Labo 3C : fonctions, tableaux, objets

Objectifs

- Construire et manipuler un objet créé à la volée
- Découvrir les types d'objets associés aux éléments HTML d'une page et les manipuler
- Manipuler des données représentées par un tableau d'objets
- (Optionnel) compléments sur les tableaux, dont les tableaux à trous

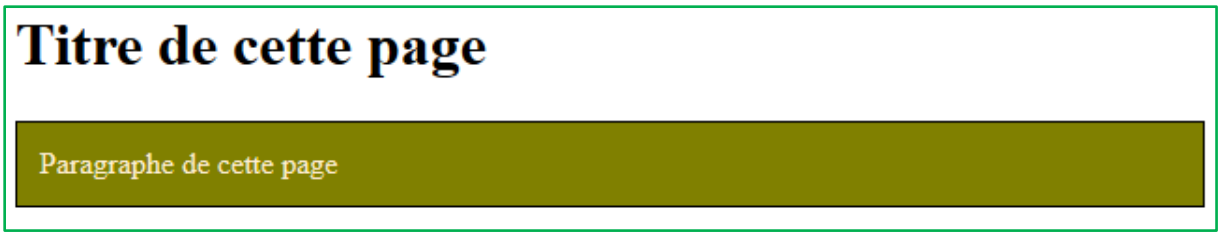
Exercice 1 : un objet à la volée lié à un élément HTML

Le but de cet exercice est de créer un objet Javascript permettant d'effectuer toute une série d'opérations sur un élément d'une page HTML.

Étape 1 : la page HTML et quelques instructions Javascript

Créez un fichier HTML et des styles CSS (vous pouvez les intégrer dans l'en-tête de la page HTML) pour une page web comportant

- un titre en <h1> avec « titre » comme identificateur et
- un paragraphe avec « para » comme identificateur, de fond « olive », avec un texte écrit en couleur « papayawhip », une bordure noire de 1 pixel d'épaisseur et un padding de 12 pixels dans les quatre directions.



Ouvrez cette page sous Firefox et, dans la console Javascript, introduisez une à une les instructions suivantes.

```
let titreHTML = document.getElementById("titre");
let paraHTML = document.getElementById("para");
```

Ces lignes permettent de placer dans les variables `titreHTML` et `paraHTML` des références vers les objets décrivant le titre et le paragraphe de la page web.

Pour vous assurer qu'il s'agit bien d'objets, vous pouvez entrer

```
titreHTML
```

dans la console, ce qui devrait afficher `<h1 id="titre">` suivi d'un symbole (un clic sur ce symbole permet d'ouvrir l'inspecteur HTML sur l'élément HTML en question). Effectuez un clic droit sur `h1` ou sur le symbole puis choisissez « Open in Variables View » pour voir le contenu de l'objet en détail dans la partie droite de la console.

Notez que cet objet a un beau petit paquet de propriétés… Notez entre autres les propriétés `id`, `innerHTML` et `nodeName`.

Vous pouvez faire de même avec `paraHTML`.

Voici quelques instructions Javascript permettant de modifier les éléments en question. Référez-vous à ces exemples pour implémenter les méthodes des étapes suivantes.

```
titreHTML.innerHTML = "Nouveau titre";
paraHTML.style.backgroundColor = "blue";
paraHTML.style.color = "white";
paraHTML.innerHTML += " Et la suite ?";
```

Étape 2 : un objet avec méthode

Effectuez les opérations suivantes directement dans la console, sous la page créée plus haut. Pour éviter de devoir taper plusieurs fois le code en cas d'erreur, vous pourriez avoir envie de tout d'abord l'écrire dans un fichier texte puis de le copier/coller vers la console morceau par morceau.

Définissez, en une instruction Javascript (utilisez donc un littéral pour objets) un objet

- appelé `elem`,
- comportant une propriété `cible` dont la valeur est `titreHTML`,
- et une méthode `changeTexte(txt)` qui remplace le texte de l'élément ciblé par celui qui est donné en argument.

Dans la méthode `changeTexte`, ne faites pas référence directement à `titreHTML` mais utilisez `this.cible`.

Vérifiez votre code en testant les lignes suivantes (ainsi que d'autres si vous le désirez).

```
elem.cible.nodeName                    // devrait afficher H1
elem.changeTexte("My title");
```

La méthode `changeTexte` a été définie directement dans le littéral de l'objet. Vérifiez que vous avez bien utilisé le raccourci syntaxique introduit par la version ES6.

Ajoutez à l'objet `elem` (qui existe déjà) une nouvelle méthode appelée `ajouteTexte(txt)` qui va ajouter le texte donné en argument au contenu de la cible. Ne redéfinissez pas l'objet `elem`… ajoutez-lui juste cette nouvelle propriété !

```
elem.ajouteTexte(" is very interesting");
```

Étape 3 : un objet dans un objet

Examinez la valeur de `paraHTML.style` dans la console.

Notez qu'il s'agit d'un objet, dont vous pouvez observer le contenu en cliquant sur CSS2Properties. Notez qu'on y retrouve les propriétés de style utilisables entre autres en CSS.

Comme `paraHTML.style` est un objet, il devrait vous sembler naturel d'en modifier les propriétés via des lignes telles que

```
paraHTML.style.backgroundColor = "blue";
```

Modifiez `elem.cible` pour qu'il s'agisse désormais de `paraHTML`. Testez l'effet de la modification avec les lignes suivantes.

```
elem.cible.nodeName                    // devrait afficher P
elem.changeTexte("My blah-blah");
```

Ajoutez à l'objet `elem` une méthode `changeFond(col)` qui modifie la couleur de fond de la cible (en `col`) et qui remplace son contenu par le texte « La couleur est désormais *col*. »

Naturellement, le changement de contenu se fera via un appel à l'autre méthode définie plus haut !

```
elem.changeFond("fuchsia");
elem.changeFond("chocolate");
```

Étape 4 : deux dernières méthodes

Ajoutez à l'objet `elem` une méthode `changeCouleur(col)` qui

- stocke dans une propriété `couleurActuelle` de `elem` la couleur donnée (`col`),
- après avoir stocké dans la propriété `ancienneCouleur` la valeur qui se trouvait avant dans `couleurActuelle`,
- et qui modifie la couleur du texte de l'élément HTML ciblé.

Notez qu'initialement, `elem` ne possèdera pas de propriété `couleurActuelle`. Ce n'est pas grave… lors du premier appel à `changeCouleur`, l'expression `this.couleurActuelle` s'évaluera à `undefined` et c'est donc cette valeur qu'on placera dans la propriété `ancienneCouleur`.

Vérifiez le contenu de l'objet `elem` avant et après l'instruction de test suivante.

```
elem.changeCouleur("black");
```

Vérifiez entre autres qu'après exécution, les propriétés `ancienneCouleur` et `couleurActuelle` soient bien présentes et possèdent les valeurs attendues.

Faites de même après l'exécution de l'instruction suivante.

```
elem.changeCouleur("blue");
```

Finalement, ajoutez à l'objet `elem` une méthode `retourCouleur()` qui rétablit la couleur précédente, d'avant le dernier changement. Utilisez les propriétés qui ont été ajoutées ci-dessus et qui sont mises à jour par la méthode `changeCouleur`. Testez votre code.

Exercice 2 : musiciens en pagaille

L'objectif de cet exercice est d'utiliser des objets contenant des informations sur des musiciens pour construire une page web.

Étape 1 : tableau d'objets contenant des objets

Considérez la déclaration Javascript suivante et examinez-en la structure.

```
let musiciens = [
  { nom: "Mozart", prénom: "Wolfgang Amadeus", image: "WAMozart.jpg",
    naissance: {jour: 27, mois: 1, année: 1756},
    mort: {jour: 5, mois: 12, année: 1791} },
  { nom: "Bach", prénom: "Johann Sebastian", image: "JSBach.jpg",
    naissance: {jour: 31, mois: 3, année: 1685},
    mort: {jour: 28, mois: 7, année: 1750} },
  { nom: "Vivaldi", prénom: "Antonio Lucio", image: "Vivaldi.jpg",
    naissance: {jour: 4, mois: 3, année: 1678},
    mort: {jour: 28, mois: 7, année: 1741} },
  { nom: "Ravel", prénom: "Joseph Maurice", image: "Ravel.jpg",
    naissance: {jour: 7, mois: 3, année: 1875},
    mort: {jour: 28, mois: 12, année: 1937} },
  { nom: "Beethoven", prénom: "Ludwig van", image: "Ludwig.jpg",
    naissance: {jour: 17, mois: 12, année: 1770},
    mort: {jour: 26, mois: 3, année: 1827} },
  { nom: "Gershwin", prénom: "Jacob", image: "Gershwin.jpg",
    naissance: {jour: 26, mois: 9, année: 1898},
    mort: {jour: 11, mois: 7, année: 1937} },
  { nom: "Bernstein", prénom: "Leonard", image: "LeoBernstein.jpg",
    naissance: {jour: 25, mois: 8, année: 1918},
    mort: {jour: 14, mois: 10, année: 1990} },
];
```

Copiez-collez cette définition dans la console Javascript puis trouvez les expressions à entrer pour faire afficher les informations suivantes :

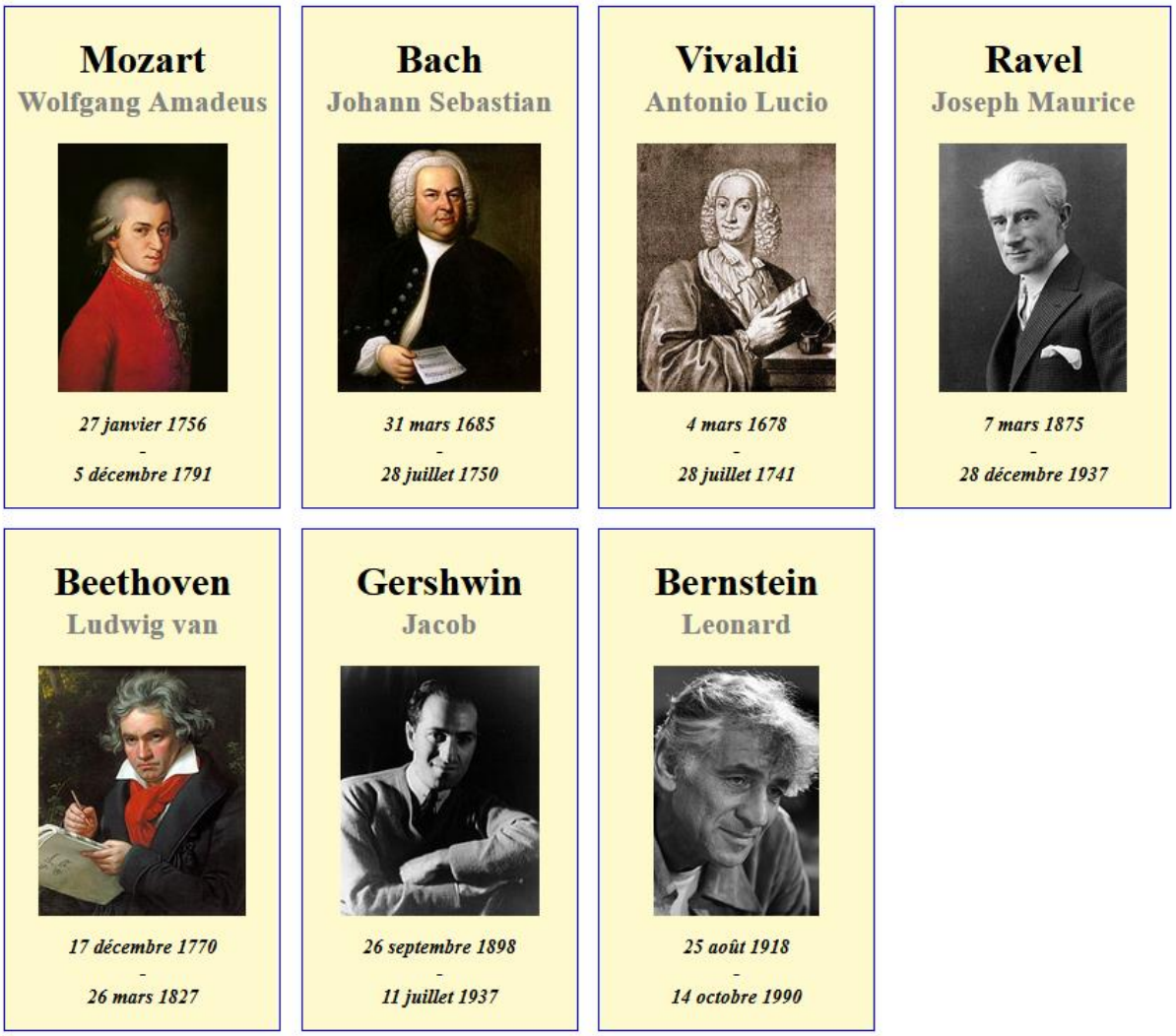
1. le prénom du 4e musicien (Joseph Maurice)
2. le numéro du mois de naissance du 3e musicien (3)
3. l'année de décès du 1er musicien (1791)

Étape 2 : la page web

Créez un répertoire et placez-y les images associées à cet exercice. Ajoutez-y un fichier HTML contenant le squelette d'une page web. Dans la balise `<body>`, ajoutez un script qui sera responsable de la production du contenu de la page (via des `document.write`).

Votre but est d'obtenir une page ressemblant à peu près à la capture d'écran qui suit. Notez que celle-ci utilise des « flexbox » pour l'affichage et que le CSS n'est plus le sujet principal du cours… donc ne perdez pas trop de temps en « peaufinage graphique ».

Toutefois, à titre d'information, quelques-uns des éléments utilisés pour obtenir cette capture d'écran sont indiqués plus bas.



Au niveau Javascript, tenez compte des conseils suivants…

- Votre script devra parcourir tous les éléments du tableau de musiciens. Vous pouvez le faire en utilisant un indice qui ira de 0 à la taille du tableau moins 1, mais il y a plus simple…
- Les gabarits de chaînes de caractères pourraient être utiles…
- Pour convertir un numéro de mois en nom de mois, il peut être utile de définir un tableau reprenant les noms de tous les mois puis d'utiliser le numéro donné pour calculer l'indice (dans le tableau) du mois recherché.
- Vous devez effectuer le même traitement pour la date de naissance et pour la date de décès… ne répétez pas le code : faites une fonction qui reçoit un objet décrivant une date et qui produit le texte à afficher, et faites-y appel deux fois !

Quelques informations sur les styles utilisés pour la capture d'écran :

- Chaque cadre correspond à un `<div>` contenant le nom du musicien (en `<h1>`), son prénom (en `<h2>`), son image et un paragraphe reprenant les dates de naissance et de décès.

- On a utilisé la règle CSS `body { display : flex ; flex-wrap : wrap ; }` pour organiser l'affichage des cadres sous la forme de « flex boxes ».
- Pour faciliter l'écriture des règles CSS, on a donné une classe (« musicien ») à chaque cadre. Chaque cadre fait 220 px de large, possède une bordure bleue de 1 pixel de largeur et un fond de couleur « lemonchiffon » ainsi qu'une marge de 8 pixels.
- Les noms et prénoms des musiciens sont centrés. Les prénoms sont en gris. On a enlevé les marges inférieures de noms et les marges supérieures des prénoms pour éviter les trop grands espaces.
- Les images sont affichées au format block (`display : block`), centrée (il faut utiliser `margin-left` et `margin-right` pour centrer un élément de type « block ») et possèdent une hauteur maximale de 200 pixels (`max-height : 200px`).
- Finalement, les paragraphes sont centrés et écrits en gras italique.

Exercice 3 : résumé de cotes automatique

Le but de cet exercice est d'implémenter des fonctions Javascript permettant de créer un tableau HTML à partir de relevés de cotes stockés dans un objet.

Étape 1 : la page HTML

Créez un document HTML et copiez-y le code suivant.

```
<!doctype HTML>
<html>
  <head>
    <meta charset="UTF-8" />
    <style>
      .réussite, .échec { text-align: center; }
      .réussite { color: blue; }
      .échec { color: red; }
      table { border-collapse: collapse; }
      th { background-color: black; color: lightgrey; padding: 4px; }
      td { padding: 4px; }
      tr.moyennes { background-color: lightgray; }
    </style>
    <script>
    </script>
  </head>
  <body>
    <script>
      afficheCotes(cotes1);
      afficheCotes(cotes2);
    </script>
  </body>
</html>
```

Dans la balise <script> de l'en-tête, ajoutez les trois définitions qui suivent.

```
let nomCours = {
  math: "Math",
  pp: "PP",
  c: "Langage C",
  oed: "OED",
  ppoo: "PP orienté objet",
  java: "Java",
  mtd: "MTD"
}
```

Cette définition reprend les codes de certains cours et leur nom complet. Assurez-vous de bien comprendre la structure utilisée. Les deux autres définitions ci-dessous listent des étudiants dont on donne le nom, la lettre de groupe et diverses cotes. Ici aussi, lisez

attentivement les définitions pour bien en comprendre la structure. Notez bien que tous les étudiants ne passent pas les mêmes cours.

```
let cotes1 = [
  {nom: "Bergamotte", groupe: "A",
   cotes: {math: 12, pp: 17, c: 14} },
  {nom: "Archibald", groupe: "C",
   cotes: {pp: 9, c: 10} },
  {nom: "Cunégonde", groupe: "D",
   cotes: {math: 7, c: 11} },
  {nom: "Diomède", groupe: "B",
   cotes: {pp: 15} },
  {nom: "Garibald", groupe: "A",
   cotes: {pp: 3, math: 5} },
  {nom: "Félix", groupe: "D",
   cotes: {c: 10} },
];
let cotes2 = [
  {nom: "Hubertine", groupe: "A",
   cotes: {mtd: 15, java: 12} },
  {nom: "Isidore", groupe: "D",
   cotes: {oed: 5, ppoo: 7, java: 6} },
  {nom: "Jacynthe", groupe: "B",
   cotes: {mtd: 12, ppoo: 3, oed: 11} },
  {nom: "Kerrouko", groupe: "C",
   cotes: {java: 5, ppoo: 7, mtd: 13} },
  {nom: "Lucienne", groupe: "C",
   cotes: {java: 18, oed: 16, mtd: 18} },
  {nom: "Mathilde", groupe: "A",
   cotes: {ppoo: 4, java: 9, mtd: 8} },
];
```

Étape 2 : le script Javascript

Votre objectif est de vous arranger pour que les appels situés dans le script du corps de la page produisent les tableaux de la capture d'écran présentée à la page suivante.

Quelques remarques et conseils :

- Avant de pouvoir construire le tableau, vous devrez extraire la liste des cours pour savoir quelles colonnes construire. Si vous décidez de stocker cette liste dans un tableau, notez que vous pouvez utiliser les méthodes prédéfinies suivantes :
  - `tab.includes(val)` : indique si le tableau contient la valeur val (renvoie un booléen) ;
  - `tab.push(val)` : ajoute la valeur val dans une nouvelle case à la fin du tableau tab

- Découpez votre code en créant des fonctions.
- La syntaxe de type « tableaux associatifs » pourrait être utiles pour accéder aux valeurs des propriétés de certains objets.
- Utilisez les boucles for-of et for-in à bon escient.
- Organisez-vous pour pouvoir calculer les éléments de la dernière ligne (les moyennes par cours).
- Les styles prédéfinis dans le document HTML peuvent aussi vous guider (et, bien évidemment, vous pouvez les modifier selon vos goûts).

Étudiant	Grp	Langage C	Math	PP
Bergamotte	A	14	12	17
Archibald	C	10		9
Cunégonde	D	11	7	
Diomède	B			15
Garibald	A		5	3
Félix	D	10		
Moyennes		11	8	11

Étudiant	Grp	Java	MTD	OED	PP orienté objet
Hubertine	A	12	15		
Isidore	D	6		5	7
Jacynthe	B		12	11	3
Kerrouko	C	5	13		7
Lucienne	C	18	18	16	
Mathilde	A	9	8		4
Moyennes		10	13	11	5

Exercice 4 : compléments sur les tableaux (facultatif)

L'objectif de cet exercice est de présenter les tableaux comme de simples cas particuliers des objets et de mettre en évidence la notion de « trous » dans un tableau.

Dans les langages comme C et Java, les tableaux correspondent à un bloc de mémoire où les valeurs des différentes cellules sont stockées côte à côte. Avec cette interprétation, la notion de « trou » dans un tableau n'a pas de sens. Mais ce n'est pas comme cela qu'il faut voir les tableaux en Javascript.

Étape 1 : tableaux et propriétés

Définissez les tableaux suivants dans la console Javascript.

```
let tab = [1, 2, 4, 8, 16];
```

puis faites afficher la valeur du tableau en entrant `tab` et cliquez sur le mot `Array` pour dévoiler le contenu de l'objet.

Dans la partie droite de la console, repérez la propriété `length` et vérifiez sa valeur. Ignorez la propriété `__proto__` pour le moment.

Observez que l'objet contient cinq autres propriétés qui, en première lecture, ressemble à des numéros de cellules et à leur contenu.

Étape 2 : le secret des crochets

Évaluez les expressions suivantes une à une dans la console.

```
tab[2]
tab["2"]
tab[4]
tab["4"]
```

En fait, de manière générale, l'écriture `obj[ind]` en Javascript se résout comme ceci :

1. On évalue la valeur de `ind`.
2. Si `ind` n'est pas une chaîne de caractères, on convertit le résultat en chaîne de caractères. (voir les règles de conversion dans les modules précédents)
3. On recherche, dans l'objet `obj` la valeur de la propriété dont le nom est la chaîne de caractères en question.

Pour vous en convaincre, entrez une à une les lignes suivantes.

```
let obj = { valeur: 13, true: 7, nan: -1, NaN : -8 };
obj.valeur
obj.true
obj.nan
obj.NaN
```

Notez qu'ici, les mots « valeur », « true », « nan » et « NaN » sont bien des noms de propriétés ; il ne s'agit pas de la valeur booléenne `true` ni de la valeur numérique `NaN`.

Évaluez les expressions suivantes.

```
obj["valeur"]
obj[true]
obj[1==1]
obj[NaN]
obj[Number("bonjour")]
```

Dans chaque cas, Javascript a évalué l'expression entre crochets puis l'a convertie en une chaîne de caractères avant d'aller chercher la propriété dont le nom correspond à cette chaîne de caractères.

Par exemple, `1==1` s'évalue à `true` (valeur booléenne) qui, une fois convertie en string, donne « true » ; on a donc cherché la valeur de la propriété « true ».

De même, `Number("bonjour")` s'évalue à `NaN` (valeur numérique) qui, une fois convertie en string, donne « NaN » ; on a donc cherché la valeur de la propriété « NaN ».

Étape 3 : tous ces nombres sont des chaînes de caractères !

Ainsi, quand on utilise une expression banale comme tab[3] en Javascript, ce qui se passe sous le capot est un peu plus compliqué que ce qu'on pourrait croire…

(1) La valeur 3 est convertie en chaîne de caractères, à savoir « 3 ».
(2) On recherche, dans l'objet tab, la valeur de la propriété dont le nom est « 3 ».

Heureusement, toutes ces manipulations sont cachées dans le moteur de Javascript et l'utilisateur peut se contenter de penser qu'on accède simplement au contenu de la case numérotée 3.

Pour que tout cela reste invisible à l'utilisateur, cela signifie qu'une affectation telle que

```
a[3] = 7;
```

doit aussi avoir un effet cohérent. Ici, l'effet doit être le suivant :

- Si l'objet `a` possède déjà une propriété appelée « 3 », modifier sa valeur en 7.
- Si l'objet `a` ne possède pas encore de propriété appelée « 3 », ajouter une telle propriété et mettre sa valeur à 7.

Étape 4 : les trous dans les tableaux

Reprenez le tableau de l'étape 1, à savoir

```
let tab = [1, 2, 4, 8, 16];
```

puis ajoutez-lui un nouvel élément en entrant

```
tab[5] = 32;
```

et observez ce qui a changé dans son contenu.

Vous devriez repérer deux changements. Tout d'abord, la longueur est passée à 6. Ensuite, on a ajouté une propriété « 5 » à laquelle on a associé la valeur 32. (Intuitivement, une 6e case ajoutée à la fin du tableau).

Exécutez maintenant la ligne suivante pour ajouter une autre valeur au tableau et observez les changements.

```
tab[8] = 256;
```

Premier changement visible, c'est lors de l'affichage de la nouvelle valeur de `tab` :

```
Array [ 1, 2, 4, 8, 16, 32, <2 empty slots>, 256 ]
```

où on voit apparaître « 2 empty slots ». Second changement, dans le contenu de l'objet `tab` : la longueur est passée à 9 et une nouvelle propriété appelée « 8 » a été ajoutée.

Notez qu'on n'a pas ajouté de propriétés appelées « 6 » ni « 7 » ! D'où la présence de deux « trous » ou « empty slots ».

Étape 5 : les trous, ça ne change rien !

Évaluez les expressions suivantes dans la console Javascript.

```
tab[6]
tab[7]
```

Dans les deux cas, vous devriez obtenir `undefined`, tout simplement parce que c'est la valeur que Javascript renvoie quand on lui demande la valeur d'une propriété qui n'existe pas.

Vous pouvez vous en convaincre en utilisant le code suivant.

```
let objetVide = {};
objetVide.valeur
```

Exécutez la boucle suivante dans la console.

```
for (let i = 0 ; i < tab.length ; i++) console.log(i + "/" + tab[i]);
```

Toutes les valeurs du tableau sont citées et, pour les « cellules » 6 et 7, la valeur donnée est `undefined`.

Une boucle « for-of » donne des résultats similaires.

```
for (let val of tab) console.log(val);
```

Elle cite également les valeurs et deux fois `undefined` pour les cellules vides (notez qu'à l'affichage, Firefox pourrait n'écrire `undefined` qu'une seule fois et indiquer un « 2 » à droite de la ligne plutôt que de répéter le mot deux fois).

Bref, jusqu'ici, un « trou » dans un tableau semble fonctionner comme s'il s'agissait d'une case contenant la valeur `undefined`.

Pour le vérifier, vous pouvez ajouter une 10e case au tableau et y placer la valeur `undefined` avant de tester l'affichage de `tab[9]` et d'exécuter à nouveau les deux boucles ci-dessus.

```
tab[9] = undefined;
```

Étape 6 : les trous, ça change tout !

Mais ce n'est pas toujours le cas… Il y a deux situations où le comportement est différent.

La première est l'utilisation du test `in` permettant de vérifier si un objet contient une propriété d'un nom donné.

```
6 in tab    // indique false car aucune propriété « 6 »
9 in tab    // indique true car il y a bien une propriété « 9 »
```

La seconde situation montrant la différence est dans l'utilisation du « for-in ».

```
for (let i in tab) console.log(i + "/" + tab[i]);
```

L'exécution de cette boucle ignore les trous (parce que « 6 » et « 7 » ne sont pas des propriétés de l'objet `tab`) mais prend bien en considération la cellule « 9 » dont la valeur est `undefined`.

Vous pouvez lire quelques détails de plus sur les « trous » dans les tableaux Javascript dans les slides du cours théorique. À titre d'exercice, vous pouvez également tenter de définir une fonction `nbTrous(tab)` qui compte le nombre de trous dans un tableau donné.