Support Initiation JS

partie 2 (ex02)

légendes pour la suite :

T: => Théorie
P: => Pratique
Mot en **gras** => mot ou expression typique de JS
texte en bleu ; : => code JS

Mise en situation :
Nous sommes une équipe de programmeurs, à laquelle un client -une école/boite de formation informatique- nous a donné le cahier des charges suivant :
Créer une application web permettant l'enregistrement à la volée d'étudiants venant s'inscrire à un des cours de programmation. La liste finale des données des étudiants devra être envoyée à l'email (saisi) du responsable de l'école.
Notre web-app devra effectuer des contrôles d'existence et de validité des données saisies (ex : email et age valide, nom et prénom présents)

Voir la démo montrant comment notre application devra fonctionner :
http://www.pixaline.net/intra/Atelier%202018/done/ex02/

Ouvrir ex02/index.html .
T:
Noter que les champs de saisie (balise <input>), les boutons (balise <button>) et certains blocs (balise), possèdent un attribut **id** avec une valeur unique pour chaque balise.
C'est cet **id unique** qui permet à JS de récupérer ces balises html et en faire des objets JS de type **Element**
Ces objets Element auront d'office différentes propriétés et fonctions natives qui nous permettront de les manipuler ;

P:
ouvrir ex02/index.js :

on commence par nos utilitaires similaires à ceux vus dans le 1er atelier :

Récupérer un Element existant dans index,html

```
function $(id) { return document.getElementById(id);}
```

Ajouter du texte dans la zone réservée à l'affichage, soit

```
function affich(tx) { $("zoneAffich").innerHTML+=tx+"<br/>";}
```

Effacer cette zone et réinitialiser la couleur du texte en noir

```
function efface() { $("zoneAffich").innerHTML=""; $("zoneAffich").style.color="black"; }
```

Dans la partie principale
// main
Une variable de type **Array** nous semble tout à fait indiquée pour stocker les élèves en mémoire d'autant que leur nombre total est indéterminé. Nous le créons à vide en début de programme :
var eleves=[] ;

On va de suite gérer les clics sur les boutons (pas vu en 1ère partie)

T :
Quelques définitions :

Les différents objets visuels d'une page html ,<input>,<button> (qu'on appelle balises en html) sont en JS des "**Element**" ; c'est à dire qu'ils appartiennent à la même classe d'objet : "Element".

Définition d'1 '**événement**' (**Event** en anglais):

C'est une action effectuée sur un Element ; action généralement déclenchée par l'utilisateur (ex : clic, survol d'un bouton ou entrée dans un champ).
Chaque événement a un type appelé **type d'événement** (event type**)**.
Un type d'événement est désigné par un littéral imposé par JS : Ex : '**click**' , '**mouseover**' , '**input**' , '**change**' etc .

Le mécanisme de programmation des événements, est de dire en JS :

« Quand un **événement** de **tel type, relié à tel élément,** se produira, alors exécute la **fonction** que j'indique ».
On dit aussi que la fonction "écoute" l'événement, qu'elle est un "écouteur d'événement", en anglais '**event listener**'.
Noter que le programmeur écrit la fonction mais ne décide pas du moment de son exécution ; ce dernier ne dépendant que de l'utilisateur.

P:
Nous n'allons ici gérer que des types "click" sur des éléments "button"

la fonction native JS qui gère ce mécanisme est **addEventListener()** ;
littéralement *ajoute un écouteur d'événement*.
Noter que plusieurs fonctions peuvent écouter un même événement même si nous n'aurons pas besoin de cette possibilité dans notre application.

on récupère l'élément, en l'occurrence le bouton d'id 'valide' ...

```
$("valide")
```

qui comme tout Element possède la fonction addEventListener ...

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

on indique le type d'événement et le nom d'une fonction (que nous allons écrire)

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

idem pour les autres button

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

Nous devons donc créer les 3 fonctions que nous développerons plus tard

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

Noter que les fonctions 'listener' reçoivent toujours un paramètre 'e' envoyé par JS, mais dont nous n'aurons pas besoin dans notre projet.

--- tester index.html ---

On continue ...
… 2 petites choses à écrire dans la partie //main :
1/ On met comme valeur par défaut du champ 'adminMail', le mail du responsable de l'école (le notre pour l'instant afin de tester notre application à la fin), voir le <input id="adminMail" /> dans index.html.

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

C'est en modifiant la prop **value** d'un Element qu'on modifie sa valeur.

2/ Le bloc 'boxEnvoi' ne doit apparaître que quand l'utilisateur cliquera sur "termine" ; on va donc au début le rendre invisible.

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

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

…/

… et 2 fonctions dans la partie
// functions
dont nous aurons besoin mais que nous développerons plus tard.
Une pour afficher les données d'1 élève après validation, qui sera aussi utilisée après l'action "terminer" pour afficher tous les élèves.
function versAffich () {}
Une pour effacer le formulaire sur click du bouton "effacer", qui sera aussi utilisée après le click du bouton "valider".
function razForm () {}

c'est fini :-)
… enfin, il n'y a plus qu'à écrire les corps des fonctions :-(

```
function valideListener (e) {
    // Cette fonction va servir pour chaque saisie d'élève :
    //      à tester la validité des <input>.
    //      à afficher les erreurs OU les données saisies.
    //      à ranger les données élève dans dans le tableau de nom 'eleves'
    //
     var msg=""; // contiendra les éventuels messages d'erreurs.
     La propriété value d'un élément <input> est toujours une string ; on la
     transforme donc en un Integer :
      var inputAge=parseInt($("age").value);
     On teste les entrées ; si il y a des erreurs de saisie alors les messages sont
     concaténés dans 'msg' :
    if ($("prenom").value=="") msg+="Le prénom est manquant"+"<br/>";
    if ($("nom").value.length==0) msg+="Le nom est manquant"+"<br/>"; //variante
    C'est un peu plus compliqué pour l'age.
    On voit l'intérêt de l'avoir mis dans la variable 'inputAge'
    if ( Number.isNaN(inputAge) || inputAge<12 || inputAge>120) msg+="L'age est
    manquant, invalide ou en dehors de la tranche 12 à 120 ans"+"<br/>";
    Pour l'email copier/coller la regex  (expression régulière en fr) :
    var re= /[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z][A-Z][A-Z]?/i;
    if (!$("email").value.match(re)) msg+="L'email est manquant ou invalide"+"<br/>";
        Les regular expressions méritent un atelier à elles seules ; donc
        soit vous chercherez un tutoriel sur les expressions régulières
        sinon c'est cadeau ! Prenez ce code tel quel pour tester simplement
        un email ;-)

     On va effacer la zone d'affichage.  (la fonction affich() ne remplace pas les
     textes ; mais les cumule)
    efface(); // efface() met aussi la couleur du texte en noir

    Si il y a des messages d'erreurs dans 'msg' on l'affiche
    if (msg!="") {
    On met avant, la <div id="zoneAffich"> en rouge en utilisant le littéral 'red' imposé
    par JS et  appartenant aussi à la syntaxe CSS :
            $("zoneAffich").style.color="red";
            affich(msg);
    }
```

…/

```
        Sinon, c'est-à-dire si msg est vide,
        on stocke les données dans le tableau eleves et on affiche la saisie :
        else {
                Création d'un object dans la variable temporaire 'eleve' ...
                var eleve={};
                 … afin de stocker chaque valeur saisie dans chacune des propriétés de cet
                 object.
                eleve.prenom     =$("prenom").value;
                eleve.nom        =$("nom").value;
                eleve.age        =inputAge;
                eleve.email      =$("email").value;
                eleve.choix      =$("choix").value;
                eleve.info       =$("info").value;
                On relie la fonction versAffich à la propriété versAffich de l'objet eleve.
                On verra plus tard l'intérêt...
                eleve.versAffich  =versAffich;
                 Notons que nous avons créé un object avec des propriétés et function comme JS le fait
                 pour ses object natifs.

                On 'pousse' notre Object à la fin de notre Array :
                eleves.push(eleve);
                On signale à l'utilisateur que tout c'est bien passé :
                affich("<b>inscription bien effectuée : </b>");
                On affiche les données saisies
                eleve.versAffich();
                On efface les champs avant la prochaine saisie.
                razForm();
        }
}
```

Note :
Toutes les lignes jusqu'à *eleves.push(eleve);* incluse, peuvent être remplacées par une seule ligne en utilisant cette syntaxe alternative :

```
eleves.push({prenom:$("prenom").value,nom:$("nom").value,age:inputAge,email:$
("email").value,choix:$("choix").value,info:$("info").value,versAffich:versAffich});
```

Qu'on peut aussi disposer par exemple, ainsi :

```
eleves.push (
        {       prenom      :$("prenom").value,
                nom         :$("nom").value,
                age         :inputAge,
                email       :$("email").value,
                choix       :$("choix").value,
                info        :$("info").value,
                versAffich  :versAffich
        }
) ;
```

A vous de voir ce qui est le plus clair pour vous ...

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

--- tester les boutons valide et efface d'index.html ---

```
function termineListener (e) {
        Cette fonction va afficher le récapitulatif de tous les élèves saisis ;
        et proposer l'envoi de cette liste par mail, au responsable (vous)
        efface();
        affich("<b>Récapitulatif de votre saisie : </b>");
        Pour chaque élève (qu'on a 'pushé' dans le tableau eleves)
        for (var i=0;i<eleves.length;i++) {
                on l'affiche en utilisant sa fonction versAffich() :
                eleves[i].versAffich();
                On affiche une petite ligne de séparation
                affich(" ----------------------------------------------");
        }
        On rend l'élément d'id 'boxEnvoi' visible
        $("boxEnvoi").style.display="block";
        On ajoute l'écouteur d'événement pour le bouton envoi de l'email
        $("envoi").addEventListener("click",envoiListener);
}
```

Il nous reste les fonctions razListener et envoiListener à écrire

```
function razListener (e) {
        razForm();

        Note :
        On peut se demander pourquoi ne pas avoir mis le corps de la fonction razForm
        dans celui de razListener afin d'utiliser l'appel direct à razListener() à la place de
        razForm() (voir plus haut à la fin de la fonction valideListener).
        L'intérêt de la fonction razForm() est de séparer la fonction listener qui contient
        un paramètre (e) de celle qui est appelée directement sans paramètre, depuis un
        autre endroit du programme.
        Si par la suite on devait ajouter des lignes à razListener ne devant pas être
        exécuter dans le cas de l'appel direct à razForm(), on pourrait le faire sans rien
        perturber. Idem si on devait ajouter un paramètre à razForm, utilisé que dans le
        cas de l'appel direct.
}
```

.../

```
function envoiListener (e) {
        Astuce :
        On récupère le récapitulatif se trouvant dans la <div> d'affichage pour le mettre
        dans le champ <input type='hidden' id="recapMail" />
        Car un <input> de type 'hidden' ne se voit pas dans la page mais sa valeur est
        bien envoyée au serveur.
       $("recapMail").value=$("zoneAffich").innerHTML;

        On va envoyer $("recapMail").value et $("adminMail").value au serveur
       $("formMail").submit();
          Détail :
             Tout Element <form> (FormElement) possède nativement une fonction
             submit qui à l'exécution va,
                appeler le programme serveur mis dans l'attribut 'action' du <form>,
                lui envoyer pour chaque <input> la valeur de son attribut 'name' et celle de
                son attribut 'value'.
             Ce qui va se passer côté serveur ne concerne plus cet atelier.
             Notons juste que le programme serveur récupérera très facilement, ces
             couples (name,value) et pourra donc effectuer l'envoi du mail.
}
```

On teste index.html, on débogue index.js
et on recommence...

Pour info, jeter un œil sur *ex02/sendMail.php*
Ce programme est là juste pour nous permettre de tester l'application jusqu'au bout.
Même sans connaître PHP, vous pourrez constater qu'il y a quelques similitudes avec JS et vu le peu de lignes vous comprendrez plus ou moins ce que fait ce programme ;-)

Pour connaître les dates des prochains ateliers Silex : https://www.silexlabs.org/

Mon email : info@pixaline.net pour me signaler si vous êtes intéressé(e)s par d'autres ateliers sur la programmation mais plus poussés :

1. Approche POO orientée prototype de JS
2. POO orthodoxe avec le langage Haxe ou TypeScript, qui compilent en JS :
3. Les patterns (modèles de programmation) POO.

Ou sur

1. La cryptographie : de César à la Quantique en passant par RSA.

Le chat Silex : https://framateam.org/silexlabs/channels/ateliers2 pour proposer des ateliers que vous voudriez animer.

--- A BIENTOT ---