
```

Pour cela, il faut utiliser le prototype

```
Personne.prototype.age = 0;
```

JavaScript

Pareillement pour les méthodes

```
Personne.prototype.getAge = function(){
  return this.age;
}

Personne.prototype.setAge = function(age){
  this.age = age;
}
```

JavaScript

Pareillement pour les méthodes

```
Personne.prototype.getAge = function(){
  return this.age;
}

Personne.prototype.setAge = function(age){
  this.age = age;
}
```

Appeler les méthodes ajoutées

```
perso.setAge(45);
console.log(perso.getAge());
```

JavaScript

Pareillement pour les méthodes

```
Personne.prototype.getAge = function(){
  return this.age;
}

Personne.prototype.setAge = function(age){
  this.age = age;
}
```

Appeler les méthodes ajoutées

```
perso.setAge(45);
console.log(perso.getAge());
```

Remarque

Ne jamais modifier le prototype d'un objet prédéfini.

JavaScript

Que désigne `this` en **JavaScript** ?

- Dans une méthode d'objet, il fait référence à l'objet auquel la méthode appartient.
- Dans une fonction, il fait généralement référence à l'objet global `window` dans un navigateur, ou `global` dans **Node.js**.
- Dans un constructeur, il fait référence à l'instance nouvellement créée.

JavaScript

Que désigne `this` en **JavaScript** ?

- Dans une méthode d'objet, il fait référence à l'objet auquel la méthode appartient.
- Dans une fonction, il fait généralement référence à l'objet global `window` dans un navigateur, ou `global` dans **Node.js**.
- Dans un constructeur, il fait référence à l'instance nouvellement créée.

Remarque

Les fonctions fléchées (**arrow functions**) ne créent pas leur propre contexte pour `this`. Elles héritent de la valeur de `this` du contexte lexical (celui dans lequel elles ont été définies).

JavaScript

Décomposition

- aussi appelée **spread operator**, ou opérateur de propagation
- consiste à ”étendre” un tableau ou un objet dans un autre tableau ou objet
- utilise l’opérateur . . .
- introduite dans
  - **ES6** en 2015 pour les tableaux
  - **ES 2018** pour les objets

JavaScript

Exemple avec les tableaux

```
const tab1 = [1, 2, 3];
const tab2 = [0, ...tab1, 4, 5];

console.log(tab2);
// affiche [0, 1, 2, 3, 4, 5]
```

JavaScript

Exemple avec les tableaux

```
const tab1 = [1, 2, 3];
const tab2 = [0, ...tab1, 4, 5];

console.log(tab2);
// affiche [0, 1, 2, 3, 4, 5]
```

Exemple avec les objets

```
const obj1 = { nom: "Wick", prenom: "John" };
const obj2 = { ...obj1, age: 45 };

console.log(obj2);
// affiche { nom: 'Wick', prenom: 'John', age: 45 }
```

JavaScript

Déstructuration

- consiste à extraire des propriétés d'un objet ou des éléments d'un tableau dans des variables distinctes
- utilise l'opérateur { } pour les objets et [ ] pour les tableaux
- introduite dans **ES6** en 2015

JavaScript

Exemple avec les tableaux

```
const tab1 = [1, 2, 3];
const [first, second] = tab1;

console.log(first);
// affiche 1

console.log(second);
// affiche 2
```

JavaScript

Exemple avec les tableaux

```
const tab1 = [1, 2, 3];
const [first, second] = tab1;

console.log(first);
// affiche 1

console.log(second);
// affiche 2
```

Exemple avec les objets

```
const obj2 = { nom: "Wick", prenom: "John", age: 45 };
const { nom, age } = obj2;

console.log(nom);
// affiche Wick

console.log(age);
// affiche 45
```

JavaScript

Objets prédéfinis

- `Math` : contenant de méthodes permettant de réaliser des opérations mathématiques sur les nombres
- `Date` : permet de manipuler des dates
- `...`

JavaScript

Exemple de méthodes de l'objet `Math`

```
Math.round(2.1); // retourne 2
Math.round(2.9); // retourne 3
Math.floor(2.1); // retourne 2
Math.floor(2.9); // retourne 2
Math.ceil(2.1); // retourne 3
Math.ceil(2.9); // retourne 3
Math.pow(2, 3); // retourne 8
Math.sqrt(4); // retourne 2
Math.abs(-2); // retourne 2
Math.min(0, 1, 4, 2, -4, -5); // retourne -5
Math.max(0, 1, 4, 2, -4, -5); // retourne 4
Math.random(); // retourne un nombre aléatoire
Math.floor(Math.random() * 10);
// retourne un entier compris entre 0 et 9
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date);
// affiche la date complète du jour
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date);
// affiche la date complète du jour
```

Créer une date à partir de valeurs passées en paramètre (les mois sont codés de 0 à 11)

```
var date = new Date(1985,7,30,5,50,0,0);
console.log(date);
// affiche Fri Aug 30 1985 05:50:00 GMT+0200
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date);
// affiche la date complète du jour
```

Créer une date à partir de valeurs passées en paramètre (les mois sont codés de 0 à 11)

```
var date = new Date(1985,7,30,5,50,0,0);
console.log(date);
// affiche Fri Aug 30 1985 05:50:00 GMT+0200
```

Créer une date à partir d'un nombre de millisecondes

```
var date = new Date(100000000000);
console.log(date);
// affiche Sat Mar 03 1973 10:46:40 GMT+0100
```

JavaScript

Exemple de méthodes pour les dates

- `getFullYear()` : retourne l'année
- `getMonth()` : retourne le mois [0-11]
- `getDate()` : retourne le jour [1-31]
- `getHours()` : retourne l'heure [0-23]
- `getMinutes()` : retourne les minutes [0-59]
- `getSeconds()` : retourne les secondes [0-59]
- `getMilliseconds()` : retourne les millisecondes [0-999]
- `getTime()` : retourne le **Timestamp** (Depuis **ES5**, on peut utiliser `Date.now()`)

JavaScript

Exemple de méthodes pour les dates

- `getFullYear()` : retourne l'année
- `getMonth()` : retourne le mois [0-11]
- `getDate()` : retourne le jour [1-31]
- `getHours()` : retourne l'heure [0-23]
- `getMinutes()` : retourne les minutes [0-59]
- `getSeconds()` : retourne les secondes [0-59]
- `getMilliseconds()` : retourne les millisecondes [0-999]
- `getTime()` : retourne le **Timestamp** (Depuis **ES5**, on peut utiliser `Date.now()`)

On peut aussi modifier les attributs de l'objet Date en utilisant les set.

JavaScript

Timestamp

- Nombre de secondes écoulés depuis le `01/01/1970`
- `01/01/1970` : date de naissance du temps dans les systèmes informatiques **UNIX**
- Utilisé par la plupart des systèmes d'exploitation modernes, les bases de données et les protocoles de communication...

JavaScript

Timestamp

- Nombre de secondes écoulés depuis le `01/01/1970`
- `01/01/1970` : date de naissance du temps dans les systèmes informatiques **UNIX**
- Utilisé par la plupart des systèmes d'exploitation modernes, les bases de données et les protocoles de communication...

Limite et solution

- Ce choix comporte des limitations, comme le problème de l'an `2038` pour les systèmes qui stockent le temps comme un entier signé de 32 bits.
- À `03:14:07 UTC le 19 janvier 2038`, ces systèmes rencontreront un débordement qui fera revenir leur compteur à une date en `1901`.
- Des solutions sont en cours de mise en œuvre : représentations de temps sur 64 bits.

JavaScript

Pour comparer les dates, on peut utiliser les opérateurs de comparaison

```
const date1 = new Date('2024-02-28');
const date2 = new Date('2024-03-01');

if (date1 < date2) {
    console.log("date1 est antérieure à date2");
} else if (date1 > date2) {
    console.log("date1 est postérieure à date2");
} else {
    console.log("date1 et date2 sont égales");
}
```

JavaScript

Pour formater une date dans un fuseau horaire particulier

```
const date = new Date();

const options = {
  timeZone: 'America/New_York',
  year: 'numeric', month: 'numeric', day: 'numeric',
  hour: 'numeric', minute: 'numeric', second: 'numeric'
};

const formatter = new Intl.DateTimeFormat([], options);
console.log(formatter.format(date));
```

JavaScript

Pour formater une date dans un fuseau horaire particulier

```
const date = new Date();

const options = {
  timeZone: 'America/New_York',
  year: 'numeric', month: 'numeric', day: 'numeric',
  hour: 'numeric', minute: 'numeric', second: 'numeric'
};

const formatter = new Intl.DateTimeFormat([], options);
console.log(formatter.format(date));
```


Remarque

En JavaScript, il n'existe pas de support direct pour spécifier un fuseau horaire particulier lors de la création d'un objet Date.

JavaScript

Valeurs acceptées par les attributs d'`options`

- `year` : `numeric` et `2-digit`
- `month` : `numeric`, `2-digit`, `long` et `short`
- `day` : `numeric` et `2-digit`
- `weekday` : `long` et `short`
- `hour` : `numeric` et `2-digit`
- `minute` : `numeric` et `2-digit`
- `second` : `numeric` et `2-digit`
- `timeZoneName` : `long` et `short`