

Les objets JS

Table des matières

I. Contexte	3
II. Concept d'objet en POO	3
A. Concept d'objet en POO	3
B. Exercice : Quiz	6
III. Les principaux objets natifs en JS	6
A. Principaux objets natifs en JS	6
B. Définition de propriétés personnalisées	11
C. Exercice : Quiz	12
IV. Essentiel	13
V. Auto-évaluation	13
A. Exercice	13
B. Test	13
Solutions des exercices	14

I. Contexte

Durée : 1 h

Prérequis : avoir étudié le cours jusqu'ici

Contexte

Les objets sont des entités, des ensembles de propriétés, qui sont l'unité même de ce qu'on appelle « *Programmation Orientée Objet* ». Certains objets vous sont déjà familiers, comme les objets `Number`, ou `Array`. Mais, pour bien comprendre leur fonctionnement, et dans une optique de maîtriser la POO en JavaScript, nous allons faire ensemble le point sur les objets dits « *natifs* ».

Il existe deux catégories d'objets en JS : les objets natifs et les objets qu'on peut appeler « *personnalisés* » ou « *définis par l'utilisateur* ». Les objets natifs sont des objets accessibles par défaut dans n'importe quel environnement JavaScript. En revanche, les objets personnalisés sont des objets qu'un utilisateur définit, en créant son propre type.

L'objectif de ce cours est donc avant tout de nous mettre au point sur les principaux objets natifs en JavaScript. Dans la première partie, nous nous mettrons au point sur le concept d'objet en POO, en faisant des rappels, et en introduisant de nouvelles notions. Puis dans la seconde partie, nous aborderons les principaux objets natifs en JS. Nous introduirons en complément le mot clé « *this* » et nous aborderons l'utilisation de prototypes.

Ce cours sera constitué de nombreux exemples de codes que vous pourrez tester via Replit. Il vous permettra de vous familiariser davantage avec la notion d'objet en POO, ce qui vous sera essentiel dans votre parcours de développeur.

II. Concept d'objet en POO

A. Concept d'objet en POO

Pour rappel, un objet est une structure de données, une entité qui contient des propriétés de type valeur (de simples variables définies sur des valeurs), et des propriétés qu'on appelle méthodes (des variables définies sur des méthodes). Une propriété est donc une paire « *clé / valeur* », c'est-à-dire qu'elle a une clé (le nom de la propriété) et une valeur (valeur ou méthode sur laquelle elle est définie).

Exemple Propriété

```
1 monObj.nomPropriete = "valeur"
```

Dans cet exemple, **nomPropriete** est la clé de la propriété tandis que **"valeur"** est la valeur de la propriété.

Les propriétés définies sur une valeur décrivent les caractéristiques de l'objet, tandis que les méthodes sont des fonctions associées à l'objet qui peuvent être appelées pour effectuer des actions ou des opérations sur l'objet ou pour renvoyer une valeur.

Si on résume, on a :

- Les « *propriétés* » qui sont définies sur une valeur.
- Les « *méthodes* » qui sont en fait des fonctions que l'on peut appeler via un objet. Ce sont des propriétés qui ne sont pas définies sur une simple valeur, mais sur une fonction.

Par ailleurs, nous pouvons noter qu'une propriété peut aussi être définie sur la référence d'un objet. Pour bien comprendre, cela veut dire que des propriétés vont nous permettre de stocker d'autres objets.

```
1 monObj.nomPropriete = new Object();
```

Dans cet exemple, nous créons un objet et nous définissons notre propriété `nomPropriete` sur la référence de cet objet. Une propriété peut donc stocker une valeur, une fonction (on parle donc de méthode) ou un objet.

Le principe est très similaire à un objet du monde réel. Par exemple, un téléphone a des caractéristiques comparables aux propriétés définies sur une valeur (son stockage, sa ram, etc.), et des mécanismes comparables aux méthodes (activation et désactivation du mode avion, etc.).

Exemple Exemple d'objet avec la représentation d'un téléphone

Voici un exemple de code où l'on crée un objet dans lequel on définit ses propriétés :

```
1 let telephone = {
2
3   //propriétés définies sur une valeur primitive
4
5   marque: "Apple",
6   ram: 4,
7   stockage: 32,
8
9   //propriétés définies sur une méthode
10
11  activateModeAvion: function() {
12    console.log("Mode avion activé");
13  },
14
15  desactiverModeAvion: function() {
16    console.log("Mode avion désactivé");
17  }
18 }
```

Dans cet exemple, nous définissons les propriétés de notre `telephone` avec une syntaxe spécifique. Nous définissons des propriétés sur des valeurs, et d'autres sur des méthodes. Comme nous l'avons dit, il est aussi possible de définir des propriétés sur des objets. Par exemple, si dans cet exemple, on définit la propriété `ram` sur un objet `Number` :

```
1 let telephone = {
2
3   //proprietes definies sur une valeur primitive
4
5   marque: "Apple",
6   ram: new Number(4),
7   stockage: 32,
8
9   //proprietes definies sur une méthode
10
11  activateModeAvion: function() {
12    console.log("Mode avion activé");
13  },
14
15  desactiverModeAvion: function() {
16    console.log("Mode avion désactivé");
17  }
18 }
```

Dans de nombreux cas, nous serons amenés à définir des propriétés sur des références d'objet.

Définition **Type, constructeur et instance**

Chaque objet a un type. Un type définit la nature des valeurs possibles d'une donnée. Le type d'un objet définit les propriétés d'un objet, sa structure même. Pour créer un objet d'un certain type (`Number`, `Array`, ou autre) il nous faut appeler le constructeur de ce type. Les objets construits sont appelés « *instances* ».

Exemple

```
1 let nombre = new Number(5216);
2
3 //Number = type
4
5 //nombre = instance de Number
6
7 //Number() = constructeur du type Number
```

Dans cet exemple, nous construisons un objet de type `Number`.

En réalité, les types d'objets sont définis par ce qu'on appelle des **Classes**. Nous n'expliquerons pas en détail dans ce cours ce que sont des classes, mais pour faire simple, les classes sont comme des modèles, des plans, qui contiennent toutes les instructions (les propriétés de type valeur, les méthodes, le constructeur, etc.) permettant de construire un objet. Par exemple, `Number` est une classe. Lorsque nous construisons un objet de type `Number`, nousinstancions la classe `Number`. Mais restons en là pour l'instant avec le concept de classes que nous aborderons plus en détail en temps voulu.

Propriétés statiques et propriétés d'instance

Nous pouvons classer l'ensemble des propriétés en deux catégories :

- Les propriétés statiques, qui sont les mêmes pour toutes les instances d'un type,
- Les propriétés d'instance, qui sont définies sur des valeurs spécifiques à chaque instance.

Exemple

Si on reprend notre nombre de type `Number` :

```
1 let nombre = new Number(5216);
2
3 console.log(Number.MAX_VALUE) //propriété statique
4
5 console.log(Number.isInteger(1)) //méthode statique
6
7 console.log(nombre.valueOf()) //méthode d'instance
```

Les méthodes statiques s'appellent donc via le nom du type (dans cet exemple `Number`), tandis que les méthodes d'instance sont appelées via la référence de l'instance (dans cet exemple `nombre`). Maintenant, parlons des principaux objets natifs en JS.

Introduction aux objets natifs

À la différence des objets personnalisés, les objets natifs sont accessibles dans n'importe quel environnement JavaScript, sans même avoir le besoin d'installer une librairie. C'est le principe même d'un objet natif, il est natif au langage, et donc compris dans le « *Core JavaScript* », qui est la partie de JS définie par ECMAScript, qui contient donc tous les éléments natifs de JS.

B. Exercice : Quiz

[solution n°1 p.15]

Question 1

Qu'est-ce qu'un objet en JS ?

- ☐ Une variable qui stocke une valeur primitive
- ☐ Une entité, un ensemble de propriétés
- ☐ Un type de donnée prédéfini

Question 2

Un objet est une instance d'une classe.

- ☐ Vrai
- ☐ Faux
- ☐ Seulement si la classe est Object()

Question 3

Qu'est-ce qu'un type ?

- ☐ Une variable qui stocke une valeur primitive
- ☐ Le nom d'un objet spécifique
- ☐ La catégorie d'un objet qui définit ses propriétés, ou d'une variable

Question 4

Qu'est-ce qu'une propriété en JavaScript ?

- ☐ Obligatoirement une méthode
- ☐ Une variable qui peut stocker par exemple une valeur dans un objet
- ☐ Le nom d'un objet spécifique

Question 5

Qu'est-ce qu'une méthode en JavaScript ?

- ☐ Une variable qui stocke une valeur dans un objet
- ☐ Une fonction appartenant à un objet
- ☐ Le nom d'un objet spécifique

III. Les principaux objets natifs en JS

A. Principaux objets natifs en JS

L'objet Object

Object est l'objet de base de tout objet en JavaScript. Quel que soit l'objet créé, qu'il soit natif ou défini par un développeur, il héritera directement ou indirectement des propriétés de base du type Object. Pour construire un objet de type Object, nous pouvons appeler simplement son constructeur.

Méthode Construire un Object

Voici un exemple d'appel du constructeur Object. Pensons bien à utiliser le mot clé « *new* » afin de créer une instance d'Object :

```
1 let monObjet = new Object();
```

Il est aussi possible de créer un objet sans appeler directement le constructeur Object() :

```
1 let monObjet = { /*définition des propriétés*/ } ;
```

Dans ce cas, nous utilisons la « *syntaxe littérale pour initialiser les objets* », et le constructeur Object() est appelé implicitement.

Voyons maintenant quelques méthodes de la classe Object, dans un exemple de code :

Exemple Quelques méthodes de Object

```
1 //Nous créons un objet en enveloppant la valeur "undefined"
2
3 let monObjet = new Object();
4
5 //On crée 2 propriétés dans notre objet
6
7 monObjet = {
8   nom:"mon objet",
9   caracteristiques:"aucune"
10 };
11
12 //Quelques méthodes statiques
13
14 console.log(Object.keys(monObjet)); //renvoie les clés des propriétés définies
15 console.log(Object.values(monObjet)); //renvoie les valeurs des propriétés définies
16 console.log(Object.entries(monObjet)); //renvoie les paires clés/valeurs des propriétés
   définies
17
18 //Quelques méthodes d'instance
19
20 console.log(monObjet.valueOf()); //renvoie la valeur primitive de l'objet, donc l'objet même
21 console.log(monObjet.toString()); //renvoie une chaîne représentant l'objet
22 console.log(monObjet.hasOwnProperty("nom")); //renvoie un booléen si l'objet a la propriété
   passée comme argument. Ne prend pas en compte les propriétés héritées "par défaut" du
   prototype Object.prototype
```

On peut noter que toutes les méthodes prédéfinies d'instance sont héritées de l'objet Object.prototype. C'est cet objet qui contient initialement toutes les méthodes dont vont hériter chaque instance de Object, comme toString(), ValueOf(), etc.

Object étant l'objet de base en JS, tous les autres types d'objets (Number, Array, etc.) vont hériter des propriétés et méthodes de Object. Nous pourrions donc utiliser ces méthodes avec les instances des autres types d'objets.

Quand nous parlons « *d'héritage* », cela veut simplement dire qu'un objet va recevoir des propriétés d'un autre objet. Dans ce cas, tous les objets JS vont recevoir (donc hériter) des propriétés d'Object.

L'objet Boolean

Le type **Boolean** permet d'envelopper un booléen dans un objet. On parle de type « *wrapper* », c'est-à-dire qu'il permet d'envelopper une valeur primitive dans un objet, pour l'utiliser comme un objet.

Méthode Construire un Boolean

Pour créer une instance de Boolean, nous pouvons appeler son constructeur.

```
1 let monObjet = new Boolean(true);
```

Exemple Quelques méthodes de Boolean

Chaque instance de Boolean va hériter des méthodes de l'objet `Boolean.prototype`. Les méthodes de l'objet `Boolean.prototype` sont héritées de l'objet `Object.prototype`. Nous pouvons donc les appeler via des instances de Boolean. Mais nous pouvons aussi appeler les méthodes statiques de `Object` en passant comme argument des instances de Boolean, comme nous aurions d'ailleurs pu le faire pour l'objet `Number`.

Nous expliquerons un peu plus bas dans une vidéo le concept de prototype, ne vous inquiétez pas.

```
1 //Nous créons un Boolean en enveloppant la valeur "true"
2
3 let monObjet = new Boolean(true);
4
5 //Quelques méthodes d'instance héritées de Boolean.prototype
6
7 console.log(monObjet.valueOf()); //renvoie la valeur enveloppée
8
9 console.log(monObjet.toString()); // renvoie une chaîne contenant la valeur enveloppée
10
11 //Quelques méthodes statiques de Object :
12
13 //Ces méthodes renvoient un tableau vide car nous n'avons défini aucune propriété
14
15 console.log(Object.keys(monObjet)); //renvoie les clés des propriétés définies
16
17 console.log(Object.values(monObjet)); //renvoie les valeurs des propriétés définies
18
19 console.log(Object.entries(monObjet)); //renvoie les paires clé/valeur des propriétés définies
```

L'objet Number

Vous connaissez maintenant bien cet objet. `Number` est un sous-type d'`Object` permettant de stocker une valeur primitive `Number` dans un objet, en l'enveloppant (c'est un type « *wrapper* »). Il permet de manipuler des nombres en passant par un objet. `Number` contient des propriétés statiques et d'instances très utiles pour manipuler les nombres.

Méthode Construire un Number

Pour construire un `Number`, nous pouvons appeler son constructeur.

```
1 let monObjet = new Number(12467);
```

Voyons quelques méthodes spécifiques de l'objet `Number` :

Exemple Quelques méthodes de Number

```
1 //Nous créons un Number en enveloppant la valeur "12467"
2
3 let monObjet = new Number(12467.28);
4
5 //Quelques méthodes statiques
6
7 console.log(Number.isInteger(16)); //renvoie un booléen indiquant si le nombre est un entier
```



```

8 console.log(Number.isSafeInteger(121152172716642751675821627)); //renvoie un booléen indiquant
  si le nombre est un entier représentable de manière safe avec Number
9 console.log(Number.parseFloat("16278.12")); //convertit une chaîne en nombre à virgule
  flottante
10
11 //Une méthode d'instance
12
13 console.log(monObjet.toFixed(1)); //renvoie une chaîne contenant la valeur du nombre arrondi
  avec un nombre de chiffres décimaux passés comme argument

```

Les méthodes d'instance sont héritées de l'objet `Number.prototype` par toutes les instances de `Number`.

Bien évidemment, nous pouvons utiliser les méthodes héritées de `Object` comme `valueOf()` ou `toString()` avec un objet `Number` :

```

1 let monObjet = new Number(12467.28);
2
3 console.log(monObjet.valueOf()); //renvoie la valeur primitive stockée dans l'objet
4
5 console.log(monObjet.toString()); //renvoie une chaîne contenant la valeur primitive stockée
  dans l'objet

```

L'objet String

En JavaScript, l'objet natif `String` est utilisé pour représenter et manipuler des chaînes de caractères. Il permet d'envelopper une valeur primitive de type `String`. Il faut voir les chaînes de caractères comme des ensemble de symboles ordonnés. Chaque lettre correspond à la valeur d'une propriété de l'objet.

Méthode Construire un objet String

Pour créer un objet `String`, nous pouvons appeler son constructeur, en passant comme argument la chaîne enveloppée :

```
1 let monObjet = new String("chaîne de caractères");
```

Nous pouvons constater que si on appelle la méthode `Object.values(monObjet)`, un tableau contenant chaque caractère de notre chaîne est renvoyé. Si nous appelons `Object.keys(monObjet)`, c'est un tableau contenant la clé de chaque propriété, chaque propriété étant un indice (0, 1, 2, etc.).

```

1 let monObjet = new String("chaîne de caractères");
2
3 console.log(Object.keys(monObjet));
4
5 console.log(Object.values(monObjet));

```

On comprend bien en ce sens qu'une chaîne de caractères est un ensemble de caractères ordonnés, qui constituent chacun une propriété.

Chaque instance de `String` hérite des méthodes de l'objet `String.prototype`.

Exemple Quelques méthodes de l'objet String

```

1 monObjet = new String("chaîne de caractères");
2
3 //Un exemple de méthode statique
4
5 console.log(String.fromCharCode(67, 72, 65, 73, 78, 69)); //renvoie une chaîne composée des
  caractères unicode passés comme arguments, dans cet exemple : "CHAINE"
6
7 //Quelques méthodes d'instance
8

```

```
9 console.log(monObjet.charAt(4)); //renvoie le caractère de la chaîne stockée ayant l'index
   passé comme argument
10 console.log(monObjet.slice(7, 9)); //renvoie une sous-chaîne de la chaîne d'origine, en
   utilisant l'index de début et de fin spécifiés
```

L'objet Array

En JavaScript, l'objet `Array` permet de stocker et manipuler des collections d'éléments, qui peuvent être de n'importe quel type, y compris des chaînes de caractères, des nombres, des objets et même d'autres tableaux. Pour créer un objet `Array`, nous pouvons appeler son constructeur ou utiliser la syntaxe littérale de tableau.

Méthode Construire un Array

```
1 let monObjet = new Array(); //appel du constructeur
2
3 let monObjet2 = []; //utilisation de la syntaxe de tableau
```

L'objet `Array` contient des méthodes statiques et d'instances spécifiques que vous connaissez déjà. Les méthodes d'instance sont héritées de l'objet `Array.prototype`.

Exemple Quelques méthodes de Array

```
1 let monObjet = new Array(1, 2); //appel du constructeur en créant un tableau avec deux
   éléments
2
3 //Une méthode statique
4
5 console.log(Array.from("test")); //renvoie un tableau à partir d'un objet itérable ou d'une
   chaîne de caractères
6
7 //Quelques méthodes d'instance
8
9 console.log(monObjet.push(3)); //rajoute un (ou plusieurs) élément à la fin du tableau
10
11 console.log(monObjet.pop()); //supprime le dernier élément du tableau
12
13 console.log(monObjet.unshift(0)); //ajoute un (ou plusieurs) élément au début du tableau
14
15 console.log (monObjet);
```

L'objet Date

L'objet natif `Date` en JavaScript est utilisé pour travailler avec des dates et des heures. Il permet de créer des instances de date, de manipuler des dates, de les afficher et de les convertir en différents formats.

Méthode Construire un objet Date

Pour construire un objet `Date`, il nous faut appeler son constructeur. Si nous ne passons pas d'argument, l'objet `Date` stockera la date actuelle, sinon, il stockera la date passée :

```
1 let monObjet = new Date(); //date actuelle
2
3 let date = new Date('2000-12-18T03:24:00'); //date spécifiée
```

Bien évidemment, l'objet `Date` a des propriétés statiques et d'instance spécifiques, mais nous ne les aborderons pas dans ce cours, elles seront introduites en temps voulu. Cependant, `Date` étant aussi un objet, chaque instance de `Date` hérite des méthodes d'instance de l'objet `Date.prototype`. Quant à lui, `Date.prototype` hérite des méthodes de l'objet `Object.prototype`. On peut donc utiliser les méthodes `valueOf()` ou `toString()` par exemple, qui ont cependant un comportement plus spécifique.

Exemple Appel de méthodes héritées de `Object.prototype`

```
1 monObjet = new Date('2000-12-18T03:24:00');
2
3 console.log(monObjet.valueOf());
4
5 console.log(monObjet.toString());
```

Vous pouvez tester ce code pour voir le comportement plus spécifique des méthodes héritées. Nous approfondirons l'objet `Date`, ses propriétés et méthodes dans un cours dédié.

L'objet `Math`

L'objet `Math` fournit des méthodes pour effectuer des opérations mathématiques. C'est un objet natif de JavaScript. À la différence de tous les objets dont nous avons parlé, l'objet `Math` est un objet statique, et donc ne peut pas être instancié. Nous pouvons cependant appeler ses méthodes statiques permettant de réaliser des calculs mathématiques.

Exemple Appel de méthodes de l'objet `Math`

Voici quelques exemples de méthodes de l'objet `Math`.

```
1 console.log(Math.sqrt(9)) //renvoie la racine carrée du nombre.
2
3 console.log(Math.random()) //renvoie un nombre aléatoire entre 0 et 1
4
5 console.log(Math.max(1, 12, 13, 52)) //renvoie le nombre max dans un ensemble de nombres
6
7 console.log(Math.min(1, 12, 13, 52)) //renvoie le nombre min dans un ensemble de nombres
8
9 console.log(Math.round(26178.516)) //renvoie le nombre arrondi à l'entier le plus proche
```

B. Définition de propriétés personnalisées

Maintenant que nous avons vu les principaux objets natifs en JS, parlons de la définition de propriétés personnalisées. Nous les aborderons au travers de deux vidéos. Dans un premier temps, voyons comment utiliser l'objet prototype pour créer des propriétés dans un objet natif.

Méthode Créer de nouvelles propriétés via l'objet prototype

<https://replit.com/@CoursJS/prototype?v=1>

Maintenant que nous avons vu comment ajouter des propriétés personnalisées dans un objet natif, introduisons le mot clé « `this` » et voyons des cas précis d'utilisation.

Méthode **Utilisation de this**

<https://replit.com/@CoursJS/this?v=1>

Voilà, vous comprenez maintenant beaucoup mieux le concept d'objet. Par ailleurs, la notion d'objet natif a moins de secrets pour vous. Vous connaissez les principaux objets natifs de JavaScript. Bien évidemment, il en existe d'autres tels que **RegExp** ou **Error**, etc. Vous avez par ailleurs appris à définir des propriétés personnalisées d'un objet.

C. Exercice : Quiz

[solution n°2 p.16]

Question 1

Comment créer un objet Array en utilisant un constructeur ?

- ☐ `let array = Array();`
- ☐ `let array = new Array();`
- ☐ `let array = []`

Question 2

Comment créer un objet Boolean en appelant un constructeur ?

- ☐ `let bool = new Boolean();`
- ☐ `let bool = Boolean.create();`
- ☐ `let bool = Boolean();`

Question 3

On peut instancier Math.

- ☐ Vrai
- ☐ Faux
- ☐ Dans certains cas

Question 4

Où utiliser le mot-clé `this` pour faire référence à l'instance courante ?

- ☐ En dehors de toute méthode
- ☐ Comme argument lors de l'appel du constructeur
- ☐ Dans la définition d'une méthode de l'objet

Question 5

Comment définir une méthode qui sera héritée par toutes les instances de Array ?

- ☐ `Array.prototype.newMethod = function() { /*instruction; */ }`
- ☐ `Array.newMethod() = function() { /*instruction; */ }`
- ☐ `Array.newMethod = function() { /*instruction; */ }`

IV. Essentiel

Pour conclure, ce cours nous a permis de mieux comprendre des concepts phares de la POO. Nous avons vu que les objets sont des entités, qui regroupent des propriétés définies sur des valeurs, et des propriétés appelées méthodes qui sont en fait définies sur des fonctions. Chaque objet a un type qui définit l'ensemble de ses propriétés. Une instance est un objet créé via un constructeur, une méthode permettant de construire un objet selon un type.

En outre, nous avons abordé les principaux objets natifs en JavaScript, notamment **Object**, **Boolean**, **Number**, **String**, **Array**, **Date** et **Math**. Vous en connaissiez déjà certains. Les objets peuvent permettre d'envelopper un type primitif pour le manipuler comme un objet. Nous pouvons distinguer deux types de méthodes : les méthodes statiques et d'instance. **Object** étant l'objet racine de tout objet en JS, chaque objet va hériter de méthodes d'instance de l'objet **Object.prototype**.

Nous avons par ailleurs mieux compris le concept de prototype. En JavaScript, le prototype est un objet qui contient les propriétés et les méthodes que toutes les instances d'un type auront en commun.

Ce cours nous a permis d'acquérir une compréhension approfondie des concepts de la POO en JavaScript et des objets natifs de base en JavaScript. Bien évidemment, nous avons abordé un nombre limité d'exemples de propriétés et de méthodes, le but étant avant tout de se concentrer sur l'ensemble des objets natifs en JS. Vous pourrez trouver pour chaque objet natif les méthodes statiques et d'instance qui lui sont propres via la documentation Mozilla : [mdn¹](https://developer.mozilla.org/fr). C'est un outil plus qu'essentiel pour un développeur en JavaScript, car elle est très complète et accessible.

V. Auto-évaluation

A. Exercice

Vous travaillez dans une entreprise de e-commerce et vous cherchez à créer un objet représentant un produit précis. Ce produit est un téléphone qui a les caractéristiques suivantes :

- Marque : Apple
- Modèle : iPhone 11
- Ram : 4
- Stockage : 128
- Prix : 500

Les propriétés **marque** et **modele** seront définies sur une instance de **String**. Les propriétés **ram**, **stockage** et **prix** seront définies sur une instance de **Number**.

Question 1

[solution n°3 p.17]

Concevoir l'objet correspondant à ces caractéristiques en commençant par appeler le constructeur **Object**. Vous devrez donc définir les propriétés du téléphone non pas sur des valeurs primitives mais sur des instances d'objets. Puis, hors de la définition de l'objet, afficher la valeur primitive stockée dans la propriété **marque** puis **ram**.

Question 2

[solution n°4 p.17]

Vous souhaitez définir des méthodes : une méthode permettra d'ajouter un montant au prix et de renvoyer le prix modifié, tandis qu'une autre permettra de retirer un montant au prix et de renvoyer le prix modifié. Enfin, une méthode permettra d'afficher le prix du produit. Définir ces méthodes et les appeler via l'objet produit.

B. Test

Exercice 1 : Quiz

[solution n°5 p.18]

Question 1

¹ <https://developer.mozilla.org/fr>

Quelle syntaxe permet de créer un objet en JS sans appeler explicitement le constructeur `Object()` ?

- ☐ `object = {}`
- ☐ `object = []`
- ☐ `object = ()`

Question 2

Quel est le constructeur d'objet natif pour les tableaux en JS ?

- ☐ `Array()`
- ☐ `Object()`
- ☐ `String()`

Question 3

Quel est le constructeur d'objet natif pour les dates en JS ?

- ☐ `Date()`
- ☐ `Time()`
- ☐ `Moment()`

Question 4

Qu'est-ce que le prototype en JS ?

- ☐ C'est une fonction qui crée des objets
- ☐ C'est un objet qui contient les propriétés héritées par les instances d'un type
- ☐ C'est une méthode qui permet la surcharge de fonctions

Question 5

Quel est le prototype de `Number` ?


- ☐ `Number.prototype`
- ☐ `Number.proto`
- ☐ `Number/prototype`

Solutions des exercices

Exercice p. 6 Solution n°1**Question 1**

Qu'est-ce qu'un objet en JS ?


- ☐ Une variable qui stocke une valeur primitive
- ☒ Une entité, un ensemble de propriétés
- ☐ Un type de donnée prédéfini

 Un objet est un ensemble de propriétés définies sur des valeurs ou des méthodes. Les propriétés définies sur des valeurs sont des variables qui stockent des valeurs, tandis que les méthodes sont des fonctions qui peuvent être appelées via l'objet, des propriétés définies sur une fonction.

Question 2

Un objet est une instance d'une classe.


- ☒ Vrai
- ☐ Faux
- ☐ Seulement si la classe est Object()

 Un objet est une instance d'une classe, c'est-à-dire une occurrence de cette classe. Nous pouvons instancier une classe grâce aux méthodes appelées « *constructeur* ».

Question 3

Qu'est-ce qu'un type ?


- ☐ Une variable qui stocke une valeur primitive
- ☐ Le nom d'un objet spécifique
- ☒ La catégorie d'un objet qui définit ses propriétés, ou d'une variable

 Un type en JavaScript est la catégorie d'un objet qui définit ses propriétés, ou la catégorie d'une variable (nous parlons alors de type primitif).

Question 4


Qu'est-ce qu'une propriété en JavaScript ?

- ☐ Obligatoirement une méthode
- ☒ Une variable qui peut stocker par exemple une valeur dans un objet
- ☐ Le nom d'un objet spécifique

 Une propriété en JavaScript est une variable qui peut stocker une valeur dans un objet ou une fonction (méthode).

Question 5


Qu'est-ce qu'une méthode en JavaScript ?

- ☐ Une variable qui stocke une valeur dans un objet
- ☒ Une fonction appartenant à un objet
- ☐ Le nom d'un objet spécifique
-  Une méthode en JavaScript est une fonction appartenant à un objet. C'est une propriété de cet objet, mais qui n'est pas définie sur une valeur mais bien sur une fonction.

Exercice p. 12 Solution n°2


Question 1

Comment créer un objet Array en utilisant un constructeur ?

- ☐ `let array = Array();`
- ☒ `let array = new Array();`
- ☐ `let array = []`
-  Pour créer un objet Array en JavaScript en utilisant le constructeur Array, on peut utiliser la syntaxe `new Array()`. On peut aussi passer comme argument les éléments du tableau séparés par des virgules.


Question 2

Comment créer un objet Boolean en appelant un constructeur ?

- ☒ `let bool = new Boolean();`
- ☐ `let bool = Boolean.create();`
- ☐ `let bool = Boolean();`
-  Pour créer un objet Boolean, on peut appeler le constructeur `Boolean()` en utilisant la syntaxe `new Boolean()`. On peut passer le booléen enveloppé comme argument du constructeur.


Question 3

On peut instancier Math.

- ☐ Vrai
- ☒ Faux
- ☐ Dans certains cas
-  Math est un objet statique, il ne peut donc pas être instancié.


Question 4

Où utiliser le mot-clé `this` pour faire référence à l'instance courante ?

- ☐ En dehors de toute méthode
- ☐ Comme argument lors de l'appel du constructeur
- ☒ Dans la définition d'une méthode de l'objet
-  Le mot-clé `this` est utilisé pour faire référence à l'instance courante. Il est souvent utilisé dans la définition d'une méthode pour se référer à l'objet qui appellera la méthode.

Question 5

Comment définir une méthode qui sera héritée par toutes les instances de Array ?

- ☒ `Array.prototype.newMethod = function() { /*instruction;*/ }`
- ☐ `Array.newMethod() = function() { /*instruction;*/ }`
- ☐ `Array.newMethod = function() { /*instruction;*/ }`
-  Pour définir une méthode d'instance personnalisée, qui sera accessible à toutes les instances de Array, on crée une méthode dans l'objet `Array.prototype`. Toutes les instances de Array hériteront des méthodes de `Array.prototype`.

p. 13 Solution n°3

Voici un code qui fonctionne :

```
1 let produit = new Object();
2
3 produit = {
4   marque: new String("Apple"),
5   modele: new String("Iphone 11"),
6   ram: new Number(4),
7   stockage: new Number(128),
8   prix: new Number(500)
9 }
10
11 console.log(produit.marque.valueOf());
12
13 console.log(produit.ram.valueOf());
```

Étant donné que nous avons défini les propriétés sur des objets, des instances, on utilise la méthode `valueOf()` pour récupérer la valeur stockée dans chaque objet.

p. 13 Solution n°4

Voici un code qui fonctionne :

```
1 let produit = new Object();
2
3 produit = {
4   marque: new String("Apple"),
5   modele: new String("Iphone 11"),
6   ram: new Number(4),
7   stockage: new Number(128),
8   prix: new Number(500),
9
10  addPrice: function(montant) {
11    this.prix += montant;
12  },
13  moinsPrice: function(montant) {
14    this.prix -= montant;
15  },
16  affPrice: function () {
17    console.log(this.prix);
18  }
19 }
```


```
20
21 produit.moinsPrice(40);
22
23 produit.addPrice(80);
24
25 produit.affPrice();
```

On peut voir qu'on peut appeler les méthodes définies, tout fonctionne.

Exercice p. 13 Solution n°5


Question 1

Quelle syntaxe permet de créer un objet en JS sans appeler explicitement le constructeur `Object()` ?

- ☒ `object = {}`
- ☐ `object = []`
- ☐ `object = ()`
-  La syntaxe `let object = {}` permet de créer un objet. Avec cette syntaxe, le constructeur `Object()` est appelé implicitement.


Question 2

Quel est le constructeur d'objet natif pour les tableaux en JS ?

- ☒ `Array()`
- ☐ `Object()`
- ☐ `String()`
-  Le constructeur d'objet natif pour les tableaux en JS est `Array()`. Il permet de créer des objets représentant des tableaux avec des éléments ordonnés et indexés.

Question 3

Quel est le constructeur d'objet natif pour les dates en JS ?

- ☒ `Date()`
- ☐ `Time()`
- ☐ `Moment()`
-  Le constructeur d'objet natif pour les dates en JS est `Date()`. Il permet de créer des objets représentant des dates et apporte des fonctionnalités utiles pour manipuler les dates.

Question 4

Qu'est-ce que le prototype en JS ?

- ☐ C'est une fonction qui crée des objets
- ☒ C'est un objet qui contient les propriétés héritées par les instances d'un type
- ☐ C'est une méthode qui permet la surcharge de fonctions

- Q En JS, chaque objet a une propriété appelée prototype qui est un objet contenant les propriétés héritées par toutes les instances d'un type.

Question 5

Quel est le prototype de Number ?

- ☒ `Number.prototype`
- ☐ `Number.proto`
- ☐ `Number/prototype`

- Q Le prototype de Number est l'objet `Number.prototype`. Il contient donc les propriétés héritées par défaut par chaque instance de Number.