Les objets JavaScript

Table des matières

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.

Méthode **Déclarer un objet avec Object()**

Pour déclarer un objet, nous allons utiliser le constructeur `Object()`.

Pour cela, nous allons créer une variable et lui affecter comme valeur la construction d'un nouvel objet (`new Object()`).

Nous allons ensuite créer ses différentes propriétés en accolant à notre variable un point, puis le nom de la propriété désirée (en la nommant en anglais, sans espaces ni caractères spéciaux), à laquelle nous allons affecter la valeur désirée avec le signe égal (=).

Exemple

```
let user = new Object()
user.firstName = 'John'
user.lastName = 'DOE'
user.age = 36
```

```
user.email = 'j.doe@email.com'
user.phoneNumbers = ['0660504030', '0123456789']
user.fullName = function() {
  return this.firstName + ' ' + this.lastName
}

console.log(user)
```

La console affiche :

```
{
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: [ '0660504030', '0123456789' ],
  fullName: [Function]
}
```

Remarque **this**

Dans la fonction ajoutée dans notre objet (`user.fullName = function()...`), nous avons utilisé le mot-clé `this` (`this.firstName`).

Celui-ci se réfère à l'objet courant au sein duquel le code est écrit (ici, il se réfère donc à : `user`).

Ainsi, si nous changeons `user` par autre chose ou que nous en créons un nouveau (via un copié/collé) en nommant la variable `user2` par exemple, la référence sera toujours la bonne, car `this` se référera cette fois à `user2`.

Méthode **Déclarer un objet avec la notation littérale**

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

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

Exemple

```
let user = {
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: ['0660504030', '0123456789'],
  fullName: function() {
    return this.firstName + ' ' + this.lastName
  }
}

console.log(user)
```

La console affichera le même résultat que précédemment.

Méthode **Créer une fonction pour construire nos objets**

Nous pouvons également créer nos propres fonctions de création d'objets, ce qui permet de les réutiliser et de faciliter l'utilisation et la modification de nos objets.

Pour cela, nous allons créer une fonction et, par convention, son nom commencera par une majuscule : `User()`.

Cette fonction prendra en paramètres les différentes valeurs (`firstNameValue`, etc.) des propriétés.

La fonction en elle-même affectera les différentes valeurs passées aux propriétés définies au sein de la fonction (`this.firstName`, etc.).

Pour utiliser notre constructeur et créer un nouvel utilisateur, nous l'appelons en lui passant les valeurs désirées (ex : `new User('John', 'DOE', ...)`).

L'avantage est que, pour ajouter un nouvel utilisateur, il suffira de rappeler à nouveau ce constructeur en y passant les valeurs désirées (ex : `new User('Jane', ...)`).

Exemple

```
function User(
  firstNameValue,
  lastNameValue,
  ageValue,
  emailValue,
  phoneNumbersValue
) {
  this.firstName = firstNameValue
  this.lastName = lastNameValue
  this.age = ageValue
  this.email = emailValue
  this.phoneNumbers = phoneNumbersValue
  this.fullName = function() {
    return this.firstName + ' ' + this.lastName
  }
}

let user = new User(
  'John',
  'DOE',
  36,
  'j.doe@email.com',
  ['0660504030', '0123456789']
)

console.log(user)
```

Syntaxe **À retenir**

Pour créer un objet, nous pouvons :

- Utiliser le constructeur `Object()`,
- Utiliser la notation littérale,
- Créer notre propre fonction constructeur.

Complément

Le JavaScript orienté objet pour débutants[1]

III. Exercice : Appliquez la notion

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

[2]

Question

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()`.

```
let user = {
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: ['0660504030', '0123456789'],
}

```

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]`).

[1] https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JS_orient%C3%A9-objet
[2] https://repl.it/

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

Exemple

```
let user = {
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: ['0660504030', '0123456789'],
  fullName: function() {
    return this.firstName + ' ' + this.lastName
  }
}

// Display : John
console.log(user.firstName)

// Display the first phone number in the array : 0660504030
console.log(user.phoneNumbers[0])

// Display the return of the fullName() function : John DOE
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

```
let user = {
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: ['0660504030', '0123456789'],
  fullName: function() {
    return this.firstName + ' ' + this.lastName
  }
}

// Display : John
console.log(user['firstName'])

// Display the first phone number in the array : 0660504030
console.log(user['phoneNumbers'][0])

// Display the return of the fullName() function : John DOE
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

```
let user = {
  firstName: 'John',
  lastName: 'DOE'
}

user.firstName = 'Bob'

// Display : Bob
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

```
let user = {
  firstName: 'John',
  lastName: 'DOE'
}

user.userName = 'jdoe'

// Display : { firstName: 'John', lastName: 'DOE', userName: 'jdoe' }
console.log(user)
```

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.

Complément

Utiliser les objets[1]

Initialisateur d'objet[2]

Les bases de JavaScript, orienté objet[3]

[1] 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
[3] https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/Basics

Méthodes du constructeur Object[1]

V. Exercice : Appliquez la notion

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

[2]

Question

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

```
const car = {
  type: 'BMW',
  color: 'bleu',
  doors: 5,
  airConditioner: true
}
```

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

Objectif

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

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.

```
const carInObject = {
  type: 'BMW',
  doors: 3,
  color: 'blue'
}

const carInArray = ['BMW', 3, 'blue'];

```

[1] https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Object#M%C3%A9thodes_du_constructeur_Object
[2] https://repl.it/

```
console.log(carInObject.color); // blue
console.log(carInArray[2]); // blue


```

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

```
const carInObject = {
  type: 'BMW',
  doors: 3,
  color: 'blue'
}

const carInArray = ['BMW', 3, 'blue'];

console.log(carInObject.length); // undefined
console.log(carInArray.length); // 3

```

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 :

```
const car1 = {
  type: 'BMW',
  doors: 3,
  color: 'blue'
}

const car2 = {
  type: 'PEUGEOT',
  doors: 5,
  color: 'grey'
}

// Dans l'objet garage nous sotckons la liste des objets car
const garage1 = {
   cars: [car1, car2]
}

const garage2 = {
   cars: [car1, car2]
}

// Pour stocker la liste des garages nous utiliseront un Array()
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[1]

VII. Exercice : Appliquez la notion

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

[2]

Question 1

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.

```
const regions = [
  {
    name: 'Occitanie',
    departement: [
      {
        name: 'Herault',
        lycees: [
          {
            name: 'Clemenceau',
            adress: {
              cp: '34060',
              numberStreet: '31',
              street: 'Avenue Georges Clemenceau',
            },
            classes : [
              {
                graduation: '1er',
                sector: 'S',
                students: [
                  {
                    name: 'Paul',
                    scores: [
                      { matiere: 'Français', note: 12 },
                      { matiere: 'Maths', note: 15 },
                      { matiere: 'Espagnol', note: 7 },
                      { matiere: 'Anglais', note: 9 },
                      { matiere: 'Histoire', note: 10 },
                    ]
                  },
                  {
                    name: 'Jeanne ',
                    scores: [
                      { matiere: 'Français', note: 16 },
                      { matiere: 'Maths', note: 10 },
                      { matiere: 'Espagnol', note: 10 },
                      { matiere: 'Anglais', note: 12 },
```

[1] https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JS_orient%C3%A9-objet
[2] https://repl.it/

```
                    { matiere: 'Histoire', note: 8 },
                  ]
                },
                {
                  name: 'Marie',
                  scores: [
                    { matiere: 'Français', note: 14 },
                    { matiere: 'Maths', note: 18 },
                    { matiere: 'Espagnol', note: 17 },
                    { matiere: 'Anglais', note: 14 },
                    { matiere: 'Histoire', note: 15 },
                  ]
                },
                {
                  name: 'Pierre',
                  scores: [
                    { matiere: 'Français', note: 7 },
                    { matiere: 'Maths', note: 4 },
                    { matiere: 'Espagnol', note: 8 },
                    { matiere: 'Anglais', note: 6 },
                    { matiere: 'Histoire', note: 8 },
                  ]
                },
                {
                  name: 'Nicolas',
                  scores: [
                    { matiere: 'Français', note: 11 },
                    { matiere: 'Maths', note: 16 },
                    { matiere: 'Espagnol', note: 8 },
                    { matiere: 'Anglais', note: 10 },
                    { matiere: 'Histoire', note: 13 },
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
]
```

Indice :

Un tableau commence à l'indice 0.

Question 2

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

```
let user = {
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: ['0660504030', '0123456789'],
}
```

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

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

```
console.log(Object.entries(user));
/*[ [ 'firstName', 'John' ],
  [ 'lastName', 'DOE' ],
  [ 'age', 36 ],
  [ 'email', 'j.doe@email.com' ],
  [ 'phoneNumbers', [ '0660504030', '0123456789' ] ],
  [ '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')`.

```
let user = {
  firstName: 'John',
  lastName: 'DOE',
  age: 36,
  email: 'j.doe@email.com',
  phoneNumbers: ['0660504030', '0123456789'],
}

let propertyLastNameExist = user.hasOwnProperty('lastName')
if (propertyLastNameExist) {
  console.log(user['lastName']) // DOE
```

```
}
```

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 :

[1]

Question

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

```
const car= {
  type: 'BMW',
  color: 'blue',
  doors: 5,
  airConditioner: true
}
```

X. Auto-évaluation

A. Exercice final

Exercice

Exercice

Cette syntaxe est correcte.

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true,
};

const type= car['type'];
```

- ○ Vrai
- ○ Faux

Exercice

Quel objet correspond aux résultats de la console ?

```
console.log(Object.keys(person));
// ['name', 'age', 'vegetarian'];

console.log(Object.value(person));
// ['Pierre', 20, false];
```

[1] https://repl.it/

- 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 ?

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};

const [, ,propertie]= Object.keys(car)

console.log(propertie)
```

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 ?

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
  getType: function(){
    return this.type;
  }
};

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.

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
  getType: function(){
    return this.type;
  }
};


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 ?

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};

const car2 = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};


console.log(car === car2);
```

- True
- False

Exercice

Que retournera la console ?

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

- color
- undefined
- null
- blue

Exercice

Que retournera la console ?

```
const person = {
  name: 'Nicolas',
  age: 20,
  getPerson: function() {
    return this.name + ' a ' + (this.age + 10) + ' ans';
  }
}
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 :

[1]

Question 1

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>'`

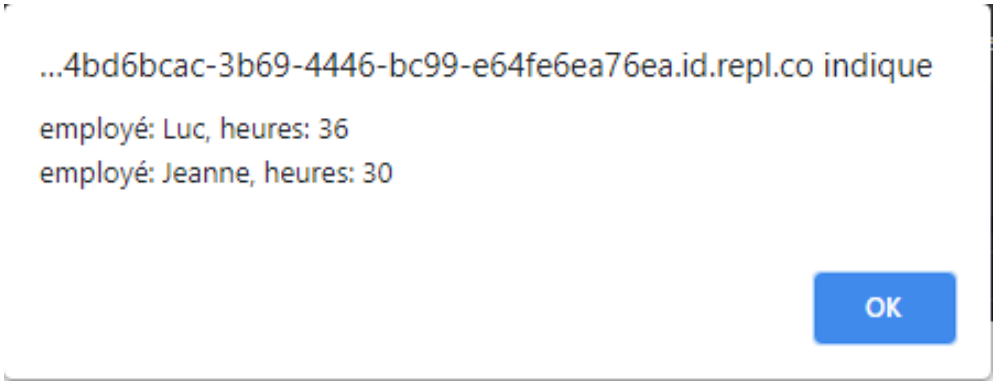
```
const workers = [
  {name: 'Benjamin', age: 25, nbHour: 35},
  {name: 'Luc', age: 45, nbHour: 36},
  {name: 'Marie', age: 23, nbHour: 35},
  {name: 'Jeanne', age: 36, nbHour: 30},
  {name: 'Jean', age: 37, nbHour: 35}
]

```

[1] https://repl.it/

Question 2

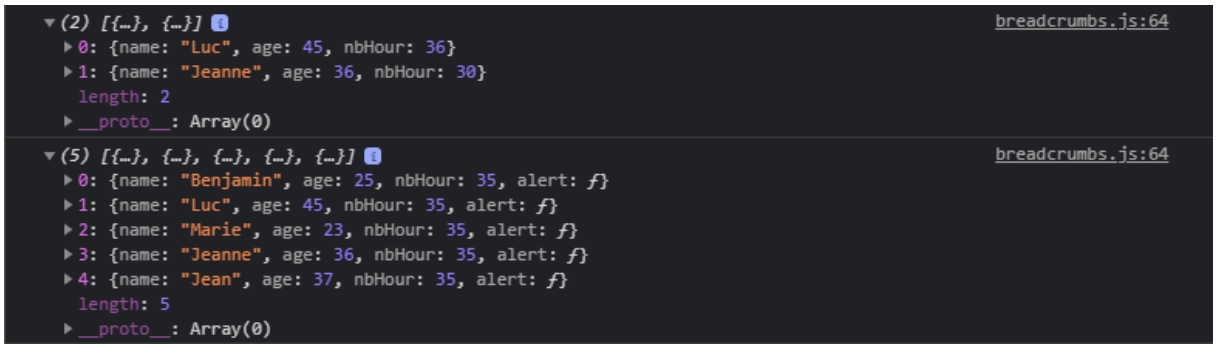
Créez maintenant le code qui trie les employés qui n'auront pas effectué exactement leurs 35 heures et qui les stocke dans un nouveau tableau. Utilisez la propriété `alert` créée précédemment pour ajouter chaque employé qui n'aura pas fait ses 35 h dans une variable `alerte`. Cette variable sera utilisée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



Question 3

Modifiez le code précédent en ajoutant la fonctionnalité permettant de mettre le nombre d'heures à jour dans le tableau `workers`. À la fin de l'exécution de la boucle, affichez en console le tableau `workers` mis à jour, ainsi que le tableau des employés n'ayant pas fait exactement leurs 35 h.

Vérifiez bien que les résultats que vous obtenez en console soient conformes aux résultats ci-dessous.



Indice :

Il y a de grandes chances pour que votre tableau d'employés n'ayant pas fait leurs 35 heures ait été mis à jour et que les propriétés `nbHour` soient égales à 35. Ceci est lié au **passage par référence**. Pour supprimer la référence à un objet, utilisez `JSON.parse(JSON.stringify(objet))`.

Solutions des exercices

Exercice p. Solution n°1

```
const car = {
  type: 'BMW',
  color: 'blue',
  doors: 5,
  airConditioner: true
}
```

Exercice p. Solution n°2

```
const car = {
  type: 'BMW',
  color: 'bleu',
  doors: 5,
  airConditioner: true
}

const doorsNumber1 = car.doors;
const doorsNumber2 = car['doors'];

console.log(doorsNumber1);
console.log(doorsNumber2);
```

Exercice p. Solution n°3

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

Exercice p. Solution n°4

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

console.log(average)

```

Exercice p. Solution n°5

```
const car = {
  type: 'BMW',
  color: 'blue',
  doors: 5,
  airConditioner: true
}

```

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

Exercice p. 14 Solution n°6

Exercice

Cette syntaxe est correcte.

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true,
};

const type= car['type'];
```

- (●) Vrai
- ( ) Faux

Exercice

Quel objet correspond aux résultats de la console ?

```
console.log(Object.keys(person));
// ['name', 'age', 'vegetarian'];

console.log(Object.value(person));
// ['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 ?

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};

const [, ,propertie]= Object.keys(car)

console.log(propertie)
```

doors

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};

```

```
// Si on décompose le code étape par étape

// On stocke les propriétés dans un tableau
const properties = Object.keys(car); // ['type', 'color', 'doors', 'airConditioner'];

// Par destructuration on stock la valeur située à l'index 2 du tableau
const [, ,propertie]= properties;
console.log(propertie) // doors

```

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 ;
- [x] car.radio = true ;
- [ ] car[radio] = true ;
- [x] car['radio'] = true ;

Exercice

Qu'affichera la console ?

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
  getType: function(){
    return this.type;
  }
};

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.

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
  getType: function(){
    return this.type;
  }
};


car.getType = car.type;
```

- [x] Vrai
- [ ] Faux

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 ?

```
const car = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};

const car2 = {
  type: 'BMW',
  color:'blue',
  doors: 3,
  airConditioner: true
};


console.log(car === car2);
```

- [ ] True
- [x] False

Exercice

Que retournera la console ?

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

- [ ] color
- [x] undefined
- [ ] null
- [ ] blue

Exercice

Que retournera la console ?

```
const person = {
  name: 'Nicolas',
  age: 20,
  getPerson: function() {
    return this.name + ' a ' + (this.age + 10) + ' ans';
  }
}
```

```
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`.

Exercice p. Solution n°7

```
// On ajoute à chaque objet une méthode qui retournera l'alerte
const workers = [
  {name: 'Benjamin', age: 25, nbHour: 35, alert: function () {return `employé: ${this.name}, heures: ${this.nbHour}`}},
  {name: 'Luc', age: 45, nbHour: 36, alert: function () {return `employé: ${this.name}, heures: ${this.nbHour}`}},
  {name: 'Marie', age: 23, nbHour: 35, alert: function () {return `employé: ${this.name}, heures: ${this.nbHour}`}},
  {name: 'Jeanne', age: 36, nbHour: 30, alert: function () {return `employé: ${this.name}, heures: ${this.nbHour}`}},
  {name: 'Jean', age: 37, nbHour: 35, alert: function () {return `employé: ${this.name}, heures: ${this.nbHour}`}}
]
```

Exercice p. Solution n°8

```
let alerte = '';

const workerHasNotHour = [];

for (let i = 0; i < workers.length; i++) {
  if (workers[i].nbHour !== 35) {
    // On ajoute le message d'alerte à la variable alerte
    alerte += `${workers[i].alert()} \n`;
    // On ajoute dans le tableau l'employé qui n'a pas fait ses 35h
    workerHasNotHour.push(workers[i]);
  }
}

alert(alerte);
```

Exercice p. Solution n°9

```
let alerte = '';

const workerHasNotHour = [];
for (let i = 0; i < workers.length; i++) {
  if (workers[i].nbHour !== 35) {
    alerte += `${workers[i].alert()} \n`;
    // On ajoute dans le tableau l'employé qui n'a pas fait ses 35h en faisant attention à supprimer la référence
    workerHasNotHour.push(JSON.parse(JSON.stringify(workers[i])));
    // On met à jour le nombre d'heures dans le tableau général
    workers[i].nbHour = 35;
  }
```

```
}

console.log(workerHasNotHour);
console.log(workers);
alert(alerte);
```