

Les dates

Table des matières

I. Contexte	3
II. L'objet Date	3
III. Exercice : Appliquez la notion	6
IV. Setters et getters	6
V. Exercice : Appliquez la notion	10
VI. Les manipulations de dates	10
VII. Exercice : Appliquez la notion	13
VIII. Formater les dates	13
IX. Exercice : Appliquez la notion	16
X. Moment.js	16
XI. Exercice : Appliquez la notion	18
XII. Auto-évaluation	19
A. Exercice final	19
B. Exercice : Défi	20
Solutions des exercices	21

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Pré-requis : Connaître les bases de JavaScript

Contexte

En développement, il est fréquent d'avoir à gérer des dates, par exemple pour afficher les dates de création et de mise à jour d'une page web aux internautes, pour calculer une différence entre deux dates, etc. Cependant, la gestion des dates peut s'avérer relativement complexe (différents fuseaux horaires, langues, formatages) : c'est pourquoi nous allons étudier dans ce cours comment gérer les dates en JavaScript avec l'objet `Date`.

II. L'objet Date

Objectifs

- Apprendre les notions de base sur l'objet `Date`
- Savoir créer une date

Mise en situation

Nous allons voir dans cette partie ce que sont l'objet `Date` et son utilisation basique, afin d'afficher dans la console du navigateur une simple date.

Méthode

Une date en JavaScript est généralement représentée de trois manières différentes :

- En format textuel, par exemple `December 17, 1995 03:24:00`
- En *timestamp UNIX*, c'est-à-dire le nombre de secondes écoulées depuis le 1^{er} janvier 1970 à minuit (UTC) (par exemple `1584028448` équivaut au 12 mars 2020 à 15:54:08)
- Sous la forme d'un objet `Date` qui en porte toutes les composantes (année, mois, jour, fuseau horaire...).

Nous allons plus nous attarder sur cette troisième forme, car c'est elle qui va nous permettre le plus facilement de manipuler une date grâce aux nombreuses méthodes que nous offre le langage.

Exemple

```
1 const date = new Date()  
2  
3 console.log(date) // Example in the Firefox console (French version) : Thu Mar 05 2020  
17:20:00 GMT+0100 (heure normale d'Europe centrale)
```

Méthode

Dans l'exemple ci-dessus, nous avons créé une variable `date` en lui assignant un nouvel objet `Date` (`new Date()`).

Comme nous le verrons plus tard, le constructeur de l'objet `Date` peut prendre des paramètres (en les plaçant entre les parenthèses) ; mais si aucun paramètre n'est renseigné, c'est la date courante qui est renvoyée.

Par exemple, le 05/03/2020 à 17 h 20, la console affiche :

- `Thu Mar 05 2020 17:20:00 GMT+0100 (heure normale d'Europe centrale)`

Remarque Fuseaux horaires

Je ne vais rien vous apprendre en vous disant qu'à un instant T, l'heure n'est pas la même partout dans le monde. C'est pour cela que lorsque l'on manipule des dates en informatique, elle possédera le plus souvent une notion de fuseau horaire. Il est donc très important, si l'on souhaite avoir le même jour en entrée et en sortie, d'afficher une date dans le même fuseau horaire avec lequel elle a été saisie.

Si votre code est exécuté dans votre navigateur (dans le cas d'une installation locale), il est fort à parier qu'il traite votre date d'entrée et de sortie avec le même fuseau horaire. Nous ne devrions donc pas avoir de problème. Néanmoins, si votre date est traitée par un autre moyen (par exemple un traitement côté serveur avec repl.it), il se peut qu'elle soit affichée avec un autre fuseau horaire.

Par exemple, si on tente d'exécuter `console.log(new Date())` dans un navigateur, on obtiendra `Thu Mar 05 2020 17:20:00 GMT+0100` (heure normale d'Europe centrale) (le format peut changer). La fin de la chaîne indique que la date est donnée au fuseau horaire GMT+1. Cependant, si on tente d'exécuter ce code dans repl.it, la console de l'interface web affichera `2020-03-05T16:20:00.000Z`. Le caractère Z indique que la date est donnée au fuseau horaire GMT. En soi, même si le décalage n'est que d'une heure, on peut passer d'un jour à l'autre si on exécute ce code entre minuit et 1h du matin. Par contre, si on vérifie la console de notre navigateur (F12 > Onglet Console), nous verrons la date dans le bon fuseau horaire.

Ce qu'il faut retenir, c'est qu'ici notre code est bon dans les deux cas, que la console affiche `Thu Mar 05 2020 17:20:00 GMT+0100` (heure normale d'Europe centrale) ou `2020-03-05T16:20:00.000Z`, car c'est le même moment à deux endroits du globe.

Exemple

```
1 const date = new Date('February 29, 2020 09:30:59')
2
3 console.log(date)
```

Méthode

Dans l'exemple ci-dessus, on a passé en paramètre une date au format chaîne de caractères.

La console affiche :

- `Sat Feb 29 2020 09:30:59 GMT+0100` (heure normale d'Europe centrale)

Si une chaîne passée n'est pas correcte, comme par exemple `const date = new Date('Test')`, alors le constructeur va renvoyer `"Invalid Date"`.

Attention

Cette méthode n'est pas celle qui est privilégiée pour créer une date, car la chaîne de caractères doit être dans un format conforme à une des normes (à RFC 1123 de l'IETF ou à ISO 8601).

Par exemple, vous remarquerez qu'elle est écrite en anglais et avec un formatage particulier.

D'autre part, la chaîne peut être analysée différemment en fonction du navigateur, ce qui retournera différentes dates pour cette même chaîne.

Heureusement, il existe d'autres paramètres qu'il est possible de passer à l'objet `Date()`, que nous allons voir ci-dessous.

Méthode La méthode peut également prendre en paramètres :

- année (obligatoire) : un entier représentant l'année (remarque : si vous ne mettez pas l'année en entier, sachez que 0 à 99 correspond aux années 1900 à 1999).
- mois (obligatoire) : un entier allant de 0 (janvier) à 11 (décembre).
- jour (facultatif) : le jour du mois sous forme d'entier (1 à 31).
- heures (facultatif) : l'heure du jour (0 à 23).
- minutes (facultatif) : les minutes (0 à 59).
- secondes (facultatif) : les secondes (0 à 59).
- millisecondes (facultatif) : les millisecondes (0 à 999).

Nous vous conseillons d'utiliser ces paramètres pour construire vos dates afin d'éviter les problèmes évoqués ci-dessus lors de l'utilisation des dates en chaînes de caractères.

Exemple

```
1 const date = new Date(2020, 1, 25)
2
3 console.log(date) // Tue Feb 25 2020 00:00:00 GMT+0100 (heure normale d'Europe centrale) {}
```

Méthode

L'exemple ci-dessus va afficher dans la console :

- Tue Feb 25 2020 00:00:00 GMT+0100 (heure normale d'Europe centrale)

Comme la notation des mois commence à 0 pour le mois de janvier, la date affichée est donc le 25 février 2020.

Si le mois (paramètre obligatoire) n'est pas renseigné, ex : `const date = new Date(2020)`, l'objet va retourner la date du :

- Thu Jan 01 1970 01:00:02 GMT+0100 (heure normale d'Europe centrale)

Complément Timestamp

En convertissant une date en timestamp, on peut plus facilement faire des comparaisons de dates, des tris, etc., étant donné qu'il s'agira alors d'une simple comparaison de nombres entiers.

Syntaxe À retenir

Pour construire et utiliser des dates en JavaScript, on utilise l'objet `Date()`.

Si on ne lui passe aucun paramètre, il retourne la date courante.

En paramètres, il ne faut pas utiliser les dates en chaînes de caractères (ex : `'February 29, 2020 09:30:59'`), mais utiliser les paramètres "année", "mois"...

Les mois vont **de 0 (janvier) à 11 (décembre)**.

Complément

Date (MDN)¹

¹ https://developer.mozilla.org/fr/docs/web/javascript/reference/objets_globaux/date

III. Exercice : Appliquez la notion

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



Question

Dans cet exercice, il nous faut créer deux dates différentes (`date` et `date2`) et les afficher dans la console avec `console.log()` :

- La date courante
- La date du 25 décembre 2020

Indice :

Attention à la numérotation des mois.

IV. Setters et getters

Objectifs

- Savoir construire une date
- Apprendre à récupérer les données d'une date

Mise en situation

Après avoir découvert l'objet `Date` dans une première partie, nous allons à présent étudier les différentes méthodes dont il dispose et qui vont nous permettre :

- Soit de construire une date (en définissant ses secondes, minutes, etc.) : **les setters**.
- Soit de récupérer ses données (jour du mois, de la semaine, etc.) : **les getters**.

Méthode `setFullYear()`, `setMonth()`, `setDate()`

Comme leurs noms l'indiquent, ces méthodes permettent de définir respectivement l'année complète, le mois ou le jour d'un objet `date` :

- **`setFullYear()`** : prend en paramètre un entier indiquant l'année et éventuellement un entier pour le mois (de 0 à 11) et un entier pour le jour (1 à 31).
- **`setMonth()`** : prend en paramètre un entier définissant le mois (de 0 à 11) et éventuellement un entier pour le jour (1 à 31).
- **`setDate()`** : prend en paramètre un entier définissant le jour. Si la valeur passée est 0, alors la date sera le dernier jour du mois précédent. Si la valeur passée est négative, la date sera définie sur les N jours à partir du dernier jour du mois précédent. *Ex : -10 signifie 10 jours avant le dernier jour du mois précédent.*

L'exemple ci-dessous affiche (dans la console Firefox) :

```
Thu Dec 31 2020 10:07:21 GMT+0100 (heure normale d'Europe centrale)
```

¹ <https://repl.it/>

Exemple

```
1 const date = new Date()
2
3 date.setFullYear(2020)
4 date.setMonth(11)
5 date.setDate(31)
6
7 console.log(date)
```

Méthode **setHours(), setMinutes(), setSeconds(), setMilliseconds()**

Comme elles l'indiquent, ces méthodes permettent de définir les heures, minutes, secondes ou millisecondes d'une date :

- **setHours()** : prend en paramètre un entier définissant l'heure (0 à 23). Peut également prendre en paramètres facultatifs les minutes, secondes et millisecondes.
- **setMinutes()** : prend en paramètre un entier définissant les minutes (0 à 59). Peut également prendre en paramètres facultatifs les secondes et millisecondes.
- **setSeconds()** : prend en paramètre un entier définissant les secondes (0 à 59). Peut également prendre en paramètre facultatif les millisecondes.
- **setMilliseconds()** : prend en paramètre un entier définissant les millisecondes (0 à 999).

L'exemple ci-dessous affiche dans la console :

```
Sat Feb 29 2020 22:30:45 GMT+0100 (heure normale d'Europe centrale)
```

Exemple

```
1 const date = new Date(2020, 1, 29)
2
3 date.setHours(22)
4 date.setMinutes(30)
5 date.setSeconds(45)
6 date.setMilliseconds(500)
7
8 console.log(date)
```

Méthode **setTime()**

Le paramètre passé est le timestamp (nombre de millisecondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée) de la date désirée.

L'exemple ci-dessous affichera :

```
Thu Oct 31 2019 00:00:00 GMT+0100 (heure normale d'Europe centrale)
```

Exemple

```
1 const date = new Date(2020, 1, 29)
2
3 date.setTime(date.getTime(1584028448))
4
5 console.log(date)
```

Méthode `getFullYear(), getMonth(), getDate()`

Ces méthodes permettent de retourner respectivement : l'année complète, le mois ou le jour d'une date donnée.

En fonction de la donnée dont nous avons besoin ou de l'affichage que nous voulons faire de celle-ci, nous pouvons donc choisir avec précision celle qui nous intéresse, sans avoir à faire de manipulations complexes.

Exemple

```
1 const date = new Date(2020, 1, 29)
2
3 console.log(date.getFullYear()) // 2020
4 console.log(date.getMonth()) // 1
5 console.log(date.getDate()) // 29
```

Méthode `getHours(), getMinutes(), getSeconds(), getMilliseconds()`

Ces méthodes permettent de retourner : l'heure, les minutes, les secondes ou les millisecondes d'une date donnée.

Exemple

```
1 const date = new Date(2020, 1, 29, 22, 30, 45, 500)
2
3 console.log(date.getHours()) // 22
4 console.log(date.getMinutes()) // 30
5 console.log(date.getSeconds()) // 45
6 console.log(date.getMilliseconds()) // 500
```

Méthode `getDay()`

- Renvoie le jour de la semaine de la date donnée.
- Retourne un entier de 0 à 6 avec 0 pour dimanche, 1 pour lundi, et jusqu'à 6 pour samedi.

Exemple

```
1 const date = new Date(2020, 1, 29)
2
3 console.log(date.getDay()) // 6
```

Méthode `getTime()`

- Renvoie le timestamp de la date donnée.

Cette méthode peut être utilisée pour affecter une nouvelle date à une date existante (avec `setTime()`).

Exemple

```
1 const date = new Date(2020, 1, 29)
2
3 console.log(date.getTime()) // 1582930800000
```

Méthode `Date.now()`

Renvoie le timestamp actuel.

Exemple

```
1 const date = Date.now()
2
3 console.log(date) // 1583751043304
```

Syntaxe **À retenir**

setFullYear(), setMonth(), setDate() : affecte l'année, le mois ou le jour à une date.

setHours(), setMinutes(), setSeconds(), setMilliseconds() : affecte les heures, minutes, secondes ou millisecondes à une date.

setTime() : affecte une date à partir d'un timestamp.

getFullYear(), getMonth(), getDate() : récupère l'année, le mois ou le jour d'une date donnée.

getHours(), getMinutes(), getSeconds(), getMilliseconds() : récupère l'heure, les minutes, secondes ou millisecondes d'une date donnée.

getDay() : récupère le jour de la semaine d'une date donnée (0 dimanche à 6 samedi).

getTime() : récupère le timestamp d'une date donnée.

Date.now() : récupère le timestamp actuel.

Complément

setFullYear()¹
setMonth()²
setDate()³
setHours()⁴
setMinutes()⁵
setSeconds()⁶
setMilliseconds()⁷
setTime()⁸
getFullYear()⁹
getMonth()¹⁰
getDate()¹¹
getHours()¹²
getMinutes()¹³

¹ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setFullYear

² https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setMonth

³ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setDate

⁴ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setHours

⁵ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setMinutes

⁶ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setSeconds

⁷ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setMilliseconds

⁸ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/setTime

⁹ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getFullYear

¹⁰ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getMonth

¹¹ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getDate

¹² https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getHours

¹³ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getMinutes

```
getSeconds()1
getMilliseconds()2
getDay()3
getTime()4
Date.now()5
```

V. Exercice : Appliquez la notion

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



Question

À partir de la date du 25 décembre 2020, nous voulons :

1. Afficher son timestamp dans la console
2. Afficher le mois dans la console
3. Lui affecter l'heure 20 h 30 et afficher la date dans la console

VI. Les manipulations de dates

Objectif

- Apprendre à manipuler les dates

Mise en situation

Dans la précédente partie nous avons vu comment construire une date et accéder à ses données, nous allons à présent voir comment les manipuler afin, par exemple, d'ajouter des jours à une date donnée et pouvoir ainsi calculer une échéance.

Méthode

Grâce aux différents getters, nous pouvons récupérer les données d'une date et, grâce aux setters, nous pouvons lui en affecter.

Comme les années, mois, jours, etc. sont des entiers, nous pouvons effectuer facilement des opérations mathématiques.

Par exemple, nous souhaitons créer une nouvelle date (`date2`) ayant 1 an, 6 mois et 15 jours de plus que notre date d'origine du 29 février 2020 (`date`).

Pour cela, on utilise les setters de `date2` afin de lui affecter comme valeurs celles que l'on récupère grâce aux getters de `date`, additionnées aux nombres désirés.

¹ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getSeconds

² https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getMilliseconds

³ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getDay

⁴ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/getTime

⁵ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/now

⁶ <https://repl.it/>

L'exemple ci-dessous affiche donc dans la console :

Mon Sep 13 2021 14:32:03 GMT+0200 (heure d'été d'Europe centrale)

Exemple

```
1 const date = new Date(2020, 1, 29)
2 const date2 = new Date()
3
4 date2.setFullYear(date.getFullYear() + 1)
5 date2.setMonth(date.getMonth() + 6)
6 date2.setDate(date.getDate() + 15)
7
8 console.log(date2)
```

Méthode **Un autre exemple avec la manipulation des heures, minutes et secondes**

À partir de la date du 29 février à 21 h 30 m 55 s, nous souhaitons créer une nouvelle date avec 4 heures, 30 minutes et 30 secondes de plus.

Nous créons donc `date2` avec la même année, le même mois et le même jour que la date initiale (`date`).

Puis, de la même manière que précédemment, nous utilisons les setters de `date2` afin de lui affecter comme valeurs celles que l'on récupère grâce aux getters de `date`, additionnées aux nombres désirés.

L'exemple ci-dessous affiche donc dans la console :

Sun Mar 01 2020 02:01:25 GMT+0100 (heure normale d'Europe centrale)

Exemple

```
1 const date = new Date(2020, 1, 29, 21, 30, 55)
2 const date2 = new Date()
3
4 date2.setFullYear(date.getFullYear())
5 date2.setMonth(date.getMonth())
6 date2.setDate(date.getDate())
7
8 date2.setHours(date.getHours() + 4)
9 date2.setMinutes(date.getMinutes() + 30)
10 date2.setSeconds(date.getSeconds() + 30)
11
12 console.log(date2)
```

Méthode

Le découpage très fin des exemples ci-dessus permet de bien comprendre. Il existe d'autres méthodes qui vont nous permettre de saisir plusieurs propriétés de l'objet `date` en une seule ligne (exemple 1), ou encore en passant directement les paramètres au constructeur de `Date()` (exemple 2).

Exemple

```
1 // Exemple 1
2 const date = new Date(2020, 1, 29, 20, 30, 45)
3 const date2 = new Date()
4
5 date2.setFullYear(
6   date.getFullYear() + 1,
```

```

7   date.getMonth() + 6,
8   date.getDate() + 15
9 )
10
11 date2.setHours(
12   date.getHours() + 4,
13   date.getMinutes() + 30,
14   date.getSeconds() + 30
15 )
16
17 console.log(date2)

1 // Example 2
2 const date = new Date(2020, 1, 29, 20, 30, 45)
3
4 const date2 = new Date(
5   date.getFullYear() + 1,
6   date.getMonth() + 6,
7   date.getDate() + 15,
8   date.getHours() + 4,
9   date.getMinutes() + 30,
10  date.getSeconds() + 30
11 )
12
13 console.log(date2)

```

Méthode Manipulations avec timestamp

Il est également possible de faire des manipulations à partir des timestamps, pour comparer les dates et les trier, par exemple.

Imaginons trois dates que nous voulons comparer, plutôt que de comparer une à une les années, mois, etc., avec les méthodes dédiées (ce qui serait long et fastidieux). Nous allons récupérer et comparer les timestamp des dates.

On peut ensuite comparer et trier ces valeurs, puisque l'on sait que la plus grande est la plus récente : le timestamp JavaScript correspond en effet au nombre de millisecondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée.

L'exemple ci-dessous affiche dans la console :

```

1582930800000
1597520700000
1597521600000

```

Exemple

```

1 const date = new Date(2020, 1, 29)
2 const date2 = new Date(2020, 7, 15, 21, 45)
3 const date3 = new Date(2020, 7, 15, 22)
4
5 console.log(date.getTime())
6 console.log(date2.getTime())
7 console.log(date3.getTime())

```

Syntaxe **À retenir**

Avec les **getters**, nous pouvons récupérer les données nécessaires pour faire des calculs.

Avec les **setters**, nous pouvons construire des dates avec ces nouvelles données calculées en JavaScript.

Grâce aux **timestamps**, nous pouvons comparer et trier plus facilement les dates.

VII. Exercice : Appliquez la notion

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



Question

À partir de la date du 25 décembre 2020 à 20 h 30, nous voulons calculer une nouvelle date qui sera dans 5 ans, 6 mois et 12 heures de plus (vous devez effectuer des calculs en JavaScript) et l'afficher dans la console.

VIII. Formater les dates

Objectif

- Apprendre à formater les dates de différentes façons en fonction de ses besoins.

Mise en situation

Maintenant que nous avons appris à créer des dates, à accéder à leurs données et à les manipuler, nous allons voir comment les formater pour obtenir un rendu correspondant à nos besoins.

Méthode **toLocaleString()**

Pour formater une date, nous allons utiliser cette méthode de l'objet `Date`.

Elle retourne une chaîne de caractères, qui correspond à une date donnée selon une locale.

Elle prend deux paramètres facultatifs :

- **Locales** : une chaîne de caractères de localisation (ex : *fr-FR pour la France, de-DE pour l'Allemagne, etc.*) ou un tableau de ces chaînes de caractères. Si ce paramètre n'est pas précisé, c'est la locale par défaut du navigateur qui sera utilisée.
- **Options** : un objet contenant toutes ou certaines propriétés, que nous n'allons pas détailler ici, mais qui permettent par exemple de définir le fuseau horaire, le format 12 heures (au lieu de 24), la représentation de l'année, du mois, etc.

L'exemple ci-dessous va afficher dans la console :

```
29/02/2020 à 21:30:45
```

```
samedi 29 février 2020 à 21:30
```

¹ <https://repl.it/>

Exemple

```

1 const date = new Date(2020, 1, 29, 21, 30, 45)
2
3 const dateDisplayed = date.toLocaleString('fr-FR')
4
5 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric', hour: '2-
  digit', minute: '2-digit' }
6 const dateDisplayedLong = date.toLocaleString('fr-FR', options)
7
8 console.log(dateDisplayed)
9 console.log(dateDisplayedLong)

```

Méthode **toLocaleDateString()**

Cette méthode est similaire à la première, sauf qu'elle retourne la date (jour, mois, année) sans l'heure. Toutefois, les paramètres dans l'objet `options` permettent quand même de tout afficher si on le souhaite.

L'exemple ci-dessous va afficher dans la console :

```

29/02/2020
samedi 29 février 2020

```

Exemple

```

1 const date = new Date(2020, 1, 29, 21, 30, 45)
2
3 const dateDisplayed = date.toLocaleDateString('fr-FR')
4
5 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
6 const dateDisplayedLong = date.toLocaleDateString('fr-FR', options)
7
8 console.log(dateDisplayed)
9 console.log(dateDisplayedLong)

```

Méthode **toLocaleTimeString()**

Cette méthode est similaire à la première, sauf qu'elle retourne directement l'heure de la date. Toutefois, les paramètres dans l'objet `options` permettent quand même de tout afficher si on le souhaite.

L'exemple ci-dessous va afficher dans la console :

```

21:30:45

```

Exemple

```

1 const date = new Date(2020, 1, 29, 21, 30, 45)
2
3 const dateDisplayed = date.toLocaleTimeString('fr-FR')
4
5 console.log(dateDisplayed)

```

Méthode Intl.DateTimeFormat

Il s'agit d'une autre façon de formater les dates, qu'il est préférable d'utiliser pour des raisons de performance si nous devons en formater un grand nombre

Il faut alors créer un objet `Intl.DateTimeFormat` et utiliser sa propriété `format` en lui passant la date. Cet objet prend en paramètres facultatifs `locales` et `options`.

L'exemple ci-dessous va afficher dans la console :

29/02/2020

samedi 29 février 2020 à 21:30

Exemple

```
1 const date = new Date(2020, 1, 29, 21, 30, 45)
2
3 const dateDisplayed = new Intl.DateTimeFormat('fr-FR').format(date)
4
5 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric', hour: '2-
  digit', minute: '2-digit' }
6 const dateDisplayedLong = new Intl.DateTimeFormat('fr-FR', options).format(date)
7
8 console.log(dateDisplayed)
9 console.log(dateDisplayedLong)
```

Syntaxe À retenir

- `toLocaleString()` : renvoie une chaîne de caractères représentant la date selon une locale.
- `toLocaleDateString()` : renvoie une chaîne de caractères représentant la date (jour, mois, année) selon une locale.
- `toLocaleTimeString()` : renvoie une chaîne de caractères représentant l'heure de la date selon une locale.
- `Intl.DateTimeFormat` : objet permettant de formater les dates, à utiliser en cas de nombreuses dates à formater.

Complément

`toLocaleString()`¹

`toLocaleDateString()`²

`toLocaleTimeString()`³

`Intl.DateTimeFormat`⁴

¹ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/toLocaleString

² https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/toLocaleDateString

³ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/toLocaleTimeString

⁴ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/DateTimeFormat

IX. Exercice : Appliquez la notion

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



Question

À partir de la date du 25 décembre 2020, nous voulons :

1. Afficher la date dans la console au format `jj/mm/aaaa` (ex : `01/11/2019`)
2. Afficher la date dans la console dans un format plus long (ex : `samedi 29 février 2020`)

Indice :

Pour la date complète, les options sont :

```
{ weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
```

X. Moment.js

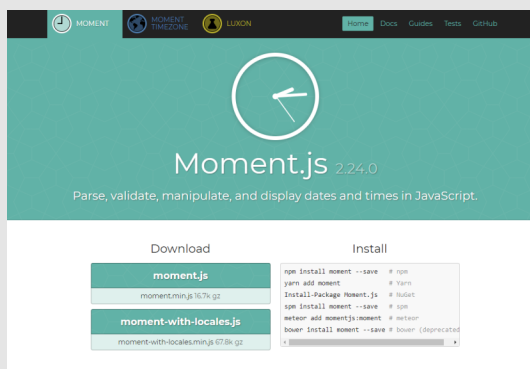
Objectif

- Apprendre les bases sur Moment.js

Mise en situation

Manipuler les dates est possible en JavaScript. Mais pour les traitements complexes, il sera nécessaire d'écrire beaucoup de code pour arriver au résultat souhaité. Des outils ont été créés pour simplifier ces manipulations : nous allons maintenant aborder Moment.js, qui est une librairie spécialisée dans la gestion de dates.

Méthode



Il sera nécessaire d'installer Moment.js manuellement dans notre projet, car il n'est pas directement intégré à JavaScript. Il existe plusieurs manières d'ajouter Moment : nous allons nous rendre sur le site officiel² et cliquer sur « download - moment-with-locales.js ».

¹ <https://repl.it/>

² <https://momentjs.com/>

Tout le code de la librairie devrait être affiché à l'écran en texte brut. Nous allons copier-coller ce code dans un fichier `moment.js`, que nous allons créer dans notre projet.

```
// moment.js
(function (global, factory) {
  typeof exports === 'object' && typeof module !== 'undefined' ? module.exports = factory() :
  typeof define === 'function' && define.amd ? define(factory) :
  (this, (function () { 'use strict';
    var hookCallback;

    function hooks () {
      return hookCallback.apply(null, arguments);
    }

    // This is done to register the method called with moment()
    // without creating circular dependencies.
    function setHookCallback(callback) {
      hookCallback = callback;
    }

    function isArray(input) {
      return input instanceof Array || Object.prototype.toString.call(input) === '[object Array]';
    }

    function isObject(input) {
      // IE8 will treat undefined and null as object if it wasn't for
      // input != null
      return input != null && Object.prototype.toString.call(input) === '[object Object]';
    }

    function isObjectEmpty(obj) {
      if (Object.getOwnPropertyNames(obj).length === 0) {
        return true;
      } else {
        for (var k in obj) {
          if (obj.hasOwnProperty(k)) {
            return false;
          }
        }
        return true;
      }
    }

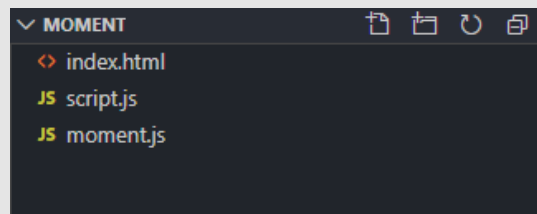
    function isEmpty(input) {
      return input === void 0;
    }

    function isNumber(input) {
      return typeof input === 'number' || Object.prototype.toString.call(input) === '[object Number]';
    }

    function isDate(input) {
      return input instanceof Date || Object.prototype.toString.call(input) === '[object Date]';
    }

    function map(arr, fn) {
      var res = [];
      for (i = 0; i < arr.length; ++i) {
        res.push(fn(arr[i], i));
      }
      return res;
    }
  }));
})(this, function () {
  // ...
});
```

Nous allons ensuite créer un autre fichier JavaScript, que nous nommerons `script.js`. Une fois les deux fichiers créés, nous devrions avoir la structure de fichier suivante :



Enfin, nous allons lier nos deux fichiers JavaScript dans le fichier `index.html`.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8" />
5     <title>Les dates</title>
6     <script type="text/javascript" src="./moment.js"></script>
7     <script type="text/javascript" src="./script.js"></script>
8   </head>
9
10  <body></body>
11 </html>
```

L'installation de Moment est alors finalisée, nous pouvons commencer à utiliser la fonction `moment()`.

Moment.js permet de créer des dates, de les formater, d'en récupérer les données (comme l'année, le mois, le jour de la semaine, le jour de l'année, la semaine de l'année, le numéro du trimestre, etc.), de manipuler les dates (addition / soustraction / différence entre 2 dates), de connaître le nombre de jours dans le mois, et bien d'autres fonctionnalités que vous pourrez retrouver dans la documentation.

Exemple

```
1 moment.locale('fr')
2 let date = moment('2020-02-29')
3
4 // Display : samedi 29 février 2020
5 console.log(date.format('dddd D MMMM YYYY'))
6
7 // Display : 5 (number of the day of the week starting with 0 for Monday)
```

```

8 console.log(date.weekday())
9
10 // Display : 1 (quarter number)
11 console.log(date.quarter())
12
13 // Display : samedi 7 mars 2020 (add 7 days to the initial date)
14 console.log(date.add(7, 'd').format('dddd D MMMM YYYY'))
15
16 let dateEnd = moment('2020-11-01')
17
18 // Display : dans 8 mois
19 console.log(dateEnd.from(date))
20
21 // Display : 239 (number of days difference between these 2 dates)
22 console.log(dateEnd.diff(date, 'days'))
23
24 // Display : false (because dateEnd is not before date)
25 console.log(dateEnd.isBefore(date))
    
```

Syntaxe À retenir

Moment.js est une alternative pour analyser, valider, manipuler et afficher les dates et heures en JavaScript, mais cela nécessite de télécharger la librairie, ce qui alourdit un peu la page chargée par l'internaute.

C'est pour cela que, pour des besoins basiques, il existe également **Day.js** qui est une alternative similaire, mais plus légère.

Complément

Moment.js¹

Day.js² / Sur GitHub³

XI. Exercice : Appliquez la notion

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



Question

Nous souhaitons écrire un script capable de comparer la date d'une commande en ligne avec une période de promotion pour savoir si la commande bénéficie de la remise.

- La commande a été passée le 22 décembre 2019 à 10:45.
- La promotion a débuté le 18 décembre 2019 à 18:00 et dure 10 jours.

¹<https://momentjs.com/>

²<https://day.js.org/>

³<https://github.com/iamkun/dayjs>

⁴<https://repl.it/>

Indice :

- Lors du calcul de la date de fin de promotion, **add()** mute la variable d'origine, effectuez le **add()** sur une copie réalisée avec **moment().clone()**.
- La fonction à utiliser pour savoir si une date est entre deux dates est **isBetween()**.

XII. Auto-évaluation**A. Exercice final****Exercice**

Exercice

`const date = new Date()` est une date créée avec la date courante.

- ☐ Vrai
- ☐ Faux

Exercice

Quelle est la meilleure syntaxe pour déclarer une date ?

- ☐ `const date = new Date('February 29, 2020 09:30:59')`
- ☐ `const date = new Date(2020, 1, 29, 9, 30, 59)`

Exercice

En JavaScript, un timestamp représente :

- ☐ Le nombre de secondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée
- ☐ Le nombre de millisecondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée

Exercice

Quelle est la bonne syntaxe pour la création de la date du 1^{er} Janvier 2020 ?

- ☐ `const date = new Date(2020, 0, 1)`
- ☐ `const date = new Date(2020, 1, 0)`
- ☐ `const date = new Date(2020, 1, 1)`
- ☐ `const date = new Date(2020, 0, 0)`

Exercice

Quelle méthode retourne le jour du mois ?

- ☐ `getDate()`
- ☐ `getDay()`

Exercice

Quelle méthode retourne le timestamp d'une date ?

- ☐ `getTimestamp()`
- ☐ `getTime()`

Exercice

`setFullYear()` permet d'affecter à une date :

- ☐ L'année
- ☐ L'heure
- ☐ Les minutes
- ☐ Le mois
- ☐ Le jour
- ☐ Les secondes

Exercice

`setDate()` permet de définir une date complète.

- ☐ Vrai
- ☐ Faux

Exercice

Quelle syntaxe permet d'afficher 29/02/2020 à partir de : `const date = new Date(2020, 1, 29, 21, 30, 45)` ?

- ☐ `console.log(date.toLocaleDateString('fr-FR'))`
- ☐ `console.log(date.toLocaleString('fr-FR'))`

Exercice

Le code ci-dessous affiche le même résultat dans la console :

```
1 const date = new Date(2020, 1, 29, 21, 30, 45)
2
3 const dateDisplayed = new Intl.DateTimeFormat('fr-FR').format(date)
4 const dateDisplayed2 = date.toLocaleDateString(date)
5
6 console.log(dateDisplayed)
7 console.log(dateDisplayed2)
```

- ☐ Vrai
- ☐ Faux

B. Exercice : Défi

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



¹ <https://repl.it/>

Question

Dans cet exercice, nous voulons gérer les dates des Jeux Olympiques d'été de 2028 à Los Angeles, qui commenceront le 21 juillet 2028 et qui dureront 16 jours. Nous voulons donc afficher toutes les informations utiles concernant cet événement, comme la date de la cérémonie d'ouverture (date de début), la date de la cérémonie de clôture (date de fin), ainsi que le nombre de jours restants entre la date actuelle et la date de début de l'événement.

Il nous faut donc :

1. Créer la date initiale (`dateStart`).
2. Calculer et créer la date de fin (`dateEnd`).
3. Afficher `dateStart` et `dateEnd` au format long (ex : *samedi 29 février 2020*) dans la console.
4. Créer la date courante `currentDate` et calculer la différence (`result`) avec la date de début, puis convertir cette différence en nombre de jours.
5. Afficher la date courante au format long (ex : *samedi 29 février 2020*) dans la console et afficher le nombre de jours de différence dans la console.

Indice :

Pour la date complète, les options sont :

```
{ weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
```

Pour calculer la différence entre la date courante et la date de début, vous allez devoir utiliser la méthode permettant de récupérer le timestamp pour chacune de ces dates, faire le calcul adéquat et convertir le résultat en jours avec la formule :

```
days=result / 1000 / 60 / 60 / 24
```

La formule ci-dessus a été détaillée pour mieux comprendre la conversion des millisecondes en secondes, minutes, heures et jours.

Solutions des exercices

Exercice p. Solution n°1

```
1 const date = new Date()
2 const date2 = new Date(2020, 11, 25)
3
4 console.log(date)
5 console.log(date2)
```

Exercice p. Solution n°2

```
1 const date = new Date(2020, 11, 25)
2
3 // 1) Timestamp
4 console.log(date.getTime())
5
6 // 2) Month
7 console.log(date.getMonth())
8
9 // 3) Add time
10 date.setHours(20)
11 date.setMinutes(30)
12 console.log(date)
13 // 3) Shorter syntax
14 date.setHours(20, 30)
15 console.log(date)
```

Exercice p. Solution n°3

```
1 const date = new Date(2020, 11, 25, 20, 30)
2
3 const date2 = new Date(
4   date.getFullYear() + 5,
5   date.getMonth() + 6,
6   date.getDate(),
7   date.getHours() + 12,
8   date.getMinutes()
9 )
10
11 // Display : Fri Jun 26 2026 08:30:00 GMT+0200 (heure d'été d'Europe centrale)
12 console.log(date2)
```

Une autre solution :

```
1 const date = new Date(2020, 11, 25, 20, 30)
2
3 const date2 = date
4
5 date2.setFullYear(
6   date.getFullYear() + 5,
7   date.getMonth() + 6
8 )
9
10 date2.setHours(date.getHours() + 12)
11
12 // Display : Fri Jun 26 2026 08:30:00 GMT+0200 (heure d'été d'Europe centrale)
```

```
13 console.log(date2)
```

Exercice p. Solution n°4

```
1 const date = new Date(2020, 11, 25, 20, 30)
2 const dateDisplayed = date.toLocaleDateString('fr-FR')
3
4 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
5 const dateDisplayedLong = date.toLocaleString('fr-FR', options)
6
7 // 1) Display : 25/12/2020
8 console.log(dateDisplayed)
9
10 // 2) Display : vendredi 25 décembre 2020
11 console.log(dateDisplayedLong)
```

Ou avec Intl.DateTimeFormat :

```
1 const date = new Date(2020, 11, 25, 20, 30)
2 const dateDisplayed = new Intl.DateTimeFormat('fr-FR').format(date)
3
4 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
5 const dateDisplayedLong = new Intl.DateTimeFormat('fr-FR', options).format(date)
6
7 // 1) Display : 25/12/2020
8 console.log(dateDisplayed)
9
10 // 2) Display : vendredi 25 décembre 2020
11 console.log(dateDisplayedLong)
```

Exercice p. Solution n°5

```
1 const orderedAt = moment('2019-12-22 10:45');
2 const specialOfferStartedAt = moment('2019-12-18 18:00');
3 const specialOfferEndedAt = specialOfferStartedAt.clone().add(10, 'days');
4 const hasReducedPrice = orderedAt.isBetween(specialOfferStartedAt, specialOfferEndedAt);
5
6 if (hasReducedPrice) {
7   console.log('La commande a bénéficié du tarif réduit !')
8 } else {
9   console.log('La commande a été effectuée en dehors de la période promotionnelle.')
10 }
11
```

Exercice p. 19 Solution n°6

Exercice

`const date = new Date()` est une date créée avec la date courante.

☒ Vrai

☐ Faux




C'est vrai, c'est une date créée avec la date courante.

Exercice

Quelle est la meilleure syntaxe pour déclarer une date ?

☐ `const date = new Date('February 29, 2020 09:30:59')`

☒ `const date = new Date(2020, 1, 29, 9, 30, 59)`


 C'est `const date = new Date(2020, 1, 29, 9, 30, 59)` qui est la meilleure syntaxe pour déclarer une date.

Exercice

En JavaScript, un timestamp représente :

☐ Le nombre de secondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée

☒ Le nombre de millisecondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée

 En JavaScript, un timestamp représente le nombre de millisecondes écoulées entre le 1^{er} janvier 1970 à 00:00:00 et une date donnée.

Exercice


Quelle est la bonne syntaxe pour la création de la date du 1^{er} Janvier 2020 ?

☒ `const date = new Date(2020, 0, 1)`

☐ `const date = new Date(2020, 1, 0)`

☐ `const date = new Date(2020, 1, 1)`

☐ `const date = new Date(2020, 0, 0)`


 Les mois vont de 0 (janvier) à 11 (décembre) : on met donc 0 pour signifier janvier, puis 1 pour désigner le 1^{er} jour.

Exercice

Quelle méthode retourne le jour du mois ?

☒ `getDate()`

☐ `getDay()`

 `getDate()` : renvoie le jour du mois.
`getDay()` : renvoie le jour de la semaine.

Exercice

Quelle méthode retourne le timestamp d'une date ?

☐ `getTimestamp()`

☒ `getTime()`

 C'est `getTime()` qui permet de retourner le timestamp d'une date.


Exercice

`setFullYear()` permet d'affecter à une date :

☒ L'année

☐ L'heure

- ☐ Les minutes
- ☒ Le mois
- ☒ Le jour
- ☐ Les secondes

 `setFullYear()` : prend en paramètre un entier indiquant l'année, et éventuellement un entier pour le mois (de 0 à 11) et un entier pour le jour (1 à 31).

Exercice

`setDate()` permet de définir une date complète.


- ☐ Vrai
- ☒ Faux

 Cette méthode permet d'affecter le jour du mois à une date.

Exercice

Quelle syntaxe permet d'afficher 29/02/2020 à partir de : `const date = new Date(2020, 1, 29, 21, 30, 45)` ?

- ☒ `console.log(date.toLocaleDateString('fr-FR'))`
- ☐ `console.log(date.toLocaleString('fr-FR'))`


 `toLocaleDateString()` est similaire à `toLocaleString()`, sauf qu'elle retourne la date (jour, mois, année) sans l'heure.

Exercice

Le code ci-dessous affiche le même résultat dans la console :

```
1 const date = new Date(2020, 1, 29, 21, 30, 45)
2
3 const dateDisplayed = new Intl.DateTimeFormat('fr-FR').format(date)
4 const dateDisplayed2 = date.toLocaleDateString(date)
5
6 console.log(dateDisplayed)
7 console.log(dateDisplayed2)
```

- ☒ Vrai
- ☐ Faux

 `Intl.DateTimeFormat` est une autre façon de formater les dates, qu'il est préférable d'utiliser pour des raisons de performance si nous devons formater de nombreuses dates.

Exercice p. Solution n°7

```
1 const dateStart = new Date(2028, 6, 21)
2 const dateEnd = new Date(
3   dateStart.getFullYear(),
4   dateStart.getMonth(),
5   dateStart.getDate() + 16
6 )
7
8 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
```

```

9
10 // Display : vendredi 21 juillet 2028
11 console.log(dateStart.toLocaleString('fr-FR', options))
12
13 // Display : dimanche 6 août 2028
14 console.log(dateEnd.toLocaleString('fr-FR', options))
15
16 // Current date
17 const currentDate = new Date()
18 // Difference between start date and current date
19 const result = dateStart.getTime() - currentDate.getTime()
20 // Converting milliseconds to days
21 const days = result / 1000 / 60 / 60 / 24
22
23 console.log(currentDate.toLocaleString('fr-FR', options))
24 console.log(days)

```

Ou avec Intl.DateTimeFormat :

```

1 const dateStart = new Date(2028, 6, 21)
2 const dateEnd = new Date(
3   dateStart.getFullYear(),
4   dateStart.getMonth(),
5   dateStart.getDate() + 16
6 )
7
8 const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' }
9
10 // Display : vendredi 21 juillet 2028
11 console.log(new Intl.DateTimeFormat('fr-FR', options).format(dateStart))
12
13 // Display : dimanche 6 août 2028
14 console.log(new Intl.DateTimeFormat('fr-FR', options).format(dateEnd))
15
16 // Current date
17 const currentDate = new Date()
18 // Difference between start date and current date
19 const result = dateStart.getTime() - currentDate.getTime()
20 // Converting milliseconds to days
21 const days = result / 1000 / 60 / 60 / 24
22
23 console.log(new Intl.DateTimeFormat('fr-FR', options).format(currentDate))
24 console.log(days)

```