Manipulation simple en Javascript
PSB
2020
Y. STROPPA

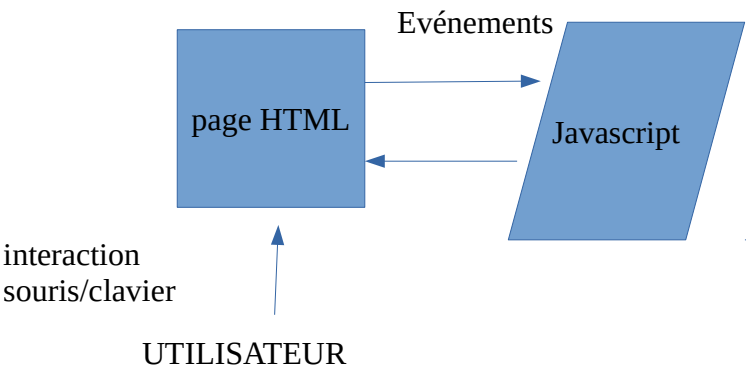
Comment se constitue un site Web .... une partie en HTML accompagnée de CSS et de fichiers JS ; La vocation des fichiers Javascript est de permettre le côté dynamique du site Web c'est à dire de pouvoir répondre aux interactions de l'utilisateur, de pouvoir échanger des données avec le serveur web ou d'autres services ....

Pour ce faire, il est nécessaire de pouvoir communiquer entre la page HTML et JavaScript comme on le fait en CSS.

Si on démarre du fichier HTML qui est chargé dans votre navigateur. Comme se passe se chargement, le navigateur lorsqu'on sollicite une URL, demande au serveur de lui fournir en réponse une structure HTML qu'il télécharge et charge dans une page. A la lecture de cette page, plusieurs événements se passent : le chargement de la page, la lecture de fichiers d'inclusions en JS eou en CSS et éventuellement leur interprétation par la navigateur. En fonction de l'écriture d'instructions Javascript qui peut se faire au chargement de la page, à la fin du chargement de la page et à l'interaction de l'utilisateur.

Tout d'abord à la lecture du fichier HTML, le navigateur définit une première structure d'éléments à partir du fichier HTML. c'est ce que l'on appelle la DOM **Document Object Model** qui permet de définir les éléments avec des propriétés visuels, contenu et comportements le tout accessible à l'aide de la variable en Javascript document.

Ensuite l'ensemble sera piloté entre événement (interaction utilisateur, Timer Javascript...) et instructions en réponse à l'événement.



Comment se passe l'interaction entre les deux éléments chargés dans la même application (navigateur). Les événements sont reçus pas la page HTML qui la relie éventuellement à un traitement associés en Javascript (click, moudown, mouseup, mousemove ...).

Plusieurs façons d'associer des événements à des objets contenu dans la DOM, c'est directement à la déclaration du fichier HTML lorsqu'on définir les Tags de notre structure avec les instructions de type onchange="code javascript", onclick="code javascript" .....

Quelques exemples :

```
<!DOCTYPE HTML>
<html>
<head>
<title>Projet PSB</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<div id="conteneur" class="center">
        <div id="contenu">
      <div id="commande_top" class="commandes">
        <input type="button" id="bt3" value="RàZ" onClick="choixRaz();"/>
        <input type="button" id="bt2" value="Verif" onClick="verifie();"/>
        <select name="config" id="config-select" onchange="chargement_();">
        </select>
      </div>
     <input type="text" id="saisie" name="saisie" valeur="54">
      <svg xmlns="http://www.w3.org/2000/svg"  id="mySVG"
            width="500" height="400" viewBox="0 0 500 500">
      </svg>
      </div>
</div>
<div id='descriptif' class="center_texte">
   <p>
      condition
   </p>
 </div>
</body>
</html>
```

Maintenant si on essaye ce fichier sans le charger sur un serveur mais uniquement en local, que se passe t-il ?

Tout se passe comme il le faut, mais lorsqu'on clique sur les boutons et si on consulte la console on s'aperçoit qu'il y a des erreurs .... normal car on n'a pas défini le contenu des fonctions en réponse au sollicitations de l'utilisateur sur les boutons ....

Plusieurs possibilités, on incorpore directement dans le fichier html le code de réponse ....

```
<input type="button" id="bt3" value="RàZ" onClick="alert('salut');"/>
```

Le comportement sera bien géré par la navigateur qui lors du clicque sur le bouton va bien relier l'événement au comportement souhaité ... mais on s'aperçoit des limitations de cette méthode au niveau du bloc d'instructions .... c'est limité et difficile à maintenir ...

Donc on va préférer définir l'appel à une fonction que l'on va définir dans le corps de notre document html dans un premier temps ...

Attention à encapsuler par des balises <script> </script> qui permettent d'indiquer au navigateur le changement de langages à utiliser

```
<script>
function choixRaz() {
	// corps de la méthode
	alert("je suis dans le corps de la methode") ;
	// fin du corps de la mtéhode
}
<script>
```

Maintenant si on souhaite séparer le corps des fonctions Javascript du HTML il suffit d'inclure une instruction HTML pour indiquer au navigateur l'insertion d'un fichier js par l'instruction suivante :
<script src="premier_fichier.js"></script>

On génére un nouveau fichier premier_fichier.js dans lequel on copie le bloc d'instructions défini :

```
function choixRaz() {
	// corps de la méthode
	alert("je suis dans le corps de la methode") ;
	// fin du corps de la mtéhode
}
```

Ensuite dans le fichier html on ajoute les lignes supplémentaires en entête du fichier
<script src="premier_fichier.js"></script>

On a le début de notre démarche, on préférera organiser notre projet en répertoire du style /js ou /css dans lesquels on installera les différents fichiers nécessaires pour définir les comportements ....

Interaction entre la partie Javacsript et HTML ....

Dans ce petit exemple on s'aperçoit que le comportement est défini à l'initialisation de la page HTML dans les instructions en HTML5 ... et qu'il permet au navigateur de connecter événement --> instructions ... maintenant la question que l'on peut se poser c'est comment le JavaScript peut interagir avec la DOM et que peut-il faire ...

Pour cela nous allons tout d'abord travailler directement dans la console pour examiner les différentes instructions et utilisation possible en "live" ...

Le tout est de récupérer un descripteur sur l'élément que l'on veut modifier grâce à la variable document.getElementById ou document.getElementByName

Exemple sur le premier bouton : var but=document.getElementById("bt3");
Si, on observe les attributs et méthodes associés à cet objet :

On peut interagir sur les propriétés de l'objet directement :

but.style.background="red" ou "yellow"
but.disabled=true ou false
but.value="nouveau contenu"
.....

Définition d'un style associé à nos boutons style1

A rajouter dans l'entête de votre fichier

```
<style>
.style1 {
      background:red;
}
</style>
```

Ensuite on peut récupérer tous les descripteurs associés à un style donné et modifier ainsi toutes les propriétés des éléments ../..

```
<input class="style1" type="button" id="bt3" value="RàZ" onClick="choixRaz();"/>
<input class="style1" type="button" id="bt2" value="Verif" onClick="verifie();"/>
```

Ce qui nous donne d'un point de vue visuel :

Maintenant si on souhaite travailler sur l'ensemble des éléments associés à un même style on peut exécuter les instructions suivan,tes :

```
var btns=document.querySelectorAll(".style1") ;

btns.forEach(e=> {
   e.style.background="yellow";
}
```

.style
balise
   input
   div
    ....

On commence à voir l'étendue des possibilités

On peut remplacer ".style1" par "input pour interagir avec tous les éléments input de la page ou avec la balise div pour interagir avec toutes les div de la page ...

Sur la partie input de type text on peut récupérer le contenu de la saisie et le modifier ....

exemple

```
var texte=document.getElementById("saisie") ;
texte.value ;
texte.value="dldl" ;
```

On souhaite maintenant modifier le comportement du bouton1 ....il suffit de récupérer un descripteur sur le bouton, et de redéfinir un autre comportement par exemple

```
function autrecomportement() {
        console.log("affiche un autre comportement") ;
}
```

Ensuite il faute le relier à la fonction :

```
var btn1=document.getElementById("bt3") ;
btn1.onclick ;
btn1.onclick=autrecomportement ;

// on peut tester
```

Donc on a la possibilité à modifier avec Javascript le comportement de notre page web à la volée en fonction de certaines situations ...

Autre exemple : essayons d'ajouter et relier les éléments entre eux, lors d'un clique sur le bouton verif on souhaite que la zone de texte se charge du contenu VERIF

Donc pour ce faire il suffit d'ajouter un module d'écoute pour le bouton ou de le surcharger si il existe et de créer une fonction qui modifie le contenu de l'input saisie

```
var verif=document.getElementById("bt2") ;
verif.onclick ;
// on peut tester
function charge_input() {
        var zone=document.getElementById("saisie") ;
        zone.value="VERIF"
} ;
verif.onclick=charge_input;
```

Une fois que vous avez mis au point les différentes

Maintenant si on souhaite générer un effet lors du survole de la souris saur la zone de saisie par exemple mettre en majuscules et en minuscules ...

Il faut ajouter deux modules d'écoute :
mouseover et mouseleave

```
var verif=document.getElementById("saisie") ;
function minuscules() {
    var sai= document.getElementById("saisie");
    sai.value= sai.value.toLowerCase();
}
function majuscules() {
    var sai= document.getElementById("saisie");
    sai.value= sai.value.toUpperCase();
}
verif.onmouseover=majuscules;
verif.onmouseleave=minuscules;
```

Maintenant on peut prendre le tout et venir l'incorporer dans un fichier js et ainsi avoir le comportement attendu.

On peut également modifier le contenu d ela DOM on venant ajouter des éléments ou en enlever ....
par exemple rendre les choses visibles ou invisibles
rajouter des items

Veuillez rajouter un bouton qui rende visible ou invisible la zone de saisie

```
<input class="style1" type="button" id="bt3" value="hide" onClick="masque();"/>
```

```
function masque() {
                    var zone_saisie=document.getElementById("saisie");
                    if (zone_saisie.style.display=="block"){
                        zone_saisie.style.display="none";
                    } else {
                        zone_saisie.style.display="block";
                    }
}
```

Ensuite lors de la sélection on souhaite rajouter un nombre d'élément en plus
Il faut ajouter une fonction addElement() qui permettra d'ajouter un nouveau élément à partir du positionnement fixe pour le moment.

Pour ajouter une zone de texte

```
var newdiv=document.createElement("div") ;
var newContentText=document.createTextNode("salut") ;
newdiv.appendChild(newContentText) ;
var currentDiv=document.getElementById("descriptif") ;
document.body.insertBefore(newdiv, currentDiv) ;
```

Pour ajouter une nouvel élément de saisie

```
var newdiv=document.createElement("INPUT") ;
newdiv.setAttribute("type","text") ;
newdiv.setAttribute("id","textYS") ;
newdiv.setAttribute("value","text") ;
document.body.appendChild(newdiv) ;
```

Utilisation de Bootstrap pour le positionnement et le caractère responsive d'un site

Notion de grid : positionnement : https://getbootstrap.com/docs/4.0/layout/grid/

Suite des exercices voir la cas étude N°1

Rappels algorithmiques et implémentations

Le principe des langages de programmation est de pouvoir construire des ensembles de blocs pour permettre la réalisation d'un ensemble de traitements qui doivent s'exécuter dans un certain ordre. Pour définir ces étapes le développeur peut s'appuyer sur les principes de base de la programmation qui sont pouvoir définir des variables pour stocker des données, pouvoir définir des blocs d'instructions de différents types : conditionnels, itératifs ...pouvoir factoriser un ensemble d'instructions à l'aide de fonction .... pour éviter de dupliquer des traitements.

Commençons à définir ce qu'est un programme en algorithmie

Un programme se définit par un nom
ensuite par des déclarations de variables
et par une succession d'instructions et de blocs d'instructions
pour arriver à un résultat

Imaginons que l'on souhaite développer une interface pour résoudre les équations du second degré et tracé la courbe de la fonction (et des racines) ... On souhaite développer une page Web qui permette à un utilisateur de rentrer les coefficients a,b et c et qui lui permette de connaître la solution de cette équation et de visualiser la courbe représentative de cette équation.

Commençons par définir une analyse de l'algorithme à mettre en place (voir méthode de résolution d'équation du second degré)

ANALYSE et SOLUTION :

Donc il faut dans un premier temps définir notre algorithme de résolution de ce type d'équation :

```
Nom du prog : résolution d'équations du second degré
Variables : a,b,c : numérique                  // coefficients de l'équation
        discriminant : numérique               // discriminant b²-4ac
        x1,x2 : numérique                      // solutions réelles de l'équation
        xi1,xi2 : alpha                        // solutions imaginaires de l'équation

// Début du programme :
        [1]Lire(a) <--  "entrez la valeur de a"
        Lire(b) <-- "entrez la valeur de b"
        Lire(c) <-- "entrez la valeur de c"

// Calcul du discriminant
        discriminant <-- b*b-4*a*c

// Analyse du discriminant
si discriminant >0 alors
        x1 <-- (b²-racine(discriminant)) / (2*a)
        x2 <-- (b²+racine(discriminant)) / (2*a)
        // affichage de la solution
```

1 Fonction de base qui permet d'entrer une valeur ou de demander à l'utilisateur la saisie dune valeur que l'on affecte directement à la variable a

```
        [2]Ecrire ("solutions réelles de l'équation :" ,x1,x2)
    fin si
    si discriminant ==0 alors
        x1 <-- b² / (2*a)
        // affichage de la solution
        Ecrire ("solution unique de l'équation :" ,x1)
    fin si
    si discriminant <0 alors
        xr1 <-- b² / (2*a)
        xi1 <-- racine(-discriminant)) / (2*a)
        // affichage de la solution
        Ecrire ("solutions imaginaires de l'équation :" ,xr1+"+i"+xi1)
        Ecrire (xr1+"-i"+xi1)
    fin si
    Fin du programme

    // une fois terminé l'analyse et le calcul des racines on peut tracer la courbe
    // pour tracer la courbe on va définir tout d'abord une fonction qui va recevoir
    // un argument (x) et retourner la valeur de la fonction
    Function f(x) : réel {
        return a*x*x+b*x+c ;
    }

    Prog tracé la courbe
    // pour tracer la courbe : on va constituer un tableau des points x,y sur un intervalle de
    borne_inf à borne_sup avec un pas de p
    // Déclarations de variables
        borne_inf, borne_sup,p  : réels
        nb_valeurs : entier   // nombre de points pour tracé la courbe
        tab_xy : tableau de couples de réels
    // Début du programme
        // un tableau de couples x,y sur l'intervalle
        Lire("entrez la valeur de la borne inférieure",  borne_inf)
        Lire("entrez la valeur de la borne supérieure", borne_sup)
        Lire("entrez le nombre de points",nb_valeurs)
        // calcul du pas
        p<-- (borne_sup -borne_inf)/nb_valeurs
        deb<-- borne_inf
        Pour i allant de borne_inf à borne_sup avec un pas de p faire
            tab_xy<-- (deb,f(deb))
            deb<-- deb+p
        Fin de pour
    Fin de programme

    // tracé de la courbe
    // il faut balayer le tableau et afficher les points sur un repère. Pour cela on va utiliser les
solutions de ***charts.js*** ou **highcharts.js** pour effectuer ce type de tracé

IMPLEMENTATION DU PROGRAMME
```

2 Fonction de base qui permet d'afficher un message à destination de l'utilisateur : sur la console et plus tard dans une interface.

Une fois que l'on sait comment résoudre ce type de problème , on peut commencer à implémenter la solution dans notre contexte Web et Javascript.

Commençons par préparer notre petit solveur : qui va permettre de résoudre et de nous renvoyer les solutions de l'équation.

La fonction que l'on va développer ressemble à ça
Dans un fichier Javascript associé à une page html

```
// fonction de resolution d'une  equation du second degre
// retour de la fonction peut etre une structure de type
// solutions : {"type" :"unique","valeurs":[valeur]}
//             {"type" :"reelles","valeurs" :[valeur,valeur]}
//             {"type" :"imaginaires","valeurs" :["x1+ix2","x1-ix2"]}
//
function solver(a,b,c){
        var discriminant = Math.pow(b,2)-4*a*c;
        var x1,x2,xi1,xr1;
        var solution={};
        if (discriminant==0) {
                x1=-b/(2*a);
                solution["type"]="unique" ;
               solution["valeurs"]=[x1];
        }else if (discriminant>0) {
                x1=(-b-Math.sqrt(discriminant)) /(2*a);
                x2=(-b+Math.sqrt(discriminant)) /(2*a);
                solution["type"]="reelles";
                solution["valeurs"]=[x1,x2];
        } else { // calcul de complexe
                xr1=-b/(2*a);
                xi1=Math.sqrt(-discriminant) /(2*a);
                solution["type"]="imaginaires";
               if (xr1<0) {
                        solution["valeurs"]=[xr1+"-i"+Math.abs(xi1),xr1+"+i"+Math.abs(xi1)];
                 }else {
                        solution["valeurs"]=[xr1+"+i"+xi1,xr1+"-i"+xi1];
                 }
        }
        return solution;
}
```

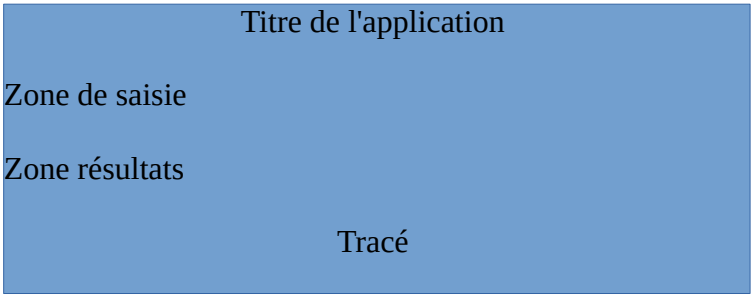
La fonction renvoie une structure contenant deux clés : type et valeur. La première clé permet de distinguer la nature de la solution(unique, réelles, imaginaires). La deuxième clé permet de stocker les valeurs trouvées.

Lorsqu'on développe ce type de code, il faut définir des cas tests afin de vérifier que tout est OK. De plus, on peut s'apercevoir dans ce code, qu'un contrôle des coefficients serait intéressant pour ce éviter de trouver dans des situations qui poseraient des problèmes. Du type si a=0....

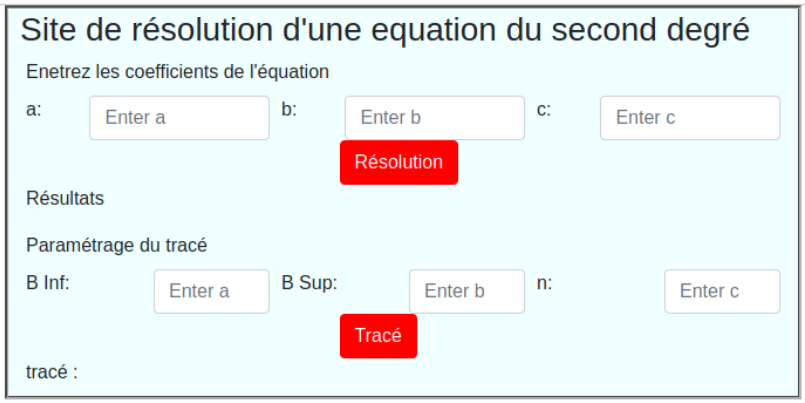
```
Tests de réf pour valider les calculs :
x²+5x+2=0   discriminant= 25-4*1*2=17 x1=-5/2-sqrt(17)/2=-
4.561552812808831
                                      x1=-5/2+sqrt(17)/2=-
0.4384471871911697
x²+x+10=0   discriminant=1-4*10=-39    x1=-0,5+3,12i   x2=-0,5-3,12i
x²+2x+1=0   discriminant=0        x1=-1
```

Une fois que vous avez vérifier le fonctionnement de votre solveur, maintenant on va s'occuper de définir l'interface pour la saisie et la visualisation des résultats

L'interface pourrait ressembler à cela



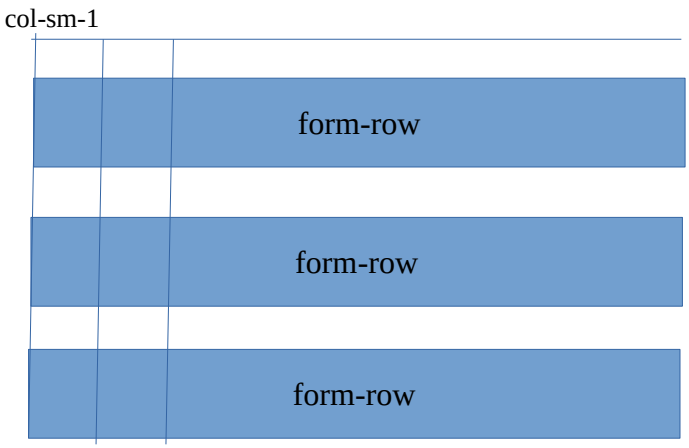
Si on construit maintenant le contenu HTML de cette interface en utilisant Bootstrap pour simplifier le positionnement des éléments et pour avoir également les propriétés responsive de la page.



Comment implémenter tout ça ?

Tout d'abord on va décrire ce que l'on veut faire et on va construire au fur et à mesure les traitements associés aux différents événements.

On constitue le squelette HTML avec les balises et styles de bootstrap. On va constituer notre page en Ligne via la classe form-row et on insère dans chaque ligne les éléments en fixant la taille des colonnes.



Il nous suffit de positionner les différents éléments HTML sur chaque ligne et de définir les différents attributs associés : valeur et l'ID. Le dernier attribut va être important par la suite car il va nous permettre à partir de JS d'accéder à ces différents éléments.

Exemple pour la ligne des coefficients :

```
<div class="form-row">
  <label class="control-label col-sm-1" for="object">a:</label>
  <div class="col col-sm-3">
        <input  type="text" class="form-control valeurs" id="val_a" name="val_a" value="" placeholder="Enter a">
  </div>
  <label class="control-label col-sm-1" for="object">b:</label>
  <div class="col col-sm-3">
        <input  type="text" class="form-control valeurs" id="val_b" name="val_b" value="" placeholder="Enter b">
  </div>
  <label class="control-label col-sm-1" for="object">c:</label>
  <div class="col col-sm-3">
        <input  type="text" class="form-control valeurs" id="val_c" name="val_c" value="" placeholder="Enter c">
  </div>
</div>
```

Le reste du fichier est accessible sur le github suivant :
https://github.com/ystroppa/tutoequa2 dans le fichier index.html

Description du fonctionnement de notre page:

Lors de la saisie des coefficients a,b et c par l'utilisateur on souhaite empêcher la saisie de caractères autres que ceux compris entre 0 et 1, les séparateurs (. ou ,) et les signes (+,-).

Comment faire pour maintenant capturer ce type d'éventement lors de la saisie de caractères, on peut associé à une zone Input les événements claviers de type onkeypress, onkeydown, et on keyup. Si vous voulez vérifier les modules d'écoute possible, se rendre dans la console de votre navigateur et récupérer un descripteur sur le premier input nommé val_a ....
var a=document.getElementById("val_a") ;
a.keypress ...

Comment faire : il faut capter le fait que l'utilisateur saisisse des caractères dans les inputs de notre page HTML ... pour se faire nous allons utiliser l'événement keypress sur les différents zones pour capter le caractères entré par l'utilisateur et éventuellement empêcher le fait qu'il saisisse un caractères qui n'est pas numérique (entier ou réel).

On va construire la fonction traite_caractère()
Le code des caractères numériques est 0 --> 49 et 9 -->57, pour le séparateur nous avons le . ou la virgule , dont les codes sont respectivement 46 et 44.

```
function  traite_caracteres(event) {
      var key = window.event ? event.keyCode : event.which;
          if (event.keyCode === 8 || event.keyCode === 46) {
              return true;
          else if (event.keyCode === 43 || event.keyCode === 45) {
              return true;
          } else if ( key < 48 || key > 57 ) {
              return false;
          } else {
              return true;
      }
},
```

Pour positionner cette gestion d'événement on peut rajouter ce code à la fin du chargement de la page.

var val_a=document.getElementById("val_a") ;
val_a.onkeypress=traite_caracteres ;

On pourra définir pour tous les inputs le même comportement on affectant à chaque input la classe valeurs et en utilisant le querySelectorAll pour positionner l'événement onkeypress sur la fonction qui traite la saisie.

```
var inputs=document.querySelectorAll(".valeurs") ;
inputs.forEach(e=> {
          e.onkeypress=traite_caracteres ;
}) ;
```

Maintenant une fois la saisie contrôlée, il faut afficher l'équation avec les valeurs saisies. Pour ce faire on a défini dans le fichier html une zone spécifique avec un id=equation.
Donc il faut lorsque l'utilisateur lâche le bouton du clavier écrire la forme de l'équation. Comme on utilise un div/paragraphe on peut modifier son contenu en jouant sur la propriété innerHTML ou textContent.

```
function affiche_equation() {
	var z=document.getElementById("equation");
	var val_a=document.getElementById("val_a");
	var val_b=document.getElementById("val_b");
	var val_c=document.getElementById("val_c");
	var a=parseFloat(val_a.value) ;
	var b=parseFloat(val_b.value) ;
	var c=parseFloat(val_c.value) ;
	var signe_a,signe_b,signe_c ;
	a<0?signe_a="" :signe_a="+" ;
	b<0?signe_b="" :signe_b="+" ;
	c<0?signe_c="" :signe_c="+" ;
	console.log(a,b,c);
	z.textContent="Equation : "+signe_a+a+"x²"+signe_b+b+"x"+signe_c+c+"=0" ;
}
```

Le code est trés détaillé on aurait pu aller plus rapidement en simplifiant les écritures

Pour adapter le comportement à la saisie des inputs

```
var inputs=document.querySelectorAll(".valeurs") ;
inputs.forEach(e=> {
		e.onkeyup= affiche_equation ;
}) ;
```

Si on intègre le tout dans notre fichier js, ça nous donne ceci

Fichier solveur.js

```
function affiche_equation() {
	var z=document.getElementById("equation");
	var val_a=document.getElementById("val_a");
	var val_b=document.getElementById("val_b");
	var val_c=document.getElementById("val_c");
	var a=parseFloat(val_a.value) ;
	var b=parseFloat(val_b.value) ;
	var c=parseFloat(val_c.value) ;
	var signe_a,signe_b,signe_c ;
	a<0?signe_a="" :signe_a="+" ;
	b<0?signe_b="" :signe_b="+" ;
	c<0?signe_c="" :signe_c="+" ;
	console.log(a,b,c);
	z.textContent="Equation : "+signe_a+a+"x²"+signe_b+b+"x"+signe_c+c+"=0" ;
}

function  traite_caracteres(event) {
	var key = window.event ? event.keyCode : event.which;
	console.log(event.keyCode);
	if (event.keyCode === 8 || event.keyCode === 46) {
		return true;
	}else if (event.keyCode === 43 || event.keyCode === 45) {
		return true;
	} else if ( key < 48 || key > 57 ) {
		return false;
	} else {
		return true;
	}
}

// mise en place des modules d'écoute associés aux fonctions
var inputs=document.querySelectorAll(".valeurs") ;
inputs.forEach(e=> {
	e.onkeypress=traite_caracteres ;
	e.onkeyup= affiche_equation ;
}) ;
```

Attention à modifier également le fichier index.html pour insérer la ligne suivante juste avant la fin du body.
<script src="solveur.js"></script>
L'insertion doit se faire après le chargement du fichier html et des éléments sinon lors de la mise en place des modules d'écoutes si les éléments ne sont pas créés vous n'allez pas avoir le comportement souhaité.

Prochaine étape c'est de connecter le bouton **résolution** à l'appel de la fonction solver, pour ce faire on a juste à associer dans le fichier index.html le lien entre onClick et la fonction appel_solver. La fonction que l'on doit générer doit gérer également les situations anormales du type ... pas de valeur pour les coefficients ou une expression qui ne correspond par à un numérique de type 21,21,21.
Une fois cette vérification effectuée, elle doit soumettre et effectuer l'appel à la fonction solver en passant les arguments a,b,c, récupérer les résultats pour les afficher dans la partie resultat de notre page HTML.

Ce qui nous donne

```
function appel_solver() {
// Extraction de a,b,c
// Vérification si numérique
// Appel de la fonction solver
// Affichage du résultat
}
```

```
function appel_solver() {
// Extraction de a,b,c
var poursuit=true ;
var a =document.getElementById("val_a") ;
var b =document.getElementById("val_b") ;
var c =document.getElementById("val_c") ;
// Vérification si numérique
if (typeof parseFloat(a.value)!= "number") {
        a.style.background="red" ;
        poursuit=false ;
}
if (typeof parseFloat(b.value)!= "number") {
        b.style.background="red" ;
        poursuit=false ;
}
if (typeof parseFloat(c.value)!= "number") {
        c.style.background="red" ;
        poursuit=false ;
}
 // on arrête à ce niveau si problème de numérique
if (!poursuit) {
        return;  // on retourne à l'appelant
}

// Appel de la fonction solver
var solution= solver(a.value, b.value,c.value);

// Affichage du résultat
var res=document.getElementById("resultat") ;
res.textContent="Résultat : "+solution["type"] +"  " + solution["valeurs"];

}
```

L'étape suivante c'est le tracé de la courbe et éventuellement le marquage des racines de l'équation sur le graphe.

Nous avons besoin pour commencer de récupérer les indications que doit fournir l'utilisateur (bornes et nombre de points) on va procéder la même façon que pour les coefficients en mettant les mêmes contraintes sur la saisie des caractères possibles dans ces champs.
Pour ce faire il nous suffit de les associer à la même classe que pour les autres ... et le tour est joué.

Ensuite il faut associer au bouton tracé la représentation de la courbe entre les bornes et avec le nombre de points demandés.
On va créer une fonction qui récupère les conditions du tracé et constitue le tableau des couples et ensuite appelle la fonction de tracé.

Mais avant de commencer il faudrait créer la fonction d'évaluation f(x)

```
function f(x) {
        var val_a=document.getElementById("val_a");
        var val_b=document.getElementById("val_b");
        var val_c=document.getElementById("val_c");
        var a=parseFloat(val_a.value) ;
        var b=parseFloat(val_b.value) ;
        var c=parseFloat(val_c.value) ;
        return a*Math.pow(x,2)+b*x+c;
}
```

On pourra vérifier à partir de la console du navigateur que l'a fonction est correcte.
Saisir les valeurs dans a,b et c
à partir de la page.html et exécutez dans la console les appels à la fonction f(0), f(10) ....

La préparation du tracé va se faire à partir d'un bloc itératif

```
function preparation_trace() {
        var s_binf=document.getElementById("binf");
        var s_bsup=document.getElementById("bsup");
        var s_nb=document.getElementById("nb");
        var binf=parseFloat(s_binf.value) ;
        var bsup=parseFloat(s_bsup.value) ;
        var nb=parseFloat(s_nb.value) ;
        // on peut commencer les itérations
        var p= (bsup-binf)/nb;
        var x=binf;
        var couples=[];
        for (var i=0; i<nb ;i++) {
                var y=f(x);
                couples.push({x,y});
                x+=p;
        }
        return couples;
}
```

Récupération des paramètres

Calcul du pas

Boucle sur les couples à construire

Une fois cette fonction définie dans le fichier solver.js on va pouvoir de la même façon la vérifier à partir de la console. : on vérifiera le tableau de valeurs qu'elle retourne ...

Maintenant il nous reste à construire la courbe, plusieurs librairies JS permettent de tracer des courbes, nous allons choisir highcharts (voir le site de demo https://www.highcharts.com/demo) qui permet de faire des représentations de haut niveau de différents types de graphe.
On récupère un bloc complet d'un tracé voisin du notre et on va l'adapter notre contexte.
Voilà l'exemple que l'on va récupérer
https://jsfiddle.net/gh/get/library/pure/highcharts/highcharts/tree/master/samples/highcharts/demo/line-basic/

Pour ce faire on s'aperçoit qu'il vous faut pour notre application, plusieurs fichiers js et css à inclure commençons par là .....
A ajouter dans le fichier index.html

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/series-label.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
<script src="https://code.highcharts.com/modules/export-data.js"></script>
<script src="https://code.highcharts.com/modules/accessibility.js"></script>
```

Ensuite il faut définir la zone dans laquelle on va dessiner le graphe

```
<figure class="highcharts-figure">
  <div id="container"></div>
  <p class="highcharts-description">
    Basic line chart showing trends in a dataset. This chart includes the
    <code>series-label</code> module, which adds a label to each line for
    enhanced readability.
  </p>
</figure>
```

Il faut bien sur l'adapter à notre contexte et l'insérer dans la dernière ligne de notre fichier index.html

```
        <div class="form-row col-sm-10">
                <figure class="highcharts-figure">
                        <div id="containerGraphe"></div>
                        <p class="highcharts-description">
                        </p>
                </figure>
        </div>
```

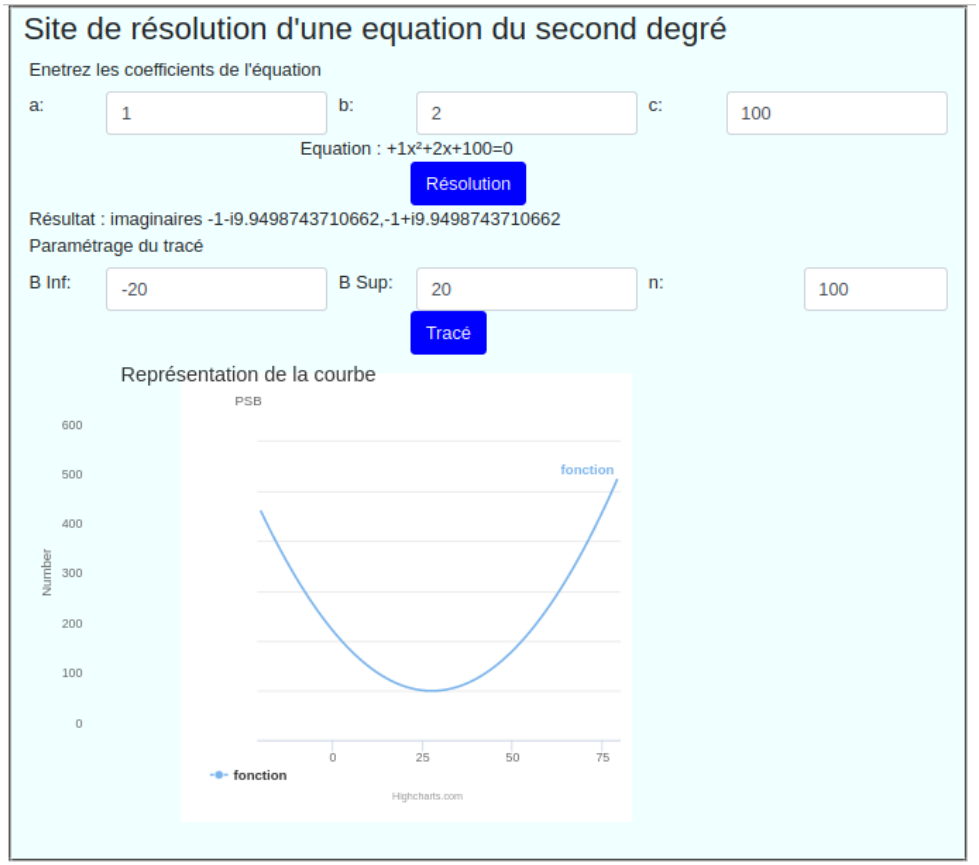
Il suffit également d'aller copier dans la partie style les éléments css

Ensuite on peut revenir sur notre fichier js pour adapter le code dans une fonction que l'on va nommée `trace_graphe(data)`
Cette fonction va recevoir notre tableau x,y

```
function trace_graphe(data){
        // chargement des données pour le tracer
        var Data=[{name: 'fonction',        data: []}];
        var Key=[]; // pour l axe des abcisses
        for (var element in data) {
                Data[0].data.push(data[element].y);
                Key.push(parseInt(data[element].x));
        }
        Highcharts.chart('containerGraphe', {
           chart: {
                scrollablePlotArea: {
                minWidth: 500
                }
        },
        title: {text: 'Représentation de la courbe '},
        subtitle: {text: 'PSB'},
        tooltip: {
        shared: true,
        crosshairs: true
        },
        yAxis: {title: {text: 'Number'}
        },
        xAxis: {Key             },
        legend: {
                layout: 'vertical',
                align: 'right',
                verticalAlign: 'middle'
        },
        plotOptions: {
                series: {
                        cursor: 'pointer',
                        point: {
      events: {
         click: function (e) {
            hs.htmlExpand(null, {
               pageOrigin: {
                  x: e.pageX || e.clientX,
                  y: e.pageY || e.clientY
               },
               headingText: this.series.name,
               maincontentText: ' sessioysns',
               width: 200
                  });
                  }
        }
                },
                marker: {
      lineWidth: 1
        },
                label: {
                        connectorAllowed: true
                },
                pointStart: Key[0]
        }
        },
        series: Data,
        responsive: {
        rules: [{
                condition: {
                        maxWidth: 800
                },
                chartOptions: {
                        legend: {
                                layout: 'horizontal',
                                align: 'center',
                                verticalAlign: 'bottom'
                        }
                }
        }]
        }
        });
}
```

On met x dans les abscisses et y en ordonnées

Une fois fini, voilà ce que l'on obtient



Solution accessible sur https://github.com/ystroppa/tutoequa2

Si on avait un peu plus de temps on pourrait prévoir sur ce petit exemple de mémoriser les représentations effectuées dans une base de données ... par exemple ou de varier la recherche de solution sur des équations plus complexes ...