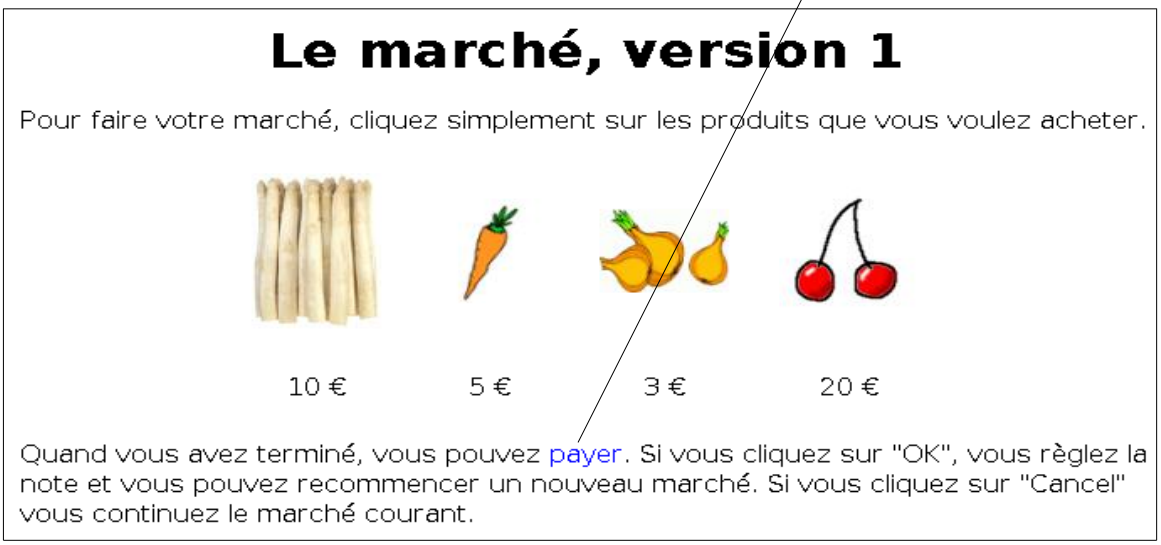
DEVELOPPEMENT WEB FRONT-END
TP N°2 - HTML/JavaScript/DOM -
(*Initiation et/ou révision*)

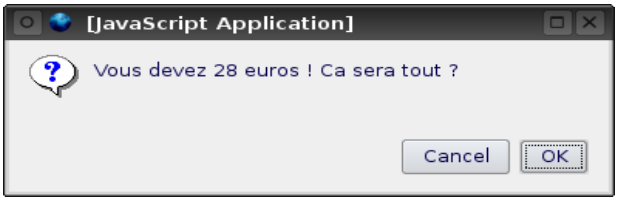
Exercice 1 : (JavaScript : E/S, tests, boucles, array, fonctions)

Le marché, version 1

On veut écrire une page à partir de laquelle on peut faire son marché. Pour remplir son panier, il suffit de cliquer sur le produit désiré. On peut cliquer plusieurs fois sur les mêmes produits. Quand on a terminé, on peut payer, ce qui a pour effet de vider le panier. On peut alors recommencer un nouveau marché.



Si on clique sur "payer" alors que le panier n'est pas vide, il faut afficher et confirmer le montant :



Si on clique sur "payer" alors que le panier est vide, il faut afficher un message du type :

Les images sont dans /home/public/2A_INFO/WEB_FRONT_END/TP2_JS_DOM/1_MARCHE/marche.zip.

1. Réalisation de la partie HTML

- Faire un tableau HTML avec :
  - 1ère ligne : les 4 images de fruits et légumes. Chaque image sera associée à un lien représentant un appel de la fonction JavaScript nommée "choisir()". Exemple :
    <td> <img src="asperges.png"></td>
  - 2ème ligne : les 4 prix en euros

- Ecrire le texte en bas "Quand vous avez terminé .....vous pouvez payer ....." en mettant un lien hypertexte su le mot "payer" représentant un appel de la fonction JavaScript nommée "payer()".

2. Réalisation de la partie JavaScript

- Déclarer une variable globale "total" qui contiendra le total à payer.
- Écrire les 3 fonctions suivantes (celles-ci sont simples dans cette version, mais elles seront enrichies dans les versions 2) :

choisir(produit) : ajoute à "total" le prix du produit passé en paramètre. Appelle la fonction "*prix()*".

prix(produit) : retourne le prix du produit passé en paramètre.

payer() : si le total est 0, afficher une boîte alerte "votre panier est vide!", sinon afficher dans une boîte de confirmation "Vous devez .... euros ! Ce sera tout ?). Dans l'affirmatif, remettre "total à" 0.

Le marché, version 2

Écrivez maintenant une page HTML pour pouvoir faire son marché de façon plus élaborée. Dans cette nouvelle version, lorsqu'on clique sur un produit, il faut ensuite préciser la quantité désirée :

Si on clique sur "panier" on doit voir s'afficher le contenu et le prix du panier courant :

Le panier doit rester trié tout le long du marché : si le client achète 1 Kg de carottes, 0.5 Kg d'asperges, puis de nouveau 1 Kg de carottes, et demande ensuite de voir l'état de son panier courant, il doit voir *2 Kg de carottes, 0.5 Kg d'asperges*.

1. **Réalisation de la partie HTML**

- idem solution 1, mais modifier le texte "Pour faire votre marché, ..." en mettant un lien hypertexte sur le mot "panier" représentant un appel de la fonction JavaScript nommée "voir_panier()".
  Dans onclick="choisir('asperges')" modifier le nom du produit par son n° d'emplacement (0).

2. **Réalisation de la partie JavaScript**

- Déclarer en plus les variables globales suivantes :
  "produits" : tableau contenant les 4 produits
  "prix" : tableau contenant le prix de chacun des produits
  "panier" : tableau contenant la quantité de chacun des produits commandés. Initialement vide.
  "total" : montant total du panier. Initialement à 0.

  choisir(numproduit) : le prix du produit dont le n° est passé en paramètre doit être multiplié par la quantité demandée à partir du prompt affiché ; à ajouter à "total".

  voir_panier() : Appelée sur clic panier. Dans une boîte *alert*, afficher le contenu et le prix du panier courant.

  payer() : idem solution 1, mais ajouter l'affichage du contenu du panier en appelant "voir_panier()".

REMARQUE DANS CET EXERCICE1

Le code JavaScript devrait être séparé du balisage HTML pour une meilleure lisibilité. Il aurait fallu supprimer les *onclick="fonction()"* et utiliser la méthode JavaScript *addEventListener()*. Ce qui permettrait d'ajouter des écouteurs d'événement même lorsque vous ne contrôlez pas le balisage HTML, comme ci-dessous :

```
element.addEventListener("mouseover", myFunction);
element.addEventListener( "click", function(){ myFunction(p1, p2); } );
```

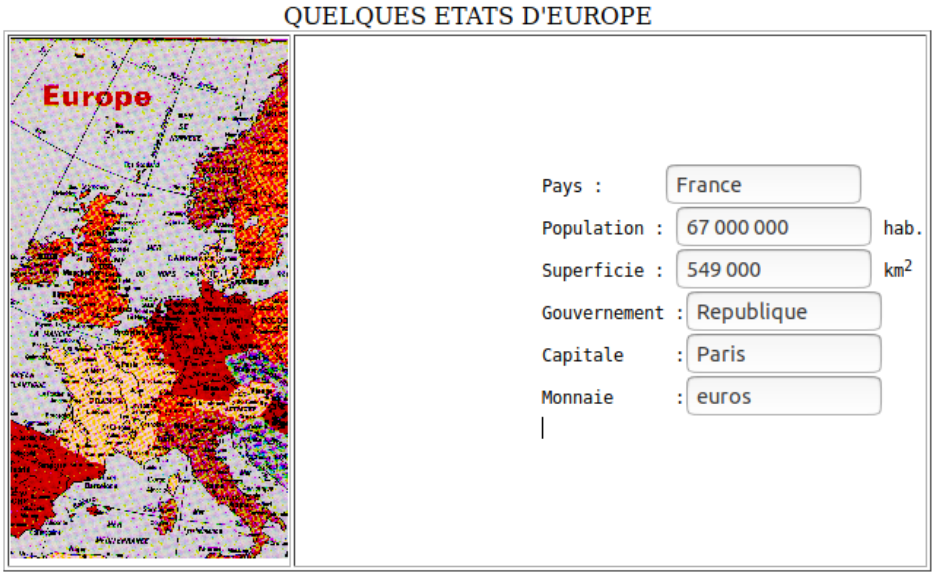
Lire :

https://www.w3schools.com/js/js_htmldom_eventlistener.asp

Dans certains exercices suivants, il faudra le faire une fois que vous aurez maîtrisé le DOM.

Exercice 2 : (JavaScript :Événements, array, fonctions, formulaire)

Dans l'exemple ci-dessous, le passage de la souris sur la France génère un événement *onmouseover* qui provoque l'appel de la fonction *affiche(France).* La fonction JavaScript *affiche(pays)* permets d'afficher les caractéristiques du pays à droite de la carte.
Le retrait de la souris de la France génère une événement *onmouseout* qui provoque l'appel de la fonction *affiche(efface)*. La fonction JavaScript *affiche(efface)* permets d'effacer les caractéristiques du pays à droite de la carte.
Enfin, sur clic de souris sur le pays, une nouvelle page web s'ouvre pour afficher l'office de tourisme du pays.



1. **Réalisation de la partie HTML**

- *balises : map, area, form, table, caption, tr, td, img, input, map*
  attributs : name, src, href, usemap, coords, shape, onmouseover, onmouseout
  Utiliser :
  - https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onmouseover
  - https://www.w3schools.com/tags/tag_area.asp

  L'image *europe.gif* se trouve dans /home/public/2A_INFO/WEB_FRONT_END/TP2_JS_DOM/2_CARTE_EUROPE/
  Idem pour le fichier *map.html* contenant les coordonnées de chacun des pays.

  Compléter le fichier *map.html* fourni : La *map* est composée de régions <*area*> représentées par un polygone *shape="poly",* de coordonnées *coords="a,b,c, ..."* sur la carte. Chaque région représente un pays européen.
  Pour chaque pays qui vous intéresse, ajouter à <area> un attribut *href* avec comme valeur une *url* du site web touristique du pays associé.
  Copier le contenu de *map.html* et le coller dans le *body* de votre fichier HTML.
  Faire un formulaire (*form)* composé d'un tableau à 2 colonnes : à gauche l'image, à droite les 6 *input*. Ajouter un attribut *name* pour chaque *input* ainsi que pour *form*. Associer à l'image la <*map*> dont l'attribut *name* est « Europe ».

2. **Réalisation de la partie JavaScript**

- Pour chacun des pays qui vous intéressent, déclarer un tableau (*Array*) composé de 6 chaînes correspondants aux propriétés suivantes : nom, population, superficie, gouvernement, capitale, monnaie. Le nom de chaque tableau est le pays.
  Exemple : *var France = ["France","67 000 000","549 000","République","Paris", "euros"];*
  Le tableau de nom *efface* est un tableau de 6 chaînes vides : *var efface = ["", "", "", "", "", ""];*
- Définir la fonction *affiche(pays)* qui affecter les valeurs des propriétés du paramètre *pays* dans les *input* du formulaire.
  Utiliser : *document*, *value* (initiation au DOM).

3. **Finition de la partie HTML**

  Retourner dans la *map* pour ajouter les 2 attributs représentant les événements suivants :
  onMouseOver : sur passage de la souris sur un pays, appelle *affiche(lepayschoisi) (ex : affiche(France))*
  onMouseOut : sur sortie de la souris, appelle la fonction *affiche(efface) où efface* est un tableau de 6 cases vides.

Exercice 3 : (array, boucles, tests)

Réaliser une page web affichant pour chaque année *x* le message "*x* est une année bissextile" si *x* est bissextile, ou affichant "*x* n'est pas une année bissextile" si *x* ne l'est pas. La ligne sera affichée en vert dans le 1er cas et en rouge dans le second. Les années choisies seront initialement réunies dans un tableau. Une année est bissextile si elle divisible par 4 mais pas par 100, ou si elle est divisible par 400.

Utiliser :
https://www.w3schools.com/jsref/met_document_queryselector.asp
https://www.w3schools.com/jsref/prop_style_color.asp

Version 1 : avec innerHTML (simple mais exécution plus lente)

https://www.w3schools.com/jsref/met_document_queryselectorall.asp

Version 2 : avec DOM (complexe mais exécution rapide)

Obtenir le même résultat que Version 1.
Utiliser :
https://www.w3schools.com/jsref/met_document_createelement.asp
https://www.w3schools.com/jsref/met_document_createtextnode.asp
https://www.w3schools.com/jsref/met_node_appendchild.asp

Exercice 4 : (E/S, tests, boucles, array, fonctions)

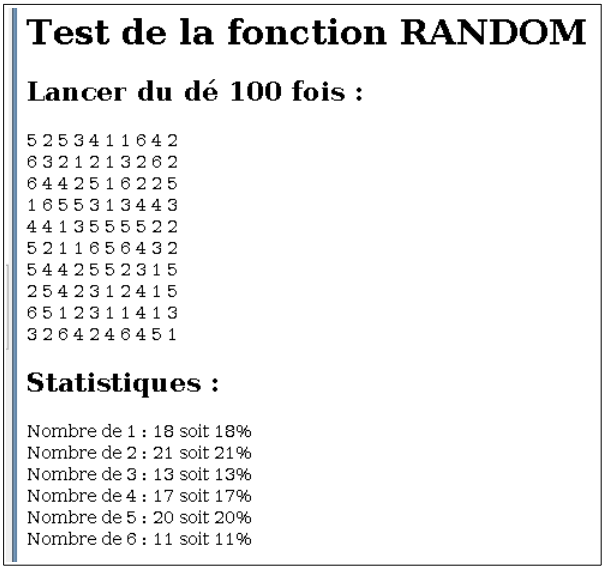
Réaliser une page web affichant les résultats de lancés d'un dé ainsi que ses statistiques. Le nombre de lancés est saisi par l'utilisateur à travers une boîte de dialogue. Pour chaque chiffre du dé, l'on calculera le nombre d'apparitions et son pourcentage (voir un exemple de page web à obtenir ci-dessous). Pour cela, un tableau (*resultat*) de taille 6 sera utilisé pour mémoriser les scores. Le programme JavaScript sera décomposé en fonctions avec un paramètre éventuel, décrites comme suit :

init() : initialise à 0 le tableau *resultat* (variable globale)

de() : retourne une nombre compris entre 1 et 6. Utiliser la fonction **Math.random()** qui donne un nombre au hasard dans l'intervalle **[0 , 1 [**, et la fonction **Math.floor(x)** qui fournit l'arrondi à l'entier le plus proche (définies avec l'objet Math).

lancer(nbfois) : lance *nbfois* le dé. Affiche les numéros obtenus. Mémorise dans le tableau *resultat* le score pour chacun des numéros.

afficher(nbfois) : afficher les statistiques (nombre d'occurrences de chaque chiffre de 1 à 6 et pourcentages).

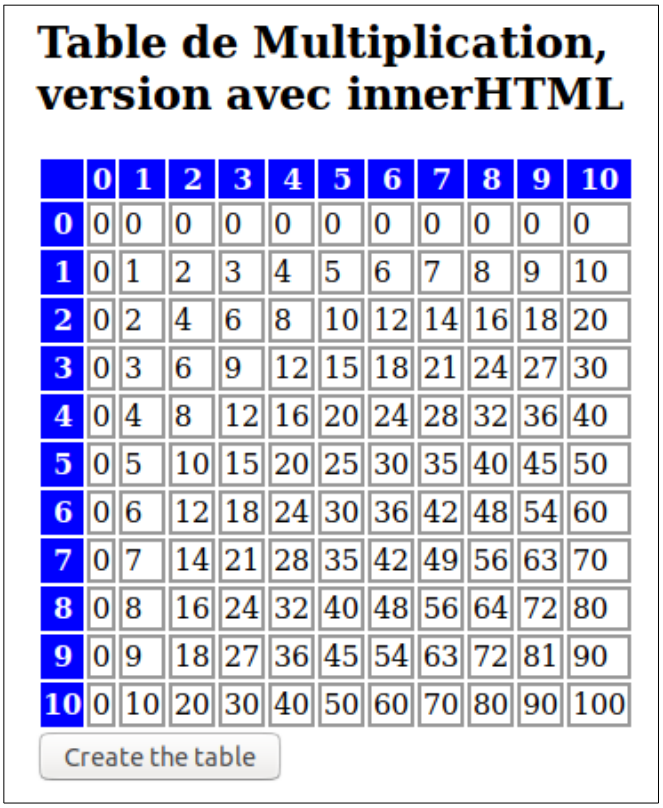


Version 1 : avec innerHTML (simple mais exécution plus lente)

Utiliser :
https://www.w3schools.com/jsref/met_document_queryselector.asp
https://www.w3schools.com/jsref/met_document_queryselectorall.asp

Exercice 5 : (E/S, boucles, fonctions)

Réaliser une page web affichant la table de multiplication selon l'exemple de la Figure ci-dessous.
Pour cela, donner un titre : "**Table de multiplication version ….**" et créer un bouton « Create the table ». Sur clic sur le bouton, une fonction *createTable()* sera appelée permettant à l'internaute de répondre aux 2 questions « *Nombre de lignes?* » puis « *Nombre de colonnes?* » pour créer la table (utiliser 2 *prompt*).
Dans la fonction *createTable(),* le tableau HTML devra être généré en JavaScript à la volée. Pour appliquer une couleur de fond et de texte particulièrement à la 1ère colonne et la 1ère ligne, utiliser la balise "<th>" et appliquer la couleur à ces éléments à l'aide du CSS DOM. Enfin, le tableau sera centré avec bordures.



	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	12	14	16	18	20
3	0	3	6	9	12	15	18	21	24	27	30
4	0	4	8	12	16	20	24	28	32	36	40
5	0	5	10	15	20	25	30	35	40	45	50
6	0	6	12	18	24	30	36	42	48	54	60
7	0	7	14	21	28	35	42	49	56	63	70
8	0	8	16	24	32	40	48	56	64	72	80
9	0	9	18	27	36	45	54	63	72	81	90
10	0	10	20	30	40	50	60	70	80	90	100

Utiliser :
https://www.w3schools.com/jsref/met_document_queryselector.asp
https://www.w3schools.com/jsref/prop_style_color.asp

Version 1 : avec innerHTML (simple mais exécution plus lente)

https://www.w3schools.com/jsref/met_document_queryselectorall.asp

Version 2 : avec DOM (complexe mais exécution rapide)

Obtenir le même résultat avec DOM, et modifier le titre du tableau par « …..version avec DOM ».

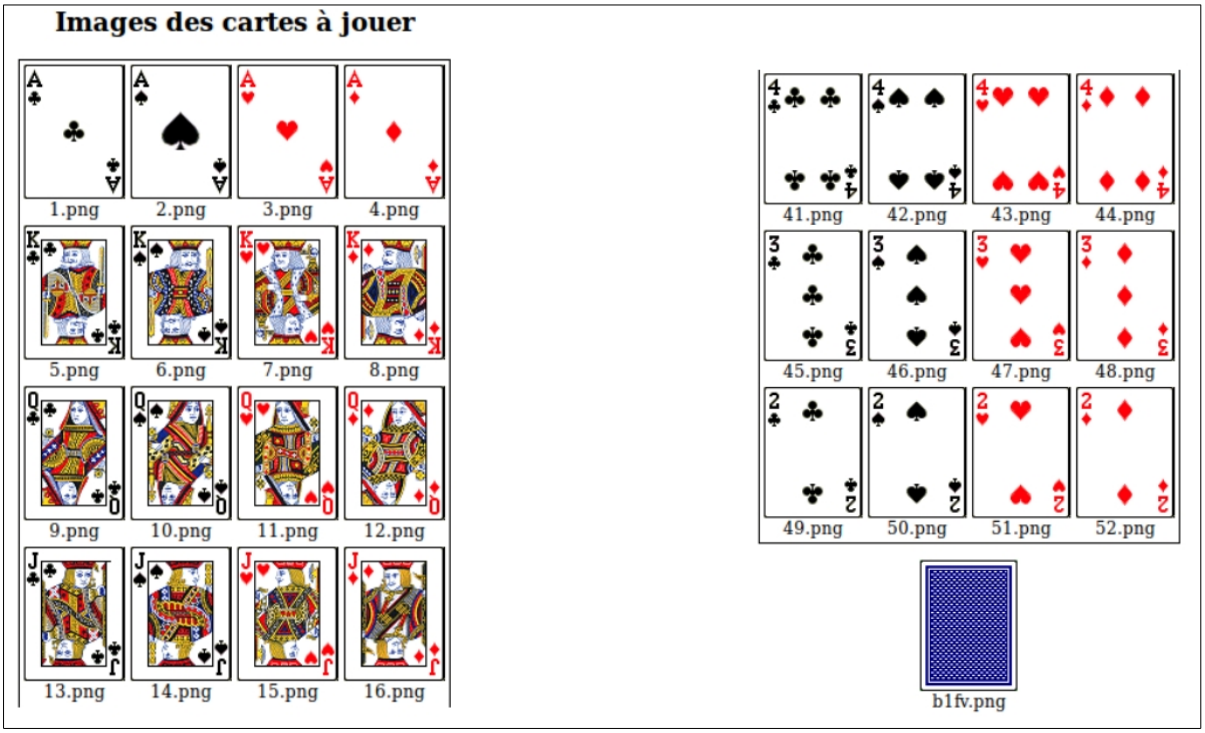
Utiliser :
https://www.w3schools.com/jsref/met_table_insertrow.asp
https://www.w3schools.com/jsref/met_tablerow_insertcell.asp

Exercice 6 : (Boucles, fonctions)

Réaliser une page web affichant l'ensemble des cartes du jeu selon le choix (32 ou 52) de l'internaute. Pour cela, donner un titre "Cartes de jeux" à votre page web. En tête de page, afficher "Images des cartes à jouer". Créer un tableau HTML centré avec bordure sur 4 colonnes. Le nom du fichier de chacune des cartes sera affiché en dessous de l'image carte. En dernière ligne, afficher la carte retournée (fichier *b1fv.png*). Les images de cartes se trouvent sous :

/home/public/2A_INFO/WEB_FRONT_END/TP2_JS_DOM/6_JEU_DE_CARTES/jeuxDeCartes.zip

Enfin, en pied de page, afficher « *Source et information sur le jeu de cartes:* https://fr.wikipedia.org/wiki/Jeu_de_cartes_français ».



Utiliser : <*select*> pour créer une liste de choix ayant pour options les valeurs 32 ou 52. Sur changement de valeur par l'internaute, une fonction JavaScript *createjeucartes()* est appelée.

Utiliser : https://www.w3schools.com/jsref/met_document_queryselector.asp

Version 1 : avec innerHTML (simple mais exécution plus lente)

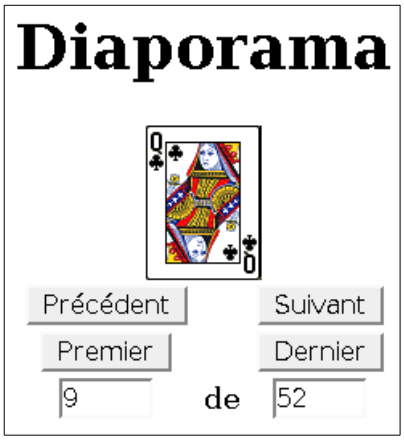
Version 2 : avec DOM (complexe mais exécution rapide)

Obtenir le même résultat avec DOM.

Utiliser :
https://www.w3schools.com/jsref/met_table_insertrow.asp
https://www.w3schools.com/jsref/met_tablerow_insertcell.asp

Exercice 7 : (Tests, boucles, fonctions, document, propriétés, événements)

On désire réaliser un diaporama permettant d'afficher une à une les cartes d'un jeu de 52 cartes. Ainsi, l'utilisateur pourra visualiser la première carte, la dernière carte, la précédente, la suivante. Le n° de carte (ici 9) en cours de visualisation sera affiché, ainsi que le nombre total de cartes. Les cartes se trouvent au même endroit que celui indiqué dans l'exercice n°6.



1. Réalisation de la partie HTML

La page web commencera par afficher le titre *Diaporama* puis l'image de la première carte (1.png). Puis sera affiché un formulaire possédant des zones *input* (rem : pour avoir des boutons bien alignés, utilisez un tableau) :

- 4 *button* pour *Précédent*, *Suivant*, *Premier*, *Dernier*. Pour chacun de ces boutons, on associera une fonction à la propriété *onClick*. Exemple pour le bouton *Précédent* : *<button id="previous">Précédent</button>*
- 2 *input* de type *text* pour afficher le n° de carte courant et le nombre total de cartes.

2. Réalisation de la partie JavaScript

Les fonctions JavaScript à définir :

- etat(nb)
  Affiche ou cache les boutons selon la valeur de nb (utiliser la propriété *disabled)*. Si nb=1, désactive le bouton *Précédent (disabled=true)*, sinon active le bouton *Précédent (disabled=false)* ; si n=52, désactive le bouton *Suivant* ; sinon sinon active le bouton *Suivant*. De même, affiche le n° de carte courant dans la zone de texte correspondante du formulaire.

- init()
  Appelée sur chargement du document (événement *load*). Initialise à 52 la variable globale *nombre* et remplit la zone de texte correspondante dans le formulaire. Appelle la fonction *etat* avec 1.

- premier()
  Appelée sur click du bouton *Premier*. Affiche la première carte. Appelle la fonction *etat* avec 1.

- dernier()
  Appelée sur click du bouton *Dernier*. Affiche la dernière carte. Appelle la fonction *etat* avec *nombre*.

- suivant()
  Appelée sur click du bouton *Suivant*. Affiche la carte suivante (c-à-d dont le n° est égal au n° de carte courant +1), si elle existe. Appelle la fonction *etat* avec ce nouveau n°.

- precedent()
  Appelée sur click du bouton *Précédent*. Affiche la carte précédente (dont le n° = n° de carte courant-1), si elle existe. Appelle la fonction *etat* avec ce nouveau n°.

Utiliser :

element.addEventListener("click", myFunction);

https://www.w3schools.com/js/js_htmldom_eventlistener.asp

Exercice 8 : (Objets, propriétés, array, boucles, fonctions)

Le but est de coder une page présentant des livres sur JavaScript en utilisant une bibliothèque représentée par un tableau (*array*) d'objets *Livre*. Chaque objet *Livre* est caractérisé par une image, un titre, un auteur, un prix et un éditeur. La liste des objets *Livre* est contenue dans le fichier *bibliotheque.js* où la fonction JavaScript *ajouter(…)* permettra de les insérer dans le tableau (*array*).
Les images se trouvent dans : */home/public/2A_INFO/WEB_FRONT_END/TP2_JS_DOM/8_OBJETS/livres.zip*.
Le fichier *bibliotheque.js* se trouve dans le répertoire ci-dessus.
Ci-contre, un aperçu de votre futur site :

Ecrire un fichier *livres.html* qui affichera les données contenues dans *bibliotheque.js*. L'affichage du tableau se fera en JavaScript dans un fichier *livres.js*

1. Partie HTML

Celle-ci contiendra une feuille de style CSS reliée, puis le titre « Livres sur JavaScript » et une déclaration d'une table vide (*<table></table>*) centrée. Puis, ajouter le chargement de *bibliotheque.js* et de *livres.js* en utilisant la balise *<script src="..."></script>*.

2. Partie JavaScript

Créer le fichier *livres.js* et déclarer un tableau (*array*) *ListeLivre* (vide). Déclarer le modèle objet *Livre*. Définir la méthode *afficherLivre()*, ainsi que la fonction *ajouter(titre, auteur, ...)* qui sera appelée à partir du fichier *bibliotheque.js*.

- **Tableau *ListeLivre***

Déclarer un tableau nommé *ListeLivre* qui contiendra l'ensemble des livres. Nombre inconnu.

- **Objet *Livre***

Écrire un fonction *Livre(titre, auteur, editeur, prix, image)* permettant de construire un objet *Livre* de titre 'titre', dont l'auteur est 'auteur', l'éditeur 'editeur', le prix 'prix' et l'image qui l'illustre 'image'.
De plus, l'objet *Livre* possédera une méthode *toString* qui sera initialisée par la méthode *afficherLivre()*.

- **Méthode *afficherLivre***

Écrire une méthode *afficherLivre()* qui permet de générer du code HTML correspondant à une ligne du tableau HTML (pour le format de la ligne, voir l'aperçu de la page web ci-contre).

- **Fonction *ajouter***

Écrire une fonction *ajouter()* permettant d'ajouter facilement un nouvel objet *Livre* au tableau *ListeLivre*. NB : On ne connaît pas le nombre de livres.
Pour afficher l'ensemble des livres, utiliser le fichier *bibliotheque.js*. Celui-ci contient plusieurs appels de la fonction *ajouter()* pour ajouter des livres au tableau (*array*) *ListeLivre*.

Une fois la réalisation de la page web terminée, il vous suffira simplement d'ajouter des livres parmi vos préférés dans le fichier *bibliotheque.js*. L'affichage des données se fera de façon indépendante et sans mise à jour des fichiers *livres.html* et *livres.js*.

Utiliser : window.addEventListener('load', fonction), querySelector(), this, toString(), innerHTML.

Exercice 9 : (Validation de formulaire)

Dans l'Exercice 2 du TP1 (HTML/CSS), vous avez construit un formulaire pour saisir de données envoyées par courrier électronique. Ici, avec du JavaScript, vous vérifierez que les valeurs des champs saisies ou sélectionnées sont valides. Si le test est positif, l'envoi se fera avec "Envoyer", sinon une boîte de dialogue (*alert*) demandera à l'utilisateur de saisir une nouvelle fois les champs erronés.
La combinaison des 2 types de validation sera utilisée :

- validation intégrée au formulaire, utilisant des attributs de validation sur éléments de formulaire HTML5. Ne nécessite pas de JavaScript, possède de meilleures performances, mais non personnalisable. Non supporté pour certains navigateurs anciens.
- validation du formulaire avec JavaScript, utilisant des propriétés de l'API de validation des contraintes, entièrement personnalisable. Supporté par tous les navigateurs.
  Voir aussi les navigateurs compatibles pour les validations : https://caniuse.com/#search=validation

1. **Reprise du formulaire**
   Reprendre le fichier HTML correspondant au formulaire de l'Exercice 2 du TP1.
   - Le code HTML ne doit pas contenir de styles (comme : attributs de styles à l'intérieur des balises, balises <*b*><*i*>, ...) mais il doit être lié à une feuille de présentation (CSS). Lire document : ***bonnes_pratiques_HTML.pdf*** situé dans le dossier TP1.
   - Le code HTML ne devra pas contenir du code JavaScript (comme : *onclick="fonction(…)"*) mais il devra inclure un fichier externe JavaScript juste avant </body>. Lire document : ***bonnes_pratiques_JS.pdf*** situé dans dossier TP2.

2. **Fichier HTML**
   Utiliser :
   attributs : *name*, *required*, *pattern*, *type* (*text*, *radio*, *number*, *mail*, *checkbox*, *button*, *submit*, *reset*), *min*, *max*,*size, action, …*
   balises : *form*, *input*, *label*, *div*, …
   S'inspirer sur : validation intégrée au formulaire utilisant des attributs de validation sur éléments de formulaire HTML5.

3. **Fichier CSS**
   Utiliser vos styles préférés. Ajouter les pseudo-classes *:valid* et *:invalid* permettant de cibler les éléments valides (bordure noire) ou invalides (bordure rouge) :

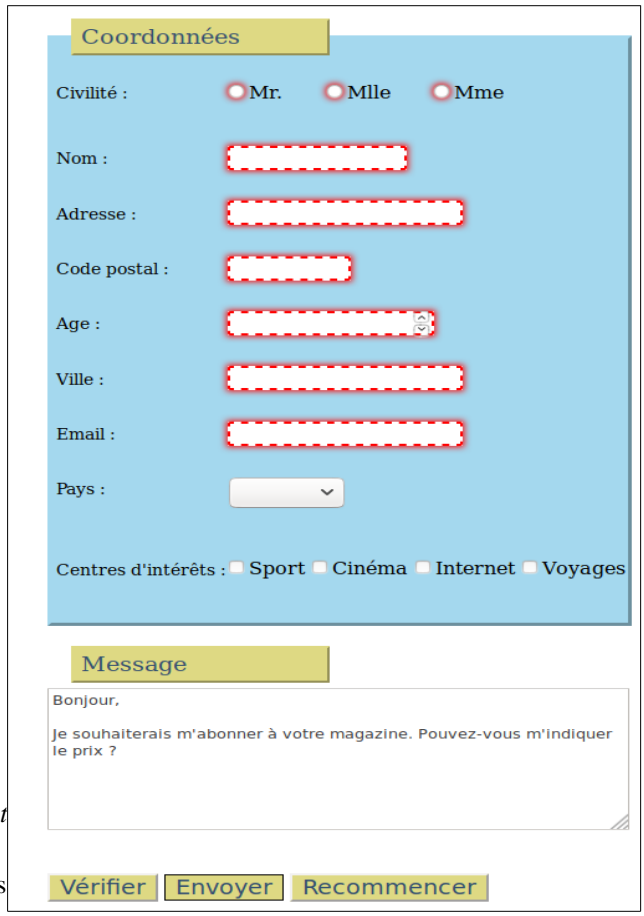
```
input:invalid {
  border: 2px dashed red;
  box-shadow: 0 0 5px 1px red;
}
input:valid {
  border: 1px solid black;
}
```

4. **Fichier JavaScript**
   Sur chargement du document, lancer une fonction *init()* permettant d'associer une variable au formulaire ainsi qu'une variable pour chaque élément du formulaire.
   Afin de personnaliser les messages d'erreurs concernant le *Code Postal*, *l'Age* et *l'Email,* définir pour chacun un gestionnaire d'évènements qui appel une fonction anonyme affichant un message personnalisé sur événement *mouseleave* :
   Code Postal → message : *Votre Code postal n'est pas un nombre composé de 5 chiffres !* (si input invalide)
   Age → message : *Votre âge doit être entre 12 et 90 ans!* (si input invalide)
   Email → message : *Adresse E-mail invalide! @ absent !* (si input invalide)
   Utiliser :
   propriétés : *validity, patternMismatch, rangeOverflow, rangeOverflow, typeMismatch, style, border, ...*
   méthodes : *setCustomValidity*

   Pour chaque élément de type *button*, définir un gestionnaire d'évènements qui appel une fonction sur événement *click* :
   Vérifier → appel de *formtest()*
   Envoyer → appel de *formtest() && securite('expédier maintenant')*
   Recommencer → appel de *securite('vider')*

   S'inspirer sur : validation du formulaire avec JavaScript utilisant des propriétés de l'API de validation des contraintes.
   Utiliser aussi : https://dmouronval.developpez.com/tutoriels/javascript/api-contrainte-validation/

Description des fonctions à définir :



1. La fonction *securite(action)*
   affiche une boite de confirmation "Êtes-vous sûr de vouloir *action* le document ?" avec pour *action* la valeur '*vider*' ou '*expédier maintenant*'. Retourne *true* ou *false* selon la réponse de l'utilisateur.

2. La fonction *formtest()*
   vérifie que les éléments du formulaire sont bien saisis et valides. Les vérifications à faire sont :
   - Vérifier que la civilité est choisie
   - Vérifier que les input texte sont saisies
   - Vérifier que le code postal est une suite de 5 chiffres
   - Vérifier que l'âge est un nombre entre 12 et 90 ans
   - Vérifier que l'email comporte un @ et un point
   - Vérifier que le pays a été sélectionné
   - Vérifier qu'au moins un centre d'intérêts a été coché

   Lors de chacune des vérifications, si le test n'est pas bon :
   - afficher une *alert* avec un message adéquat.
     Exemple : *alert("Adresse E-mail invalide! @ absent !")*;
   - Repositionner le curseur sur la zone concernée
   - retourner *false*.

   Au fur et à mesure des vérifications, construire une chaîne *resultat* pour mémoriser les valeurs choisies ou saisies.
   Enfin, afficher dans une boîte *alert* le récapitulatif des réponses saisies, en utilisant la variable *resultat* :
   Récapitulatif :
   ****
   Nom : Brun
   Prenom : Myriam
   Adresse : .....
   ...
   Centre d'intérêts : Cinéma Voyages

Indications :

Vérifier que la *civilité* est choisie :
utiliser les propriétés : *length* (nombre de boutons radio), *value*, *checked*

Vérifier que les input texte sont saisies :
utiliser les propriétés : *length* (nombre de *input*), *elements*, *type*, *value*, *name*
utiliser la méthode : *focus()*

Vérifier que le code postal est une suite de 5 chiffres :
utiliser les méthodes : *focus(), test(),* selon le format */^[0-9]{5}$/*

Vérifier que l'âge est un nombre compris entre 12 et 90 ans :
utiliser la méthode : *focus()*
utiliser la fonction : *isNaN()*

Vérifier que l'email comporte un @ et un point :
utiliser les propriétés : *length* (longueur du email)
utiliser les méthodes : *indexOf(), focus()*

Vérifier que le pays a été sélectionné :
utiliser les propriétés : *selectedIndex, text*
utiliser la méthode : *focus()*

Vérifier qu'au moins un centre d'intérêts a été coché :
utiliser les propriétés : *length* (nombre de cases à cocher), *checked, value*