Story-board JMD

Initiation à JS

Connaître un minimum le HTML
  dont balises <html>, <body>, , <input>, <button>
et les styles CSS :
  dont attributs color, width, text-align, etc
voir http://fr.html.net/tutorials/html/

Vérification préalable :
  Q: qui ne sait pas ce qu'est un éditeur de texte ?
  Q: tout le monde a-t-il bien téléchargé Flashdevelop ou Sublime text ou Visual Studio Code ?
  Q: qui a déjà programmé dans un autre langage et connaît donc les bases de la programmation ?
  Q: qui ne voit pas du tout ce qu'est la programmation ou un langage de programmation ?

Préambule et défs

Déf d'un edit de texte : Comme un TT il permet de saisir un texte mais sans fioritures, juste une suite de mots, phrases, instructions compréhensibles par une machine. C'est l'outil de base du programmeur.

Déf de la programmation :
  C'est l'écriture d'un programme ,
    qui est une suite d'instructions (d'ordres) simples, données à la machine pour qu'elle exécute 1 ou plusieurs tâches complexes qu' on appelle aussi applications (web, mobile).
    Ces instructions sont écrites dans un langage à la syntaxe rigoureuse qu'on appelle Langage de programmation.

    J Script est un langage de programmation ; universel (tout type d'applications) comme les autres. Sa particularité ? Il s'exécute dans (il est compris, interprété par) le navigateur (chrome, firefox, etc) contrairement à ceux qui s'exécute directement sous Windows ou Mac Os.
    Mais noter qu'il existe une version "serveur" de JS qui est Node JS (ateliers Silex conseillés).

Conclusion : On va écrire du javascript avec notre éditeur et on testera dans notre navigateur...

Initier au JS en 2heures = gageure !

on va commencer doucement mais ça ira de plus en plus vite.
  1 heure d'initiation théo et pratique avec exemples courts, simples, sans trop de sens ni de liens entres eux.
  1 heure pour écrire un vrai programme qui a du sens.

Les bases de js

P: Download de https://goo.gl/EPTNJ5 https://github.com/flashline/Atelier-JS-basic
Ouvrir Atelier-bases-js/ex01/index.html
Demander aux collègues de vérifier que tout le monde a son dossier projet ouvert dans son éditeur, notamment pour ceux qui n'aurait que le *blocnote*. Je donne explication pour ceux qui on choisi flashdevelop :
Ouvrir Atelier-JS-basic.fdproj

Expliquer : *<script src="index.js"></script>*

Ouvrir ex01.js

---------- EX01 ----------

T:
Un prog manipule essentiellement des données, des valeurs que le programmeur stocke dans des **variables.**
Les variables peuvent être comparées à des contenants (des tiroirs, boites, bouteilles par ex.) dont le contenu peut **varié.**
On peut aussi se les représenter une adresse -à laquelle on donne aussi un **nom-** et qui permet l'accès à une donnée : Contrairement à un tiroir ou une bouteille, on ne transfert pas les contenus. Quand on copie le contenu d'une var x vers une var y, la var x garde son contenu intacte.

P:
Création de variables
var prenomEleve ;
var => mot clés imposé par JS
; => termine chaque instruction.
prenomEleve =>
c'est le nom choisi par le programmeur
certains caractères sont interdit (espace, +, -, etc)
usages :
On choisi (contrairement au maths) des noms "parlant"
a à z et A à Z et 0 à 9 (pas d'accent en fr)
commence par minuscule. Les majuscules remplace les espaces (chameau)
Il y a plusieurs types de valeurs (donc de variables aussi)
les basiques sont
Les chaînes de caractères alpha-numérique (soit du texte)
ex : *106 av. Ph Auguste* est une chaine
Les nombres (entier, décimaux)

assignation d'une valeur alpha-numérique
prenomEleve = "Jean-Michel" ;
le signe égal => ne signifie pas égale au sens math
mais mettre l'expression de droite dans la variables à gauche.
Les valeurs alpha/num sont tjrs entre " (sinon dans ce cas, JS comprendrait contenu de la var Jean moins celui de Michel)

var ageEleve = 26 ;
On peut donc créer et assigner ds même instruction.
Les nombres sont sans " "
on choisit le type nombre (type JS *Number)* quand on sait qu'on aura besoin de faire des calculs ou des tris numériques. Ex : 26<100 sinon "100" > "26"

Petite question ?

Si je crée :
var codePostal ;
dois-je choisir
codePostal=18270 ;
ou
codePostal="18270" ;
Votez !

Solution : codePostal="18270" ; car
1/ codePostal=03120 ; donnerait 3120
2/ on ne fait généralement pas d'opérations entre les codes postaux.

pour tester :
codePostal="03120" ;
//puis codePostal = 03120 ;

P:
pour tester :

document.write("Mon prénom est "+prenomEleve) ;
document.write(" et j'ai "+ageEleve+" ans.
Mon code postal est "+codePostal) ;
on reverra plus tard les notions expliquant document.write() .
Pour l'instant on se dit juste que ça permet de tester notre programme dans la page web.
Les '+' ne font pas ici d'addition.

--- tester index.html ---

Autres manip avec les valeurs et variables

```
document.write("<br/><br/>--- Concaténation avec signe +");
var nomEleve = "Delettre";
// concaténation
var nomComplet = prenomEleve+" "+nomEleve;
// le <br> ou <br/> c'est du html et signifie passage à la ligne.
document.write("<br>Nom complet "+nomComplet) ;
```

---- TEST -----

```
//variables Number et calcul
var ageEleve1 = 26 ;
var ageEleve2 = 56 ;
var ageEleve3 = 12 ;
var ageEleve4 = 32 ;
var ageEleve5 = 18 ;
var ageMoyen=(ageEleve1+ageEleve2+ageEleve3+ageEleve4+ageEleve5)/5;
// les opérateurs de calcul basiques sont + - / *
document.write("<br>Moyenne des ages : "+ageMoyen+" ans ") ;
ageMoyen=parseInt(ageMoyen);
document.write("<br>Moyenne des ages "+ageMoyen+" ans ") ;
// expression typique avec les 4 opérateurs
document.write("<br/><br/>--- Calcul avec les 4 opérateurs");
var ht=200; var acompte=80; var tva=20;
var aPayer=ht+(ht*tva/100)-acompte ;
// La priorité des * et / sur les – et +, évite certaines ( ) et permet d'écrire.
aPayer=ht+ht*tva/100-acompte ;
// si on veut mettre ht en facteur on doit mettre les parenthèses
//aPayer=ht*(tva/100+1)-acompte  ;
document.write("<br>Net à payer "+aPayer+" €" ) ;
```

Q:

// Faut-il être malin en 'factorisant' les expressions (ou pire
// en remplaçant **/100** par X **10^-2)**
ou bien être le plus compréhensible pour le programmeur suivant ?

---- TEST -----

```
// Les tableaux soit les vars de type Array ou variables indicées
// Quand on a les cas suivants :
// un grand nombre de valeurs (généralement de même catégorie) ou
// un nombre indéterminé de valeurs à stocker...
// le principe est de regrouper la liste des valeurs  dans une même variable
// et de donner à chaque valeur une position de 0 à N
// création array vide
P:
var eleves = [ ] ;
//
eleves[0] = "jeanmi";
eleves[1] = "martine";
eleves[2] = "camel";
eleves[4] = "robert";
eleves[3] = "nicolas";
//
document.write("<br><br>--- Les tableaux (type js Array) :<br>");
// faire copier/coller du 'for'
for (var i=0;i<eleves.length;i++) document.write(eleves[i]+"<br/>") ;
// On expliquera plus tard cette structure de boucle 'for' (etc)
// ici c'est juste pour permettre le test
//       juste dire que la var i va varier de 0 à 4 donc 5 élèves

                                                    ---- TEST -----
// variante pour charger un Array
eleves.push("Joëlle");
eleves.push("René");
eleves.push("Ginette");
eleves.push("Fernande");
// Quand JS exécute la créa du tab à vide avec eleves=[ ], il associe automatiquement
// à eleves un ensemble de fonctions et variables typique d'un Array.
// on verra plus tard ces notions quand on abordera les Function et Object JS
// pour l'instant admettons par exemple que :
// .push() est une fonction qui ajoute un élément en fin tableau et
// .length est une var ou propriété qui contient le nombre d'éléments
document.write("<br>Variante :<br>");
for (var i=0;i<eleves.length;i++) document.write(eleves[i]+"<br/>");
document.write("Nombre d'élèves : "+eleves.length);

                                                    ---- TEST -----
```

les fonctions

```
// appel de fonctions fournies par js (api js, fonctions natives)
// Une fonction JS exécute une tache précise ; retourne parfois un résultat.
//       elle a un nom (similaire à un nom de vars mais choisi par JS) suivi d'une
//       liste de paramètres d'aucun à n ; séparés par une virgule)
alert("bonjour "+eleves[3]);
document.write("<br/><br/>--- Les fonctions (type JS Function) ");
document.write("<br>je suis la fonction write de l'objet document");
// En JS, tous les fonctions sont rattachées à une variable, un 'objet JS' qui lui-même est rattaché à un objet de niveau supérieur sauf l'objet window qui est le niveau le plus élevé.
// Même les fonctions comme alert() ou confirm() devraient s'écrire :
window.alert("je suis la fonction alert de l'objet window (le plus élevé dans la hiérarchie) ");
window.document.write("<br>je suis la fonction write() de document qui lui-même appartient à l'objet window");
// A noter que c'est vrai aussi pour nos variables
document.write("<br/>window.ageEleve : "+window.ageEleve);
```

---- TEST -----

```
// Comme vu précédemment avec les Array, dès qu'on créé une variable, elle possède un ensemble de fonctions natives JS selon sont type.
// On va par ex voir String.substr() qui extrait une sous chaîne à partir d'une chaîne.
document.write("<br/><br>--- Fonction String.substr()  ");
var chaine="abcdefghij" ;
var sousChaine=chaine.substr(3,2);
document.write("<br>"+sousChaine); //  de
// la propriété length de String contient le nombre de caractères.
document.write("<br>longueur de chaine = "+chaine.length);
//
//Array.pop() qui est le contraire de Array.push()
document.write("<br/><br/>--- Fonction Array.pop() ");
document.write("<br><br>"+eleves.pop());
document.write("<br><br>"+eleves.pop());

// A noter que les fonctions sont des variables comme les autres (de type JS Function)
var affich= alert;
affich("J'ai mis alert() dans la variable 'affich'. Je  l'exécute par affich() ");
```

---- TEST -----

```
// Créer ses propres fonctions
// Une fonction programmeur est un bout de code avec un nombre d'instructions plus ou moins important.
// Exemple de petite fonction très pratique.
// Voir dans index.html le <div id="header" >
// Pour récupérer un élément html (div,button,etc) la syntaxe de la function est un peu longue :
// document.getElementById(id de l'élément).
// On va donc créer une fonction qui aura comme nom un simple $ par exemple. (exemple qui confirme la règle)
// Cette function recevra 1 seul argument (paramètre, var en entrée) : l'id de l'élément html
// Nous allons le faire avec les 2 syntaxes possibles.
// 1ère syntaxe
function $ (id) {
        return document.getElementById(id);
}
/ détail :
//              function est le mot-clé imposé par JS
//              $ est le nom qui nous permettra d'appeler (d'exécuter) la fonction
//              Entre les () on a 0 ou plusieurs arguments, séparés par des virgules, pour passer des valeurs à la fonction.
//                      Exemple $("id_de_mon_element"); mettra auto la valeur "id_de_mon_element" dans la var id.
//                      A noter qu'il y a pas besoin d'initialiser les vars arguments par 'var' etc.
//              Le corps de la function est entre des {}
//              Si la fonction doit retourner un résultat le mot clé return est nécessaire
//
// 2ème syntaxe
// var $ = function (id) { return document.getElementById(id) } ;
// on teste
$("header").innerHTML="Je mets ce que je veux dans le sous-titre";
$("header").style.color="red";
$("header").innerHTML+="<br>avec la couleur qui me plait.";

/**/
// 2ème exemple avec 2ème syntaxe:
affich = function (msg) {document.write("<br>"+msg)};
affich ("--- Exemple de ma fonction affich()");
affich ("Hello, dear taxes payers");
// Dans la partie II on fera d'autres fonctions plus longues.
```

---- TEST -----

```
/**/
// Les boucles et exécution conditionnelle
//      le for          =>    répétition de n fois, un bloc d'instructions
//      le while        =>    exécution d'un même bloc tant que la condition est vraie
//      le if           =>    exécution d'un bloc 1 seul fois SI la condition est vraie
//                            ; sinon exécuter le bloc suivant (facultatif)
//
// syntaxe du for :
var max=5;
affich ("<br>--- le bloc FOR");
affich ("affiche les "+max+" 1ers élèves :");
for (var i=0;i<max;i=i+1) {
        affich ( eleves[i]+" a le n° "+(i+1));
}
// Détail
//      for          =>  mot clé
//      var          =>  si la var existe déjà le var n'est pas nécessaire
//      i=0;         =>  i démarrera avec 0 comme valeur
//      i=i+1        =>  à chaque fin de block (entre {}) i sera incrémenté d'un pas de 1
//      i<max        =>  si cette condition est devient fausse, alors JS sort de la boucle.
//                                  les opérateurs de comparaison sont <,>,==,!=,<=,>=
//
// Si je voulais afficher les multiples de 3, de 6 à 30 j'écrirai  :
affich ( "Les multiples de 3, de 6 à 30 avec FOR ");
for (var i=6;i<31;i=i+3) {
        affich ( i);
}
//
// Syntaxe du while
var i=6;
affich ( "<br>--- Les multiples de 3, de 6 à 30 avec WHILE ");
while (i<31) {
        affich (i);
        i+=3; // i=i+3 ;
}
```

---- TEST -----

```
// Syntaxe du if ... else
affich ( "<br>--- Les branchements conditionnels avec IF ELSE ");
var i=10;
if (i%5==0) {
        affich ( ""+i+" est un multible de 5.");
}
else  {
        affich ( ""+i+" N'est PAS un multible de 5.");
}
```

% est un opérateur qui renvoi le reste de la division : i/5

Détail du if :

if et else	sont les mots clé
entre les ( )	condition qui est vraie ou fausse
	si vrai on exécute le bloc entre { } suivant le if
	si faux on exécute le bloc entre { } suivant le else

```
//
// on ajoute un while
i=10;
while (i<30) {
        if (i%5==0) {
                affich ( ""+i+" est un multible de 5.");
        }
        else  {
                affich ( ""+i+" N'est PAS un multible de 5.");
        }
        i++; // i=i+1
}
```

Détail du while:

while	est le mot clé
entre les ( )	condition tant que vraie le bloc entre { } est exécuté
i++;	généralement une instruction modifie le test conditionnel
	facultatif en théorie mais donne une boucle sans fin...

// Note: IF n'est pas obligé d'être couplé à un ELSE. Mais assez fréquent.

---- TEST -----

```
// Les vars de type Object qu'on appelle des objets en fr.
//
// Déf simple :
//       Un objet est une variable qui regroupe en sont sein :
//              de 0 à plusieurs variables (propriétés, attributs)
//              de 0 à plusieurs fonctions
//              de 0 à plusieurs autres objets.
//              un objet peut aussi être vide
//
        De même qu'un tableau regroupait plusieurs éléments en leur donnant un
        indice différent, un objet regroupe plusieurs éléments (membres) en leur
        donnant un nom.
//
//
// Syntaxe
//
affich ( "<br>--- Les vars de type Object ");
var eleve={}; // C'est tout ! objet crée à vide
// Je remplis mon objet : <nom de l'objet>.<nom la variable>
eleve.prenom="Jean-Michel";
eleve.nom="Delettre";
eleve.age=45;
eleve.email="info@pixaline.net";
// on teste
affich (eleve.prenom+" "+eleve.nom+" est agé de "+eleve.age+" ans."+"<br/>"+
                                    "Son adresse mail est "+eleve.email+".<br/><br/>");
//
// Nous allons créer une function que nous ajouteront à chaque objet 'eleve'
var affichCetEleve = function() {
        affich ("<br>"+this.prenom+" "+this.nom+" est agé de "+this.age+" ans."+"<br/>"+
                                    "Son adresse mail est
"+this.email+".<br/>------------------------------<br/>");
};
eleve.affich=affichCetEleve;
//
// A l'exécution, le mot clé 'this' aura comme valeur, l'objet précis à partir duquel
// on appellera la fonction affich() ; soit eleve.affich()
// attendre l'exemple à venir !...

// On a donc un objet avec 4 propriétés et 1 fonction.

eleve.affich() ;
```

---- TEST -----

```
// On va utiliser le tableau 'eleves' pour stocker nos objets 'eleve'
eleves=[];
eleves.push(eleve);
//
eleve={};
// Attention cette syntaxe très simple fait bcp de choses.
// Elle dit à JS met dans 'eleve' un nouvel Object et ne touche
// plus à l'objet que j'y avais mis précédemment.
// En d'autres termes eleves[0] n'est pas affecté.
// Si par contre je faisais directement eleve.prenom="Marie"; eleve.nom="Lombard"; etc
// alors l'objet avec jean-michel etc, serait écrasé.
eleve.prenom="Marie";
eleve.nom="Lombard";
eleve.age="18";
eleve.email="marie.lombard@laposte.net";
eleve.affich=affichCetEleve;
eleves.push(eleve);
//
eleve={};
eleve.prenom="Eric";
eleve.nom="Enneke";
eleve.age="32";
eleve.email="e.neke@gmail.com";
eleve.affich=affichCetEleve;;
eleves.push(eleve);
//
// on teste//
affich("Boucle de tous mes objets 'eleve'");
for (var i=0;i<eleves.length;i++) {
        eleves[i].affich();
}
```

détail :

on parcourt le tableau (Array) 'eleves'
dont chaque élément est un objet (Object) 'eleve'
avec sa fonction affich() on affiche toutes ses propriétés (variables).

---- TEST -----

//
// Les intérêts d'avoir une structure de type Object sont entres autres :
// 1/ Toutes les propriétés et fonctions sont BIEN regroupées
// et donc isolées dans chaque objet.
// chacun ses affaires ! L'objet **encapsule** les données !
// (Les objets de catégorie différente ont chacun leurs propriétés ;
// les objets de même nature (comme les 'eleve' de notre exemple) ont
// juste, des contenus différents.
// 2/ Si une appli a besoin de plusieurs catégories d'objet
// exemple des 'eleve', 'professeur', 'proviseur', etc ;
// Ces différentes catégories auront vraisemblablement des variables
// commune comme prenom et nom
// mais par exemple, 'appreciation', 'niveau', etc appartiendront à 'eleve'
// mais pas à 'professeur' ni 'proviseur'.
// Par contre tous ces objets pourront avoir une function de même nom
// (affich()) mais qui se comportera complètement différemment selon
// la catégorie. (en POO orthodoxe cela s'appelle le polymorphisme)
//

---------- EX02 ----------

Mise en situation :
Une école/boite de formation veut une webapp pour inscrire des élèves à des cours de programmation.
La liste finale sera envoyée à l'email du directeur ou autre.

Faire la démo :
http://www.pixaline.net/intra/Atelier%202018/done/ex02/

Descendre rapidement l' **index.html** en insistant sur :
L'id unique des <input> : prenom, nom ,etc ; qui permet à JS de récupérer le champ <input> en tant que variable. (plus précisément en tant qu'objet JS natif avec toutes ses propriétés et fonctions natives)
Les id des <button> : valide, termine, etc qui seront gérer par js (html ne sait pas faire)

montrer index.nu.html sans le js :
http://www.pixaline.net/intra/Atelier%202018/done/ex02/index.nu.html

ouvrir ex02.js :

on commence par nos utilitaires :

```
function $(id) { return document.getElementById(id);}
function affich(tx) { $("zoneAffich").innerHTML+=tx+"<br/>";}
function efface() { $("zoneAffich").innerHTML=""; $("zoneAffich").style.color="black"; }
```

// dans main
Pour garder les élèves en mémoire on va utiliser un 'tableau' :

```
var eleves=[] ;
```

On va de suite gérer les click sur les buttons (pas vu en 1ère partie)
T :
Quelques définitions :
Les différents objets visuels d'une page html ,<input>,<button> (qu'on appellent balises, blocs, div, span en html) sont en js des "**Element**" ; ils appartiennent tous à la même classe d'objet "Element".

Def d'1 "**événement**" :
Une action liée à un Element ; action généralement déclenchée par l'utilisateur (ex : click, survol d'un bouton ou image).
Les types d'action ex : click et survol, sont appelés **types d'événements**.
Un type d'événement est désigné par un littéral (chaîne de caractère) imposé par JS : Ex : "click" ; "mouseover" ; "input" ; "change" etc ;

Le principe de programmation des événements, est de dire à JS :
quand un événement de **tel type, relié à tel élément,** se produira ; alors exécute la fonction que je t'indique. Le programmeur écrit la fonction mais ne décide pas de son exécution.

Nous n'allons gérer que des types "click" sur des éléments "button"
P:
la fonction native JS qui correspond est addEventListener() ;
littéralement ajoute un écouteur d'événement car plusieurs function peuvent écouter un même événement.
on récupère l'élément origine de l'événement qui comme tout Element possède la function addEventListener.

```
$("valide").addEventListener() ;
```

on indique le type d'événement et notre function

```
$("valide").addEventListener("click",valideListener);
```

idem pour les autres button

```
$("termine").addEventListener("click",termineListener);
$("razForm").addEventListener("click",razListener);
```

On doit donc faire les 3 functions. On les remplira plus tard

```
function valideListener (e) {affich('valide') ;}
function termineListener (e) {affich('termine') ;}
function razListener (e) {affich('raz') ;}
```

--- test rapide --

En attendant
… 2 petites choses de début de programme:
on met comme valeur par défaut du champ adminMail, le mail du "directeur d'école" (le votre), voir le html...

```
$("adminMail").value="info@pixaline.net";
```

C'est en modifiant la prop 'value' d'un Element qu'on modifie sa valeur.
'value' correspond aussi à l'attribut html de même nom.

le bloc 'boxEnvoi' ne doit apparaître que quand on click sur "termine" ; on va donc le rendre invisible.

```
$("boxEnvoi").style.display="none";
```

Toutes les variables (ou propriétés) JS qui ont un équivalent dans les CSS sont rassemblées dans l'objet style que possède tous les éléments.

… et 2 fonctions dont nous aurons besoin.

Une fonction pour afficher les données d'1 élève après validation, qui sera aussi utilisée après l'action "terminer"

```
var versAffich = function  () {}
```

Une pour effacer le formulaire sur click du bouton "effacer", qui sera aussi utilisée après l'action "valider"

```
function razForm () {}
```

c'est fini … y a plus qu'à écrire les corps des functions.

```
function valideListener (e) {
	// Cette fonction va servir pour chaque élève :
	//	tester la saisie
	//	afficher les erreurs OU les données saisies.
	//	ranger les données élève dans dans le tableau élèves
	//
	 var msg=""; // contiendra les messages d'erreurs
	 var inputAge=parseInt($("age").value); // La propriété value d'un élément <input>
	 est toujours une string ; on la transforme donc en un Integer
	 // test : si erreurs de saisie alors msg est chargé
	 if ($("prenom").value=="") msg+="Le prénom est manquant"+"<br/>";
	 if ($("nom").value=="") msg+="Le nom est manquant"+"<br/>";
	 if ( Number.isNaN(inputAge) || inputAge<12 || inputAge>120) msg+="L'age est
	 manquant ou invalide"+"<br/>"; // on voit intérêt d'avoir mis dans une var Int
	 // copier/coller la regex
	 var re= /[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z][A-Z][A-Z]?/i;
	 if (!$("email").value.match(re)) msg+="L'email est manquant ou invalide"+"<br/>";
	    // soit vous irez voir un tuto sur les expressions régulières (reg exp en (en))
	    // sinon c'est cadeau ! prenez ce code tel quel pour tester simplement
	    // un email !
	 //
	 // On va effacer la zone d'affichage.  (elle cumule les messages ; ne les remplace
	 // pas)
	 efface(); // efface() remet aussi la couleur du texte en noir
	 //
	 // si il y a des erreurs je les affiche
	 if (msg!="") {
	        // je mets ma "zoneAffich" en rouge
	        $("zoneAffich").style.color="red"; // littéral 'red' est imposé par JS et
	         // appartient aussi à la syntaxe CSS
	        affich(msg);
	 }
	 //
	 // sinon je stocke les données dans mon tableau eleves :
```

```
        else {
                var eleve={}; // crée un object
                eleve.prenom=$("prenom").value;
                eleve.nom=$("nom").value;
                eleve.age=inputAge;
                eleve.email=$("email").value;
                eleve.choix=$("choix").value;
                eleve.info=$("info").value;
                // je relie la fonction versAffich à la propriété versAffich de l'objet eleve.
                // On verra plus tard l'intérêt...
                eleve.versAffich=versAffich;
                    // on a créé un object 'eleve' avec des propriétés et function comme JS
                    // le fait avec ses object natifs
                //
                //
                eleves.push(eleve); // je mets l'élève à la fin du tableau eleveS.
                affich("<b>inscription bien effectuée : </b>");
                eleve.versAffich();
                razForm();
        }
}
```

```
function versAffich () {
        // Cette fonction est affectée à chacun des object eleve et affiche des propriétés
        de cet object. La syntaxe utilisée pour viser les bonnes variables est this
        this signifie l'object auquel cette fonction appartient.
        affich (this.prenom+" "+this.nom+" agé(e) de "+this.age+" ans,"+"<br/>"+
                        "email "+this.email+", est inscris en "+this.choix
                );
}
function razForm () {
        $("prenom").value="";
        $("nom").value="";
        $("age").value="";
        $("email").value="";
        // $("choix").value="Java Script.";
        $("info").value="";
};
```

--- test rapide ---

```
function termineListener (e) {
        // Cette fonction va afficher le récapitulatif de tous les 'eleve' saisis ;
        // et proposer l'envoi de cette liste par mail, au directeur (vous)
        efface();
        affich("<b>Récapitulatif de votre saisie : </b>");
        // Pour chaque élève du tableau élèves...
        for (var i=0;i<eleves.length;i++) { // un tableau commence tjrs à zéro
        //… on l'affiche grâce à sa fonction versAffich() de l'objet 'eleve' que
        // l'on a 'pushé' dans le tableau eleves
                eleves[i].versAffich();
                //
                if (eleves[i].info!="")
                    affich("Informations supplémentaires : <br/>"
                    +eleves[i].info);
                affich(" ----------------------------------------------");
        }
        //
        // on rend l'élément boxEnvoi visible
        $("boxEnvoi").style.display="block";
        // le littéral 'block' est imposé par JS et appartient aussi à la syntaxe CSS
        //
        // et on ajoute l'écouteur d'événement pour le bouton envoi de l'email
        $("envoi").addEventListener("click",envoiListener);
}
// reste function  envoiListener et razListener

function razListener (e) {
        // le seul intérêt de la fonction razForm() séparée, est de ne pas appeler  une
        // fonction listener en dehors d'un addEventListener
        razForm();
}
```

```
function envoiListener (e) {
        // Astuce. On récupère le récapitulatif affiché pour le mettre dans le champ <input
        type='hidden' />
        //
        $("recapMail").value=$("zoneAffich").innerHTML;
        // on envoi au server
        $("formMail").submit();
            // le submit() va faire que JS va appeler le programme serveur mis
            // dans l'attribut 'action' du formulaire et lui envoyer les champs
            // de name 'adminMail' et 'recapMail' (même si ce dernier est 'hidden')
            // (voir index.html)
}
```

--- On teste et débogue ---

NB :
j'ai écris le prog server en php. Je ne le détaille pas, mais il est dans votre dossier.

// Conclusion :

Le corrigé sera sur github à la même adresse avant demain soir
ainsi que le support d'ici quelques jours

si ok, envoyer mails à info@pixaline.net si vous voulez d'autres ateliers
même thème mais plus poussés :

- 1/ avec approche POO façon JS
- 2/ Utilisation de langage comme TypeScript ou Haxe
  qui compile en JS et
  permettent une POO plus orthodoxe.
- 3/ les pattern (modèles de programmation) , etc