Intro Programmation Web

Javascript - intro

Pierre Blarre - Florian Rodriguez

Plan

- Qu'est-ce que Javascript ?
- Historique
- Syntaxe
- Variables
- Variable et types
- Types de données
- Opérateurs et structures de contrôle
- Fonctions
- Objets
- Tableaux
- DOM
- Événements

Qu'est-ce que Javascript ?

- Langage de programmation de scripts
- Utilisé pour rendre les pages web interactives
- Exécuté côté client
- Créé en 1995 par Brendan Eich
- Standardisé par ECMA International
- ECMAScript est le standard
- Javascript est l'implémentation de référence

Historique

- 1995 : Création de Javascript par Brendan Eich
- 1996 : Javascript est standardisé par ECMA International
- 1997 : ECMAScript 1
- 1999 : ECMAScript 3
- 2009 : ECMAScript 5
- 2015 : ECMAScript 6 (ES6)
- 2016 : ECMAScript 7 (ES7)
- 2017 : ECMAScript 8 (ES8)
- 2018 : ECMAScript 9 (ES9)
- 2019 : ECMAScript 10 (ES10)
- 2020 : ECMAScript 11 (ES11)
- 2021 : ECMAScript 12 (ES12)
- 2022 : ECMAScript 13 (ES13)
- 2023 : ECMAScript 14 (ES14)
- 2024 : ECMAScript 15 (ES15)

Syntaxe

- Sensible à la casse
- Les instructions sont séparées par des points-virgules `;`
- Les blocs de code sont délimités par des accolades `{}`
- Les commentaires sont délimités par `//` ou `/* */`
- Les variables sont déclarées avec `var`, `let` ou `const`
- Les chaînes de caractères sont délimitées par des guillemets simples `'` ou doubles `"`
- Les nombres peuvent être entiers ou décimaux
- Les tableaux sont déclarés avec des crochets `[]`
- Les objets sont déclarés avec des accolades `{}`
- Les fonctions sont déclarées avec le mot-clé `function`
- Les opérateurs sont utilisés pour effectuer des opérations sur des variables et des valeurs

Association document HTML - JS

- Le code Javascript peut être inclus dans une page HTML de différentes manières
  - Dans la balise `<script>`
  - Dans un fichier externe
  - Dans un événement HTML
  - Dans un lien HTML

Dans la balise `<script>`

- Le code Javascript peut être inclus dans la balise `<script>`

```
<script>
  alert("Hello World");
</script>
```

Copy

- Le code Javascript est exécuté lorsque la page est chargée

Dans un fichier externe

- Le code Javascript peut être inclus dans un fichier externe

```
<script src="script.js"></script>
```

Copy

- Le fichier `script.js` contient le code Javascript
- C'est la méthode recommandée pour inclure du code Javascript:
  - Le code Javascript est séparé du code HTML
  - Le code Javascript peut être réutilisé
  - Le code Javascript est plus facile à maintenir
  - Le code Javascript est mis en cache par le navigateur

Dans un événement HTML

- Le code Javascript peut être inclus dans un événement HTML

```
<button onclick="alert('Hello World')">Click me</button>
```

Copy

Click me

- Le code Javascript est exécuté lorsque le bouton est cliqué

Dans un lien HTML

- Le code Javascript peut être inclus dans un lien HTML

```
<a href="javascript:alert('Hello World')">Click me</a>
```

Copy

Click me

- Le code Javascript est exécuté lorsque le lien est cliqué

Variables

- Les variable sont déclarées avec les mots-clés `var`, `let` ou `const`
- Avant ES6, on utilisait `var` pour déclarer des variables
- Depuis ES6, on peut utiliser `let` et `const` pour déclarer des variables
- `let` permet de déclarer des variables dont la valeur peut être modifiée
- `const` permet de déclarer des variables dont la valeur ne peut pas être modifiée

Var

- `var` fonctionne comme let mais avec des différences (ceci n'est pas forcément clair pour le moment, c'est normal):
  - `var` peut être redéclaré
  - `var` a une portée de fonction alors que `let` a une portée de bloc
- **Règle pour le moment**:
  Utilisez `let` quand vous ne pouvez pas utiliser `const`. N'utilisez jamais `var`.

Variable et types

- Javascript est un langage de programmation dynamiquement typé
  - Le type des variables n'est pas défini lors de la déclaration
  - Le type des variables est déterminé lors de l'exécution du programme

```
let x = 5; // x est un nombre
typeof x; // number
x = "Hello"; // x est une chaîne de caractères
typeof x; // string
```

Copy

```
"string"
```

- Javascript est un langage de programmation faiblement typé
  - Les variables peuvent changer de type
  - Les opérations peuvent être effectuées sur des variables de types différents

```
let x = 5;
let y = "5";
x + y; // "55"
```

Copy

```
"55"
```

Types de données

- Les types de données sont divisés en deux catégories
  - Types de données primitifs
    - Nombre
    - Chaîne de caractères
    - Booléen
    - Undefined
    - Null
  - Types de données par référence (objects)
    - Objet
    - Tableau

Les primitifs

- Les types de données primitifs sont stockés directement dans la variable
- **Nombre**
  - Les nombres sont des valeurs numériques
  - Les nombres peuvent être entiers ou décimaux
  - Les nombres peuvent être positifs ou négatifs
  - Les nombres peuvent être écrits avec ou sans décimales
  - Valeurs spéciales : `Infinity`, `-Infinity`, `NaN` (Not a Number)
- **Chaîne de caractères**
- **undefined**
  - Une variable non initialisée a la valeur `undefined`
- **null**
  - La valeur `null` signifie l'absence de valeur. pointe vers un objet inexistant.

```
let x;
let y = null;
typeof x; // undefined
typeof y; // object
```

Copy

```
"object"
```

- Booléen

Objets

- Les objets sont des variables qui peuvent contenir de nombreuses valeurs
- Les objets sont déclarés avec des accolades {}
- Les objets sont des variables de type référence
- Les objets peuvent contenir des propriétés et des méthodes

Copy

```
let person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
person.fullName(); // "John Doe"
```

```
"John Doe"
```

Objets comparaison

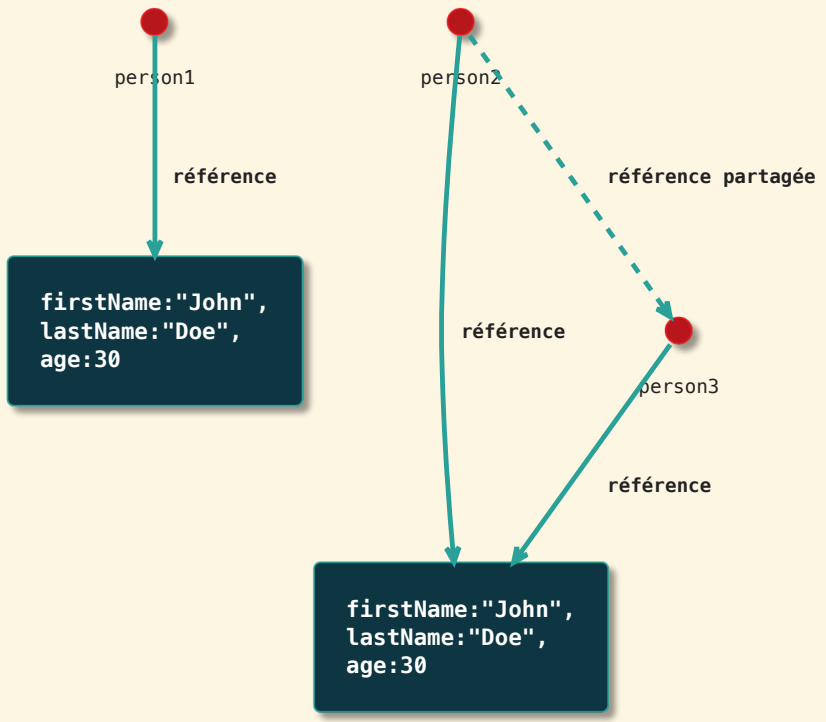
- Les objets `person1` et `person2` identiques ?

```
let person1 = {
  firstName: "John",
  lastName: "Doe",
  age: 30
};
let person2 = {
  firstName: "John",
  lastName: "Doe",
  age: 30
};
let person3 = person1;

person1 == person2; // false
person1 == person3; // true
```

Copy

```
true
```



Opérateurs et structures de contrôle

- Opérateurs et structures de contrôle standard (C++, Java, etc.)
- Opérateurs:
  - Arithmétiques
    - `+`, `-`, `*`, `/`, `%`, `++`, `--`
  - Comparaison
    - `=`, `!`, `<`, `>`, `< =`, `= >`
  - Logiques
    - `&&`, `||`, `!`
- Structures de contrôle
  - Conditionnelles
    - `if`, `else`, `else if`, `switch`
  - Boucles
    - `for`, `while`, `do-while`, `for-in`, `for-of`

Opérateurs de comparaison

- Deux opérateurs de comparaison: égalité faible et égalité stricte
  - == et != pour l'égalité faible
  - === et !== pour l'égalité stricte
  - L'égalité faible compare les valeurs
  - L'égalité stricte compare les valeurs et les types

```
5 == "5"; // true
5 === "5"; // false
5 != "5"; // false
5 !== "5"; // true

false == 0; // true
false === 0; // false
'' == 0; // true
```

Copy

```
true
```

Fonctions

- Déclaration de fonction:

```
function myFunction(x, y) {
  return x + y;
}
```

Copy

```
undefined
```

  - myFunction est le **nom** de la fonction
  - x et y sont les **paramètres** de la fonction
  - x + y est le **corps** de la fonction
  - Les types des paramètres et de la valeur de retour ne sont pas définis
  - Le retour de la fonction `return` est facultatif (si absent, la fonction retourne `undefined`)

- Appel de fonction:

```
myFunction(5, 3)
```

Copy

```
8
```

  - Appel de la fonction `myFunction` avec les **arguments** 5 et 3

Fonctions anonymes

- Fonctions anonymes:

```
let myFunction = function(x, y) {
  return x + y;
};

undefined
```

Copy

  - Fonction sans nom affectée à la variable `myFunction`

Fonctions fléchées

- Fonctions fléchées:

```
let myFunction = (x, y) => x + y;
```

Copy

```
undefined
```

  - Fonction fléchée avec les paramètres x et y et le corps x + y
  - Les fonctions fléchées sont plus courtes et plus lisibles que les fonctions anonymes

Callbacks

- Les fonctions peuvent être passées en tant que paramètres à d'autres fonctions

```
function myFunction(callback) {
  callback();
}
myFunction(function() {
  console.log("Hello");
});
```

Copy

```
"Hello"
undefined
```

  - La fonction `myFunction` prend une fonction `callback` en paramètre

Callback - autre exemple

- Exemple de callback avec setTimeout

```
setTimeout(function() {
  console.log("Hello");
}, 1000);
```

Copy

```
88
```

  - La fonction `setTimeout` appelle la fonction `callback` après 1000 millisecondes

Callback - autre exemple

Copy

```
function sum10(x) {return x + 10;}
function multiply10(x) {return x * 10;}

function processArray(arr,operation){
  for (let i=0; i<arr.length; i++) {
    arr[i] = operation(arr[i]);
  }
}

let arr = [1, 2, 3, 4, 5];
processArray(arr, sum10);
console.log(arr); // [11, 12, 13, 14, 15]

processArray(arr, multiply10);
console.log(arr); // [110, 120, 130, 140, 150]
```

```
Array [
  11,
  12,
  13,
  14,
  15,
]
Array [
  110,
  120,
  130,
  140,
  150,
]
```

Fonctions - arguments passés par valeurs

- Les arguments sont passés par valeurs

```
function myFunction(x) {
  x = 10;
}
let y = 5;
myFunction(y);
console.log(y); // 5
```

Copy

```
5
undefined
```

  - La variable `y` n'est pas modifiée par la fonction `myFunction`

Fonctions - arguments passés par référence

- Les objets sont passés par référence

```
function myFunction(obj) {
  obj.name = "John";
}
let person = {name: "Doe"};
myFunction(person);
console.log(person.name); // "John"
```

Copy

```
"John"
undefined
```

  - La propriété `name` de l'objet `person` est modifiée par la fonction `myFunction`

Fonctions - arguments par défaut

- Les fonctions peuvent avoir des arguments par défaut

```
function myFunction(x = 10) {
  return x;
}
myFunction(); // 10
myFunction(5); // 5
```

Copy

```
5
```

  - La valeur par défaut de x est 10

Fonctions - nombre d'arguments

- Les fonctions peuvent avoir un nombre variable d'arguments

```
function myFunction(...args) {
  return args;
}
myFunction(1, 2, 3); // [1, 2, 3]
```

Copy

```
Array [
  1,
  2,
  3,
]
```

  - L'opérateur de décomposition `...` permet de passer un nombre variable d'arguments

Fonctions - trop ou pas assez d'arguments

- Les fonctions peuvent être appelées avec trop ou pas assez d'arguments
  - Les arguments supplémentaires sont ignorés
  - Les arguments manquants sont définis à `undefined`

```
function myFunction(x, y) {
  return x + y;
}
myFunction(5); // NaN
```

Copy

```
NaN
```

  - La fonction `myFunction` retourne `NaN` car `y` est `undefined`

- Si deux fonctions ont le même nom, la dernière définition écrase les précédentes, même si elles ont des paramètres différents

```
function myFunction(x,y) {
  return 1;
}
function myFunction() {
  return 2;
}
myFunction('testx','testy'); // 2
```

Copy

```
2
```

  - La fonction `myFunction` retourne 2 car la dernière définition écrase la première

Classes d'objets

- Les classes sont des modèles pour créer des objets
- Les classes sont déclarées avec le mot-clé `class`
- Les classes ont des propriétés et des méthodes
- Les objets sont créés à partir de classes avec le mot-clé `new`
- Les objets sont des instances de classes

Déclaration de classe

Copy

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
  fullName() {
    return this.firstName + " " + this.lastName;
  }
}

const person = new Person("John", "Doe");
const person2 = new Person("Jane", "Doe");
```

```
undefined
```

«Classes» de base

- Les classes de base sont des classes prédéfinies
  - `Object` : classe de base pour tous les objets
  - `Array` : classe de base pour tous les tableaux
  - `Date` : classe de base pour toutes les dates
  - `String` : classe de base pour toutes les chaînes de caractères
  - `Number` : classe de base pour tous les nombres
  - `Boolean` : classe de base pour tous les booléens
  - `Function` : classe de base pour toutes les fonctions

Chaînes de caractères

- type objet prédéfini `String`

```
let chaine = "essai";
let chaine = 'essai';
let chaine = new String("essai");
```

Copy

```
SyntaxError: Identifier 'chaine' has already been declared
```

- Nompre de caractères

```
chaine.length; // 5
```

Copy

```
ReferenceError: chaine is not defined
```

- + Concatenation

```
chaine = chaine + " une autre chaine";
```

Copy

```
ReferenceError: chaine is not defined
```

- Nombreuses méthodes prédéfinies

Methodes prédéfinies

- `charAt(i)` : retourne le caractère à l'index spécifié
- `indexOf(ch, i)` : retourne l'index de la première occurrence d'une chaîne de caractères
- `lastIndexOf(ch, i)` : retourne l'index de la dernière occurrence d'une chaîne de caractères
- `split(ch)` : divise une chaîne de caractères en un tableau de sous-chaînes
- `match(exp)` : recherche les sous-chaînes correspondant à l'expression régulière `exp`
- `replace(exp, ch)` : remplace les sous-chaînes correspondant à l'expression régulière `exp` par `ch`
- `substring(i, j)` : extrait une sous-chaîne entre les index `i` et `j`
- `substr(i, n)` : extrait `n` caractères à partir de l'index `i`
- `toLowerCase()` : convertit une chaîne de caractères en minuscules
- `toUpperCase()` : convertit une chaîne de caractères en majuscules

Tableaux

- séquence d'éléments accessibles par un index
  - Javascript étant faiblement typé, un tableau peut contenir des éléments de types différents
- Création

```
let tab = [1, 2, 3, 4, 5, 'a', 'b', 'c'];
let tab = new Array(10); // alloue un tableau de 10 éléments
let tab = new Array(); // si pas de taille, ajustement dynamique
let tab = [];
```

Copy

```
SyntaxError: Identifier 'tab' has already been declared
```

- `[]` pour accéder aux éléments

```
tab[0] = 10;
tab[1] = 20;
tab[2] = 30;
```

Copy

```
ReferenceError: tab is not defined
```

- `length` pour obtenir la taille

```
tab.length; // 3
```

Copy

```
ReferenceError: tab is not defined
```

Methodes prédéfinies

- `push(e)` : ajoute un élément à la fin du tableau
- `pop()` : dépile et retourne le dernier élément du tableau
- `shift()` : dépile et retourne le premier élément du tableau
- `unshift(e)` : ajoute un élément au début du tableau
- `slice(i, j)` : extrait une sous-liste entre les index `i` et `j`

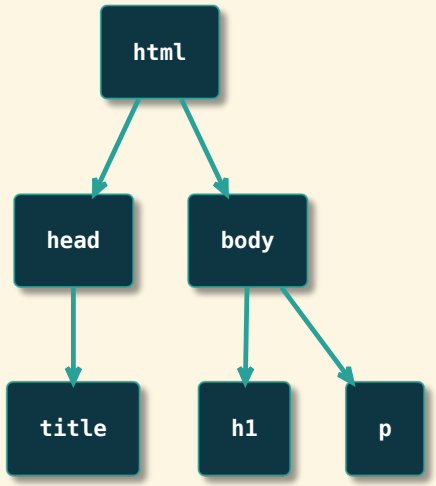
API DOM

- Le Document Object Model (DOM) est une interface de programmation pour les documents HTML et XML
- Le DOM représente la structure d'un document sous forme d'objets
- Le DOM est une interface de programmation standardisée par le W3C
- Le DOM est utilisé pour accéder, modifier, ajouter ou supprimer des éléments HTML

L'arbre DOM

- Quand une page est chargée, le navigateur crée un modèle de l'arbre DOM correspondant à la structure de la page

```
<html>
<head>
        <title>Titre de page</title>
</head>
<body>
        <h1>Niveau 1</h1>
        <p>Un premier paragraphe</p>
</body>
</html>
```



- les éléments dans le DOM sont des noeuds de type `Node`

Le type Document

- représente l'ensemble du document HTML/XML (contenu de la fenêtre)
- accessible par la variable globale `document`
- `document.documentElement` : retourne l'élément `html` du document
- `document.head` : retourne l'élément `head` du document
- `document.body` : retourne l'élément `body` du document
- d'autre propriétes intéressantes:
  - `document.title` : titre de la page
  - `document.URL` : URL de la page
  - `document.domain` : domaine de la page
  - `document.lastModified` : date de la dernière modification de la page
  - `document.cookie` : cookies de la page
  - `document.referrer` : URL de la page précédente

Accès aux éléments

- `document.getElementById(id)` : retourne l'élément avec l'ID spécifié
- `document.getElementsByTagName(name)` : retourne une liste d'éléments avec le nom de balise spécifié
- `document.getElementsByClassName(name)` : retourne une liste d'éléments avec la classe spécifiée
- `document.querySelector(selector)` : retourne le premier élément qui correspond au sélecteur CSS spécifié
- `document.querySelectorAll(selector)` : retourne une liste d'éléments qui correspondent au sélecteur CSS spécifié

Modification des éléments

- `element.innerHTML` : définit ou retourne le contenu HTML de l'élément
- `element.textContent` : définit ou retourne le contenu textuel de l'élément
- `element.style.property` : définit ou retourne la valeur d'une propriété CSS
- `element.setAttribute(attribute, value)` : définit ou modifie la valeur d'un attribut
- `element.appendChild(node)` : ajoute un nœud à la fin de la liste des enfants d'un nœud parent
- `element.removeChild(node)` : supprime un nœud enfant d'un nœud parent
- `element.replaceChild(newNode, oldNode)` : remplace un nœud enfant par un autre nœud
- `element.addEventListener(event, function)` : ajoute un écouteur d'événements à un élément

Création d'éléments

- `document.createElement(tagName)` : crée un élément HTML avec le nom de balise spécifié
- `document.createTextNode(text)` : crée un nœud de texte avec le texte spécifié
- `document.createAttribute(name)` : crée un attribut avec le nom spécifié

Manipulation des attributs

- `element.getAttribute(attribute)` : retourne la valeur de l'attribut spécifié
- `element.setAttribute(attribute, value)` : définit ou modifie la valeur de l'attribut spécifié
- `element.removeAttribute(attribute)` : supprime l'attribut spécifié
- `element.hasAttribute(attribute)` : retourne `true` si l'élément a l'attribut spécifié

Exemple

Ceci est un div de test avec l'id test qui contient un div avec la classe exemple:

Hello World

```
let div = document.createElement("div");
let exemple = document.querySelector("#test .exemple");
exemple.innerHTML = "";
div.innerHTML = "Hello World";
div.style.color = "red";
exemple.appendChild(div);
```

Copy

```
HTMLDivElement {}
```

Événements

- Les événements sont des actions qui se produisent dans le navigateur
- Les événements peuvent être déclenchés par l'utilisateur ou par le navigateur
- Les événements sont utilisés pour déclencher des fonctions
- Les événements sont associés à des éléments HTML
- Les événements sont déclenchés par des actions comme le clic de souris, le survol de la souris, le chargement de la page, etc.

Gestionnaire d'événements

- `element.addEventListener(event, function)` : ajoute un écouteur d'événements à un élément
  - `event` : nom (type) de l'événement
  - `function` : fonction à exécuter lorsque l'événement est déclenché (accepte un seul paramètre `event`)
- `element.removeEventListener(event, function)` : supprime un écouteur d'événements d'un élément

Types d'événements

- `click` : l'utilisateur clique sur un élément
- `mouseover` : l'utilisateur survole un élément
- `mouseout` : l'utilisateur quitte un élément
- `keydown` : l'utilisateur appuie sur une touche du clavier
- `keyup` : l'utilisateur relâche une touche du clavier
- `load` : la page est chargée
- `resize` : la fenêtre du navigateur est redimensionnée
- `scroll` : l'utilisateur fait défiler la page
- `focus` : un élément obtient le focus
- `blur` : un élément perd le focus
- `submit` : un formulaire est soumis
- `change` : la valeur d'un élément change

Exemple

Click me

```
let btn = document.querySelector("#btn");
btn.addEventListener("click", function() {
  alert("Hello World");
});
```

Copy

```
undefined
```

Gestionnaire d'évenements sur l'attribut d'un élément html

- le nom de l'attribut est le nom de l'événement précédé de `on`
- la valeur de l'attribut est le code Javascript à exécuter
- exemple: `element.onclick`

```
let btn2 = document.querySelector("#btn2");
btn2.onclick = function() {
  alert("Hello World");
};
```

Copy

```
TypeError: Cannot set properties of null (setting 'onclick')
```

```
<button id="btn2" style="font-size:25px" onclick="alert('Hello World')">Click me</button>
```

Copy

Click me

Utile

- `console.log()` : affiche un message dans la console du navigateur
- `alert()` : affiche une boîte de dialogue avec un message
- `prompt()` : affiche une boîte de dialogue avec un champ de saisie
- `confirm()` : affiche une boîte de dialogue avec un message et des boutons OK et Annuler
- `setTimeout(function, milliseconds)` : appelle une fonction après un délai spécifié
- `clearTimeout()` : annule un délai d'exécution défini avec `setTimeout()`

Méthodes utiles (Map, Filter, Reduce, etc)

- `map()` : applique une fonction à chaque élément d'un tableau et renvoie un nouveau tableau
- `filter()` : filtre les éléments d'un tableau en fonction d'une fonction et renvoie un nouveau tableau
- `reduce()` : réduit les éléments d'un tableau à une seule valeur en fonction d'une fonction
- `forEach()` : exécute une fonction pour chaque élément d'un tableau
- `sort()` : trie les éléments d'un tableau
- `find()` : renvoie la première valeur d'un tableau qui satisfait une condition

Map

- `map()` applique une fonction à chaque élément d'un tableau et renvoie un nouveau tableau

```
let arr = [1, 2, 3, 4, 5];
let arr2 = arr.map(x => x * 2);
console.log(arr2); // [2, 4, 6, 8, 10]
```

Copy

```
Array [
  2,
  4,
  6,
  8,
  10,
]
undefined
```

Filter

- `filter()` filtre les éléments d'un tableau en fonction d'une fonction et renvoie un nouveau tableau

```
let mots = ["bonjour", "test", "petit", "anticonstitutionnellement", "essai"];
let motsLongs = mots.filter(mot => mot.length > 5);
console.log(motsLongs); // ["bonjour", "anticonstitutionnellement"]
```

Copy

```
Array [
  "bonjour",
  "anticonstitutionnellement",
]
undefined
```

Reduce

- `reduce()` réduit les éléments d'un tableau à une seule valeur en fonction d'une fonction

```
let arr = [1, 2, 3, 4, 5];
let somme = arr.reduce((acc, val) => acc + val, 0);
console.log(somme); // 15
```

Copy

```
15
undefined
```

forEach

- `forEach()` exécute une fonction pour chaque élément d'un tableau

```
let arr = [1, 2, 3, 4, 5];
arr.forEach(x => console.log(x));
```

Copy

```
1
2
3
4
5
undefined
```

On peut combiner les méthodes

- `map()`, `filter()`, `reduce()`, `forEach()` peuvent être combinées pour effectuer des opérations complexes

```
let arr = [1, 2, 3, 4, 5];
let somme = arr.map(x => x * 2).filter(x => x > 5).reduce((acc, val) => acc + val, 0);
console.log(somme); // 24
```

Copy

```
24
undefined
```

Exemple

- Exemple:
  On souhaite la somme des ages des chiens dans le tableau `data` en faisant une traduction «age de chien» -> «age humain»

```
data = [
  {
    name: 'Butters',
    age: 3,
    type: 'dog'
  },
  {
    name: 'Lizzy',
    age: 6,
    type: 'dog'
  },
  {
    name: 'Red',
    age: 1,
    type: 'cat'
  },
  {
    name: 'Joey',
    age: 3,
    type: 'dog'
  },
];
```

Copy

```
Array [
  Object {
    "age": 3,
    "name": "Butters",
    "type": "dog",
  }
```

Solution1 - Boucle for

- Solution 1:
  Utilisation d'une boucle `for`

```
let somme = 0;
for (let i = 0; i < data.length; i++) {
  if (data[i].type === 'dog') {
    somme += data[i].age * 7;
  }
}
console.log(somme); // 84
```

Copy

```
84
undefined
```

Solution2 - Méthodes

Copy

```
let ages = data
.filter((animal) => {
  return animal.type === 'dog';
}).map((animal) => {
  return animal.age * 7;
}).reduce((sum, animal) => {
  return sum + animal.age;
});
```

```
undefined
```

Solution3 - Méthodes

Copy

```
let isDog = (animal) => {
  return animal.type === 'dog';
}
let ageHumain = (animal) => {
  return animal.age * 7;
}
let sum = (sum, animal) => {
  return sum + animal;
}

let ages = data
.filter(isDog)
.map(ageHumain)
.reduce(sum);

console.log(ages); // 84
```

```
84
undefined
```