chap3 Assignation par décomposition

Affectation par décomposition

Vous pouvez affecter par décomposition des variables pré-définies comme suit :

```
let a, b;
[a, b] = [10, 20];
```

Si vous ne souhaitez affecter que quelques variables et récupérer le reste de l'assignation dans un tableau, vous pouvez utiliser le spread operator :

```
let a, b, rest;
[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(rest); // [30, 40, 50]
```

Vous pouvez également sauter des arguments dans l'assignation :

```
let a, b;
[, , a, b] = [10, 20, 11, 111];

console.log(a, b); // 11 111
```

De même vous pouvez par décomposition assigner des valeurs à des variables en fonction des clés d'un littéral, vous devez cependant dans ce cas utiliser les mêmes noms de variable que les clés du littéral :

```
const student = { mention: "AB", note: 12 };
const { mention, note } = student;

console.log(mention); // AB
console.log(note); // 12
```

Il est parfois intéressant de renommer les clés, dans ce cas il faudra utiliser la syntaxe suivante :

```
const student = { mention: "AB", note: 12 };
const { mention: m, note: n } = student;

console.log(m); // AB
console.log(n); // 12
```

On peut également le faire par imbrication :

```
const st = {
  name: "Antoine",
  family: {
    mother: "Yvette",
    father: "Michel",
    sister: "Sylvie",
  },
  age: 49,
};

const {
  name,
  family: { sister },
} = st;
```

Vous pouvez également destructurer un littéral en argument d'une fonction :

?

```
const student = { mention: "AB", note: 12 };
const infoStudent = ({ mention, note }) => "info : " + mention + "note : " + note;

infoStudent(student);

//notez que vous pouvez également définir la fonction infoStudent sans vous souciez de l'ordre des clés :
const infoStudent_bis = ({ note, mention }) => "info : " + mention + "note : " + note;
```

01 Exercice permutations

- Permutez les valeurs a et b suivantes :

```
let a = 1, b = 2;
```

- Soient les trois variables a, b, et c permutez les valeurs dans l'ordre suivant :
- a <- b
- b <- c
- c <- a

```
let a = 1, b = 2, c = 4;
```

02 Exercice assigner par décomposition

1. Calculez la moyenne des notes de l'étudiant. Modifiez la référence du littéral.
2. Puis assignez les valeurs **name**, **notes** et **average** dans les constantes name, notes et average dans le script courant.

```
const student = {
  name: "Alan",
  notes: [10, 16, 17],
  average: null,
};

// TODO ...

// constantes
console.log(name, notes, average);
```

03 Exercice iterate destructuring

Affichez le nom et le nom de la soeur de chaque étudiant en utilisant une boucle for of dans le littéral students :

// Nom : Alan soeur : Sylvie

```
const students = [
  {
    name: "Alan",
    family: {
      mother: "Yvette",
      father: "Paul",
      sister: "Sylvie",
    },
    age: 35,
  },
  {
    name: "Bernard",
    family: {
      mother: "Martine",
      father: "Cécile",
      sister: "Sophie",
    },
    age: 55,
  },
];
```

Le spread operator

Vous pouvez effectuer une fusion des deux tableaux en JS à l'aide de l'opérateur spread :

```
let numbers1 = [1, 2, 3, 4, 5, 7, 8, 9, 10];
let numbers2 = [11, 12, 13];

let numMerge = [...numbers1, ...numbers2];
```

Vous pouvez également fusionner deux littéraux :

```
const st1 = { s1: "Alan", s2: "Alice" };
const st2 = { s3: "Bernard", s4: "Sophie" };

const stMerge = { ...st1, ...st2 };
console.log(stMerge);
//{s1: "Alan", s2: "Alice", s3: "Bernard", s4: "Sophie"}
```

Le spread operator peut servir également pour mettre à jour une clé dans un littéral.

```
const st11 = { s1: "Alan", s2: "Alice" };
const st22 = { s2: "Bernard", s4: "Sophie" };

const stMerge = { ...st11, ...st22 };

console.log(stMerge);
// {s1: "Alan", s2: "Bernard",  s4: "Sophie"}
```

Un autre exemple de mise à jour avec cette technique.

```
const state = {
  name: "",
  age: 25,
  email: "alan@alan.fr",
};

const newState = { ...state, email: "sophie@sophie.fr" };
// {name: "", age: 25, email: "sophie@sophie.fr"}
```

04 Exercice push value

Soient les données suivantes. Créez un tableau strNumbers et pushez-y chacune de ses valeurs.

```
const str1 = ["one", "two"];
const str2 = ["three", "four"];
```

Nom de propriété calculé et décomposition

Vous pouvez utiliser une variable pour définir une clé d'un littéral. Dans la syntaxe ci-dessous, il faut utiliser les crochets pour que JS interprète la variable comme une clé du littéral.

```
const state = {
  name: "",
  email: "alan@alan.fr",
};

// définition d'une clé dynamique
let name = "email";
const newState = { ...state, [name]: "bernard@bernard.fr" };
```

06 Challenge Exercice accumulator

Créez une fonction récursive permettant de retourner la somme des nombres d'un tableau de valeurs numériques. Utilisez un paramètre facultatif pour accumuler les valeurs de la somme. Ce paramètre sera initialisé à zéro.

Utilisez la méthode shift() qui permet de dépiler la première valeur d'un tableau. Dans votre fonction récursive définissez une condition d'r ıon la fonction continuera de s'appeler elle-même indéfiniment (Stack Overflow).

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// retourne la première valeur du tableau en la supprimant du tableau
numbers.shift();

function accumulator(numbers, acc = 0) {
  // Une condition d'arrêt et retourner la somme des valeurs du tableau
  // dans la fonction on ré-appelle la fonction elle-même
  // accumulator(numbers, 10);
}

console.log(accumulator(numbers)); // doit retourner 55
```

07 Challenge deep copy

Faites une copie profonde de l'objet suivant :

```
const students = [
  {
    name: "Alan",
    family: {
      mother: "Yvette",
      father: "Paul",
      sister: "Sylvie",
    },
    age: 35,
  },
  {
    name: "Bernard",
    family: {
      mother: "Martine",
      father: "Cécile",
      sister: "Sophie",
    },
    age: 55,
  },
];
```

Modifié le: vendredi 22 septembre 2023, 13:07