Module 13 (laboratoire)
Exercices récapitulatifs

Exercice 1 : JSON et canvas

La nouvelle norme HTML5 a amené une série d'innovations. Parmi celles-ci, on trouve la nouvelle balise `<canvas>` (ce qui signifie « toile pour peindre » en anglais). Cette balise permet d'insérer au sein d'une page HTML un cadre dans lequel il est possible de « peindre » divers éléments graphiques. Pour modifier le contenu du *canvas*, on utilise Javascript et l'API prévue à cet effet.

> « **API** » (**Application Programing Interface**, ou interface de programmation) est un terme générique qui désigne l'ensemble des éléments informatiques conçus pour permettre ou faciliter l'accès à certaines fonctionnalités. Il peut s'agir par exemple d'une bibliothèque de fonctions (en non orienté objet) ou de classes et de méthodes (en orienté objet) liées à un thème donné.
>
> Le DOM est un exemple d'API (il s'agit d'un ensemble de « classes » et de méthodes Javascript permettant de manipuler le contenu d'une page HTML).

Dans une première partie, on présente quelques éléments de l'API des canvas. Ensuite, on propose quelques exercices simples pour voir comment utiliser cette API. Puis, en troisième lieu vient un énoncé correspondant à un mini-projet relatif aux canvas (pour construire des dessins) et au JSON (pour « sauvegarder » ces dessins).

1.1 Les canvas du côté HTML

La balise HTML5 `<canvas>` permet de définir une zone rectangulaire sur laquelle il sera ensuite possible de « peindre » divers composants. Elle s'utilise comme suit :

```
<canvas width="600" height="400"></canvas>
```

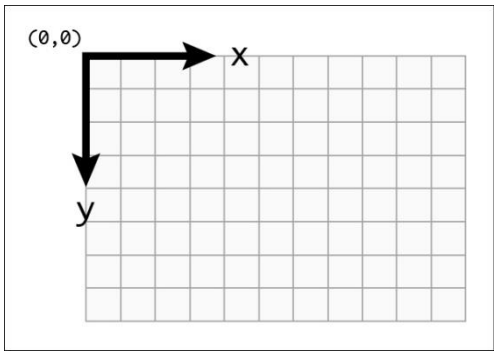
En plus des attributs `width` et `height` qui spécifient la taille de la zone rectangulaire, il peut être utile de spécifier…

- un identifiant `id="…"` pour pouvoir cibler le *canvas* en question dans le code Javascript ;
- un style CSS de type inline `style="…"` ou, mieux, une ou plusieurs classes faisant référence à des règles CSS via `class="…"` (par défaut, le *canvas* est affiché sans aucune couleur de fond et sans bordure).

À l'intérieur de la balise `<canvas>`, on peut ajouter un texte alternatif qui sera affiché par les navigateurs incapables de produire un rendu visuel de cette balise. Par exemple :

```
<canvas width="600" height="400">
  Votre navigateur n'est pas capable
  d'afficher ce contenu ; mettez-le à jour !
</canvas>
```

À l'intérieur du cadre rectangulaire du *canvas*, on repère la position des éléments graphiques grâce à un système de coordonnées standard. Comme bien souvent en informatique, on situe l'origine (c'est-à-dire le point de coordonnées 0, 0) en haut à gauche. La première coordonnée (x) augmente lorsqu'on se déplace vers la droite, tandis que la seconde (y) augmente lorsqu'on se déplace vers le bas (voir graphique ci-contre).



1.2 Une API pour le dessin en 2 dimensions

Avant de pouvoir dessiner sur le `<canvas>`, il faut récupérer une référence Javascript non seulement vers l'élément HTML ciblé mais vers son « contexte ».

```
let monCanvas = document.getElementById("canvas1");
let sonContexte = monCanvas.getContext("2d");
```

La variable `monCanvas` contient une référence à l'objet représentant le *canvas* en tant qu'élément au sein de la page HTML. C'est via cette variable qu'on pourra par exemple modifier le style CSS du cadre (pour ajouter une bordure ou un effet d'ombre).

La variable `sonContexte`, quant à elle, cible le contenu du *canvas* et permet de le modifier en le considérant comme un espace graphique à 2 dimensions (dans ce cas-ci). L'objet ciblé par cette variable porte de nombreuses propriétés (attributs et méthodes) permettant de dessiner sur le *canvas*.

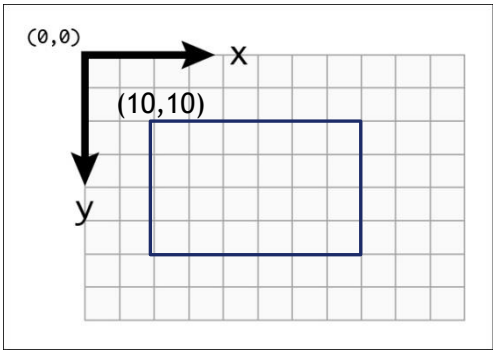
Dans la suite de ce document, on supposera que la variable Javascript `ctx` contient le contexte 2D d'un *canvas*.

[a] Dessiner un rectangle

Un rectangle est déterminé par les coordonnées de son point supérieur gauche, sa longueur et sa hauteur.

```
ctx.strokeRect(10, 10, 30, 20);
```

La méthode `strokeRect` permet de tracer le contour d'un rectangle. Elle prend quatre arguments : les coordonnées du coin supérieur gauche du rectangle, sa largeur puis sa hauteur. Le code ci-dessus dessinera donc un rectangle dont le coin supérieur gauche se situe en (10,10), avec une largeur de 30 unités et une hauteur de 20 unités (voir ci-contre).



Pour dessiner un rectangle plein, on utilisera la méthode `fillRect` au lieu de `strokeRect`. Dans les deux cas, les arguments sont les mêmes.

L'API propose une autre méthode liée aux rectangles : `clearRect` permet d'effacer une portion rectangulaire du *canvas*. Cela peut se révéler utile pour effacer tout le contenu actuel. Si la variable `canvas` se réfère à l'élément HTML, la ligne de code suivante permet de purger tout le contenu du *canvas*.

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

[b] Préciser le style d'un tracé ou le style de remplissage

Pour indiquer le style à utiliser pour tracer des lignes (le contour d'un rectangle par exemple), on utilise principalement les deux outils suivants.

```
ctx.strokeStyle = "#FF0000";
ctx.lineWidth = 5;
```

Les propriétés `strokeStyle` et `lineWidth` permettent d'indiquer respectivement la couleur et l'épaisseur de trait à utiliser pour les prochains tracés. Si on désire utiliser autre chose que les valeurs standards (couleur noire #000000 et épaisseur de 1 pixel), il faut donc les modifier avant de faire appel à une méthode permettant de tracer un trait.

Pour ceux qui voudraient aller plus loin, on peut noter que `strokeStyle` permet également de définir des couleurs sous la forme de dégradés (gradients) et, qu'en plus de `lineWidth`, il existe d'autres propriétés telles que `lineCap`, `lineJoin` et `MiterLimit` pour définir avec plus de précision la manière dont les extrémités de lignes sont dessinées.

```
ctx.fillStyle = "#FF0000";
```

Pour préciser le style de remplissage, on peut se servir de la propriété `fillStyle` pour indiquer la couleur à utiliser (voir exemple ci-dessus). Ici aussi, il est possible d'utiliser un dégradé, voire même un motif à répéter.

[c] Dessiner un chemin

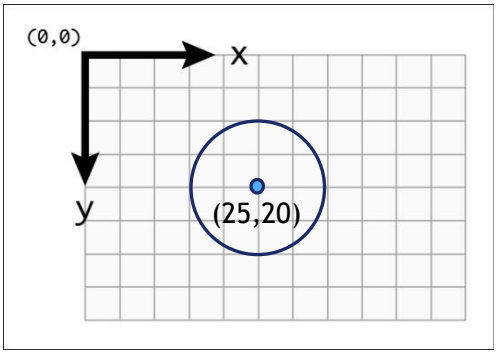
On appelle « chemin » une suite de segments de lignes, qui peuvent former une forme fermée (chemin clos) ou ouverte (chemin ouvert). Intuitivement, un chemin se forme en déplaçant le crayon d'une position à une autre. La plupart du temps, le déplacement d'une position à la suite s'effectue en ligne droite (d'autres méthodes permettent de se déplacer en suivant une courbe).

```
ctx.beginPath();
ctx.moveTo(10, 10);
ctx.lineTo(10, 40);
ctx.lineTo(40, 40);
ctx.closePath();
ctx.stroke();
```

Le code ci-dessus va tracer les contours d'un triangle : il s'agit d'un chemin commençant au point (10,10) puis allant vers le point (10,40) puis vers le point (40,40) avant de revenir à son point de départ (chemin fermé). La méthode `beginPath` permet d'indiquer qu'on commence un nouveau chemin ; la méthode `moveTo` déplace le crayon vers le point de départ ; la méthode `lineTo` trace une ligne vers le point indiqué ; et la méthode `closePath` boucle le chemin.

Finalement, la méthode `stroke` indique qu'il faut tracer le chemin en cours, en utilisant les paramètres de tracé (`strokeStyle` et `lineWidth`) actuels. Pour colorer l'intérieur du chemin, on peut utiliser la méthode `fill` au lieu de `stroke`.

[d] Dessiner un cercle



Les cercles sont considérés comme des cas particuliers de chemins. Pour représenter un cercle, on utilise une méthode plus générale appelée `arc` et permettant de dessiner n'importe quel arc de cercle.

```
ctx.beginPath();
ctx.arc(25, 20, 10, 0, 2 * Math.PI);
ctx.stroke();
```

La méthode `arc` nécessite les arguments suivants (dans l'ordre) :

- les coordonnées du centre du cercle (25, 20 dans l'exemple) ;
- le rayon du cercle (10 dans l'exemple) ;
- les angles qui déterminent le début et la fin de l'arc à tracer ; si on désire tracer le cercle tout entier, il faut aller (en radians) de l'angle 0 à l'angle 2π.

Pour dessiner l'intérieur du cercle (cercle plein), il suffit d'utiliser la méthode `fill` au lieu de `stroke`. Et, comme plus haut, les paramètres de style s'appliquent normalement.

[e] Écrire du texte

Deux méthodes permettent d'écrire du texte. La première effectue le tracé du contour des lettres alors que la seconde colore l'intérieur des lettres (cette dernière correspond à un affichage « normal » de texte).

```
ctx.strokeText("Contour des lettres", 20, 30);
ctx.fillText("Bonjour !", 10, 10);
```

En plus du texte à afficher, ces méthodes nécessitent des coordonnées. La manière dont ces coordonnées sont utilisées dépend des valeurs de deux propriétés : `textAlign` (alignement horizontal) et `textBaseline` (alignement vertical). Les tableaux suivants présentent les principales valeurs que ces paramètres peuvent prendre.

`ctx.textAlign="…"`	Signification
`start / left`	Le texte commence à la position indiquée (il est écrit à la droite de la position)
`end / right`	Le texte se termine à la position indiquée (il est écrit à la gauche de la position).
`center`	Le texte est centré horizontalement sur la position indiquée.

`ctx.textBaseline="…"`	Signification
`top`	Le texte est écrit en-dessous de la position indiquée.
`bottom`	Le texte est écrit au-dessus de la position indiquée.
`middle`	Le texte est centré verticalement sur la position indiquée.
`alphabetic`	Le bas des lettres repose sur la position indiquée (mais les queues dépassent vers le bas). C'est la valeur par défaut.

Pour faciliter le positionnement du texte, on peut aussi utiliser la méthode `measureText`, qui permet de connaître la largeur qu'un texte va occuper une fois affiché à l'écran.

```
let largeurTexte = ctx.measureText("Texte à afficher").width;
```

Finalement, le format du texte est dicté par la valeur de la propriété `font`.

```
ctx.font = "italic bold 30px Arial";
```

La propriété `font` s'utilise avec une syntaxe similaire à celle des règles CSS. Dans la chaîne de caractères, on peut préciser le style (italic par exemple), son poids (bold par exemple), sa taille en pixels et la police de caractères à utiliser.

1.3 Ce que ce document ne dit pas…

Ce document n'a fait que survoler l'API liée à l'édition du contenu des *canvas* en 2 dimensions. Référez-vous à des sites spécialisés (comme w3schools.com) pour un aperçu des autres fonctionnalités).

Certains navigateurs commencent (ou continuent) à travailler sur d'autres types de contextes pour aborder les *canvas*, dont certains contextes orientés 3D (WebGL par exemple).

D'autre part, comme l'API reste assez primitive, diverses bibliothèques se sont développées ici et là sur le net (certaines gratuites, d'autres payantes) pour offrir des possibilités de plus haut niveau. Une simple recherche sur les termes « javascript canvas library » permet de mettre en évidence certaines de ces bibliothèques (telles que KineticJS, GoJS, Fabric, bHive…).

2. Quelques exercices pour se familiariser avec canvas

Créez une page HTML contenant le code suivant.

```
<html>
  <head>
    <meta charset="UTF-8"/>
  </head>
  <body>
    <canvas id="canvas" width="800" height="600"
            style="border:1px solid black">
    </canvas>
    <button onclick="go();">Go !</button>
  </body>
</html>
```

Dans la partie « head », ajoutez un script Javascript pour définir la fonction `go()`, qui se déclenchera lors d'un clic sur le bouton. Ce script accomplira les actions suivantes (conseil : codez les actions une par une pour pouvoir les tester au fur et à mesure).

1. Tracer un rectangle dont le coin supérieur gauche est en (100,100) et le coin inférieur droit est en (400,300).
2. Dessiner un carré plein de couleur rouge (#FF0000) dont le coin supérieur gauche est en (200,200) et le coin inférieur droit est en (300,300).
3. Dessiner un cercle plein de couleur vert (#00FF00) dont le centre est en (400,300) et de rayon 150.
4. Afficher « Bonjour » en plein milieu du canevas, en Arial taille 30.
5. Afficher « Titre du canvas » contre le bord supérieur du canevas, centré verticalement.

6. Demander un nombre à l'utilisateur (via prompt) puis recouvrir le canevas avec une grille de rectangles de couleurs différentes (si le nombre entré est 4, la grille sera de taille 4×4).

3. Mini-projet sur les canvas et JSON

Le but final de cet exercice est de créer une page HTML permettant à l'utilisateur de réaliser une création graphique sous la forme d'un *canvas* auquel il pourra ajouter des cercles de couleurs diverses et donner un titre. Il lui sera également possible de sauvegarder sa création sous la forme d'une chaîne de caractères (selon le standard JSON) qu'il pourra copier/coller dans un fichier en lieu sûr, puis la réutiliser plus tard pour retrouver sa création de départ.

Étape 1

Créez tout d'abord le document HTML qui servira de base à cet exercice. Celui-ci comportera les éléments suivants :

- un *canvas* de taille 800 px × 600 px avec un bord noir d'un pixel de largeur (donnez-lui également un identificateur pour pouvoir le repérer facilement) ;
- un paragraphe constitué de 4 boutons :
    - un bouton « Ajouter » lié à la fonction `ajouter()`, qui permettra d'ajouter une forme sur le *canvas* ;
    - un bouton « Nommer » lié à la fonction `nommer()`, qui permettra de donner un nom à l'œuvre (ou de modifier son nom) ;
    - un bouton « Exporter » lié à la fonction `exporter()`, qui permettra d'exporter l'œuvre sous la forme d'une chaîne de caractères JSON ;
    - un bouton « Importer » lié à la fonction `importer()`, qui permettra d'importer une description d'œuvre au format JSON ;
- une zone de texte qui servira à afficher (en cas d'exportation) ou à recevoir (en cas d'importation) la description de l'œuvre au format JSON.

```
<textarea id="cadreTexte" style="width:800px" rows="8"></textarea>
```

Ajoutez dans la partie « head » ou dans un fichier .js des fonctions `ajouter`, `nommer`, `exporter` et `importer` qui, pour l'instant, seront vides. Vérifiez que tout est en ordre en affichant la page sur votre navigateur.

Étape 2

Pour que vous puissiez tester l'exportation et l'importation en vous échangeant des descriptions au format JSON, il faut tout d'abord se mettre d'accord sur le format des objets et le nom des attributs. Voici la structure de « classe » à utiliser.

Un dessin (fonction constructrice `Dessin`) sera représenté par un objet possédant les attributs suivants :

- `nom` : titre du dessin (initialement une chaîne vide) ;
- `contenu` : tableau reprenant les éléments graphiques du dessin ;

et les méthodes suivantes :

- `ajouter(elem)` : pour ajouter un nouvel élément graphique au dessin (le nouvel élément sera ajouté à la fin du tableau) ;

- `redessiner()` : pour redessiner le *canvas* (elle devra : effacer tout le *canvas*, puis dessiner les éléments graphiques, puis afficher le titre en Arial de 30px de manière à ce que celui-ci apparaisse centré horizontalement et collé contre le bord supérieur du *canvas*).

Un élément graphique (fonction constructrice `Elem`) sera représenté par un objet possédant les attributs suivants :

- `posX` : coordonnée en X du centre du cercle ;
- `posY` : coordonnée en Y du centre du cercle ;
- `rayon` : longueur du rayon ;
- `coul` : couleur de l'élément (chaîne de caractères) ;

et la méthode suivante :

- `redessiner()` : pour redessiner l'élément en question sur le *canvas*.

Naturellement, les méthodes seront placées sur les prototypes, pas sur chaque objet.

Définissez les fonctions constructrices ainsi que les méthodes.

Alternativement, vous pouvez réaliser ces définitions en utilisant la nouvelle syntaxe de classe de l'ES6.

Étape 3

Les méthodes `redessiner` (pour le dessin et pour les éléments) utilisent des références à l'élément HTML correspondant au *canvas* et à son contexte. Recréer à chaque fois ces références via `document.getElementById` n'est sans doute pas une bonne idée, car cela implique de nombreuses recherches à travers tout le document.

(Dans ce cas-ci, le document n'est pas très grand, mais autant prendre de bonnes habitudes !)

Une première solution pourrait être de réaliser cette recherche une fois pour toute et d'utiliser des variables globales. Peut-on se contenter d'ajouter les deux lignes suivantes au début du script qui se trouve dans la section <head> ? Pourquoi ?

```
const canvas = document.getElementById("…");
const ctx = canvas.getContext("2d");
```

Comment résoudre ce problème ? Plusieurs méthodes sont possibles mais la plus élégante consiste sans doute à attendre que l'entièreté de la page HTML soit chargée avant d'initialiser ces variables. Pour cela,

- définissez une fonction `init()` qui se chargera d'initialiser ces deux variables (assurez-vous qu'elles soient bien déclarées de manière globale) ;
- ajoutez la ligne suivante dans le script, de sorte que la fonction `init` soit appelée automatiquement dès la fin du chargement complet de la page web.

```
window.onload = init;
```

Note/Rappel. On aurait pu se passer de donner un nom à la fonction `init` en utilisant une syntaxe comme `window.onload = function () { … }`.

Étape 4

La méthode `redessiner` pour le dessin fait appel à la méthode `redessiner` pour chacun des éléments contenus dans le dessin. Vous avez sans doute implémenté cela en utilisant une boucle `for` ? Vous pouvez tenter de réécrire cette implémentation en utilisant plutôt la méthode `forEach`.

Étape 5

Le dessin en cours sera représenté par une variable globale. Déclarez et initialisez-la.

Codez la fonction `nommer`, qui est appelée lors d'un clic sur le second bouton et est censée demander le titre de l'œuvre à l'utilisateur (via prompt) puis mettre à jour l'affichage en faisant appel à la méthode `redessiner` du dessin.

Étape 6

Implémentez maintenant la fonction `ajouter`, censée demander à l'utilisateur les diverses informations d'un élément (coordonnées du centre, rayon et couleur) qui devra être ensuite ajouté au contenu du dessin et dessiné à l'écran.

Les trois premières informations demandées à l'utilisateur doivent être des nombres… vérifiez que c'est bien le cas ! Pensez clean code et factorisation du code : ne répétez pas 3 fois le même code ! Une fonction auxiliaire pourrait être la bienvenue !

Et pour aller plus loin dans la vérification, testez non seulement qu'il s'agit bien de nombre mais, en plus, (a) que la coordonnée en x se situe entre 0 et la largeur du *canvas*, (b) que la coordonnée en y se situe entre 0 et la hauteur du *canvas*, et (c) que le rayon du cercle est suffisamment petit pour que le cercle tout entier tienne dans le *canvas*. Là encore, creusez-vous la tête pour faire tous ces tests sans répéter de code inutilement.

Étape 7

Maintenant que les fonctionnalités de base sont codées, il est temps de s'intéresser aux options d'importation et d'exportation. Celles-ci utiliseront la zone `textarea` soit pour afficher la chaîne d'exportation au format JSON soit pour en lire le contenu et l'importer.

Vous pouvez utiliser la syntaxe suivante pour manipuler le contenu de la zone textarea.

```
document.getElementById("cadreTexte").value = …    // exportation
txt = document.getElementById("cadreTexte").value // importation
```

L'API liée au format JSON propose deux méthodes permettant d'effectuer les conversions nécessaires : `JSON.stringify(obj)` donne la chaîne de caractères JSON correspondant à l'objet en question, alors que `JSON.parse(chaine)` renvoie un objet correspondant à la chaîne de caractères donnée.

Dans le cas de l'exportation, il suffit d'afficher la chaîne produite dans la zone textarea.

Dans le cas de l'importation, il faut non seulement charger en mémoire l'œuvre correspondant à la chaîne donnée mais également déclencher son affichage en appelant la méthode `redessiner`. Cela peut nécessiter quelques ajustements supplémentaires…

Étape 8

Pour aller plus loin : autorisez non seulement des cercles mais également des rectangles dans le *canvas*. (Dans un premier temps, ignorez l'exportation et l'importation via JSON).

Pour ce faire, revoyez la structure des « classes » en ne gardant dans Elem que les propriétés communes aux cercles et aux rectangles (à savoir la couleur et les coordonnées d'un point) puis en créant deux « sous-classes » Cercle et Rectangle. À part cette restructuration et la mise en place d'un héritage, aucune autre modification ne devrait être nécessaire dans cette partie du code.

Étape 9

Finalement, modifiez votre code pour que l'importation et l'exportation fonctionnent à nouveau. Cela peut nécessiter quelques changements dans la structure des objets…

Exercice 2 : examen janvier 2016

Dans le jeu « Simon », on présente à l'utilisateur une séquence qu'il doit ensuite répéter. Il peut par exemple s'agir de pousser, dans le bon ordre, sur une séquence de boutons colorés.

Première partie

Dans le cadre de cet exercice, on va supposer qu'il s'agit plutôt d'une séquence d'entiers positifs (comme par exemple 11, 17, 2, 14, 9) qui est affichée pendant un court moment et que l'utilisateur doit ensuite rentrer les uns après les autres. On va s'intéresser ici au code qui permettra de (a) créer une séquence aléatoire et (b) de tester les entrées de l'utilisateur.

Fonction constructrice. Créez une fonction constructrice permettant de créer des objets représentant une séquence aléatoire. La fonction constructrice prendra deux arguments : (a) la longueur de la séquence, c'est-à-dire le nombre de valeurs aléatoires à générer et (b) un maximum que les valeurs ne devront pas dépasser. Ainsi, pour les arguments 5 et 20, la fonction constructrice devrait créer une séquence de 5 valeurs comprises (au sens large) entre 0 et 20.

Dans l'objet créé, on retiendra (a) un tableau reprenant toutes les valeurs de la séquence et (b) un entier indiquant le nombre de valeurs correctes déjà entrées par l'utilisateur (initialement, zéro).

Note. Pour générer les nombres aléatoires, vous pouvez utiliser la fonction Math.random() qui donne un nombre aléatoire compris dans l'intervalle [0,1[(le nombre peut être 0 mais n'est jamais 1).

Méthode. Ajoutez aux objets représentant des séquences une méthode permettant de vérifier la valeur entrée par l'utilisateur. La méthode en question, appelée `verification(valeur)`, renverra comme résultat deux booléens (trouvez une solution pour le faire !).

• Le premier des booléens indiquera si l'entrée de l'utilisateur est correcte (vrai = il a rentré le nombre attendu ; faux sinon).
• Le second des booléens indiquera si la partie est terminée (vrai = l'utilisateur a fini d'entrer la séquence ou s'est trompé ; faux = l'utilisateur ne s'est pas trompé et il doit encore entrer un ou plusieurs nombres). En d'autres termes, le second booléen indiquera si on doit demander un autre nombre à l'utilisateur.

Utilisation. La page HTML contient entre autre le code suivant.

```
<label for="valeurUtilisateur">Valeur : </label>
<input type="text" id="valeurUtilisateur" />
<button id="bEntrer">Entrer</button>
<p id="resultat"></p>
```

On supposera que la variable globale `seqEnCours` contient l'objet représentant la séquence en cours (dont l'utilisateur a peut-être déjà entré quelques valeurs).

Écrivez du code Javascript pour que, lors d'un clic sur le bouton Entrer, on lise la valeur entrée par l'utilisateur dans le champ « input » (on supposera qu'il s'agit bien d'un entier) et qu'on la teste à l'aide de la méthode `verification` définie plus haut. En fonction des informations retournées par cette méthode, on affichera un message dans le paragraphe identifié par « resultat » :
• en cas d'erreur, « Perdu ! »
• en cas de victoire (l'utilisateur a entré tous les nombres correctement), « Bravo ! »
• en cas d'entrée correcte non finale (l'utilisateur doit entrer d'autres nombres), « Continuez… »

Deuxième partie

Jusqu'ici, on a utilisé une séquence de nombres que l'utilisateur devait répéter. Utilisons désormais des « boutons ». Chacun de ces « boutons » sera en fait représenté par un « div » placé sur la page HTML et caractérisé par deux couleurs (une couleur « éteinte », comme darkgreen, et une couleur « allumée », comme lightgreen).

On suppose qu'on a déjà garni un tableau appelé boutons et contenant un certain nombre d'objets Javascript. Chacun de ces objets Javascript possède trois attributs :
• `offColor` : nom de la couleur HTML correspondant au bouton éteint (par exemple darkgreen) ;
• `onColor` : nom de la couleur HTML correspondant au bouton allumé (par exemple lightgreen) ;
• `elem` : l'élément HTML du DOM qui correspond au bouton (il s'agit d'un « div »).

On a réutilisé le code de la question précédente pour générer une séquence seqEnCours dont chaque entier correspond à un bouton (0 pour le 1er bouton du tableau boutons, 1 pour le 2e, etc.). Le but final de cette question est d'écrire une fonction qui va « présenter » la séquence, en « allumant » tour à tour les boutons correspondant aux entiers de seqEnCours.

1. Écrire une fonction `colore(numBouton,couleur)` qui change la couleur de fond du div correspondant au bouton numéro `numBouton` en couleur.

2. En utilisant la fonction précédente, définir des fonctions `allume(numBouton)` et `eteint(numBouton)` qui allument (en `onColor`) ou éteignent (en `offColor`) un bouton donné.

3. En utilisant éventuellement les fonctions précédentes, définir une fonction `appuie(numBouton)` qui allume le bouton en question pendant 2 secondes.

4. En utilisant la variable globale `seqEnCours` (qu'on supposera déjà garnie correctement), définir une fonction `presenteSequence()` qui va présenter la séquence en allumant tour à tour les boutons correspondant à la séquence. Chaque bouton restera allumé pendant 2 secondes et il y aura une pause de 1 seconde entre le moment où un bouton s'allume et le suivant s'éteint.