

# Les objets JavaScript

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Qu'est-ce qu'un objet JavaScript ?</b>	<b>3</b>
<b>III. Exercice : Appliquez la notion</b>	<b>5</b>
<b>IV. Les propriétés d'un objet JavaScript</b>	<b>5</b>
<b>V. Exercice : Appliquez la notion</b>	<b>8</b>
<b>VI. Différences entre l'objet Object() et l'objet Array()</b>	<b>8</b>
<b>VII. Exercice : Appliquez la notion</b>	<b>10</b>
<b>VIII. Les méthodes de l'objet Object()</b>	<b>12</b>
<b>IX. Exercice : Appliquez la notion</b>	<b>13</b>
<b>X. Déstructuration et opérateur spread</b>	<b>14</b>
<b>XI. Exercice : Appliquez la notion</b>	<b>16</b>
<b>XII. Auto-évaluation</b>	<b>17</b>
A. Exercice final .....	17
B. Exercice : Défi .....	20
<b>Solutions des exercices</b>	<b>20</b>

## I. Contexte

**Durée :** 1 h

**Environnement de travail :** Repl.it

**Pré-requis :** Connaître les bases de JavaScript et des tableaux

### Contexte

Lors de nos développements, nous avons utilisé des variables qui contenaient des textes, nombres ou encore des tableaux pour la gestion des listes. Cependant, pour des besoins plus avancés, ces types ne sont pas suffisants.

Par exemple, au lieu de stocker le nom de l'utilisateur dans une variable, son âge dans une autre, etc. ou de les stocker dans un tableau qui ne nous permettra pas de retrouver à quoi correspond une valeur, nous allons créer une variable contenant un objet possédant toutes les informations d'un utilisateur.

## II. Qu'est-ce qu'un objet JavaScript ?

### Objectif

- Apprendre ce qu'est un objet et comment le déclarer

### Mise en situation

Afin de manipuler plus facilement des données complexes, nous allons utiliser les objets. Par exemple, pour stocker toutes les informations d'un utilisateur et les retrouver facilement en une seule variable contenant les valeurs associées aux propriétés définies (ex : `firstName`, `lastName`, `email`, `age`, `phoneNumber`).

### Définition

Un objet va donc contenir des associations propriété / valeur (ex : `firstName: 'John'`).

Ainsi, si nous créons tous nos utilisateurs via le même objet, il sera plus facile de les manipuler.

Une propriété d'objet peut contenir toutes sortes de données (texte, nombre, tableau, objet, fonction, etc.).

Contrairement aux tableaux, un objet possède des propriétés (`firstName`, etc.) et il est donc plus facile de retrouver et modifier une valeur, car il suffit d'appeler la propriété correspondante.

Nous allons donc voir les différentes manières pour déclarer un objet.

### Méthode Déclarer un objet avec la notation littérale

Nous pouvons également utiliser la notation littérale pour créer notre objet.

Ici, nous listons les propriétés associées à leur valeur entre accolades : pour affecter une valeur à la propriété, nous utilisons les deux points (:), suivis d'un espace (pour la lisibilité).

### Exemple

```
1 let user = {
2   firstName: 'John',
3   lastName: 'DOE',
4   age: 36,
5   email: 'j.doe@email.com',
6   phoneNumbers: ['0660504030', '0123456789'],
```

```

7   }
8 }
9
10 console.log(user)

```

La console affiche :

```

1 1  {
2 2    firstName: 'John',
3 3    lastName: 'DOE',
4 4    age: 36,
5 5    email: 'j.doe@email.com',
6 6    phoneNumbers: [ '0660504030', '0123456789' ],
7 8  }
8 9
9 10 John

```

### Méthode Déclarer un objet en utilisant une fonction avec les mots « new » et « this »

L'avantage d'utiliser une fonction pour créer un objet est que celle-ci va servir de modèle et nous permettra de créer plusieurs objets avec les mêmes propriétés mais des valeurs différentes.

Nous créons tout d'abord notre fonction sur la base du modèle précédent :

```

1 function User(firstName, lastName, age) {
2   this.firstName = firstName
3   this.lastName = lastName
4   this.age = age
5   this.sayHello = function() {
6     return 'Hello ' + this.firstName + ' ' + this.lastName
7   }
8 }

```

Notez ici l'utilisation du mot clé « **this** », il correspond par définition à l'objet courant. Grâce à lui nous pouvons d'une part réutiliser nos propriétés au sein de notre fonction (cf : fonction `this.sayHello()`) Il permet d'une autre part d'accéder aux différentes propriétés depuis l'extérieur de notre fonction en le remplaçant par le nom de notre objet. Nous verrons ceci en exemples plus bas, ce sera plus parlant.

Nous allons ensuite grâce à notre fonction **User()** et le mot clé « **new** », créer deux utilisateurs :

```

1 let user1 = new User("John", "Doe", 36)
2 let user2 = new User("Jane", "Smith", 42)

```

Nous pouvons maintenant afficher en console nos objets et leurs propriétés :

```

1 // Afficher nos 2 utilisateurs
2 console.log(user1) // User {firstName: 'John', lastName: 'Doe', age: 36, sayHello: f}
3 console.log(user2) // User {firstName: 'Jane', lastName: 'Smith', age: 42, sayHello: f}
4
5 // Accéder aux propriétés de nos objets en remplaçant le mot clé 'this' par le nom de l'objet
6 console.log(user1.firstName) // John
7 console.log(user2.firstName) // Jane
8
9 // Utiliser la fonction sayHello() créée au sein de notre fonction User()
10 console.log(user1.sayHello()) // Hello John Doe
11 console.log(user2.sayHello()) // Hello Jane Smith

```

**Syntaxe**    **À retenir**

Pour créer un objet, nous pouvons :

- Utiliser la notation littérale,
- Utiliser une fonction avec les mots clés « new » et « this ».

**Complément**

Le JavaScript orienté objet pour débutants<sup>1</sup>

### III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

**Question**

[solution n°1 p.21]

Instanciez un objet `car` de manière littérale, qui aurait comme propriétés : `'type'`, `'color'`, `'doors'`, `'airConditioner'`.

L'attribut `type` est de type `string` ; `color` est de type `string` ; `doors` de type `number` ; et `airConditioner` de type `boolean`.

Les valeurs de chaque propriété doivent respecter les types.

### IV. Les propriétés d'un objet JavaScript

**Objectif**

- Savoir accéder aux propriétés et les manipuler

**Mise en situation**

Nous savons créer des objets avec leurs propriétés et nous allons voir maintenant comment manipuler celles-ci, afin des les afficher ou de les modifier.

**Méthode**    **Les propriétés**

Une propriété d'un `Object()` peut stocker tous les types. Ainsi, chaque propriété d'un objet est indépendante des autres. Si nous prenons l'exemple d'un utilisateur, la propriété `'firstName'` sera un `string`, `'age'` sera un `number` et `'phoneNumbers'` un `Array()`.

```
1 let user = {  
2   firstName: 'John',  
3   lastName: 'DOE',  
4   age: 36,  
5   email: 'j.doe@email.com',  
6   phoneNumbers: ['0660504030', '0123456789'],  
7 }  
8
```

<sup>1</sup> [https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JS\\_orient%C3%A9-objet](https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JS_orient%C3%A9-objet)

<sup>2</sup> <https://repl.it/>

### Méthode Avec le point (.)

Pour accéder à une propriété, nous pouvons utiliser le nom de notre variable suivi d'un point, puis du nom de la propriété.

Si la valeur de la propriété est un tableau, alors nous pouvons la manipuler comme tel, par exemple en affichant la première valeur (`user.phoneNumbers[0]`).

Si la valeur de la propriété est une fonction, alors il faut lui ajouter des parenthèses pour l'appeler (`user.fullName()`).

### Exemple

```
1 let user = {
2   firstName: 'John',
3   lastName: 'DOE',
4   age: 36,
5   email: 'j.doe@email.com',
6   phoneNumbers: ['0660504030', '0123456789'],
7   fullName: function() {
8     return this.firstName + ' ' + this.lastName
9   }
10 }
11
12 // Display : John
13 console.log(user.firstName)
14
15 // Display the first phone number in the array : 0660504030
16 console.log(user.phoneNumbers[0])
17
18 // Display the return of the fullName() function : John DOE
19 console.log(user.fullName())
```

### Méthode Avec les crochets ([])

Il est également possible de manipuler les propriétés à la manière d'un tableau en accolant à la suite de notre variable le nom de la propriété entre crochets.

Le nom de la propriété devra être mis entre apostrophes au sein des crochets (`user['firstName']`).

### Exemple

```
1 let user = {
2   firstName: 'John',
3   lastName: 'DOE',
4   age: 36,
5   email: 'j.doe@email.com',
6   phoneNumbers: ['0660504030', '0123456789'],
7   fullName: function() {
8     return this.firstName + ' ' + this.lastName
9   }
10 }
11
12 // Display : John
13 console.log(user['firstName'])
14
15 // Display the first phone number in the array : 0660504030
16 console.log(user['phoneNumbers'][0])
17
```

```
18 // Display the return of the fullName() function : John DOE
19 console.log(user['fullName']())
```

### Méthode Modifier une valeur de propriété

Pour modifier la valeur d'une propriété, nous allons l'appeler avec l'une des méthodes ci-dessus et lui affecter une nouvelle valeur.

### Exemple

```
1 let user = {
2   firstName: 'John',
3   lastName: 'DOE'
4 }
5
6 user.firstName = 'Bob'
7
8 // Display : Bob
9 console.log(user.firstName)
```

### Méthode Ajouter une nouvelle propriété

Pour ajouter une nouvelle propriété, même si la variable contenant l'objet est déjà créée, nous utilisons notre variable, puis nous y accolons notre nouvelle propriété avec la méthode choisie, et nous lui affectons la valeur désirée.

### Exemple

```
1 let user = {
2   firstName: 'John',
3   lastName: 'DOE'
4 }
5
6 user.userName = 'jdoe'
7
8 // Display : { firstName: 'John', lastName: 'DOE', userName: 'jdoe' }
9 console.log(user)
```

### Méthode Boucler sur les propriétés d'un objet avec for...in

Cette instruction va créer une boucle permettant d'exécuter un bloc d'instructions sur chaque propriété d'un objet.

### Exemple

Ici l'exemple est simple car l'objet n'a que deux propriétés, mais il pourrait évidemment en avoir beaucoup plus.

```
1 1 let user = {firstName: 'John', lastName: 'DOE'}
2 2
3 3 for (let property in user) {
4 4   console.log(property)
5 5 }
```

L'exemple ci-dessus, sur un objet, va afficher dans la console :

firstName

lastName

### Syntaxe À retenir

Pour accéder aux propriétés d'un objet, nous pouvons :

- Utiliser le point (ex : `user.firstName`)
- Utiliser les crochets et apostrophes (ex : `user['firstName']`)

Il est possible de rajouter une propriété en utilisant une des méthodes ci-dessus et en y affectant une valeur avec le signe égal.

On peut également utiliser l'affectation de valeur pour modifier la valeur d'une propriété existante.

Enfin, on utilise la méthode `for...in` pour boucler sur les propriétés d'un objet.

### Complément

Utiliser les objets<sup>1</sup>

Initialisateur d'objet<sup>2</sup>

Les bases de JavaScript, orienté objet<sup>3</sup>

Méthodes du constructeur `Object`<sup>4</sup>

## V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



### Question

[solution n°2 p.21]

Affectez à deux variables différentes, de deux manières différentes, le nombre de portes de la voiture décrite par l'objet ci-après.

```
1 const car = {
2   type: 'BMW',
3   color: 'bleu',
4   doors: 5,
5   airConditioner: true
6 }
```

## VI. Différences entre l'objet `Object()` et l'objet `Array()`

### Objectif

- Apprendre la différence entre un `Object()` et un `Array()`

1 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser\\_les\\_objets](https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_objets)

2 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Initialisateur\\_objet](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Initialisateur_objet)

3 <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/Basics>

4 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets\\_globaux/Object#M%C3%A9thodes\\_du\\_constructeur\\_Object](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Object#M%C3%A9thodes_du_constructeur_Object)

5 <https://repl.it/>



## Mise en situation

Dans vos développement, vous croiserez systématiquement deux types d'objets : `Object()` et `Array()`. Nous traiterons ici des différences et des similitudes entre ces deux objets.

### Rappel `Array()`

L'objet `Array()` peut être assimilé à une liste.

Pour récupérer un élément de la liste, on va se servir de son index (place dans le tableau).

Le premier index d'un tableau est égal à 0.

## Différences

Lorsque l'on souhaite récupérer une valeur dans un `Array()`, on va itérer sur ses index, tandis que, pour un `Object()`, on va itérer sur ses propriétés.

```
1 const carInObject = {
2   type: 'BMW',
3   doors: 3,
4   color: 'blue'
5 }
6
7 const carInArray = ['BMW', 3, 'blue'];
8
9 console.log(carInObject.color); // blue
10 console.log(carInArray[2]); // blue
11
12
```

Il n'est pas possible de connaître le nombre d'éléments présents dans un `Object()` sans le transformer en `Array()`.

```
1 const carInObject = {
2   type: 'BMW',
3   doors: 3,
4   color: 'blue'
5 }
6
7 const carInArray = ['BMW', 3, 'blue'];
8
9 console.log(carInObject.length); // undefined
10 console.log(carInArray.length); // 3
11
```

## Similitude

Un `Object()` peut stocker un `Array()`, et inversement. Tout dépendra des besoins. Si un objet possède des caractéristiques différentes, on utilisera un `Object()`. L'`Array()` va, dans la plupart des cas, nous servir à stocker des listes d'éléments de même type.

Reprenons l'exemple des voitures :

```
1 const car1 = {
2   type: 'BMW',
3   doors: 3,
4   color: 'blue'
5 }
6
7 const car2 = {
```

```

8   type: 'PEUGEOT',
9   doors: 5,
10  color: 'grey'
11 }
12
13 // Dans l'objet garage nous stockons la liste des objets car
14 const garage1 = {
15   cars: [car1, car2]
16 }
17
18 const garage2 = {
19   cars: [car1, car2]
20 }
21
22 // Pour stocker la liste des garages nous utiliseront un Array()
23 const garages = [garage1, garage2]

```

### Syntaxe À retenir

- On ne peut pas directement itérer sur un `Object()` via l'index des propriétés.
- On ne peut pas connaître directement le nombre de propriétés contenues dans un objet `Object()` sans le transformer en `Array()`.

### Complément

Le JavaScript orienté objet pour débutants<sup>1</sup>

## VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



### Question 1

[solution n°3 p.21]

Dans le cadre d'une étude statistique au niveau national, on a stocké chaque moyenne de chaque élève par classe et par matière.

À partir de cet objet, écrivez un code retournant en console la note de Jeanne en Anglais.

```

1 const regions = [
2   {
3     name: 'Occitanie',
4     departement: [
5       {
6         name: 'Hérault',
7         lycees: [
8           {
9             name: 'Clemenceau',
10            adress: {
11              cp: '34060',
12              numberStreet: '31',

```

<sup>1</sup> [https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JS\\_orient%C3%A9-objet](https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JS_orient%C3%A9-objet)

<sup>2</sup> <https://repl.it/>

```

13     street: 'Avenue Georges Clemenceau',
14 },
15 classes : [
16     {
17         graduation: '1er',
18         sector: 'S',
19         students: [
20             {
21                 name: 'Paul',
22                 scores: [
23                     { matiere: 'Français', note: 12 },
24                     { matiere: 'Maths', note: 15 },
25                     { matiere: 'Espagnol', note: 7 },
26                     { matiere: 'Anglais', note: 9 },
27                     { matiere: 'Histoire', note: 10 },
28                 ]
29             },
30             {
31                 name: 'Jeanne ',
32                 scores: [
33                     { matiere: 'Français', note: 16 },
34                     { matiere: 'Maths', note: 10 },
35                     { matiere: 'Espagnol', note: 10 },
36                     { matiere: 'Anglais', note: 12 },
37                     { matiere: 'Histoire', note: 8 },
38                 ]
39             },
40             {
41                 name: 'Marie',
42                 scores: [
43                     { matiere: 'Français', note: 14 },
44                     { matiere: 'Maths', note: 18 },
45                     { matiere: 'Espagnol', note: 17 },
46                     { matiere: 'Anglais', note: 14 },
47                     { matiere: 'Histoire', note: 15 },
48                 ]
49             },
50             {
51                 name: 'Pierre',
52                 scores: [
53                     { matiere: 'Français', note: 7 },
54                     { matiere: 'Maths', note: 4 },
55                     { matiere: 'Espagnol', note: 8 },
56                     { matiere: 'Anglais', note: 6 },
57                     { matiere: 'Histoire', note: 8 },
58                 ]
59             },
60             {
61                 name: 'Nicolas',
62                 scores: [
63                     { matiere: 'Français', note: 11 },
64                     { matiere: 'Maths', note: 16 },
65                     { matiere: 'Espagnol', note: 8 },
66                     { matiere: 'Anglais', note: 10 },
67                     { matiere: 'Histoire', note: 13 },
68                 ]
69             }
70         ]

```

```

71     }
72   ]
73 }
74 ]
75 }
76 ]
77 }
78 ]

```

### Indice :

Un tableau commence à l'indice 0.

### Question 2

[solution n°4 p.21]

Écrivez un code retournant en console la moyenne générale de Nicolas.

### Indice :

Utilisez la méthode `Math.round()` pour arrondir un résultat au dixième.

## VIII. Les méthodes de l'objet Object()

### Objectif

- Utiliser les méthodes de l'objet `Object()`

### Mise en situation

Pour effectuer des opérations sur les `Object()`, vous devrez la plupart du temps utiliser les méthodes internes à l'objet.

#### Méthode Transformer un Objet() en Array()

Pour effectuer certaines opérations, il va être nécessaire de transformer notre `Objet` en `Array`. Pour ce faire, `Object` possède des méthodes permettant de faciliter son utilisation, par exemple pour connaître les propriétés d'un objet, les valeurs des propriétés, les propriétés avec valeurs associées, etc. Nous allons ici en citer quelques-unes :

- `Object.keys()` : permet de récupérer un tableau contenant la liste des propriétés de l'objet donné.
- `Object.values()` : permet de récupérer un tableau contenant la liste des valeurs des propriétés de l'objet.
- `Object.entries()` : renvoie un tableau à plusieurs dimensions contenant les propriétés avec les valeurs associées.

#### Exemple

```

1 let user = {
2   firstName: 'John',
3   lastName: 'DOE',
4   age: 36,
5   email: 'j.doe@email.com',
6   phoneNumbers: ['0660504030', '0123456789'],
7 }

1 console.log(Object.keys(user));
2 // ['firstName', 'lastName', 'age', 'email', 'phoneNumbers']

1 console.log(Object.values(user));
2 // ['John', 'DOE', 36, 'j.doe@email.com', ['0660504030', '0123456789']]

```

```

1 console.log(Object.entries(user));
2 /* [ ['firstName', 'John' ],
3    [ 'lastName', 'DOE' ],
4    [ 'age', 36 ],
5    [ 'email', 'j.doe@email.com' ],
6    [ 'phoneNumbers', [ '0660504030', '0123456789' ] ],
7    [ 'fullName', [Function] ] ] */

```

### Méthode HasOwnProperty()

Il n'est pas toujours nécessaire de modifier notre `Object()` en `Array()`. Lorsque nous voulons vérifier qu'une propriété existe dans l'objet et que nous connaissons son nom, nous pouvons utiliser `Object.hasOwnProperty('property')`.

```

1 let user = {
2   firstName: 'John',
3   lastName: 'DOE',
4   age: 36,
5   email: 'j.doe@email.com',
6   phoneNumbers: ['0660504030', '0123456789'],
7 }
8
9 let propertyLastNameExist = user.hasOwnProperty('lastName')
10 if (propertyLastNameExist) {
11   console.log(user['lastName']) // DOE
12 }

```

### Syntaxe À retenir

- `Object` propose différentes méthodes permettant de connaître les propriétés et valeurs d'un objet.

## IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



### Question

[solution n°5 p.21]

À partir de l'objet suivant, stockez dans un tableau les propriétés, et dans un autre les valeurs :

```

1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 5,
5   airConditioner: true
6 }

```

1 <https://repl.it/>

## X. Déstructuration et opérateur spread

### Objectifs

- Comprendre comment fonctionne la déstructuration JavaScript apparue avec ES6 et ses avantages.
- Savoir utiliser les opérateurs `spread`.

### La déstructuration en JavaScript

La déstructuration consiste en l'assignation de variables provenant d'un objet ou d'un tableau en s'appuyant sur leur structure. Il y a deux syntaxes différentes pour les objets et les tableaux.

```

1 // L'objet "personne" a des propriétés nom, age et ville
2 2   let personne = {
3 3     nom : "Thomas",
4 4     age: 35,
5 5     ville: "Paris"
6 6   };
7
8 8   // Avec les méthodes classiques, il faut déclarer une variable et lui assigner la
    propriété de l'objet séparément à chaque fois.
9 9   let nom = personne.nom;
10 10  let age = personne.age;
11 11  let adresse = personne.ville;
12
13 13  // Avec la déstructuration, il suffit d'une seule ligne
14 14  // Définition des variables nom, age et ville, que l'on renomme en adresse, pour leur
    appliquer les valeurs des propriétés du même nom de l'objet personne
15 15  let { nom, age, ville: adresse } = personne;
16
17 17  console.log("nom : ", nom, " age : ", age, " adresse : ", adresse);
18 18  // Affichera nom : Thomas age : 35 adresse : Paris

```

La syntaxe est la suivante :

`let {nom de chaque propriété de l'objet souhaité} = nom de l'objet ;`

Si on souhaite que la variable ait un nom différent de la propriété existante, il faut la renommer comme ceci : `nom de la propriété : nomsouhaité` pour la variable.

Cette méthode est plus simple et rapide que la méthode habituelle.

Pour un tableau, la syntaxe est sensiblement identique mais on utilise des croches `[]`.

```

1 1   // Le tableau
2 2   let langages = ["JavaScript", "Java", "PHP", "Go"];
3
4 4   // Avec les méthodes classiques, il faut déclarer une variable et lui assigner la
    valeur de chaque élément du tableau séparément.
5 5   let langage1 = langages[0];
6 6   let langage2 = langages[1];
7 7   let langage4 = langages[3];
8
9 9   // Avec la déstructuration, il suffit d'une seule ligne.
10 10  let [langage1, langage2, langage3, langage4] = langages;
11
12 12  // On peut aussi procéder comme suit
13 13  let [langage1, langage2, langage3, langage4] = ["JavaScript", "Java", "PHP", "Go"];
14
15 15  console.log("langage1 : ", langage1, " langage2 : ", langage2, " langage3 : ",
    langage3, " langage4 : ", langage4);
16 16  // Affichera langage1 : JavaScript, langage2 : Java, langage3 : PHP, langage4 : Go

```

La syntaxe est la suivante :

```
let [nom de chaque variable] = tableau ;
```

Il est également possible de pratiquer la déstructuration sur plusieurs niveaux, dans un objet comme dans un tableau. Si la valeur n'existe pas dans le premier niveau, elle retournera `undefined`. cependant, s'il y a besoin d'accéder à un niveau inférieur à `undefined`, une erreur sera retournée, due au `nesting`.

#### Remarque Omission de clés

Il est possible de déstructurer un tableau en omettant certaines de ses clés en ne l'affectant dans aucune variable. Dans ce cas, on oublie simplement d'en spécifier une.

```
1 // 1
2 2 let [langage1, langage2, langage3] = ["JavaScript", "Java", "PHP", "Go"];
3 3 console.log(langage1, langage2, langage3); // JavaScript Java PHP
4 4
5 5 // 2
6 6 let [langage1, , langage3] = ["JavaScript", "Java", "PHP", "Go"];
7 7 console.log(langage1, langage3); // JavaScript PHP
8 8
9 9 // 3
10 10 let [, langage2, langage3] = ["JavaScript", "Java", "PHP", "Go"];
11 11 console.log(langage2, langage3); // Java PHP
```

1. On affecte JavaScript dans langage1, Java dans langage2, PHP dans langage3 et Go dans rien.
2. On affecte JavaScript dans langage1, Java dans rien, PHP dans langage2 et Go dans langage3.
3. On affecte JavaScript dans rien, Java dans langage1, PHP dans langage2 et Go dans langage3.

### L'opérateur spread

L'opérateur `spread` permet de développer un objet itérable (comme un tableau ou une chaîne de caractères) lorsqu'on a besoin de plusieurs arguments faisant partie de ses propriétés.

```
1 let maDate = [1985, 12, 10];
2 2 new Date(...maDate);
3 3 // Cette notation équivaut à new Date(1985, 12, 10)
4 4
5 5 let maChaineDeCaractères = "ma chaîne";
6 6 [...maChaineDeCaractères]
7 7 // Cette notation équivaut à ["m", "a", " ", "c", "h", "a", "i", "n", "e"]
```

Il peut être mélangé à d'autres arguments :

```
1 let maDate = [1985, 12, 10];
2 2 new Date(...maDate, 50);
3 3 // Cette notation équivaut à new Date(1985, 12, 10, 50)
```

Cet opérateur permet de faciliter plusieurs traitements fréquemment nécessaires, tels que la concaténation de plusieurs itérables, la déstructuration dans un tableau, etc.

```
1 // Concaténation de plusieurs itérables
2 2 let mesLangages = ["JavaScript", "Java"];
3 3
4 4 // Anciennement avec ES5
5 5 ["PHP", "Go"].concat(mesLangages);
6 6
7 7 // Avec ES6
8 8 let lang = ["PHP", "Go", ...mesLangages]
9 9 console.log(lang)
10 10 // Retournera ["PHP", "Go", "JavaScript", "Java"]
```

```

11 11
12 12
13 13 // Déstructuration dans un tableau
14 14
15 15 let mesLangages = ["JavaScript", "Java", "PHP", "Go"];
16 16
17 17 // Anciennement avec ES5
18 18 let premierMot = mesLangages[0];
19 19
20 20 // "JavaScript"
21 21 let restant = mesLangages.slice(1);
22 22 console.log(restant)
23 23 // ["Java", "PHP", "Go"]
24 24
25 25 // Avec ES6
26 26 let [premierMot, ...restant] = mesLangages;
27 27 // On obtient le même résultat avec moins d'opérations à effectuer.

```

### Syntaxe À retenir

La déstructuration et l'opérateur `spread` permettent de simplifier le code et de le rendre plus clair et concis, en évitant des traitements répétitifs inutiles.

### Complément

Affectation par déstructuration<sup>1</sup>

Syntaxe de déstructuration<sup>2</sup>

## XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



### Question

[solution n°6 p.22]

En utilisant la déstructuration, la méthode `splice()` et l'opérateur `spread`, comment obtenir ces résultats à partir de `myVariable` ?

```

1 let myArray= [1,2,3,4,5,6]
2 // code...
3 console.log(myNumber) // 2
4 // code...
5 console.log(myArray) // [5,6]
6 // code...
7 console.log(myArray) // [2,5,6]

```

### Indice :

La méthode `splice()` permet d'insérer, supprimer ou modifier les éléments d'un tableau.

1 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

2 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Spread\\_syntax](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Spread_syntax)

3 <https://replit.com/>



## XII. Auto-évaluation

### A. Exercice final

#### Exercice 1

[solution n°7 p.22]

Exercice

Cette syntaxe est correcte.

```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true,  
6 };  
7  
8 const type = car['type'];
```

- ☐ Vrai
- ☐ Faux

Exercice

Quel objet correspond aux résultats de la console ?

```
1 console.log(Object.keys(person));  
2 // ['name', 'age', 'vegetarian'];  
3  
4 console.log(Object.value(person));  
5 // ['Pierre', 20, false];
```

- ☐ const person = {name : 'Pierre', age : '20', vegetarian : 'false'};
- ☐ const person = {name : 'Pierre', age : 20, vegetarian : false};
- ☐ const person = {name : Pierre, age : 20, vegetarian : false};

Exercice

Quel résultat sera affiché en console ?

```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true  
6 };  
7  
8 const properties = Object.keys(car)  
9  
10 console.log(properties)
```

Exercice

On veut ajouter une propriété `radio` dont la valeur serait `true` à l'objet `car` de l'exercice précédent. Quelles syntaxes sont justes ?

- ☐ `car.hasOwnProperty('radio')=true;`
- ☐ `car.radio = true;`
- ☐ `car[radio] = true;`
- ☐ `car['radio'] = true;`

#### Exercice

Qu'affichera la console ?

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 3,
5   airConditioner: true,
6   getType: function(){
7     return this.type;
8   }
9 };
10
11 console.log(car.getType());
```

- ☐ Error, getType is not a function
- ☐ BMW
- ☐ type
- ☐ {type: 'BMW'}

#### Exercice

On peut changer la valeur d'une propriété de cette manière.

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 3,
5   airConditioner: true,
6   getType: function(){
7     return this.type;
8   }
9 };
10
11
12 car.getType = car.type;
```

- ☐ Vrai
- ☐ Faux

#### Exercice

Quelle méthode doit-on utiliser pour stocker les propriétés et les valeurs d'un `Object` dans un tableau ?

#### Exercice

Que retournera la console si on compare deux objets de type `Object` exactement identiques ?

```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true  
6 };  
7  
8 const car2 = {  
9   type: 'BMW',  
10  color: 'blue',  
11  doors: 3,  
12  airConditioner: true  
13 };  
14  
15  
16 console.log(car === car2);
```

- ☐ True
- ☐ False

#### Exercice

Que retournera la console ?

```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true  
6 };  
7 console.log(car[1])
```

- ☐ color
- ☐ undefined
- ☐ null
- ☐ blue

#### Exercice

Que retournera la console ?

```
1 const person = {  
2   name: 'Nicolas',  
3   age: 20,  
4   getPerson: function() {  
5     return this.name + ' a ' + (this.age + 10) + ' ans';  
6   }  
7 }  
8 console.log(person.getPerson());
```

## B. Exercice : Défi

Dans une entreprise, le dirigeant veut vérifier que tous ses employés ont effectué leurs 35 heures.

Pour cela, il se base sur un tableau représentant ses employés.

Chaque employé est représenté par un `Object` et le nombre d'heures effectuées correspond à la propriété `nbHour`.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



### Question 1

[solution n°8 p.25]

Dans un premier temps, ajoutez à chaque objet une propriété `alert`, qui sera une fonction qui retournera un `string` avec le nom de l'employé et le nombre d'heures effectuées :

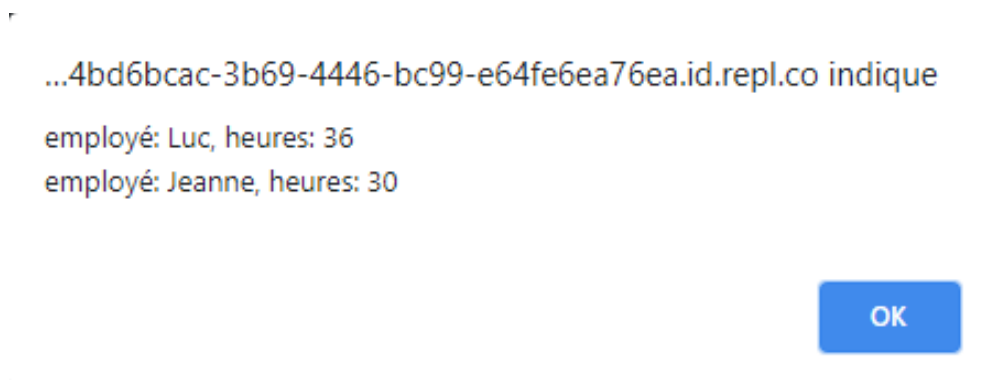
- `'employé: <nom>, heures: <nombre heures>'`

```
1 const workers = [
2   {name: 'Benjamin', age: 25, nbHour: 35},
3   {name: 'Luc', age: 45, nbHour: 36},
4   {name: 'Marie', age: 23, nbHour: 35},
5   {name: 'Jeanne', age: 36, nbHour: 30},
6   {name: 'Jean', age: 37, nbHour: 35}
7 ]
8
```

### Question 2

[solution n°9 p.26]

Créez maintenant le code qui filtre les employés qui n'auront pas effectué exactement leurs 35 heures. Utilisez la propriété `alert` créée précédemment pour ajouter chaque employé qui n'aura pas fait ses 35 h dans une variable `alerte`. Cette variable sera utilisée dans une alerte lorsque la totalité du code aura été exécuté. L'alerte sera de ce format :



## Solutions des exercices

1 <https://repl.it/>

**p. 5 Solution n°1**

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 5,
5   airConditioner: true
6 }
```

**p. 8 Solution n°2**

```
1 const car = {
2   type: 'BMW',
3   color: 'bleu',
4   doors: 5,
5   airConditioner: true
6 }
7
8 const doorsNumber1 = car.doors;
9 const doorsNumber2 = car['doors'];
10
11 console.log(doorsNumber1);
12 console.log(doorsNumber2);
```

**p. 10 Solution n°3**

```
1 const noteEnglishJeanne =
  regions[0].departement[0].lycees[0].classes[0].students[1].scores[3].note;
2 console.log(noteEnglishJeanne)
```

**p. 12 Solution n°4**

```
1 let average = 0
2 // boucle sur le tableau des étudiant de la classe
3 const notesNicolas = regions[0].departement[0].lycees[0].classes[0].students[4].scores ;
4 for (let i = 0 ; i < notesNicolas.length ; i++) {
5   average += Math.round((notesNicolas[i].note / notesNicolas.length) * 10) / 10;
6 }
7
8 console.log(average)
9
```

**p. 13 Solution n°5**

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 5,
5   airConditioner: true
6 }
7
```

```
8 const properties = Object.keys(car);
9 const values = Object.values(car);
10 console.log(properties); //['type', 'color', 'doors', 'airConditioner'];
11 console.log(values); //['BMW', 'blue', 5, true];
```

#### p. 16 Solution n°6

```
1 let myArray= [1,2,3,4,5,6];
2   const [,myNumber] = myArray;
3   console.log(myNumber) // 2
4   myArray.splice(0, 4);
5   console.log(myArray) // [5, 6]
6   myArray= [myNumber, ...myArray];
7   console.log(myArray) // [2, 5, 6]
```

#### Exercice p. 17 Solution n°7

##### Exercice

Cette syntaxe est correcte.

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 3,
5   airConditioner: true,
6 };
7
8 const type= car['type'];
```

☒ Vrai

☐ Faux

*En JavaScript, on peut accéder aux valeurs d'un Object de cette manière : `const value = monObjet['propriete']`.*

*On dit qu'un Object JavaScript est aussi un tableau associatif.*

##### Exercice

Quel objet correspond aux résultats de la console ?

```
1 console.log(Object.keys(person));
2 // ['name', 'age', 'vegetarian'];
3
4 console.log(Object.value(person));
5 // ['Pierre', 20, false];
```

☐ `const person = {name: 'Pierre', age: '20', vegetarian: 'false'};`

*Attention au typage, age est de type number et vegetarian de type boolean.*

☒ `const person = {name: 'Pierre', age: 20, vegetarian: false};`

☐ `const person = {name: Pierre, age: 20, vegetarian: false};`

*Attention au typage, age est de type number et vegetarian de type boolean.*

##### Exercice

Quel résultat sera affiché en console ?

```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true  
6 };  
7  
8 const properties = Object.keys(car)  
9  
10 console.log(properties)
```

doors



```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true  
6 };  
7  
8 // On stocke dans la variable properties les propriétés de l'objet car  
9 console.log(properties) // ['type', 'color', 'doors', 'airConditioner']
```

### Exercice

On veut ajouter une propriété `radio` dont la valeur serait `true` à l'objet `car` de l'exercice précédent. Quelles syntaxes sont justes ?

☐ `car.hasOwnProperty('radio')=true;`

*La méthode `hasOwnProperty('property')` retourne un booléen indiquant si la propriété existe dans l'Object.*

☒ `car.radio = true;`

☐ `car[radio] = true;`

*Pour accéder ou ajouter une propriété via un tableau associatif, la propriété doit être écrite en `string`.*

☒ `car['radio'] = true;`

### Exercice

Qu'affichera la console ?

```
1 const car = {  
2   type: 'BMW',  
3   color: 'blue',  
4   doors: 3,  
5   airConditioner: true,  
6   getType: function(){  
7     return this.type;  
8   }  
9 };  
10  
11 console.log(car.getType());
```

☐ Error, `getType` is not a function

*`getType()` est une fonction.*

☒ BMW

- ☐ type  
*this* représente l'objet courant. Dans notre cas, l'objet courant est *car*. Donc *this.type* revient à écrire *car.type*.  
 Le résultat est donc BMW.
- ☐ {type: 'BMW'}  
*this* représente l'objet courant. Dans notre cas, l'objet courant est *car*. Donc *this.type* revient à écrire *car.type*.  
 Le résultat est donc BMW.

### Exercice

On peut changer la valeur d'une propriété de cette manière.

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 3,
5   airConditioner: true,
6   getType: function(){
7     return this.type;
8   }
9 };
10
11
12 car.getType = car.type;
```

- ☒ Vrai
- ☐ Faux  
*getType* est une propriété dont le type est *function*. Quel que soit le type d'une propriété, le typage faible de JavaScript permet d'y accéder et de changer son type et sa valeur.

### Exercice

Quelle méthode doit-on utiliser pour stocker les propriétés et les valeurs d'un `Object` dans un tableau ?

`entries()`

### Exercice

Que retournera la console si on compare deux objets de type `Object` exactement identiques ?

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 3,
5   airConditioner: true
6 };
7
8 const car2 = {
9   type: 'BMW',
10  color: 'blue',
11  doors: 3,
12  airConditioner: true
13 };
14
15
16 console.log(car === car2);
```



☐ True

*JavaScript ne peut pas comparer deux objets de type `Object`. Pour comparer ces deux objets, il faut itérer sur les propriétés et/ou les valeurs de manière indépendante, pour chacune d'entre elles.*

☒ False

### Exercice

Que retournera la console ?

```
1 const car = {
2   type: 'BMW',
3   color: 'blue',
4   doors: 3,
5   airConditioner: true
6 };
7 console.log(car[1])
```

☐ color

*Un `Object` est un tableau associatif. Cela signifie que l'on peut accéder aux valeurs via un tableau et que celles-ci ne sont pas liées à des index, mais à des propriétés.*

*La propriété 1 n'existe pas, donc la console retournera `undefined`.*

☒ undefined

☐ null

*Un `Object` est un tableau associatif. Cela signifie que l'on peut accéder aux valeurs via un tableau et que celles-ci ne sont pas liées à des index, mais à des propriétés.*

*La propriété 1 n'existe pas, donc la console retournera `undefined`.*

☐ blue

*Un `Object` est un tableau associatif. Cela signifie que l'on peut accéder aux valeurs via un tableau et que celles-ci ne sont pas liées à des index, mais à des propriétés.*


*La propriété 1 n'existe pas, donc la console retournera `undefined`.*

### Exercice

Que retournera la console ?

```
1 const person = {
2   name: 'Nicolas',
3   age: 20,
4   getPerson: function() {
5     return this.name + ' a ' + (this.age + 10) + ' ans';
6   }
7 }
8 console.log(person.getPerson());
```

Nicolas a 30 ans

 Attention au typage lorsqu'on utilise l'opérateur '+'. `this.age` est de type `number`. Comme il est additionné entre parenthèses avec 10, les deux chiffres s'additionnent.

`this` permet d'accéder aux propriétés de l'objet courant. Dans notre cas, `person`.

`this.name` équivaut à `person.name`.

`this.age` équivaut à `person.age`.

```
1 // On ajoute à chaque objet une méthode qui retournera l'alerte
2 const workers = [
3   {name: 'Benjamin', age: 25, nbHour: 35, alert: function () {return `employé: ${this.name},
4     heures: ${this.nbHour}`}},
5   {name: 'Luc', age: 45, nbHour: 36, alert: function () {return `employé: ${this.name},
6     heures: ${this.nbHour}`}},
7   {name: 'Marie', age: 23, nbHour: 35, alert: function () {return `employé: ${this.name},
8     heures: ${this.nbHour}`}},
9   {name: 'Jeanne', age: 36, nbHour: 30, alert: function () {return `employé: ${this.name},
10    heures: ${this.nbHour}`}},
11   {name: 'Jean', age: 37, nbHour: 35, alert: function () {return `employé: ${this.name},
12    heures: ${this.nbHour}`}}
13 ]
```

**p. 20 Solution n°9**

```
1 let alerte = '';
2
3 const workerHasNotHour = [];
4
5 for (let i = 0; i < workers.length; i++) {
6   if (workers[i].nbHour !== 35) {
7     // On ajoute le message d'alerte à la variable alerte
8     alerte += `${workers[i].alert()} \n`;
9   }
10 }
11
12 alert(alerte);
```