Objectif

Affichez les en-têtes d'une structure hétérogène.

Etude préalable

Commençons par un cas simple : une structure homogène. Autrement dit, chaque objet de la structure possède les mêmes clés.

Exemple

Dans le tableau suivant, nous disposons d'une structure commune à l'ensemble des étudiants. Les en-têtes [name,marks,img] seront déduites de la structure pour représenter les en-têtes du tableau.

<table>
<tr><th>BD</th><th>représentation</th></tr>
<tr>
<td>

```
const students = [
  {
    name: "Alice",
    marks: [15, 5],
    img: "",
  },...
]
```

</td>
<td>
<table>
<tr><td>name</td><td>marks</td><td>img</td></tr>
<tr><td>Alice</td><td>15,5</td><td>👩‍🦰</td></tr>
<tr><td>…</td><td></td><td></td></tr>
</table>
</td>
</tr>
</table>

Commençons donc notre analyse par ce cas simple. C'est-à-dire, la structure de chaque étudiant est homogène.

Base de données homogènes

Une structure homogène garantit que chaque élément dispose de la même structure.

Exemple 

Considérons la BD suivante, où pour chaque étudiant nous disposons des mêmes informations.

```
1. const students = [
2.   {
3.     name: "Alice",
4.     marks: [15, 5],
5.     img: "https://randomuser.me/api/portraits/women/1.jpg",
6.   },
7. ...
8. ]
```

🪛Mission

Nous voudrions obtenir les attributs de la structure commune. Les attributs sont les clefs des objets.

Analyse

La structure est commune. Nous pouvons simplement prendre le premier étudiant pour représenter l'objet référent de l'étude.

L'objectif est d'extraire les clefs (keys) de l'objet référent.

Object référent	Keys de l'objet référent
students[0] { name: "Alice", marks: [15, 5], img: "https://randomuser.me/api/portraits/women/1.jpg", },...]	name marks img

Nous allons découvrir qu'il est très simple de déterminer les clés (keys) d'un objet.

Object.keys

The `Object.keys()` static method returns an array of a given object's own enumerable string-keyed property names. ref

En action

Tapez le code dans l'onglet **Try it** du cours de Mozilla ref

Try it

JavaScript Demo: Object.keys()

```
const  student =  {
    name: "Alice",
    marks: [15, 5],
    img: "https://randomuser.me/api/portraits/women/1.jpg",
  }


console.log(Object.keys(student));
// Expected output:  Array ["name", "marks", "img"]
```

Run ›

Reset

```
> Array ["name", "marks", "img"]
```

Try it

Nous pouvons manipuler très facilement Object.keys d'un objet.

🪛Etudiez les codes suivants.

```
const  student =  {
    name: "Alice",
    marks: [15, 5],
    img: "https://randomuser.me/api/portraits/women/1.jpg",
  }

console.log(Object.keys(student));
// Expected output:  Array ["name", "marks", "img"]
```

```
9.
10. for (const key in student) {
11.     console.log(key);
12. }
13.
14. const keys = Object.keys(student);
15. for (const key of keys) {
16.     console.log(key);
17. }
18.
19. keys.forEach((key) => {
20.     console.log(key);
21. });
22.
23. const transformedKeysToUppercase = keys.map((key) => {
24.     return key.toUpperCase();
25. });
26. console.log(transformedKeysToUppercase);
27.
28. const transformedKeys = keys.map((key) => key.charAt(0).toUpperCase()
        + key.slice(1));
29. console.log(transformedKeys);
```

Nous verrons en fin de dossier comment transformer ces clefs en en-têtes du tableau à afficher.

🥷Cas des structures hétérogènes

Notre mission se complique lorsque les structures sont **hétérogènes**. c'est classique dans les base de données comme Mongodb.

🪛Mission

Nous voudrions obtenir tous les attributs de la structure hétérogène.

Analyse

La structure n'est pas commune à chaque objet. Ainsi, nous ne pouvons pas simplement prendre le premier étudiant pour objet référent.

🆘Il faut extraire les clefs (keys) dans tous les objets.

Voici une illustration du problème. Seule la clef "name" est commune aux objets.

Étudiants	Keys
const students = [{ **name**: "Alice", cohort: "DDI", }, { **name**: "Mohamed", marks: [15, 5], img: "https://randomuser.me/..", },	name cohort marks img

{
name: "Bob",
}
]	

Analyse

🥷Le problème pour déterminer l'ensemble des clefs est plus délicat pour une structure hétérogène.

Il reste facile de comparer les keys données par différents étudiants.

```
const  students =  [
    {
        name: "Alice",
        cohort: "DDI",
    },
    {
        name: "Mohamed",
        marks: [15, 5],
        img: "https://randomuser.me/api/portraits/women/1.jpg",

    },
    {
        name: "Bob",
    }
]

```

```
17.
18. console.log(Object.keys(students[0]));
19. // Expected output:  Array ["name", "cohort"]
20.
21. console.log(Object.keys(students[1]));
22. // Expected output:  Array ["name", "marks", "img"]
```

Analyse

Nous pouvons facilement afficher les clefs de chaque objets.

🪛Evaluez le code

```
1.  students.forEach(student => {
2.      console.log(Object.keys(student));
3.  });
```

Collecte de toutes les clés uniques

Pour obtenir une liste complète de toutes les clés uniques du tableau d'élèves, vous pouvez utiliser un **ensemble** pour éviter les doublons.

Utilisation de Set

Les ensemble garantissent l'unicité des éléments !

Rappel : add ref

Ecrire le code suivant dans l'onglet Try it

Try it

JavaScript Demo: Set.prototype.add()

```
1  const entetes = new Set();
2  entetes.add("name")
3  entetes.add("name")
4  entetes.add("marks")
5  entetes.add("name")
6
7
8
9  for (const item of entetes) {
10   console.log(item);
11 }
12 // > "name"
13 //> "marks"
```

Run ›

Reset

```
> "name"
> "marks"
```

🆘Nous allons nous servir de cette méthode pour déterminer les clés.

🪛Evaluez

1. const allKeys = new Set();
2. students.forEach(student => {
3. Object.keys(student).forEach(key => allKeys.**add**(key));
4. });
5. console.log(Array.from(allKeys));

Nous allons étudier une autre solution utilisant reduce.

Utilisation de reduce

Voici une utilisation de reduce pour agréger des clés.

🥷Vous pouvez utiliser la fonction **reduce** pour regrouper toutes les clés dans un seul tableau, puis **filtrer** les doublons.

🪛Etudiez le code suivant

```
const allKeys = students.reduce((keys, student) => {
    return keys.concat(Object.keys(student));
}, []);
const uniqueKeys = [...new Set(allKeys)];
console.log(uniqueKeys);
```

Il est intéressant de créer une fonction réutilisable dans notre code.

Function

Nous pourrions utiliser une fonction pour obtenir toutes les clés. Nous utilisons `Array.from` pour transformer l'ensemble en tableau.

🪛Etudiez le code suivant

```
function getAllKeys(arr) {
    const keys = new Set();
```

```
    arr.forEach(obj => {
        Object.keys(obj).forEach(key => keys.add(key));
    });
    return Array.from(keys);
}

console.log(getAllKeys(students));
```

Pause

Réflexion

Vous ne connaissez pas encore la méthode flat

🪛Try it

Etudiez le code suivant

Try it

JavaScript Demo: Array.flat()

```
const entetes = [["name", "cohort"],["name", "marks", "img"],["name"]];

console.log(entetes.flat());
// expected output: > Array ["name", "cohort", "name", "marks", "img", "name"]
```

En complément de **flat**, nous pourrions utiliser **Set.**

Tapez le code suivant

Try it

JavaScript Demo: Array.flat()

```
const entetes = [["name", "cohort"],["name", "marks", "img"],["name"]];

console.log(entetes.flat());
// expected output: > Array ["name", "cohort", "name", "marks", "img", "name"]


console.log([...new Set(entetes.flat())])
// expected output: > Array ["name", "cohort", "marks", "img"]
```

🥷Reste à comprendre comment obtenir dans le problème qui nous intéresse le tableau des en-têtes ?

const entetes = [["name", "cohort"],["name", "marks", "img"],["name"]];

Utilisation de flat

L'utilisation de flat peut simplifier le processus de collecte de toutes les clés uniques du tableau des étudiants.

Voici comment vous pouvez l'utiliser et pourquoi c'est une bonne idée :

Explication

Aplatissement des tableaux (**Flattening Arrays**:)

La méthode flat est utilisée pour aplatir des tableaux imbriqués. Dans ce cas, elle peut aider à combiner toutes les clés de chaque objet étudiant en un seul tableau.

Simplification de la collecte des clés (**Simplifying Key Collection**:)

En aplatissant le tableau de clés, vous pouvez facilement supprimer les doublons et obtenir une liste complète de toutes les clés uniques.

🥷Mise en œuvre

Voici comment vous pouvez utiliser flat pour atteindre notre objectif :

```
const students = [
    {
        name: "Alice",
        cohort: "DDI",
    },
    {
```

```
        name: "Mohamed",
        marks: [15, 5],
        img: "https://randomuser.me/api/portraits/women/1.jpg",
    },
    {
        name: "Bob",
    }
];

// Collect all keys from each student object
const allKeys = students.map(student => Object.keys(student));

// Flatten the array of arrays into a single array
const flattenedKeys = allKeys.flat();

// Remove duplicates by converting to a Set and back to an array
const uniqueKeys = [...new Set(flattenedKeys)];

console.log(uniqueKeys);
// Expected output: ["name", "cohort", "marks", "img"]
```

limit is the sky : last code

🧙‍♂️Découvrez par vous même mapflat. : ref

🥷mapFlat

Je vous laisse étudier mapFlat.

TryIt

Try it

JavaScript Demo: Array.flatMap()

```
const students = [
  { name: "Alice" },
  { name: "Mohamed", mark: 1 },
  { name: "Bob",cohort: "DDI",},
];


heaaders = students.flatMap((student) => Object.keys(student))

console.log(heaaders);
// Expected output:  Array ["name", "name", "marks", "name", "cohort"]

```

Run ›

Reset

```
> Array ["name", "name", "marks", "name", "cohort"]
```

En action

🪛Etudiez le code suivant.

```
// Get all unique headers from the student data
const headers = Array.from(
  new Set(studentData.flatMap((student) => Object.keys(student)))
);
```

Explication du code

LE code récupère tous les en-têtes uniques du tableau studentData.

```
1. const headers = Array.from(
2.   new Set(studentData.flatMap((student) => Object.keys(student)))
3. );
```

Voici une explication étape par étape :

🆘explication :

- studentData.**flatMap**((étudiant) => Object.keys(étudiant)) :

flatMap est utilisé pour mettre en correspondance chaque objet étudiant avec ses clés, puis pour aplatir le résultat en un seul tableau.

Object.keys(student) renvoie un tableau de clés pour chaque objet étudiant.

flatMap combine tous ces tableaux de clés en un seul tableau plat.

🆘explication :

- new Set(...) :

Un **Set** est utilisé pour stocker des valeurs uniques. En passant le tableau de clés aplati au constructeur de Set, il supprime toutes les clés en double.

explication

- Array.from(...) :

Array.**from** reconvertit le Set en tableau. Nous obtenons ainsi un tableau d'en-têtes uniques.

Bilan

Pourquoi cette approche est efficace

Concision : L'utilisation de flatMap et de Set rend le code concis et facile à lire.

Efficacité : Cette approche combine efficacement le mappage et l'aplatissement en une seule étape, ce qui réduit le nombre d'itérations.

Unicité : L'utilisation de Set garantit que tous les en-têtes sont uniques sans vérification supplémentaire.

Exemple

Étant donné les données suivantes sur les étudiants :

```
const studentData = [
  { name: "Alice", cohort: "DDI" },
  { name: "Mohamed", marks: [15, 5], img:
  "https://randomuser.me/api/portraits/women/1.jpg" },
  { name: "Bob" }
];
```

Le code produira :

```
const headers = Array.from(
  new Set(studentData.flatMap((student) => Object.keys(student)))
);
```

```
4.
5. console.log(headers);
6. // Expected output: ["name", "cohort", "marks", "img"]
```

Cette sortie représente toutes les clés uniques trouvées dans le tableau studentData.

Affichage

Nous voulons afficher les entêtes dans un tableau !

Disposant des en-têtes dans un tableau, il est facile de générer les éléments HTML.

```
1.  <thead class="thead-dark">
2.      <tr>
3.        ${headers.map((header) => `<th>${header}</th>`).join("")}
4.      </tr>
5.   </thead>
```

La balise <thead> sera intégrée dans la balise <table>

Résumé :

```
//Get all unique headers from the student data using reduce
```

```
const headers = Array.from(
  studentData.reduce((acc, student) => {
    Object.keys(student).forEach((key) => acc.add(key));
    return acc;
  }, new Set())
);

//Get all unique headers from the student data using map and flat
const headers = Array.from(
  new Set(studentData.map((student) => Object.keys(student)).flat())
);

// Get all unique headers from the student data
const headers = Array.from(
  new Set(studentData.flatMap((student) => Object.keys(student)))
);
```