Développement Web
TP n° 2 : Objets

Dans ce TP, vous allez manipuler les objets et découvrir quelques techniques classiques de JavaScript.

Exercice 1 : Clone

1. Donnez au moins une façon de réaliser une copie de surface d'un objet. Proposez des alternatives si vous en connaissez plusieurs (c'est normalement le cas).
2. On se propose de réaliser une copie profonde d'un objet. Les propriétés qui sont elles-mêmes des objets sont clonées récursivement. Les fonctions, bien qu'elles soient des objets, ne sont pas copiées et préservées telles quelles. Écrivez la fonction `deepClone` qui implémente ces spécifications. Vous pouvez tester votre code en exécutant :

```
const obj = {
  a: 1,
  b: { c: 2, d: 3 },
  e: null,
  f() {
    console.log(this.a + this.b.c + this.b.d);
  },
};

const clone = deepClone(obj);
clone.f(); // 6
delete obj.b.c;
obj.f(); // NaN
clone.f(); // 6
```

3. Exécutez le code suivant. Il est probable que vous rencontriez un problème.

```
const circularObj = {};
circularObj.autoRef = circularObj;
deepClone(circularObj);
```

4. Expliquez ce qu'il se passe et proposez une solution.

Exercice 2 : Listes chaînées

Une liste chaînée est représentée par un objet qui a une propriété `head` et un certain nombre de méthodes. La propriété `head` est `null` pour une liste vide ou contient le premier maillon du chaînage pour une liste non-vide. Un maillon est un objet de la forme `{value, next}` où `value` est la valeur contenue dans le maillon et `next` référence le maillon suivant (`null` si c'était le dernier maillon).

1. Définissez un constructeur `List` qui créé une liste vide.
2. Ajoutez une méthode `isEmpty` qui renvoie `true` si la liste courante est vide ; `false` sinon.
3. Ajoutez une méthode `add` qui ajoute un élément en tête de liste.
4. Ajoutez une méthode `get` qui renvoie l'élément de la liste à l'indice donné. Si l'indice passé est invalide, la méthode renvoie `undefined`. À ce stade, les tests suivants devraient passer.

```
const l = new List();
console.log(l.isEmpty()); // true
l.add(3);
l.add(2);
```

```
l.add(1);
console.log(l.isEmpty()); // false
console.log(l.get(0), l.get(1), l.get(2)); // 1, 2, 3
console.log(l.get(-1), l.get(3), l.get(null)); // undefined, undefined, undefined
```

5. Modifiez `add` pour qu'il accepte un nombre quelconque d'arguments. Le premier argument est le nouvel élément de tête et ainsi de suite.
6. Modifiez le constructeur pour qu'il accepte un nombre quelconque d'arguments de la même manière que `add`. Ne dupliquez pas de code.

```
const l2 = new List(4, 5, 6);
l2.add(1, 2, 3);
alert([0, 1, 2, 3, 4, 5].map((i) => l2.get(i))); // [1, 2, 3, 4, 5, 6]
```

7. Un itérateur est un objet qui possède une méthode sans paramètre `next`. Lorsqu'on appelle cette méthode, elle renvoie :
   — `{ done: false, value }` pour indiquer que l'itération est en cours avec `value` pour valeur courante ou
   — `{ done: true }` pour indiquer que l'itération est terminée.
   Définissez une fonction, `getIterator` telle que `getIterator(list)` renvoie un itérateur sur `list`. Voici un exemple d'utilisation :

```
function toArray(list) {
  const iterator = getIterator(list);
  const array = [];
  for (let { done, value } = iterator.next(); // premier état de l'itérateur
       !done; // itération finie ?
       { done, value } = iterator.next()) { // état suivant de l'itérateur
    array.push(value); // ajout de la valeur courante
  }
  return array;
}
console.log(toArray(l2)); // [1, 2, 3, 4, 5, 6]
```

8. Modifiez le constructeur, pour que la propriété `Symbol.iterator` soit liée à la fonction qui renvoie un itérateur sur la liste courante. Ne dupliquez pas de code. Si vous faites les choses correctement, vos listes deviennent itérables et l'appel suivant fonctionne.

```
console.log(Array.from(l2)); // [1, 2, 3, 4, 5, 6]
```

9. Exécutez le code suivant.

```
const iterable = { [Symbol.iterator]: l2[Symbol.iterator] };
console.log(Array.from(iterable)); // [ 1, 2, 3, 4, 5, 6 ]
```

   Est-ce le résultat attendu ?
   — Si oui, essayez de comprendre pourquoi la question peut se poser.
   — Si non, adaptez votre code pour obtenir le bon résultat.