Introduction à la programmation avec javascript

Buts du cours

Initier les élèves à la pensée computationnelle et aux algorithmes. À la fin du cours, les élèves savent imaginer des algorithmes simples et savent ce qu'est une fonction récursive. Ils peuvent programmer des interactions simples avec le DOM (**Document Object Model)** d'une page web, c'est à dire, l'interface visuelle à l'intérieur du navigateur web.
Exemples : Créer un bouton qui change la couleur, la position et la rotation d'un bloc dans une page web.

Montrer la relation entre le javascript et une page web. Javascript est un langage de programmation assez simple avec une syntaxe similaire à la majorité des langages de programmation, mais il est spécialement adapté pour le web.

L'enseignant peut tracer des liens avec le CSS pour les élèves ayant suivi le cours HTML/CSS. Il peut aussi porter l'attention des élèves sur les balises et id. Les élèves n'ayant pas suivi le cours HTML/CSS devront apprendre à repérer au moins la balise script et le nom du script, ainsi que les id des balises. Il y a moins besoin de s'attarder sur le reste. En tout fin de cours, le lien entre HTML et javascript est poussé un peu plus loin, afin de donner un avant-goût de ce que l'on peut faire avec javascript.

Public cible

Enfants de 12 à 16 ans (de 8P Harmos à 1ère année du Collège/Gymnase/Lycée). Les enfants plus jeunes n'ont pas encore vu à l'école les principes de base d'un système de coordonnées en 2 dimensions et n'ont pas encore assez d'expérience dans la résolution de problèmes mathématiques.

Connaissances prérequises

Programme général de mathématiques de l'enseignement public jusqu'en 8P Harmos ou équivalent. Aucune connaissance préalable en programmation requises. Aucune connaissance en HTML/CSS requises, bien que ce cours soit donné dans le cadre du cours "Crée ton site web 2.0" donné par trimestre dans les activités extra-scolaires ou à l'année si le cours est donné en institut.

Utilisation du présent document

Ce document est le manuel de l'enseignant. Il n'est pas prévu de le donner aux élèves.

Les activités sont décrites ici selon un scénario hypothétique qui vise à développer petit à petit les connaissances des élèves en programmation, tout en évitant des répétitions avec le cours HTML/CSS, ceci dans le but de permettre à d'éventuels nouveaux élèves de suivre le présent cours. C'est à l'enseignant qu'il revient de développer certains points, ou au contraire d'en survoler, selon la dynamique de classe à laquelle il fait face. Ce document se veut suffisamment détaillé pour un cours d'introduction. En cas de besoin, les deux documents d'aide-mémoire sont aussi là pour approfondir ou résumer le contenu du cours.

Dans ce document les mots en **GRAS** sont définis dans le document aide-mémoire Vocabulaire. Ces mots peuvent faire l'objet de petits quizz ponctuels dans le cours afin d'entretenir les connaissances des élèves et ajouter un peu d'interactivité.

Les deux documents d'aide-mémoire (vocabulaire et fonctions) sont prévus pour être fournis aux élèves. Il est important de les inciter à les lire pour répondre aux questions de l'enseignant ou pour résoudre leurs bugs, avant de leur donner la réponse. Les élèves devraient être autant autonomes que possible, mais comme ils n'ont pas le manuel de l'enseignant, l'intervention de l'enseignant est nécessaire pour présenter l'activité et la commencer.

Tous les exemples de codes javascript présentés dans ce document sont un copié-collé des fichiers javascripts fournis en annexe dans le dossier du nom correspondant à l'activité. Pour le déroulement des activités, il est préférable de fournir aux élèves tous les fichiers de cours et les aides mémoires au tout début, mais de retirer les fichiers javascripts des dossiers à partir de l'activité 5. Ainsi, les fichiers html appelleront un script.js qui n'est plus existant dans le dossier et les élèves devront le recréer. Pour tester les scripts, il est important que les fichiers soient nommés "script.js" exactement et, ensuite, il suffit d'ouvrir le fichier html dans un navigateur en double-cliquant dessus.

Si vous travaillez avec des MAC, safari ne permet généralement pas d'afficher le code source de la page consultée en un clic droit dessus. préférez l'utilisation de firefox ou chrome. Faites aussi attention au navigateur google chrome. L'ensemble du cours peut être effectué sans connexion internet, mais le navigateur google chrome contient un petit jeu très bruyant lorsqu'on essaie de l'utiliser sans connexion internet.

Matériel

- Ordinateur (1 par élève)
- Clé USB (Les élèves devraient en avoir une s'il n'est pas possible d'avoir un espace travaux sur les pc à disposition.)
- Aides mémoires (Aide-memoire_Vocabulaire et Aide-memoire_javascript)
- Fichiers de cours (HTML / CSS / JS selon les activités)
- Manuel de l'enseignant (ce document)

Activités

Activité 0

Si les élèves n'ont pas suivi le début du cours "Crée ton site web 2.0" et n'ont pas de connaissances en HTML, utilisez cette activité pour expliquer la construction d'une page web.

Si les élèves ont suivi le cours précédent, vous pouvez passer rapidement sur cette activité en guise de rappel (ça ne fait jamais de mal).

Profitez de cette activité pour créer un compte scratch pour chaque élèves (nom au choix de l'élève, bien les noter sur une feuille !, mot de passe: futurekids, mail: futurekids@gmail.com), nous en aurons besoin plus tard.

Une page web est un document texte dans lequel des marqueurs indiquent au navigateur comment il doit afficher le contenu de la page. Une page web est écrite dans un langage appelé **HTML**. On y utilise aussi une feuille de style **CSS** pour y mettre des couleurs et changer les positions des objets ainsi que du code **JAVASCRIPT** pour y programmer des éléments dynamiques/interactifs.

Ouvrez la page "exemple_commentaires.html" dans un éditeur texte ou un navigateur et décortiquez le code source avec les élèves. Pas besoin d'entrer dans les détails, vous pouvez simplement lire les commentaires et répondre aux éventuelles questions. Le fichier "exemple.html" est identique, mais sans commentaires et avec des exemples d'utilisation des balises style et script. Vous pouvez aussi lire les fichiers css et js, mais sans approfondir. Nous en aurons le temps plus tard.

Activité 1

Introduction. Demander aux élèves : "Qu'est-ce que la programmation/un programme ?" Recueillir les réponses et les discuter.
On peut dire qu'un programme est une suite d'instructions à effectuer pour une machine. ça peut aller de la résolution de calculs à l'exécution de tâches complexes comme le déplacement d'un bras robotisé pour la construction d'objets.
ça ne se limite pas à juste une liste d'instructions. ça peut aussi être une répétition finie ou infinie d'actions (une horloge ne s'arrête que quand il n'y a plus d'énergie).

"Qu'utilise-t-on pour programmer ?"
Recueillir les réponses et discuter.
On utilise un **LANGAGE DE PROGRAMMATION** et un éditeur de texte (comme pour HTML/CSS, si les élèves ont fait le cours site web). Mais avant d'écrire du code, il faut aussi apprendre à parler le langage de l'ordinateur. On ne peut pas simplement écrire du texte et espérer qu'il fasse ce qu'on attend. On ne peut pas non plus écrire n'importe comment. Comme pour HTML/CSS, il faut respecter une **SYNTAXE**. Qu'est-ce que la syntaxe ? Comme en français, c'est la manière d'écrire une phrase correcte et compréhensible.

“Si / on éc.rit un-e phra;;se comme ça”

On ne comprend rien, car les règles de la ponctuation ne sont pas bien respectées.
En programmation, si la syntaxe n’est pas correcte, le programme ne fonctionnera pas et il ne se passera rien. Il est donc important de bien se relire.

Avant de commencer avec la programmation en dur, il faut aussi comprendre ce qu’est un **ALGORITHME**. Dit simplement, un algorithme, c’est une suite d’opérations pour résoudre un problème. Par exemple, pour trouver l'hypoténuse d’un triangle rectangle à partir des deux autres côtés, il faut commencer par prendre la formule de base $a^2 + b^2 = c^2$, puis appliquer une racine carrée des deux côtés de l’égalité pour connaître la valeur de c. Cette suite d’opérations est un algorithme de résolution du problème “Quelle est la longueur de l’hypoténuse”.

Donc, pour écrire un **PROGRAMME**, il faut penser à l’**ALGORITHME**, l’écrire dans le **LANGAGE** choisi avec une **SYNTAXE** correcte pour que l’ordinateur puisse comprendre la suite d’opérations qu’il doit effectuer afin de produire le résultat voulu. Notre exemple est une suite d’opération finie, mais il est possible d’avoir des algorithmes qui ne finissent jamais tant que la **CONDITION DE SORTIE** n’est pas rencontrée. Par exemple, si on écrit un programme qui incrémente de 1 une variable chaque seconde, sans lui poser de condition, le programme s’exécutera à l’infini. Dans un jeu vidéo, la fonction qui définit les actions à faire selon le bouton pressé est aussi une boucle infinie qui regarde à chaque image du jeu, si un bouton a été pressé.

Inventez un algorithme ou utilisez ceux proposés ci-dessous. Vous pouvez faire la même chose avec une addition en colonne ou une multiplication en colonne. Ce sont des algorithmes. Les portes de super marché fonctionnent aussi avec un algorithme.

Vous pouvez aussi utiliser Scratch (voir activité 2) pour commencer à introduire la notion d’algorithme (si l’exemple de l’hypoténuse n’intéresse pas les élèves et si vous n’arrivez pas à trouver d’idée facile à verbaliser comme le super-marché)

Hypothénuse

/***** Variables*****/

cote_a;
cote_b;
hypothénuse;

somme_a-b;

/**** DEBUT****/

somme_a-b = cote_a^2 + cote_b^2;

```
hypothénuse = racine_carree(somme_a-b);

/**** FIN ****/
```

Supermarché

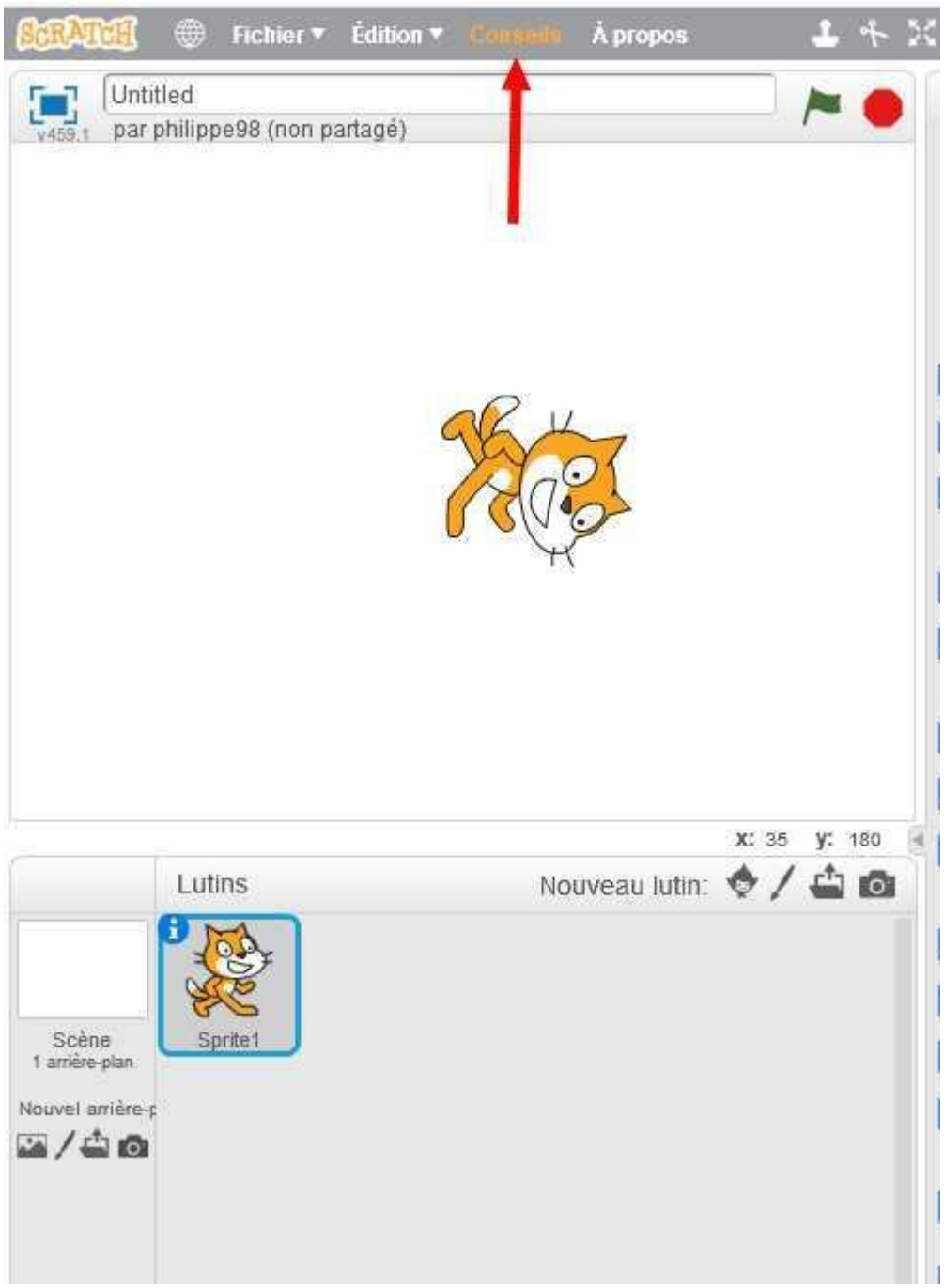
```
/*** DEBUT ***/

répéter indéfiniment
        Si quelqu'un passe devant le capteur, alors ouvrir la porte
        attendre 5 secondes
        fermer la porte

/*** FIN ***/
```

Activité 2

Utiliser scratch pour commencer la programmation de base avec des commandes simples.

Commencer par expliquer l'interface:

Les blocs de commandes sont classés par type.

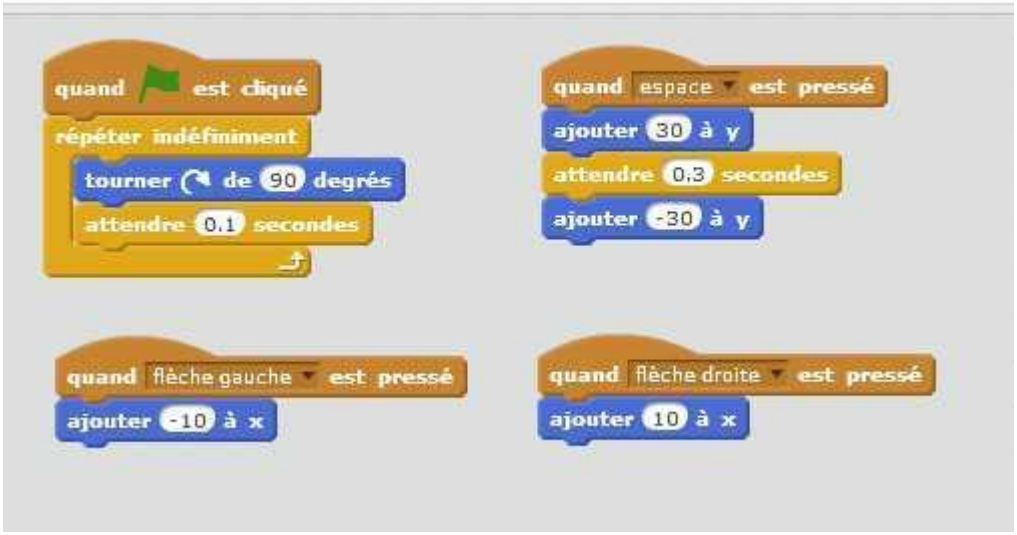


L'écran de travail se présente comme cela. Vous pouvez ajouter des lutins (des objets) à la scène dans le menu en bas. Le drapeau vert en haut de la fenêtre permet de commencer le programme et le bouton rouge sert à stopper le programme.

Cliquez sur "conseils" et faite l'activité de prise en main chez vous avant le cours, de manière à au moins comprendre la manipulation de base de scratch et l'expliquer aux élèves plus facilement.

Scratch permet d'écrire du script à l'aide de blocs de programmation. Voyez avec les élèves s'ils arrivent à déplacer le chat de la scène de base vers la droite ou la gauche

Laisser environ 10 minutes pour que les élèves jouent et découvrent les mécaniques.

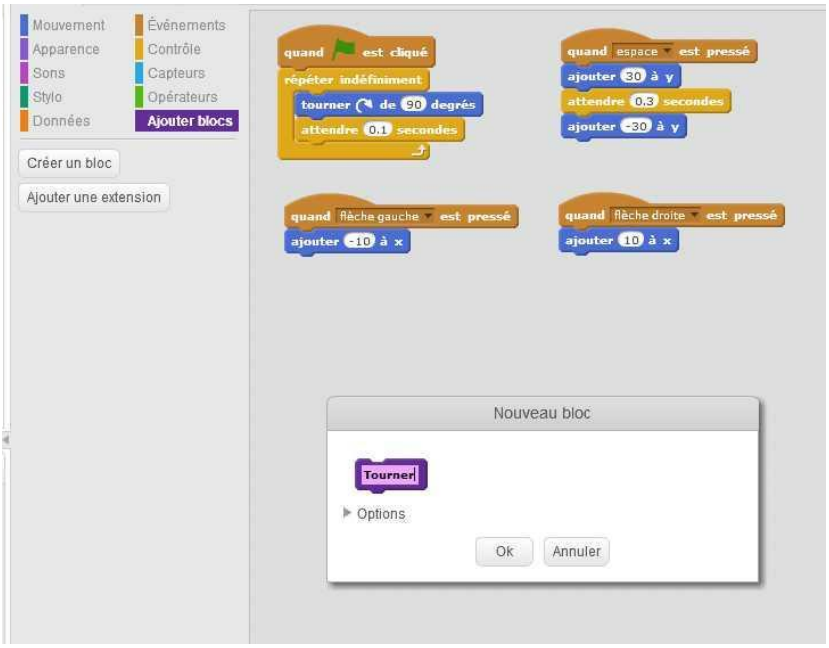


Exemple de programme pour déplacer le chat avec les flèches, le faire sauter avec espace et le faire tourner comme une toupie sur lui même en continu.

Une fois que les élèves ont joué, parler des blocs de fonctions:

Les fonctions, sont des blocs de commandes que l'on peut prédéfinir. Ensuite, il suffit de faire un **APPEL** à cette fonction pour exécuter son code, c'est plus rapide que de tout faire directement dans le programme et ça évite des copié-collé massifs dans le code, si on a besoin d'exécuter plusieurs fois la même suite d'instructions. C'est très important pour le poids du programme et pour sa vitesse d'exécution.

Laisser les élèves expérimenter une dizaine de minutes supplémentaires avec les fonctions. Leur lancer comme défi d'écrire le code le plus court possible pour faire se déplacer le chat sur la page.



On peut ajouter que quand la touche “D” est pressée, on effectue aussi la fonction “droite”, sans avoir besoin de la recopier en entier (et ce un nombre illimité de fois avec de nouvelles touches).

Explications sur pourquoi on utilise ces fonctions:

Les jeux vidéos sont actuellement très lourd (plusieurs Gigaoctets). C’est principalement dû au fait qu’il y a énormément de code pour toutes les actions possibles, en plus des graphismes, musiques, qui sont des éléments très volumineux. Si on ne cherchait pas à faire d’optimisation, un jeu comme Minecraft, qui fait un peu moins de 100 Mo, pourrait être bien plus volumineux, si les personnages étaient beaucoup plus détaillés et si le code ne contenait aucune procédures (peut-être plusieurs Gigaoctets). Il n’est pas vraiment agréable de jouer à un jeu qui ne répond pas instantanément aux commandes.

On ne peut pas bien s’en rendre compte sur des petits codes, mais il faut toujours y penser. Il n’existe jamais une seule façon d’écrire du code, donc soyez créatifs !

Activité 3

Nous allons commencer par décrire un programme. Pour cela, nous n’allons pas utiliser un langage de programmation formel, mais nous n’allons pas non plus écrire en français

chaque action du programme. Nous allons utiliser un langage intermédiaire que l'on appelle **PSEUDO CODE**. Le pseudo code, est un moyen d'écrire un squelette de programme sans avoir à s'embêter avec une syntaxe de langage particulier, c'est un travail que l'on fait plus tard lorsque l'on code le programme. C'est un peu comme faire un dessin, avant de construire une maison avec des briques.

Lisez les documents aide mémoire et définissez une syntaxe commune pour le pseudo code. Vous pouvez vous inspirer des exemples ci-dessous. Il n'y a pas de convention pour le pseudo code, soyez donc créatif, mais clair et précis. Vous pouvez aussi continuer d'utiliser scratch pour mettre en forme des algorithmes.

Pour commencer avec un exemple de pseudo code simple, écrire au tableau l'algorithme suivant (sans les commentaires en gris) :

Algorithme de calcul

Variables
a: entier;
//avant de commencer le programme, on prévient qu'on utilise une variable entière appelée "a".

DEBUT

a = 8; //on affecte la valeur 8 à a
a = a+2; //on ajoute 2 à la valeur précédente de a et on l'affecte à a
a = a*8; //on multiplie la valeur précédente de a par 8 et on l'affecte à a
a = a/2; //on divise la valeur précédente de a par 2 et on l'affecte à a

FIN

Discuter avec les élèves:

- Que fait cet algorithme ? (Réponses attendues : on affecte une valeur à a et on la modifie plusieurs fois. On additionne 2, puis on multiplie par 8 et enfin on divise par 2. On fait plusieurs calculs successifs sur a. etc...)
- combien vaut a ? (Réponse : 40)
- Expliquer l'algorithme avec les commentaires en gris du pseudo code
- Comment pourrait-on l'écrire en une seule ligne ? (Réponse : (8+2)*8/2 en suivant l'ordre normal des opérations. À noter que l'ordinateur peut résoudre ce calcul aussi de cette façon et on obtient le même résultat)

Proposer un problème à résoudre en écrivant un algorithme qui trouve la solution (pseudo code). Utiliser scratch ou des feuilles de papier pour faire écrire les élèves; éventuellement les faire venir au tableau.

Exemple : Donner un algorithme qui calcule tous les diviseurs de 457 et les affiche à l'écran à la suite.

```
Algorithme diviseurs

variables

a, b : entier;

DEBUT

a = 457;
b = 0;

Répéter tant que b < 458
{
        b = b+1;
        si a % b == 0 alors
        {
                afficher b;
        }

}

FIN
```

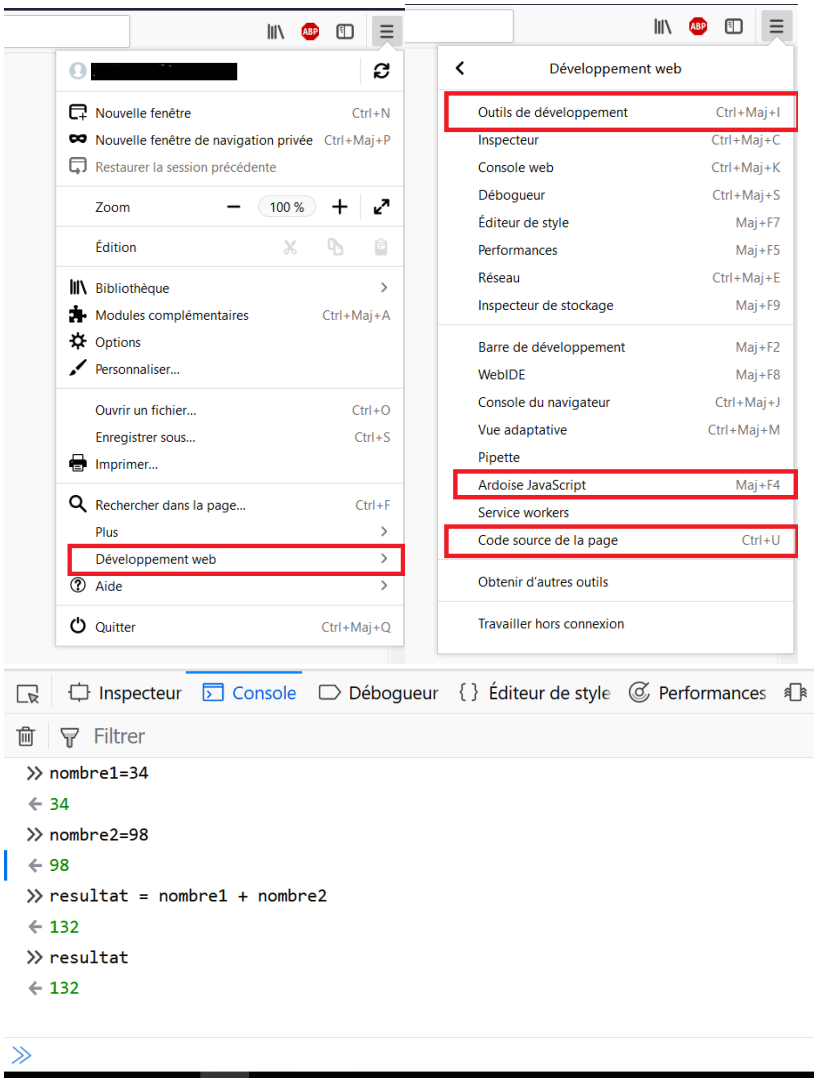
Cet algorithme affiche la valeur de b à chaque fois que le reste de la division entière (**MODULO**) est égale à zéro (c'est à dire que 457 divisé par b donne un résultat entier, sans reste). Sinon, il se contente d'incrémenter b. Cet algorithme n'est pas parfait, mais un ordinateur le calcule plus vite que nous à la main.

Spoiler : 457 est un nombre premier.

voir la page diviseurs.html

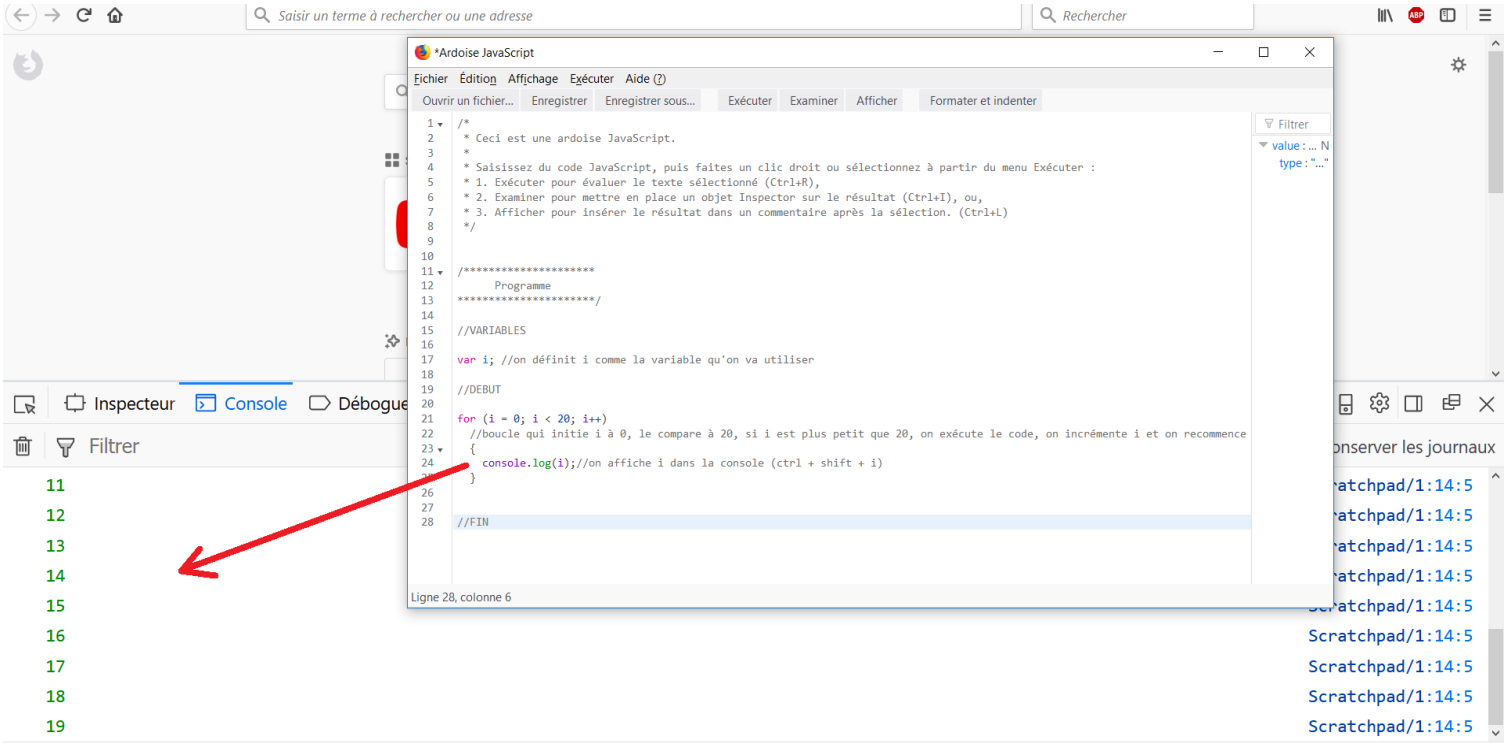
Activité 4

Il est temps de commencer à apprendre la syntaxe générale du js. Utiliser la console de **FIREFOX** (Outils de développements : ctrl+shift+i) et commencer à attribuer des valeurs à des variables : nombre1 = 34 "enter" nombre2 = 98 "enter" resultat= nombre1 + nombre2 "enter" resultat "enter" résultat est correct ?



Le nom des variables doit toujours commencer par une lettre minuscule, ne peut contenir que lettres non accentuées et chiffres, ainsi que tirets haut et bas. Rien d'autre, c'est important. Pourquoi ? Parce que les langages de programmation ont leur base en anglais. Si l'on commence à utiliser des caractères inconnus, le programme ne fonctionnera pas bien, car le compilateur javascript n'est pas conçu pour comprendre plus que les caractères romains de base (en réalité, il est possible d'utiliser des caractères accentués, mais cela peut entraîner des bugs et ce n'est en règle générale pas une bonne habitude à prendre. Javascript est permissif, mais tous les langages ne sont pas aussi flexibles).

Il existe aussi dans FireFox l'outil "ardoise javascript" (SHIFT + F4). C'est comme un bloc de texte dans lequel on peut écrire un programme et l'exécuter. Dans la console, le langage est exécuté au moment d'appuyer sur enter; dans l'ardoise, on peut écrire plusieurs commandes sans les exécuter, pour former un plus grand programme.



Le code ci-dessous peut être copié dans l'ardoise tel quel.

```
/**********************
     Programme
**********************/

//VARIABLES

var i; //on définit i comme la variable qu'on va utiliser

//DEBUT

for (i = 0; i < 20; i++) /*boucle qui initie i à 0, le compare à 20, si i est plus petit que 20, on exécute le code, on incrémente i et on recommence*/
 {
   console.log(i);//on affiche i dans la console (ctrl + shift + i)
 }


//FIN
```

Laissez les élèves adapter les algorithmes vus précédemment selon ce qu'ils pensent. Leur montrer au besoin la syntaxe des boucles et des conditions. Voir les scripts des fichiers des activités précédentes et l'aide mémoire_fonctions.

```
for(variable; condition; incrément){
        /*code*/
}
```

```
if(condition){
        /*code*/
} else {
        /*optionnel, si besoin d'effectuer quelque chose de particulier si la condition if n'est pas respectée. S'il ne faut rien faire, ne pas mettre else*/
}
```

```
while(condition vraie){
        /*tant que la condition est vraie, le code s'exécute en boucle. Fonction dangereuse s'il n'y a pas de condition de sortie !*/
}
```

N'oubliez pas les point-virgule, en fin de chaque ligne !
N'oubliez pas de fermer les accolades et parenthèses !
Les noms de variables commencent par une minuscule et on n'utilise jamais de caractères spéciaux !

Il est possible d'enregistrer les codes écrits dans l'ardoise sous format .js en cliquant sur le bouton "**ENREGISTRER**".

Activité 5

Nous allons maintenant créer un fichier javascript. Nous allons donc ouvrir sublime text (ou l'éditeur texte à disposition). En bas à droite de la fenêtre sublime text, il y a le type de document, par défaut "plain text". En cliquant dessus, on peut le changer sur "javascript". On peut aussi sauver le fichier vide et préciser l'extension ".js" à la fin du nom du fichier pour le définir comme fichier javascript. Nommez ce fichier "script.js" et sauvez-le dans le même dossier que le fichier "exemple.html". Ainsi, notre script sera utilisé dans cette page HTML sans que les élèves n'aient à éditer du HTML.

Si le script ne marche pas (dans l'ordre croissant de difficulté à vérifier):

- vérifiez que le fichier HTML n'a pas été modifié par erreur. Il doit contenir <script src="script.js"></script> tout en bas du document, dans le body.
- Vérifiez le nom du fichier. script.js sans majuscule
- vérifiez le code. Est-ce que les parenthèses sont correctement ouvertes et fermées ? les noms de variables sont ils identiques d'une ligne à l'autre ? y a-t-il des caractères spéciaux ?

Vous pouvez coller l'un des scripts des activités précédentes afin d'effectuer des tests rapides. Ouvrez la page HTML dans FIREFOX et ouvrez la console (ctrl + shift + i) pour voir les messages écrits par console.log et tester les variables. Si vous avez pris le script de l'activité 3, vous pouvez écrire le nom d'une variable et appuyer sur "entrer" pour voir sa valeur après exécution du script. Essayez avec a, b et tableau. Le principe est le même avec les autres scripts.

Le script est exécuté dès l'ouverture de la page web. S'il y a une erreur, rien ne vous le dira directement, mais votre script ne fonctionnera pas. Vous pouvez ouvrir la console du navigateur si le script ne marche pas et vous y trouverez un message d'erreur vous indiquant la ligne qui ne va pas et pourquoi.

Javascript est souple et nous ne sommes pas obligés de mettre un point virgule à la fin de chaque ligne. C'est tout de même une bonne habitude à prendre, donc bien inciter les élèves à l'écrire correctement.

Les élèves ayant fait le cours HTML/CSS peuvent aussi savoir que la balise <script> permet d'écrire directement dans le html du code javascript (comme la balise <style> permet de faire du CSS). On préfère tout mettre dans des fichiers externes pour que le site soit affiché plus vite. La page web est lue de haut en bas par le navigateur et tout est exécuté au fur et à mesure. Dans une configuration classique, le navigateur va commencer par lire le head et détecter le CSS, ensuite il affiche le body de la page en tenant compte du CSS et enfin le javascript est compilé. Le site est donc déjà lisible avant que le javascript ne soit compilé. Si on met <script> dans le head, le navigateur va commencer par compiler le code, ce qui ralentit l'affichage du reste. La page est aussi plus légère si elle ne contient pas tout le code et le CSS, c'est donc optimisé. C'est important aujourd'hui pour faire des bons sites.

Comme exemple de programme à commencer, vous pouvez proposer une simple calculatrice dans laquelle on entre deux nombres et les 4 opérations de base sont effectuées (+, -, *, /)

par exemple:

```
/**********************
        Variables
**********************/

var nombre1 = 42; /*mettez le nombre que vous voulez*/
var nombre2 = 42; /*mettez le nombre que vous voulez*/

var somme;
var différence;
var produit;
var quotient;
```

```
/*************************
        DEBUT
*************************/

somme = nombre1 + nombre2;
différence = nombre1 - nombre2;
produit = nombre1 * nombre2;
quotient = nombre1 / nombre2;

console.log("somme :" + somme + "\n" + "différence :" + différence + "\n" + "produit :" + produit + "\n" + "quotient :" + quotient); /*Dans ce code, le symbole + signifie la concaténation des éléments, pas une addition arithmétique. \n signifie un retour à la ligne.*/

/*************************
        FIN
*************************/
```

Notez qu'il est possible de réassigner les valeurs des variables en console, comme nous le faisions dans l'activité précédente, mais il faut aussi recalculer à la main. On doit donc changer le script et réactualiser la page pour changer les valeurs sans tout retaper à la main en console (ce qui ne change rien d'un point de vue rapidité).

Activité 6

Le programme précédent est très fonctionnel, mais peu pratique d'utilisation. Il faut ouvrir la console pour voir les résultats et éditer le code pour choisir d'autres nombres. Les activités suivantes vont servir à l'améliorer.

Nous allons commencer à agir sur le **DOM** (Document Object Model) d'une page web. En d'autres termes, nous allons modifier le contenu de la page web, sans éditer de document HTML. Ouvrez la page "calculatrice_act7.html" dans le navigateur. Elle est blanche. Faites un clic droit sur cette page et choisissez "afficher le code source" (ne marche pas sous safari si vous utilisez un mac, préférez firefox ou chrome). Vous remarquez que le body n'est pas aussi vide que l'on pourrait le croire. Il y a des DIV qui ne contiennent aucun texte, mais ont un ID correspondant au contenu que l'on veut y afficher (resultats, somme, différence, produit, quotient). Les élèves qui ont suivi le cours HTML devraient comprendre facilement, pour les autres, vous pouvez simplement expliquer que la balise DIV indique qu'il doit y avoir un contenu à cet endroit sur la page et l'ID est un nom que l'on donne pour les différencier entre elles.

Reprenons le code de la fois passée. Créez un nouveau fichier javascript et copiez le script de la fois passée (c'est important d'habituer les élèves à créer des fichiers aux bons endroits).

Ce script nous oblige à ouvrir la console pour voir les résultats. Ce n'est pas pratique. Nous allons donc voir une commande javascript qui va nous être très utile.

document.getElementById()
Attention aux majuscules et minuscules ici. C'est une source d'erreur importante !

Cette commande est utile parce qu'elle scanne le **document** courant (la page web où le code est exécuté) à la recherche d'un **élément** marqué avec l'**id** spécifié en paramètre.

Nous allons commencer par afficher tous les résultats au même endroit, comme avec la console. Pour cela, utilisez l'ID "resultats" (pas d'accents, souvenez-vous).

```
/**********************
        Variables
**********************/

var nombre1 = 42; /*mettez le nombre que vous voulez*/
var nombre2 = 42; /*mettez le nombre que vous voulez*/

var somme;
var différence;
var produit;
var quotient;


/**************************
        DEBUT
**************************/

somme = nombre1 + nombre2;
différence = nombre1 - nombre2;
produit = nombre1 * nombre2;
quotient = nombre1 / nombre2;

console.log("somme :" + somme + "\n" + "différence :" + différence + "\n" + "produit :" +
produit + "\n" + "quotient :" + quotient);

document.getElementById("resultats").innerHTML = "somme :" + somme + "<br>" +
"différence :" + différence + "<br>" + "produit :" + produit + "<br>" + "quotient :" +
quotient;
```

```
/**************************
        FIN
**************************/
```

Notez la différence avec la ligne console.log. On n'utilise pas \n pour un retour à la ligne, car on écrit du HTML sur la page web, il faut donc utiliser la balise
 qui signifie le retour à la ligne en HTML. Vous pouvez profiter de l'occasion pour rappeler aux élèves que l'on peut souvent copier-coller beaucoup de choses quand on écrit du code, mais il faut toujours faire attention au contexte pour corriger les petites différences.

Si vous ouvrez la console, vous pourrez constater que les résultats sont les mêmes. Les élèves peuvent essayer de modifier les variables dans le script ou d'en faire des nouvelles si l'envie leur prend.

Mais cette façon de faire n'est pas pratique si on a besoin d'avoir ces résultats dans des balises séparées. Nous allons donc rajouter un getElementById pour chaque id.

```
document.getElementById("somme").innerHTML = "somme :" + somme;

document.getElementById("différence").innerHTML = "différence :" + différence;

document.getElementById("produit").innerHTML = "produit :" + produit;

document.getElementById("quotient").innerHTML = "quotient :" + quotient;
```

On notera que c'est bien plus lisible pour l'humain et on comprend de suite quelle balise affiche quoi. Même si cela nous fait écrire 4 lignes. Notez aussi que comme les résultats sont dans des balises différentes, il n'y a plus besoin de demander un retour à la ligne. (Les élèves qui ont suivit le cours HTML peuvent savoir que c'est parce que DIV est une balise block qui ne s'affiche pas en ligne par défaut).

Rafraîchissez la page ou réouvrez le document "Calculatrice_act6.html"
Le résultat devrait être le suivant :

somme :84
différence :0
produit :1764
quotient :1

somme :84

différence :0

produit :1764

quotient :1

Laissez les élèves expérimenter d’autres nombres s’ils en ont envie.

Défi optionnel: Afficher aussi les puissances à l'aide d'une boucle for. Pour cela, créez une variable puissance et lui affecter la valeur voulue et créer une variable tampon qui contiendra le résultat intermédiaire de chaque étape du calcul. Affectez la valeur 1 à la variable tampon.

```
for(i = 0; i < puissance; i++){

        tampon = tampon * nombre1;

}
```

Et bien sûr, affichez le résultat de tampon en console.log ou en innerHTML, comme avec les résultats précédents.

La fonction javascript Math.pow(nombre, puissance) résout cela en une seule ligne de code, mais les élèves devraient réfléchir à comment résoudre les problèmes mathématiquement.

Activité 7

On aimerait pouvoir définir les nombres directement dans la page web et les modifier à la volée, sans avoir besoin de modifier du code. La page "calculatrice_act8.html" contient des boutons pour les opérateurs et des champs de textes pour les deux nombres. Ils ne peuvent pas fonctionner sans code.

Créez un fichier javascript en insistant sur chacune des étapes, car, cette fois, nous allons avoir besoin de **FONCTIONS**.

```
/**********************
        Variables
**********************/
var nombre1;
var nombre2;
var resultat;

var plus = document.getElementById("addition");
var moins = document.getElementById("soustraction");
var fois = document.getElementById("multiplication");
var divise = document.getElementById("division");
```

Pour les variables, nous avons besoin des deux nombres, ainsi qu'une variable qui stocke le résultat.

On crée aussi des variables pour chacun des boutons qui contiennent les **SÉLECTEURS** pour s'éviter des les écrire plus tard. Ce n'est pas indispensable, mais c'est intéressant de montrer aux élèves que l'on peut stocker autre chose que des nombres et du texte dans les variables; on peut aussi stocker des bouts de code simple. Attention cela dit, entrer toute une opération dans une variables n'est pas correct, il existe des fonctions pour cela.

```
/************************
        fonctions
************************/

function addition() {
        nombre1 = document.getElementById("num1").value;
        nombre2 = document.getElementById("num2").value;
        resultat = parseFloat(nombre1, 10) + parseFloat(nombre2, 10); //pour écrire des nombres fractionnaires, utilisez la virgule, pas le point

        document.getElementById("resultats").innerHTML = nombre1 + " + " + nombre2 + " = " + resultat;;
}

function soustraction() {
        nombre1 = document.getElementById("num1").value;
        nombre2 = document.getElementById("num2").value;
        resultat = parseFloat(nombre1, 10) - parseFloat(nombre2, 10); //pour écrire des nombres fractionnaires, utilisez la virgule, pas le point

        document.getElementById("resultats").innerHTML = nombre1 + " - " + nombre2 + " = " + resultat;
}

function multiplication() {
        nombre1 = document.getElementById("num1").value;
        nombre2 = document.getElementById("num2").value;
        resultat = parseFloat(nombre1, 10) * parseFloat(nombre2, 10); //pour écrire des nombres fractionnaires, utilisez la virgule, pas le point

        document.getElementById("resultats").innerHTML = nombre1 + " x " + nombre2 + " = " + resultat;;
}

function division() {
        nombre1 = document.getElementById("num1").value;
        nombre2 = document.getElementById("num2").value;
        resultat = parseFloat(nombre1, 10) / parseFloat(nombre2, 10); //pour écrire des nombres fractionnaires, utilisez la virgule, pas le point
```

```
        document.getElementById("resultats").innerHTML = nombre1 + " / " + nombre2 + "
= " + resultat;;
}
```

Toutes les fonctions sont très similaires. Attention aux accolades “{ }” une fonction se définit toujours avec le mot-clef “function” puis le nom qu l’on veut donner à notre fonction. Encore une fois, ne pas utiliser de majuscules au début du nom ou de caractères spéciaux. Ici, nous opterons sur des noms évidents, en rapport avec les 4 opérations que l’on veut faire. Les parenthèses “()” vides signifient que notre fonction ne prend aucun paramètre. Si nous voulions utiliser des paramètres, nous les écririons ici, séparés par des virgules (voir les fonctions getElementById() ou parseFloat() ci-dessus pour une illustration de fonction prenant des paramètres).

Lorsque l’on appelle l’une des fonctions, elle commence par assigner aux variables nombre1 et nombre2 le contenu des champs de texte en récupérant l’ID de chacun des champs et en appelant la fonction “.value”. Ensuite, on assigne à la variables resultat le calcul voulu par la fonction. Le “parseFloat” permet de convertir les valeurs des champs en nombre en virgule flottante (nombre décimaux, nombre fractionnaires). Dans les champs de texte, les nombres doivent être écrits avec le symbole virgule “,” et non pas le point “.”.

Enfin, on modifie le contenu du div “resultats” de la page HTML en y inscrivant le résultat. Pour un résultat plus esthétique, on peut ajouter le calcul d’origine ou en écrivant quelque chose comme “Le résultat est…”. Les élèves peuvent être créatifs à ce niveau, tant que la variable résultat est utilisée. Le texte est écrit entre guillemets et les variables sont écrites telles quelles. Les symboles + entre les chaînes de texte et les variables permettent de **CONCATÉNER** le texte et les nombres. Si on écrivait directement nombre1 + nombre2, le calcul serait effectué à cet endroit et nous n’aurions pas besoin de la variable resultat, mais cette pratique n’est pas recommandable si l’on vise à écrire des codes plus longs, notamment pour le contrôle d’erreurs.

```
/*************************
        DEBUT
*************************/
plus.addEventListener('click', addition);
moins.addEventListener('click', soustraction);
fois.addEventListener('click', multiplication);
divise.addEventListener('click', division);

/*************************
        FIN
*************************/
```

Le programme en lui-même attend qu’il se passe quelque chose, ici l’eventListener attend un “click” pour lancer la fonction correspondante au bouton. Ici, on utilise les variables créées au début du code plus, moins, fois, divise qui contiennent les sélecteurs. On aurait pu écrire :

```
document.getElementById(“addition”).addEventListener(‘click’, addition);
```

Pour un résultat totalement identique.

Les élèves peuvent apporter leurs modifications personnelles selon leurs envies. Pour afficher les éléments de la page HTML d’une autre manière, il faut faire du CSS. C’est le programme d’un autre cours, donc ne pas trop s’attarder là dessus si les élèves le demandent. Vous pouvez vous en tenir à mettre des margin entre les blocs.

Activité 8

Quand vous créez un site web, vous pourriez vouloir cacher des pages ou des informations derrière un mot de passe. Ceci n’est pas fait en javascript, car c’est dangereux.

Demander aux élèves pourquoi. (Réponse attendue : On peut accéder au code source de la page en faisant simplement un clic droit. On peut lire les fichiers externes appelés dans une page html.)

Puisqu’on sait cela, allons voir la page “securite_act9.html”. Il y a un méchant qui embête les élèves de futureKids. Nous allons trouver son mot de passe pour pouvoir le donner à Tristan. Tristan a besoin du mot de passe pour accéder à d’autres informations ailleurs. On sait que c’est le même.

Laissez les élèves chercher. Ils peuvent évidemment cliquer sur le lien securite.js qui est appelé, mais ce n’est pas là que l’on trouve le mot de passe. S’ils utilisent cette solution les rediriger vers l’objectif qui est de trouver le mot de passe.

Le but de cet exercice est d’entraîner la lecture de code. Les élèves peuvent rencontrer des difficultés à trouver le mot de passe. C’est une occasion de rappeler que les commentaires avec // ou /* */ servent à documenter le code pour faciliter sa lecture. Comme on ne cache pas d’informations sensibles sur une page HTML, il n’y a pas de mal à rendre la lecture très facile en ajoutant une grande quantité de commentaires.

Les élèves pourraient vouloir malgré tout savoir comment les sites tels que facebook ou les forums font pour cacher les mots de passe de milliards d’utilisateurs. Pour cela on utilise les bases de données. On y accède avec des requêtes PHP ce que nous n’allons pas voir ici. Il est aussi possible d’y accéder en javascript, mais c’est plus compliqué. Enfin, cela nécessite l’installation d’une base de données. La base de données peut être créée dans une autre langage, le SQL, par exemple, mais cela, nous ne le verrons pas ici. Si vous voulez créer des sites avec un login et un espace membre, utilisez les moyens déjà existants, pas la peine de vous fatiguer à le coder vous-même. Des **CMS (Content Management System)** comme wordpress, phpbb, storytlr, etc. permettent de créer des sites complexes très facilement. Vous pouvez ensuite ajouter vos script et vos pages HTML personnels pour que votre site soit à votre image. Ce genre de site nécessite des mises à jours régulières pour

protéger vos données, mais c'est aussi compliqué que de cliquer sur un bouton et attendre le téléchargement.

En règle générale, la sécurité sur le web est une chose dont vous ne devriez pas vous occuper vous-même, à moins d'avoir des très bonnes connaissances dans le sujet. Si vous avez besoin de zones sécurisées, installez un CMS et maintenez-le à jour, c'est le plus simple.

Activité 9

Nous allons créer un autre bouton interactif ici. Nous allons voir comment naviguer dans un **TABLEAU** ou **array** en anglais. Un tableau est une variable comme les autres, mais il contient plusieurs valeurs. Ces valeurs ont chacune une position dans le tableau, qui commence à zéro !

```
/*****************
Variables
******************/

var tableau = [1, 2, 3, 4, 5, 6, 7, 8, 9];

var bouton = document.getElementById("bouton"); //sélecteur du bouton
var resultat = document.getElementById("nombre_de"); //sélecteur du h1 qui affiche le résultat
var aleatoire; //stocke un nombre aléatoire au clic du bouton
```

Comme on peut le voir dans ce bout de code, le tableau se définit avec des crochets "[ ]" et les valeurs sont séparées par des virgules. Il est possible d'entrer aussi bien des nombres entiers ou non (les nombres non entiers s'écrivent avec un point !) que des caractères ou des chaînes (écrits entre guillemets). Il est possible de mélanger les types de données. Laissez les élèves construire un tableau aussi grand qu'ils le veulent et contenant ce qu'ils veulent. Pour la facilité, nous ferons un lancé de dé entre 1 et 9.

```
/***************
        Fonction
****************/

function lancer() {
```

```
	/*
	Math.random retourne un nombre, non entier, aléatoire compris entre 0 et 1.
	Math.floor retourne le plus grand nombre entier inférieur ou égal
	tableau.length retourne le nombre d'éléments contenus dans le tableau, ici 9.
	*/

	aleatoire = Math.floor(Math.random() * tableau.length); //le tableau contient 9 valeurs. La position 0 est la première du tableau, la position 8 est donc la neuvième et dernière.

	resultat.innerHTML = tableau[aleatoire];

	console.log(tableau[aleatoire]); //pour avoir un historique des nombres lancés
}
```

Cette fonction génère un nombre aléatoire. Comme expliqué dans les commentaires, nous avons besoin d'utiliser le Math.random pour obtenir un nombre aléatoire, nous le multiplions par la taille du tableau, ici, c'est facile, et nous demandons un Math.floor de cette valeur pour ne garder qu'un nombre entier.
Math.random retourne un nombre compris entre 0 et 1 sur un intervalle ouvert [0, 1[. Ce qui signifie que 0 fait partie des valeurs retournées, mais 1 en est exclu. Il faut ajouter +1 dans le calcul du nombre aléatoire pour pouvoir tomber sur 9, mais nous n'en avons pas besoin ici, puisque 8 est la dernière position du tableau (cela vaut pour toute taille de tableau, tableau.length suffit).

On peut ensuite afficher dans la page et en console la valeur contenue à la position "aleatoire" du "tableau". Si aleatoire vaut 6, tableau[6] = 5, tableau[0] = 1, tableau[8] = 9.
Il faut insister sur ce point: en programmation, on ne commence pas à 1, mais à 0, ce qui décale les valeurs de position dans un tableau.

```
/************
DEBUT
*************/
bouton.addEventListener('click', lancer);

/**********
FIN
***********/
```

Il ne reste qu'à ajouter un event listener sur le bouton, comme dans l'activité précédente, et c'est fini !

Si les élèves sont perturbés par les nombres (position 0 = 1, position 8 = 9), entrez des caractères ou des noms dans le tableau : var tableau = [“premier mot”, “un autre mot”, ...]; C’est perturbant, mais nécessaire à comprendre comment naviguer dans un tableau.

Activité 10

Nous allons voir une utilisation des variables booléennes. Pour rappel, ce sont les variables logiques “vrai” et “faux”; “true” et “false” en anglais. On peut s’en servir dans de nombreuses situations. Dans cette leçon, on va s’en servir comme d’un interrupteur (toggle, en anglais). Si la variable est vraie, on affiche le contenu, sinon on le cache.

Ouvrez la page true_false.html dans le navigateur et cliquez sur les boutons pour observer le comportement des boutons. Comment pourrait-on programmer cela ? Discutez avec les élèves. Lisez le code source de la page pour trouver les sélecteurs dont on aura besoin. pour changer des id, ici ce sont des classes.

```
/*****************************************
                Variables
*****************************************/
//Initialisées à true pour afficher tout par défaut.
var web = true;
var ev3 = true;
var scratch = true;

//sélecteurs
var siteWeb = document.querySelector(".siteWeb");
var legoEV3 = document.querySelector(".ev3");
var coursScratch = document.querySelector(".scratch");
```

querySelector est une autre façon de sélectionner des éléments dans la page. Il faut ensuite écrire le nom de l’identifiant ou de la classe de la balise en suivant les mêmes règles que dans du code CSS : querySelector(“.class”) ou querySelector(“#id”). C’est la même chose que si l’on utilisait getelementById, mais cela marche indifféremment avec id ou class.

Ici, on initie trois variables sur “true” et les sélecteurs de nos balises HTML.

```
/*****************************************
                Fonctions
*****************************************/
//Les fonctoin ci-dessous servent à inverser les valeurs et l'affichage. Elles sont exécutées onclick.

function toggleTous() {

//si les trois variables sont vraies, on les met toutes sur faux et on cache toutes les balises
```

```
        if (web == true && ev3 == true && scratch == true) {

        web = false;
        siteWeb.style.display = "none";

        ev3 = false;
        legoEV3.style.display = "none";

        scratch = false;
        coursScratch.style.display = "none";
        } else {

//Sinon, on les met toutes sur vrai et on affiche toutes les balises

        web = true;
        siteWeb.style.display = "inline-block";

        ev3 = true;
        legoEV3.style.display = "inline-block";

        scratch = true;
        coursScratch.style.display = "inline-block";

        }
}

function toggleWeb() {

        web = !web; //notation logique pour inverser la variable. Le point d'exclamation signifie "non" (not). not true = false, not false = true.

        if (web == false) {

        //si on est sur false, on change le style de la balise, comme en CSS

        siteWeb.style.display = "none";
        } else {
        siteWeb.style.display = "inline-block";
        }
}

function toggleEV3() {
        ev3 = !ev3;
        if (ev3 == false) {
        legoEV3.style.display = "none";
        } else {
        legoEV3.style.display = "inline-block";
```

```
        }
}

function toggleScratch() {
        scratch = !scratch;
        if (scratch == false) {
        coursScratch.style.display = "none";
        } else {
        coursScratch.style.display = "inline-block";
        }
}
```

On remarque plusieurs choses.

- L'opérateur logique d'égalité : "==". Il permet de tester l'égalité ou l'équivalence. Cela fonctionne aussi avec les caractères, les chaînes et les nombres. "===" permet de tester une égalité stricte, plus forte que l'équivalence, surtout pour les chaînes de caractères. 1 == 1 est vrai. 2 == 1 est faux. Pratique pour les conditions "if".
- L'opérateur logique d'inversion "!". Il nous sert à créer l'interrupteur de vrai vers faux. Il n'y a pas besoin de savoir sur quelle valeur est la variable, on l'inverse de toute façon de vrai à faux ou de faux à vrai. Ne marche qu'avec les variables booléennes.
- "sélecteur".style.display. C'est quelque chose que les élèves pourraient avoir vu en CSS. En CSS, on écrirait "div { display : none; }" par exemple, mais on ne peut pas programmer une inversion de cette valeur en CSS. On utilise donc cette fonction javascript qui permet de modifier une valeur de style. On peut aussi faire "sélecteur".style.color = 'blue'; ou bien "style.font-weight = 'bold';". Les élèves qui connaissent le CSS peuvent retrouver ici plusieurs fonctionnalités dont ils peuvent changer les valeurs comme en CSS.

Enfin, on remarque qu'il n'y a pas de DEBUT et FIN dans ce script. Pourquoi ? Laissez les élèves essayer de deviner en regardant le html et le javascript.

La réponse se trouve dans le HTML. Les boutons sont tous créés ainsi :
<button class="showAll" onclick="toggleTous()">Tous</button>
On remarque la présence du paramètre "onclick".

C'est une autre façon d'écrire addEventListener, mais au lieu d'y mettre dans le javascript, c'est dans le HTML. On peut faire ça, parce que les boutons sont généralement prévus pour être cliqués, c'est donc parfois plus rapide de faire ainsi. Il est simplement nécessaire que le nom de la fonction appelée par le bouton soit le même que la fonction décrite dans le script. N'oubliez pas les parenthèses, vides ici, car nous n'avons pas de paramètres dans notre fonction, mais il serait possible d'en avoir besoin. Pour une entrée de valeur par l'utilisateur, par exemple.

Cette leçon est la dernière et vise à renforcer le lien entre javascript et le cours HTML. Javascript est un langage web, il est donc important d'en parler dans le cours "Crée ton site web 2.0", mais il n'est pas nécessaire de connaître javascript si on ne veut que faire des

sites statiques. Comme nous l'avons vu, dans d'autre cas, php est plus approprié, mais cela mériterait un autre cours à part entière. Il y a déjà bien assez à faire pour maîtriser tout ce que nous avons vu dans ce cours.

S'il reste du temps de cours après cette leçon, les élèves peuvent travailler sur des projets personnels en s'aidant des aides-mémoire. S'ils ont un projet personnel du cours HTML, ils peuvent essayer d'y ajouter des fonctions. S'ils n'ont pas suivi le cours HTML, vous pouvez les aider à créer un formulaire ou des boutons en HTML.

Vous trouverez des exemples de projets dans le dossier "projets".

Activité 11 (bonus)

Cette activité va traiter du calcul binaire. Avant de commencer, demandez aux élèves "Qu'est-ce que le binaire?" (Réponses attendues : C'est du code compris par les ordinateurs. ça s'écrit avec seulement 0 et 1. C'est un moyen de compter en base 2/puissances de 2. C'est une traduction de notre langage en langage machine.)

Discutez les réponses des élèves.

Avant de savoir comment décrypter un programme en binaire, il faut comprendre qu'écrire une suite de 0 et de 1, c'est écrire un nombre. "Computer", en anglais, signifie "calculateur", ce qui est plus représentatif de ce que fait effectivement un ordinateur. Ainsi, les lettres, les couleurs, les images, tout est reçu sous forme de nombres par l'ordinateur et par le calcul il nous affiche un résultat compréhensible par l'humain.

0100100001100101011011000110110001101111001000000111011101101111011001011011000110010000100000000100001

Est le code pour écrire : "Hello world !"

Mais ceci n'est vrai que si l'on sait que ce code représente une **CHAÎNE** de caractères codés sur 8-bit (d'où le <meta charset="utf-8"> que les élèves pourraient avoir vus en HTML). Si l'on sépare ce code en blocs de 8 chiffres, on peut les traduire par leurs lettres respectives et on peut aussi se rendre compte que les espaces sont codés. l'ordinateur n'a dès lors pas besoin d'espace entre les blocs de 8 bit, car cela ne signifie rien dans une lecture séquentielle du code.

Cependant, si cette grande suite de 0 et 1 n'était pas une **CHAÎNE**, mais un **NOMBRE**, alors en le convertissant en base 10, on aurait le nombre : 2'867'908'381'536'507'370'900'587'552'801

Ce résultat n'a rien à voir. Comment l'ordinateur sait-il qu'il doit afficher un nombre ou une suite de caractères ? C'est le travail d'instructions spécifiques, mais nous n'allons pas approfondir le sujet. Si l'enseignant a des connaissances et que les élèves veulent vraiment le savoir, il est possible d'en parler informellement quand l'occasion se présente.

Comment pouvons nous lire un nombre simple comme 10110_2 (ne pas lire “Dix mille cent dix” ! C’est “ Un zéro un un zéro, en base deux”) ? Pour répondre à la question, nous allons commencer avec une base que nous connaissons bien: la base 10. D’ailleurs, afin de différencier le binaire de la base 10, nous écrirons en indice de chaque nombre, la base dans laquelle il est écrit : 10110_2 (binaire), 10110_{10} (base 10). Ces deux nombre sont totalement différents !

Discutez ces points avec les élèves :

- Combien avons nous de chiffres ? (10, de 0 à 9)
- Comment écrit-on le nombre “deux mille quatre cent cinquante sept” en base 10 (notre base habituelle pour compter) ?

Les élèves vont tous répondre “deux quatre cinq sept”. Mais qu’est-ce que ça veut dire vraiment ? Laissez réfléchir les élèves et essayez de discuter.

On veut dire par là qu’on additionne 7 unités, 5 dizaines, 4 centaines, et 2 milliers =7+50+400+2000 = 2457.

Pensons cela en base 10, maintenant : une dizaine, c’est 10, mais une unité, c’est quoi ? Ce n’est pas juste 1, c’est 10^0 (dix puissance zéro). quand on dit “7 unités”, on ne dit pas “7 x 1”, mais “7 x 10^0”. Quand on compte en base 10, on compte en puissances de 10, quand on compte en base 2, on compte en puissances de 2. Voir avec les élèves s’ils ont compris jusque là. Décrypter au besoin le nombre:

$2 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 7 \times 10^0$
$= 2 \times 1000 + 4 \times 100 + 5 \times 10 + 7 \times 1$
$= 2000 + 400 + 50 + 7$
$= 2457$

Et pour le binaire ? C’est exactement la même logique, mais demandez aux élèves “ combien y a-t-il de chiffres en binaire (base 2) ?” Ils devraient répondre 2 : 0 et 1. En base 10, il y a 10 chiffres : 0 à 9. En base 2, il y a 2 chiffres : 0 et 1. Mais comment écrit-on 2 en base 2 ?

Dix en base Dix s’écrit $10_{10} = 1 \times 10^1 + 0 \times 10^0 = 10 + 0$

Deux en base Deux s’écrit …… $10_2 = 1 \times 2^1 + 0 \times 2^0 = 2 + 0$

Et c’est pour ça qu’on ne peut pas dire que 2 s’écrit “dix” en base 2. Ça s’écrit “un zéro”. Perturbant ? Passez le temps nécessaire pour faire comprendre cela aux élèves. Nous sommes dans une activité bonus qui vise à élargir la compréhension des nombres. Tant pis si vous ne la terminez pas.

Reprenons le nombre donné en exemple plus haut : 10110_2 Demandez aux élèves combien il vaut en base 10.

10110_2
$= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

= 1 x 16 + 0 x 8 + 1 x 4 + 1 x 2 + 0 x 1
= 16 + 0 + 4 + 2 + 0
= 22_{10}

Savoir décomposer un nombre, c'est un bon moyen de faire du calcul mental facilement. Si les élèves n'ont pas compris, il est possible de prendre pour exemple des nombres moins grands, mais la technique est toujours la même.

Utilisez la page "binaire.html" dans le dossier activité 4 pour convertir des nombres en base 2.

Si vous avez compris cela, vous avez aussi partiellement compris pourquoi les tailles de mémoires des clefs USB sont toujours des puissances de deux (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024,...) et pourquoi les processeurs de pc sont notés 32-bit ou 64-bit (la taille des instructions). Comme avant, si l'enseignant a des connaissances et les élèves un vif intérêt, il est possible d'en parler informellement, sinon, les inviter à rechercher par eux-même.

Nous avons vu une autre application du binaire : les booléens. Pas de mathématiques ici, mais de la logique (les deux vont souvent de pair). 0 = FAUX et 1 = VRAI. Nous n'avons que deux choix possibles.