1 OBJECTIF

Cette activité permet de lire et modifier du code HTML ou de lire et modifier les champs des formulaires grâce à Javascript. Elle permet également de faire évoluer l'application en fonction des problèmes et demandes rencontrées.

2 SITUATION INITIALE

Nous allons proposer une application pour s'entraîner au calcul mental.

Le code suivant est utilisé :

```
<!doctype html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <title>Calcul mental</title>
</head>
<body>
      <h1>Calcul mental</h1>
      <p>calculez la somme des nombres et saisissez dans le champ réponse.</p>
      <div id="cadreNombre">nombre A<input type="text" id="nombreA" value="15"
readonly></div>
      +
      <div id="cadreNombre">nombre B<input type="text" id="nombreB" readonly></div>
      <div id="cadreReponse">Réponse<input type="text" id="reponse">
      <input type="button" id="repondre" value="repondre" onclick="repondre()"></div>
</body>
<script src="js/calcul.js"></script>
</html>
```

Enregistrer ce code dans votre répertoire de travail avec un nom cohérent, comme TP_Javascript_4_formes.html

Note : il est impératif que le script se trouve à la fin de la page HTML.

Il faut ensuite créer le script calcul.js.

Savez-vous où placer le script calcul.js ?

2.1 MODIFICATION DU SCRIPT JAVASCRIPT

Créez un répertoire que vous appellerez « JS » et à l'intérieur, créez le script « calcul.js »

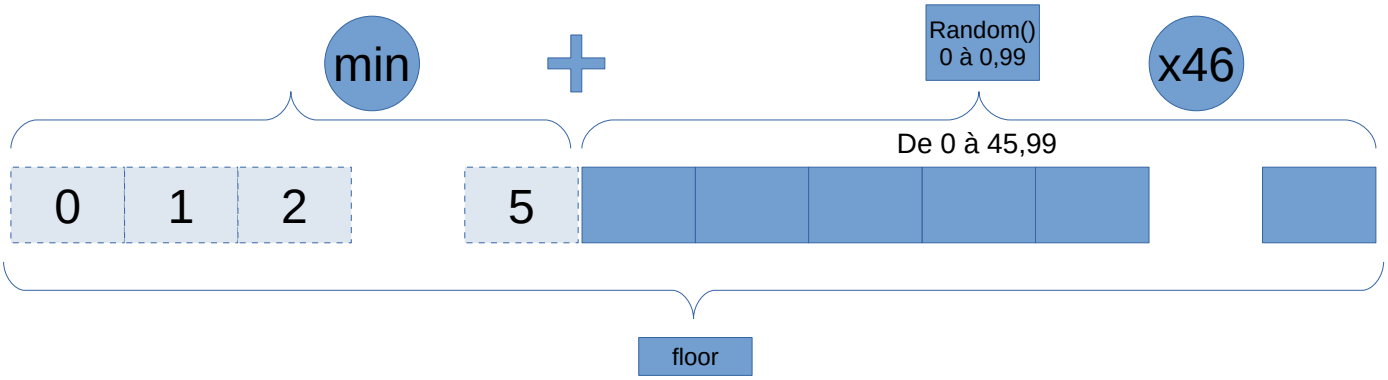
```
// Ce script va permettre d'écrire dans les champs nombreA et nombreB
// et lire la réponse de l'utilisateur, lorsqu'il cliquera le bouton.

function tirageNombre(min, max) {
    // cette fonction tire un nombre au hasard entre min et max
    let nombre = Math.floor(Math.random() * (max-min+1)) + min
        return nombre
}
for (let t=0; t < 50; t++) {
      console.log(tirageNombre(5,50))
}
```

Cette première fonction mérite quelques explications :

on veut tirer un nombre entre 5 et 50 inclus, pour cela nous devons ajouter « min » au résultat pour être certain que le nombre sera supérieur ou égal à « min ».

Si nous ajoutons « min », il faut que le maximum, soit la différence entre « max » et « min » (c'est-à-dire l'écart entre les deux nombres).

Cependant, il faut ajouter un à cet écart, car la fonction floor() va couper le résultat à l'entier inférieur.



On peut vérifier facilement que cela marche en choisissant un nombre « min » égal à zéro et un « max » égal à cinquante, on a alors 50-0+1 = 51 : dans le meilleur des cas, nous aurons 0,9999 x 51 = 50,949. En tronquant le résultat, on trouve bien 50,

Lors de l'affichage de la page HTML, vous pourrez voir le résultat de la fonction tirage : les tirages sont compris entre 5 et 50 inclus.

Corrigez votre code et cherchez les erreurs avant de passer à la section suivante, si vous ne voyez pas la liste de nombres.

2.2 ÉCRITURE DES CHAMPS

JavaScript est capable d'écrire dans les champs que nous avons définis, en utilisant leurs ID.

Il va en effet, rechercher dans le DOM de la page, où se trouve cet ID. On peut utiliser deux instructions qui font le travail de manière similaire.

La plus rapide :

```
index = document.getElementById("nombreA")
```

La plus élégante (le signe # indique qu'il s'agit d'un ID) :

```
index = document.querySelector("#nombreA")
```

Voici donc le nouveau code JavaScript (remplace le précédent) :

```
// Ce script va permettre d'écrire dans les champs nombreA et nombreB
// et lire la réponse de l'utilisateur, lorsqu'il cliquera le bouton.

function tirageNombre(min, max) {
      // cette fonction tire un nombre au hasard entre min et max
      let nombre = Math.floor(Math.random() * (max-min+1)) + min
      return nombre
}

function ecrireNombre(ID, valeur) {
      document.querySelector(ID).value = valeur
      // document.getElementById(ID).value = valeur          // autre possibilité
}

function lireNombre(ID) {
      return Number(document.querySelector(ID).value)
}

// Partie du code qui s'exécute dès l'appel du script dans la page web
for (let t=0; t < 50; t++) {
      console.log(tirageNombre(5,50))
}

// tester l'écriture des nombres. Utilisez "nombreA" si vous préférez la syntaxte
getElementById()
ecrireNombre("#nombreA", tirageNombre(1,50))
ecrireNombre("#nombreB", tirageNombre(1,50))
```

Note : je n'ai pas commenté les fonctions ecrireNombre() et lireNombre() qui sont explicites.

> Pouvez-vous ajouter les commentaires pour ces deux nouvelles fonctions ?

2.3 LECTURE DES CHAMPS

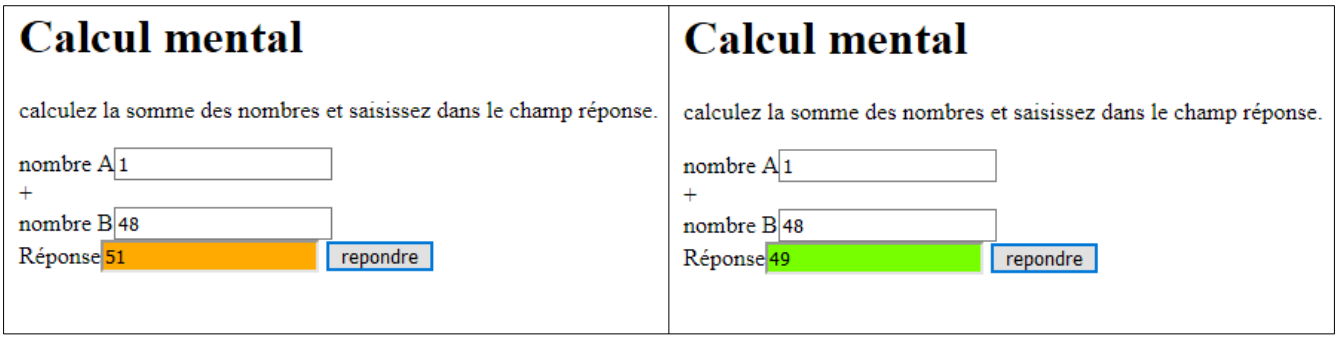
Lorsque l'utilisateur clique sur le bouton, nous l'avons programmé pour appeler la fonction repondre().

Celle-ci n'a pas de paramètres, et JavaScript va devoir lire les 3 champs pour vérifier si l'utilisateur a bien répondu.

Ajoutez la fonction suivante au code JavaScript :

```
function repondre() {
	// cette fonction lit les champs et vérifie la bonne ou mauvaise réponse
	let resultatJuste = lireNombre("#nombreA") + lireNombre("#nombreB")
	let resultatUser = lireNombre("#reponse")

	// test d'égalité et modification de la couleur du champ réponse
	if (resultatUser == resultatJuste) {
		document.querySelector("#reponse").style.backgroundColor = "#77FF00"
	} else {
		document.querySelector("#reponse").style.backgroundColor = "#FFAA00"
	}
}
```

Le résultat est à peu près le suivant :



3 AMÉLIORATIONS

3.1 AMÉLIORATION N°1 : LE STYLE

3.1.1 Problème

Les remontées des utilisateurs sont les suivantes : notre système de calcul mental fonctionne correctement… mais il est très laid !

3.1.2 Solution

Il est possible d'ajouter facilement son propre code CSS, mais travailler avec un framework CSS reste la solution la plus simple pour obtenir une page correcte.

Nous allons découvrir le framework « MetroUI » disponible sur https://metroui.org.ua/

Nous utiliserons la version CDN (c'est-à-dire en ligne). Voici le nouveau code, avec en rouge, les spécificités de MetroUI.

```
<!doctype html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<title>Calcul mental</title>
	<link rel="stylesheet" media="screen" type="text/css" href="monstyle.css"/>
	<link rel="stylesheet" href="https://cdn.metroui.org.ua/v4/css/metro-
all.min.css">
</head>
<body>
	<h1>Calcul mental</h1>
	<p>calculez la somme des nombres et saisissez dans le champ réponse.</p>
	<div class="cell-5">
	<div class="form-group"><input type="text" data-role="input" data-
prepend="Nombre A:" id="nombreA" readonly></div>
	<div class="form-group">+</div>
	<div class="form-group"><input type="text" data-role="input" data-
prepend="Nombre B:" id="nombreB" readonly></div>
	<div class="form-group">=</div>
	<div class="form-group"><input type="text" data-role="input" data-
prepend="Réponse:" id="reponse"></div>
	<div class="form-group"><input class="pure-button" type="button" id="repondre"
value="repondre" onclick="repondre()"></div>
	</div>

	<script src="https://cdn.metroui.org.ua/v4/js/metro.min.js"></script>
	<script src="js/calcul.js"></script>
</body>
</html>
```

Le résultat est le suivant :

L'interface est plus propre et ressemble plus à une application moderne.

La documentation de MetroUI est très claire et facile à utiliser, c'est une alternative à BootStrap et c'est la raison pour laquelle je vous l'ai proposé. Vous pouvez bien sûr, utiliser votre framework préféré.

Ce qu'il faut retenir :

- un framework est composé à minima d'un fichier CSS, et est souvent accompagné par un fichier JavaScript
- Pour une meilleure interactivité, il contient souvent le framework jQuery, mais depuis quelques années, jQuery a tendance à disparaître, il est donc recommandé de ne pas choisir d'exemples de codes en jQuery
- Les frameworks utilisent les attributs « class » pour déterminer l'aspect et le fonctionnement des éléments utilisés

3.2 AMÉLIORATION N°2 : REJOUER

3.2.1 Problème

Les utilisateurs ont remonté le problème suivant : une fois que le joueur a obtenu une réponse juste, il faut relancer le programme en rafraîchissant la page du navigateur. Et dans ce cas (si on n'appuie pas sur la touche CTRL ou SHIFT) le champ réponse contient l'ancienne réponse

3.2.2 Solution

Il faut créer une fonction de réinitialisation, qui recalcule tous les champs. Cela signifie simplement qu'il faut modifier la fin du script :

Remplacer

```
ecrireNombre("#nombreA", tirageNombre(1,50))
ecrireNombre("#nombreB", tirageNombre(1,50))
```

Par

```
function init() {
	ecrireNombre("#nombreA", tirageNombre(1,50))
	ecrireNombre("#nombreB", tirageNombre(1,50))
	document.querySelector("#reponse").value = ""
	document.querySelector("#reponse").style.backgroundColor = "white"
}

// Lancement d'une séquence de calcul
init()
```

au passage, ne pas oublier de mettre à jour les commentaires de l'entête :

```
// Ce script va permettre d'écrire dans les champs nombreA et nombreB
// et lire la réponse de l'utilisateur, lorsqu'il cliquera le bouton.
// v1.0 calcul des champs
// v1.1 ajout du framework CSS MetreoUI
// v1.2 ajout du bouton pour relancer le code
```

Il suffit maintenant de rajouter un bouton qui appellera cette fonction dans le code HTML (pour ma part, je l'ai mis dans le même bloc « DIV » que l'autre bouton :

```
<div class="form-group">
	<input class="pure-button" type="button" id="repondre" value="repondre"
onclick="repondre()">
	<input class="pure-button" type="button" id="relancer" value="relancer"
onclick="init()">
</div>
```

3.3 AMÉLIORATION N°3 : POURCENTAGE DE RÉUSSITE

3.3.1 Demande

Un instituteur s'est intéressé à notre programme. Il aimerait voir apparaître le pourcentage de réussite des écoliers avec une barre horizontale.

3.3.2 Solution

Il faut commencer par ajouter deux compteurs :

- comptage du nombre de clic sur le bouton « Répondre »
- comptage du nombre de bonnes réponses (passage dans le bon 'IF')

Et ensuite, proposer un élément de type barre de progression. La documentation de MetroUI indique comment modifier les paramètres : https://metroui.org.ua/progress.html

La formule est pourcentage = nbreBonneReponse/nbreCoupJoue*100

Dans la partie HTML du code, on ajoute donc une barre (ici, on utilise le framework MetroUI)

```
<div class="form-group">
      <div id="preussite">Réussite</div>
      <div data-role="progress" id="progression" data-type="buffer" data-value="0"
data-buffer="0"></div>
</div>
```

Puis dans la partie JavaScript, on ajoute en variable globale les compteurs :

```
let nbreBonneReponse = 0
let nbreCoupJoue = 0
init()
```

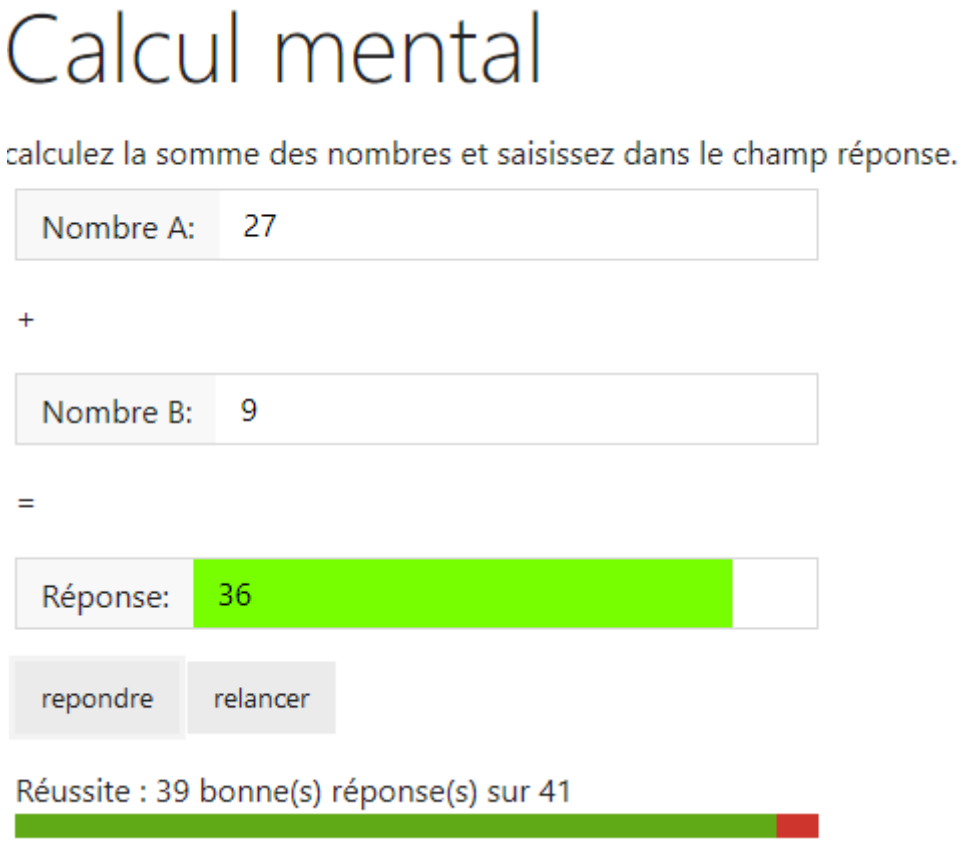
puis dans la fonction repondre() on incrémente les compteurs et on ajoute un appel à une nouvelle fonction :

```
function repondre() {
      nbreCoupJoue++
      let resultatJuste = lireNombre("#nombreA") + lireNombre("#nombreB")
      let resultatUser = lireNombre("#reponse")

      // test d'égalité et modification de la couleur du champ réponse
      if (resultatUser == resultatJuste) {
            document.querySelector("#reponse").style.backgroundColor = "#77FF00"
            nbreBonneReponse++
      } else {
            document.querySelector("#reponse").style.backgroundColor = "#FFAA00"
      }
      reussite(nbreBonneReponse, nbreCoupJoue)
}
```

Enfin, on crée la nouvelle fonction, qui accepte deux paramètres :

```
function reussite(bonne, coup) {
	// modifier la barre de réussite (MetroUI)
	let progression = Metro.getPlugin("#progression", "progress")
	progression.buff(100)
	progression.val(bonne/coup*100)
	document.querySelector("#preussite").innerHTML = "Réussite : "+bonne+" bonne(s) réponse(s) sur "+coup
}
```

Désormais, le code affiche la page suivante après quelques essais :



3.4 AMÉLIORATION N°4 : EMPÊCHER LA TRICHE

3.4.1 Problème

Notre instituteur s'est aperçu que les écoliers les plus malins, lorsqu'ils trouvent un résultat juste, s'amuse à cliquer plusieurs fois sur le bouton [Répondre], ce qui augmente leurs statistiques de réussite.

3.4.2 Solution 1

Nous pouvons interdire l'usage du bouton [Répondre] tant que le bouton [Relancer] n'a pas été cliqué.

En cas de bonne réponse :

```
document.querySelector("#repondre").disabled=true
```

Et dans la fonction init() on revalide le bouton :

```
document.querySelector("#repondre").disabled=false
```

Dans l'idéal, il faut faire l'inverse pour le bouton [Relancer], pour éviter que les élèves ne recherchent que des opérations faciles.

3.4.3 Solution 2

Nous pouvons colorier le champ en vert quelques instants et automatiquement relancer le calcul (le bouton [Relancer] devient caduc)

C'est la solution retenue ici avec l'utilisation de la fonction setTimeout(fonction, délai), à placer à l'intérieur de la fonction repondre(), dans le 'if' de la réponse juste.

```
document.querySelector("#repondre").disabled=true;
setTimeout(init, 1000)
```

```
function init() {
	ecrireNombre("#nombreA", tirageNombre(1,50))
	ecrireNombre("#nombreB", tirageNombre(1,50))
	document.querySelector("#reponse").value = ""
	document.querySelector("#reponse").style.backgroundColor = "white"
	document.querySelector("#repondre").disabled=false
}
```

Et suppression du bouton relancer sur la page HTML.