

L'objet Date

Table des matières

I. Contexte	3
II. Formater des dates avec l'objet Date	3
A. Introduction.....	3
B. Le constructeur Date	3
C. Méthodes et formatage de dates	5
D. Exercice	8
III. Les getters et setters de Date	9
A. Introduction.....	9
B. Les getters de Date	9
C. Les setters de Date	10
D. Exercice	13
IV. Essentiel	14
V. Auto-évaluation	14
A. Exercice	14
B. Test	14
Solutions des exercices	15

I. Contexte

Durée : 1 h

Prérequis : Avoir étudié le cours jusqu'ici

Contexte

Pour développer un site web dynamique, la plupart du temps, il va falloir trouver une solution pour représenter des dates. Cela ne veut pas simplement dire écrire une date statique, mais obtenir la date ou l'heure actuelle, locale, système ou universelle, choisir un fuseau horaire, formater des dates, formater des heures, etc.

En JavaScript, nous pouvons utiliser l'objet **Date**. Cet objet permet de créer des représentations de dates statiques ou dynamiques sous la forme d'objets. Il apporte donc des propriétés dont des méthodes permettant de créer, de traiter et de manipuler les dates d'une manière performante.

Dans ce cours, nous verrons comment utiliser l'objet **Date** en JavaScript. Dans une première partie, nous parlerons du formatage de dates, et dans la seconde partie, nous aborderons plusieurs **getters** et **setters** de l'objet **Date** utiles pour traiter et manipuler les dates. Nous utiliserons donc de nombreux exemples de codes que vous pourrez tester via votre IDE. Ce cours sera aussi composé de vidéos et d'exercices pratiques que vous pourrez réaliser dans l'environnement Replit.

Ce cours ne sera pas exhaustif, vous aurez donc certainement besoin au cours de votre parcours de développement web de vous référer à la documentation Mozilla. Une page décrit l'objet **Date** et ses propriétés :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Date

II. Formater des dates avec l'objet Date

A. Introduction

Avant d'aborder les points fondamentaux de ce cours, faisons le point sur une notion importante.

Attention

Différence entre temps local et temps universel

Dans ce cours, vous verrez à plusieurs reprises les concepts de temps local et de temps universel. Le temps local est basé sur l'heure locale du système d'exploitation de l'utilisateur, qui peut varier en fonction de sa localisation géographique et des paramètres de fuseau horaire de son système. Si vous êtes en France et que votre ordinateur est paramétré sur le fuseau horaire français, alors, le temps local correspondra au fuseau horaire français.

Le temps universel, également appelé Temps Universel Coordonné (UTC), est une norme de temps de référence utilisée pour coordonner les activités à travers le monde. Il est indépendant de toute localisation géographique et ne change pas avec les fuseaux horaires.

B. Le constructeur Date

Créer une date

Pour créer une date, nous allons instancier **Date**, et donc appeler son constructeur. Si nous ne passons pas d'arguments lors de l'appel du constructeur, la date sera définie par défaut sur la date et l'heure actuelles. En revanche, nous pouvons passer comme arguments des informations déterminant une date précise. Nous pouvons noter que le constructeur de **Date** définit des dates sur le fuseau horaire local.

Méthode Appeler le constructeur sans passer de paramètres

Pour commencer, instancions **Date** sans passer de paramètres. Nous pouvons voir que la méthode **toString()** va renvoyer une chaîne contenant la date et l'heure locale dans un format spécifique :

Exemple

```
1 const date = new Date();
2
3 console.log(date.toString()); //date et heure actuelle dans une chaîne
```

Nous pouvons constater que la date et l'heure correspondent bien à la date et l'heure actuelle avec le fuseau horaire local.

Méthode Appeler le constructeur en définissant une date précise

Instancions **Date** en passant comme arguments ordonnés des nombres correspondant à l'année, au mois, au jour, à l'heure et à la minute.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14);
2
3 console.log(date.toString()); //Fri Dec 01 2024 10:14:00 GMT+0100 (heure normale d'Europe
4 centrale)
5
```

Un point important, nous indiquons 11 comme mois, et la date créée est en décembre. C'est normal, le paramètre **monthIndex** commence à 0 pour janvier et va jusqu'à 11 pour décembre. Nous pouvons aussi passer un nombre supérieur à 11, dans ce cas, le nombre de mois supplémentaires est ajouté à la représentation de la date.

Nous aurions aussi pu indiquer les secondes, puis les millisecondes :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.toString()); //Fri Dec 01 2023 10:14:10 GMT+0100 (heure normale d'Europe
4 centrale)
```

Mais nous aurions aussi pu ne pas indiquer d'heure, dans ce cas, une heure par défaut (00:00:00) aurait été définie :

```
1 const date = new Date(2023, 11, 1);
2
3 console.log(date.toString()); //Fri Dec 01 2023 00:00:00 GMT+0100 (heure normale d'Europe
4 centrale)
```

Nous passons donc autant d'arguments que nous voulons, l'important étant qu'ils soient passés dans l'ordre suivant : année, mois, jour, heures, minutes, secondes, millisecondes.

Méthode Appeler le constructeur en définissant une date dans une chaîne

Il est aussi possible de définir une date précise en passant comme argument du constructeur une chaîne de caractères contenant une date par exemple avec le format ISO 8601 (une norme internationale pour la représentation de dates et d'heures), ou avec une représentation textuelle valide (comme celles renvoyées par la méthode d'instance **toString()** de **Date**).

Exemple

```
1 const date = new Date("2023-01-17T12:30:00+02:00");
2
3 console.log(date.toString()); //Tue Jan 17 2023 11:30:00 GMT+0100 (heure normale d'Europe
centrale)
4
5 const date2 = new Date("Tue Jan 17 2023 11:30:00 GMT+0100");
6
7 console.log(date2.toString()); //Tue Jan 17 2023 11:30:00 GMT+0100 (heure normale d'Europe
centrale)
```

C. Méthodes et formatage de dates

Maintenant que nous savons créer une date grâce au constructeur de **Date**, voyons quelques méthodes intéressantes de **Date** et intéressons-nous au formatage de dates.

Il faut savoir que, pour générer une date et une heure, JavaScript se sert d'une date repère correspondant au 1^{er} janvier 1970 à 00:00:00 UTC. Pour obtenir la date et l'heure actuelle, JavaScript récupère le nombre de millisecondes écoulées depuis cette date jusqu'à maintenant. Nous allons d'ailleurs pouvoir le constater en appelant certaines méthodes.

Méthode **Méthodes statiques de Date**

Il existe plusieurs méthodes statiques de **Date**.

La méthode **Date.now()** renvoie le nombre de millisecondes écoulées depuis le 1^{er} janvier 1970 à 00:00:00 UTC.

La méthode **Date.parse()** renvoie, quant à elle, le nombre de millisecondes écoulées depuis le 1^{er} janvier 1970 à 00:00:00 UTC jusqu'à la date passée comme argument sous la forme d'une représentation textuelle valide.

La méthode **Date.UTC()** renvoie le nombre de millisecondes écoulées depuis le 1^{er} janvier 1970 à 00:00:00 UTC, mais permet de passer comme arguments les mêmes arguments que le constructeur **Date()**.

Exemple

```
1 console.log(Date.now()); //nombre de millisecondes écoulées depuis la date repère
2
3 console.log(Date.parse("2023-01-17T12:30:00+02:00")); //1673951400000
4
5 console.log(Date.UTC(2023, 11, 1, 10, 14, 10, 58)); //1701425650058
```

Méthode **Méthodes d'instances de conversion**

Chaque instance de **Date** hérite des méthodes de l'objet **Date.prototype**. Parmi ces méthodes, il y'a des méthodes permettant de convertir les dates dans un format spécifique. Parmi ces méthodes figurent :

- **toDateString()** qui renvoie dans une chaîne la partie « *date* » de la date (donc sans l'heure)
- **toTimeString()** qui renvoie dans une chaîne la partie « *heure* » de la date (donc sans la date)
- **toISOString()** qui renvoie une date au format ISO 8601
- **toUTCString()** qui renvoie une chaîne contenant la date selon le fuseau horaire UTC

Exemple

Voici un exemple pour chacune de ces quelques méthodes d'instance :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.toString()); //Fri Dec 01 2023
4
5 console.log(date.toTimeString()); //10:14:10 GMT+0100 (heure normale d'Europe centrale)
6
7 console.log(date.toISOString()); //2023-12-01T09:14:10.058Z
8
9 console.log(date.toUTCString()); //Fri, 01 Dec 2023 09:14:10 GMT
```

Fondamental Méthode valueOf()

La méthode **valueOf()** permet de renvoyer la valeur primitive enveloppée par l'objet **Date**. La valeur primitive enveloppée par une instance de **Date** correspond au nombre de millisecondes écoulées depuis le 1^{er} janvier 1970 à 00:00:00 UTC, qui permet donc de représenter la date.

Exemple

Prenons un exemple :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.valueOf()); //1701422050058
```

Méthode Formater une date

Voyons maintenant comment formater une date dans une chaîne de caractères. Nous allons donc définir le format d'affichage de notre date et de notre heure. Pour formater notre date, nous allons utiliser 3 méthodes, en choisissant celle qui est la plus appropriée en fonction de ce qui est souhaité :

- **toLocaleDateString()** : si on s'intéresse uniquement à la partie « date ».
- **toLocaleTimeString()** : si on s'intéresse uniquement à la partie « heure ».
- **toLocaleString()** : si on s'intéresse aux parties « date » et « heure ».

Nous pouvons très bien appeler ces méthodes sans passer de paramètres, ce qui renverra une chaîne représentant la date, l'heure où les deux, mais avec le format local (propre à la région locale choisie) :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.toLocaleDateString()); //01/12/2023
4
5 console.log(date.toLocaleTimeString()); //10:14:10
6
7 console.log(date.toLocaleString()); //01/12/2023 10:14:10
```

Nous pouvons constater que, avec ces méthodes, la date et l'heure sont renvoyées dans un format français. En effet, c'est la **locale "fr-FR"** qui est automatiquement prise en compte, puisqu'il s'agit du format local (si votre environnement est bien paramétré en français). Nous pouvons tester des formats propres à d'autres localités par exemple pour le Canada qui est un autre pays francophone :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.toLocaleDateString("fr-CA")); //2023-12-01
4
5 console.log(date.toLocaleTimeString("fr-CA")); //10:14:10
6
7 console.log(date.toLocaleString("fr-CA")); //2023-12-01 10:14:10
```

Nous pouvons constater que le format d'affichage de l'heure n'est pas le même.

Maintenant, voyons comment formater une date en utilisant la méthode **toLocaleDateString()**.

La première chose à faire, c'est de créer un objet, que nous allons appeler **options**, qui va contenir différentes propriétés indiquant le format des différentes unités de temps. Nous pouvons définir les propriétés suivantes :

- **weekday** : pour définir si le jour de la semaine doit être affiché comme un texte complet (par exemple, "lundi") ou en abrégé (par exemple, "lun").
- **year** : pour définir si l'année doit être affichée en format numérique à 2 ou 4 chiffres, ou en format complet (par exemple, "2023").
- **month** : pour définir si le mois doit être affiché en texte complet (par exemple, "mars") ou en abrégé (par exemple, "mar").
- **day** : pour définir si le jour doit être affiché en format numérique avec ou sans zéro devant (par exemple, "01" ou "1").
- **hour** : pour définir si l'heure doit être affichée avec le format 12 ou 24 heures.
- **minute** : pour définir si les minutes doivent être affichées avec ou sans zéro devant (par exemple, "08" ou "8").
- **second** : pour définir si les secondes doivent être affichées avec ou sans zéro devant (par exemple, "08" ou "8").
- **timeZoneName** : pour afficher le nom de la zone horaire (par exemple, "heure d'été d'Europe centrale").

Ces propriétés peuvent être définies sur différentes valeurs selon le type de propriété. Par exemple, pour la propriété **weekday**, on peut utiliser les valeurs **"long"** pour afficher le nom complet du jour de la semaine, **"short"** pour l'abréviation, ou **"narrow"** pour une version très courte. La valeur **"numeric"** permettra pour certaines propriétés d'indiquer qu'on cherche à afficher un nombre. Définissons donc l'objet **options** :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 let options = {day: 'numeric', weekday: 'short', month: 'long', year: 'numeric'};
```

Maintenant, nous pouvons appeler la méthode **toLocaleDateString()** en passant 2 arguments correspondant aux paramètres : **locales** et **options**. Le paramètre **locales** permet de spécifier la locale, c'est-à-dire l'option indiquant selon quelle norme d'affichage doit être présentée la date. D'un pays à un autre, le format d'affichage des dates et heures peut être différent. Pour formater la date nous utiliserons **"fr-FR"**, afin que le formatage se fasse selon la norme française. Le paramètre **options** permet de passer l'objet contenant les options de formatage. Appelons donc notre méthode **toLocaleDateString()** en passant les arguments :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 let options = {day: 'numeric', weekday: 'short', month: 'long', year: 'numeric'};
4
5 console.log(date.toLocaleDateString("fr-FR", options)); //ven. 1 décembre 2023
```

Nous pouvons constater que ça fonctionne, nous avons une date formatée, qui prend en compte la locale française et les options spécifiées. Avec cette utilisation de **toLocaleDateString()**, seulement les propriétés définies dans l'objet **options** sont représentées dans la date formatée. Par exemple, si nous avons défini uniquement la propriété **day** :

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 let options = {day: 'numeric'};
4
5 console.log(date.toLocaleDateString("fr-FR", options)); //1
```

Uniquement le jour du mois au format numérique est affiché. Voyons maintenant dans une vidéo comment formater des dates et heures avec les méthodes **toLocaleTimeString()** et **toLocaleString()**. Le système reste le même, mais nous verrons d'autres propriétés que nous pouvons définir dans l'objet **options**.

D. Exercice

Question 1

[solution n°1 p.17]

Instancier **Date** pour représenter la date suivante : le 03 janvier 2023, 00:00:00, heure locale.

```
1 const date = //code
2
3 console.log(date.toString());
```

Question 2

[solution n°2 p.17]

Définir **FormattedDate** sur la date stockée avec le format français : dd/mm/yyyy.

```
1 const date = new Date (2023, 2, 1);
2
3 const FormattedDate = //code
4
5 console.log(FormattedDate);
```

Question 3

[solution n°3 p.17]

Définir **FormattedDate** sur la date stockée avec le format français similaire à : Lundi 6 avril 2022

```
1 const date = new Date (2023, 2, 1);
2
3 let options = //
4
5 const FormattedDate = //
6
7 console.log(FormattedDate);
```

Question 4

[solution n°4 p.17]

Définir **FormattedTime** sur l'heure stockée avec le format : hh:mm (00:00).

```
1 const date = new Date (2023, 2, 1, 10, 30, 12);
2
3 let options = //code
4
5 const FormattedTime = //code
6
7 console.log(FormattedTime);
```


Question 5

[solution n°5 p.17]

Afficher dans la console la date avec le fuseau horaire universel UTC.

```
1 const date = new Date (2023, 2, 1, 10, 30, 12);  
2  
3 //code
```

III. Les getters et setters de Date

A. Introduction

Comme vous le savez, en JavaScript, les getters et les setters sont des méthodes qui permettent de récupérer ou de définir la valeur d'une propriété sans passer directement par cette propriété. Ils permettent donc de contrôler l'accès aux propriétés d'instance d'une classe. Voyons quelques getters et setters par défaut de l'objet **Date**.

B. Les getters de Date

getFullYear()

Le getter **getFullYear** permet d'obtenir l'année (écrite avec 4 chiffres) de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);  
2  
3 console.log(date.getFullYear()); //2023
```

getMonth()

Le getter **getMonth()** permet d'obtenir le nombre représentant le mois (entre 0 pour Janvier et 11 pour Décembre) de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);  
2  
3 console.log(date.getMonth()); //11
```

getDate()

Le getter **getDate()** permet d'obtenir le jour du mois (entre 1 et 31) de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);  
2  
3 console.log(date.getDate()); //1
```

getDay()

Le getter **getDay()** permet d'obtenir le jour de la semaine (entre 0 pour dimanche et 6 pour samedi) de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getDay()); //5
```

getTime()

Le getter **getTime()** permet d'obtenir le nombre de millisecondes depuis le 1^{er} janvier 1970 à 00:00:00 UTC jusqu'à la date de l'objet **Date**.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getTime()); //1701422050058
```

getHours()

Le getter **getHours()** permet de récupérer l'heure de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getHours()); //10
```

getMinutes()

Le getter **getMinutes()** permet de récupérer les minutes de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getMinutes()); //14
```

getSeconds()

Le getter **getSeconds()** permet de récupérer les secondes de l'objet **Date**. Il fait référence au temps local.

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getSeconds()); //10
```

C. Les setters de Date

setFullYear()

Le setter **setFullYear()** permet de modifier l'année de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getFullYear()); //2023
4
5 date.setFullYear(2024);
6
7 console.log(date.getFullYear()); //2024
```

setMonth()

Le setter **setMonth()** permet de modifier le mois de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getMonth()); //11
4
5 date.setMonth(9);
6
7 console.log(date.getMonth()); //9
```

setDate()

Le setter **setDate()** permet de modifier le jour du mois de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getDate()); //1
4
5 date.setDate(12);
6
7 console.log(date.getDate()); //12
```

setHours()

Le setter **setHours()** permet de modifier l'heure de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getHours()); //10
4
5 date.setHours(11);
6
7 console.log(date.getHours()); //11
```

setMinutes()

Le setter **setMinutes()** permet de modifier les minutes de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getMinutes()); //14
4
5 date.setMinutes(1);
6
7 console.log(date.getMinutes()); //1
```

setSeconds()

Le setter **setSeconds()** permet de modifier les secondes de l'objet **Date**. Il fait référence au temps local.

Exemple

```
1 const date = new Date(2023, 11, 1, 10, 14, 10, 58);
2
3 console.log(date.getSeconds()); //10
4
5 date.setSeconds(1);
6
7 console.log(date.getSeconds()); //1
```

Getters et Setters faisant référence au temps universel

Nous avons principalement parlé de getters et de setters qui font référence au temps local. Il existe cependant des getters et setters faisant référence au temps universel comme :

- **getUTCDate()**
- **setUTCDate()**
- **getUTCMonth()**
- **setUTCMonth()**
- **getUTCFullYear()**
- **setUTCFullYear()**
- **getUTCHours()**
- **setUTCHours()**
- **Etc.**

Tous ces getters et setters jouent exactement le même rôle que les getters et setters listés, mais à la différence qu'ils prennent pour référence le temps universel (UTC) et non le temps local.

Méthode

Utilisation de getters pour afficher une heure dans un format spécifique

Voyons maintenant une vidéo nous montrant comment utiliser les getters de **Date** pour afficher une heure dans un format davantage personnalisé.

D. Exercice

Question 1

[solution n°6 p.18]

Utiliser les getters pour afficher l'heure dans la console avec le format : « *hh heures et mm minutes* » (par exemple « *10 heures et 30 minutes* »).

```
1 const date = new Date (2023, 2, 1, 10, 30, 12);
2
3 let heure = //code
4
5 console.log(heure);
```

Question 2

[solution n°7 p.18]

Modifier le jour de notre objet **date** en le définissant sur 28 (sans modifier la première ligne).

```
1 const date = new Date (2023, 4, 1);
2
3 //code
4
5 console.log(date.toLocaleDateString());
```

Question 3

[solution n°8 p.18]

Nous cherchons à réaliser dans la classe **Evenement** une méthode qui affiche une chaîne contenant le jour du mois et le mois suivi du nom de l'évènement (par exemple : « *20/10 : rdv médical* »). Cette méthode s'appellera **getResume()**. Définir cette méthode en utilisant des getters de **Date**.

```
1 class Evenement {
2   constructor(date, evenement) {
3     this.date = date;
4     this.evenement = evenement;
5   }
6
7   getResume() {
8     //code
9   }
10 }
11
12 const dt = new Date(2023, 10, 20);
13
14 const event = new Evenement(dt, "rdv medical");
15
16 event.getResume(); //appel de la méthode
```

Question 4

[solution n°9 p.18]

Modifier l'année de la date en utilisant le setter approprié. La définir sur 2024.

```
1 const date = new Date (2023, 10, 2);
2
3 //code
4
5 console.log(date.toLocaleDateString());
```

Question 5

[solution n°10 p.19]

Nous utilisons une date définie sur le fuseau horaire UTC+4 (nous pouvons voir dans la chaîne ISO un **+ 04:00** qui définit le fuseau horaire). Le nombre d'heures est de 10. Afficher dans la console le nombre d'heures au même moment dans les pays où le fuseau universel (UTC) est la norme.

```
1 const date = new Date ("2023-12-01T14:30:00+04:00");
2
3 //code
```

IV. Essentiel

L'objet **Date** en JavaScript est un outil essentiel pour travailler avec des dates et des heures. Ce cours a exploré deux aspects clés de l'utilisation de cet objet : le formatage des dates et l'utilisation des getters et setters de l'objet.

Dans la première partie du cours, nous avons appris à utiliser le constructeur **Date()** pour créer des objets représentant des dates et heures. Nous avons également vu comment formater les dates à l'aide des méthodes **toLocaleString()**, **toLocaleDateString()** et **toLocaleTimeString()**. Ces méthodes permettent de convertir les dates en chaînes de caractères avec un format personnalisé pour la date et l'heure, selon le fuseau horaire local.

Dans la deuxième partie du cours, nous avons exploré les getters et setters de l'objet **Date**. Les getters nous permettent de récupérer des informations sur une date spécifique, telles que le jour de la semaine, le jour du mois, le mois et l'année. Les setters, quant à eux, nous permettent de modifier des éléments spécifiques d'une date, tels que l'année, le mois ou le jour.

En résumé, l'objet **Date** est un outil essentiel pour travailler avec des dates et des heures en JavaScript. En utilisant le constructeur **Date()** et les méthodes de formatage, nous pouvons créer et afficher des dates personnalisées. Avec les getters et setters de l'objet, nous pouvons récupérer et modifier des éléments spécifiques d'une date. La maîtrise de l'objet **Date** est donc essentielle pour tout développeur JavaScript, qui à un moment ou à un autre va être amené à traiter des dates et des heures.

V. Auto-évaluation

A. Exercice

Vous cherchez à mettre au point un système de calendrier basé sur des scripts réalisés en POO. La première étape pour vous est de définir une classe **Event** qui contient les différentes propriétés d'un événement : son titre, sa date et sa description. Ces trois propriétés sont des propriétés d'instance dont la valeur sera définie à chaque appel du constructeur de **Event**. La propriété **date** sera définie sur un objet **Date**, qui sera instancié dans le constructeur même. Lors de l'appel du constructeur **Event**, l'utilisateur pourra passer pour le paramètre **date** une chaîne de caractères au format ISO 8601.

Question 1

[solution n°11 p.19]

Définir la classe **Event** avec toutes ces informations, et l'instancier.

Question 2

[solution n°12 p.19]

Définir une méthode d'instance permettant d'afficher pour chaque événement une chaîne avec le format :

"----- jj/mm/aaaa -----"

Titre de l'évènement : {titre}

Description : {description}"

Le format "jj/mm/aaaa" correspond si nous prenons un exemple à "12/07/2023".

Appeler la méthode définie.

B. Test

Exercice 1 : Quiz

[solution n°13 p.19]

Question 1

Quelle méthode permet de récupérer le jour de la semaine d'une date en JavaScript ?

- ☐ `getDay()`
- ☐ `getMonth()`
- ☐ `getFullYear()`

Question 2

Quelle méthode permet de modifier le jour d'une date ?

- ☐ setTime()
- ☐ setDate()
- ☐ fromMilliseconds()

Question 3

Comment créer une instance de l'objet **Date** à partir d'une chaîne de caractères représentant une date en JavaScript avec le format ISO 8601 ?

- ☐ En utilisant le constructeur Date() sans argument
- ☐ En utilisant le constructeur Date() et en passant la chaîne comme argument
- ☐ En utilisant la méthode toISOString() de l'objet Date

Question 4

Quelle méthode permet de modifier le mois d'une date en JavaScript ?

- ☐ setMonth()
- ☐ setDate()
- ☐ setYear()

Question 5

Comment obtenir la partie « *heure* » d'une date sous forme d'une chaîne de caractères avec un format personnalisé en JavaScript ?

- ☐ toLocaleTimeString()
- ☐ toTimeString()
- ☐ toLocaleString()

Solutions des exercices

p. 8 Solution n°1

Sortie attendue : "Tue Jan 03 2023 00:00:00 GMT+0100 (heure normale d'Europe centrale) "

```
1 const date = new Date (2023, 0, 3);  
2 console.log(date.toString());
```

p. 8 Solution n°2

Sortie attendue : « 01/03/2023 »

```
1 const date = new Date (2023, 2, 1);  
2  
3 const FormatedDate = date.toLocaleDateString();  
4  
5 console.log(FormatedDate);
```

p. 8 Solution n°3

Sortie attendue : « mercredi 1 mars 2023 »

```
1 const date = new Date (2023, 2, 1);  
2  
3 let options = {year: 'numeric', month: 'long', day: 'numeric', weekday: 'long'};  
4  
5 const FormatedDate = date.toLocaleDateString("fr-FR", options);  
6  
7 console.log(FormatedDate);
```

p. 8 Solution n°4

Sortie attendue : « 10:30 »

```
1 const date = new Date (2023, 2, 1, 10, 30, 12);  
2  
3 let options = {hour: 'numeric', minute: 'numeric'};  
4  
5 const FormatedTime = date.toLocaleTimeString("fr-FR", options);  
6  
7 console.log(FormatedTime);
```

p. 9 Solution n°5

Sortie attendue : « Wed, 01 Mar 2023 09:30:12 GMT »

```
1 const date = new Date (2023, 2, 1, 10, 30, 12);  
2  
3 console.log(date.toUTCString());
```

p. 13 Solution n°6

Sortie attendue : « 10 heures et 30 minutes »

```
1 const date = new Date (2023, 2, 1, 10, 30, 12);
2
3 let heure = `${date.getHours()} heures et ${date.getMinutes()} minutes`;
4
5 console.log(heure);
```

p. 13 Solution n°7

Sortie attendue : « 28/05/2023 »

```
1 const date = new Date (2023, 4, 1);
2
3 date.setDate(28);
4
5 console.log(date.toLocaleDateString());
```

p. 13 Solution n°8

Sortie attendue : « 20/10 : rdv medical »

```
1 class Evenement {
2   constructor(date, evenement) {
3     this.date = date;
4     this.evenement = evenement;
5   }
6
7   getResume() {
8     console.log(this.date.getDate() + "/" + this.date.getMonth() + " : " + this.evenement)
9   }
10 }
11
12 const dt = new Date(2023, 10, 20);
13
14 const event = new Evenement(dt, "rdv medical");
15
16 event.getResume(); //appel de la méthode
```

p. 13 Solution n°9

Sortie attendue « 02/11/2024 »

```
1 const date = new Date (2023, 10, 2);
2
3 date.setFullYear(2024);
4
5 console.log(date.toLocaleDateString());
```

p. 13 Solution n°10

Sortie attendue : « 10 »

```
1 const date = new Date ("2023-12-01T14:30:00+04:00");
2
3 console.log(date.getUTCHours());
```

Nous pouvons voir que la console affiche 10. Donc, quand, dans un pays ayant pour fuseau horaire UTC+4, une montre affiche 14 heures, alors, l'heure universelle (UTC) affiche 10 heures.

p. 14 Solution n°11

Voici un script qui fonctionne :

```
1 class Event {
2
3   constructor(titre, date, description) {
4     this.titre = titre;
5     this.date = new Date(date); //le paramètre date sera passé comme argument du constructeur
6     this.description = description;
7   }
8 }
9
10 const event = new Event("rdv medical", "2023-12-10T14:30:00", "visite medicale");
```

Date. A ne pas confondre avec l'argument d'instance qui lui est défini sur l'instance de Date.

p. 14 Solution n°12

Voici un script qui fonctionne :


```
1 class Event {
2
3   constructor(titre, date, description) {
4     this.titre = titre;
5     this.date = new Date(date);
6     this.description = description;
7   }
8
9   affEvent() {
10    let dateFormatted = this.date.toLocaleDateString("fr-FR"); //Nous formatons la date avec
11    console.log(`----- ${dateFormatted} -----\nTitre de l'évènement :
12    ${this.titre}\nDescription : ${this.description}`);
13  }
14
15  const event = new Event("rdv medical", "2023-12-10T14:30:00", "visite medicale");
16
17  event.affEvent();
```

Exercice p. 14 Solution n°13

Question 1

Quelle méthode permet de récupérer le jour de la semaine d'une date en JavaScript ?


- ☒ `getDay()`
- ☐ `getMonth()`
- ☐ `getFullYear()`

 La méthode **getDay()** permet de récupérer le jour de la semaine d'une date en JavaScript. Elle renvoie un nombre entier compris entre 0 et 6 qui représente respectivement dimanche et samedi.

Question 2

Quelle méthode permet de modifier le jour d'une date ?


- ☐ `setTime()`
- ☒ `setDate()`
- ☐ `fromMilliseconds()`

 La méthode **setDate()** permet de modifier le jour de la date (jour du mois). C'est un setter de **Date**.

Question 3

Comment créer une instance de l'objet **Date** à partir d'une chaîne de caractères représentant une date en JavaScript avec le format ISO 8601 ?


- ☐ En utilisant le constructeur `Date()` sans argument
- ☒ En utilisant le constructeur `Date()` et en passant la chaîne comme argument
- ☐ En utilisant la méthode `toISOString()` de l'objet `Date`

 Pour créer une instance de l'objet **Date** à partir d'une chaîne de caractères représentant une date en JavaScript, il faut appeler le constructeur **Date()** en passant comme argument la chaîne avec le format ISO 8601.

Question 4


Quelle méthode permet de modifier le mois d'une date en JavaScript ?

- ☒ `setMonth()`
- ☐ `setDate()`
- ☐ `setYear()`

 La méthode **setMonth()** permet de modifier le mois d'une date en JavaScript. Les méthodes **setDate()** et **setYear()** permettent quant à elles de modifier respectivement le jour du mois et l'année d'une date.

Question 5

Comment obtenir la partie « *heure* » d'une date sous forme d'une chaîne de caractères avec un format personnalisé en JavaScript ?

- ☒ toLocaleTimeString()
- ☐ toTimeString()
- ☐ toLocaleString()
-  La méthode **toLocaleTimeString()** permet d'obtenir l'heure d'une date sous forme de chaîne de caractères avec un format propre à une localité et personnalisé.