TP JavaScript - DOM et objets

Exercice 1 - L'archive **annexes.tgz** contient 3 fichiers pour cet exercice : **tablist.html**, **tablist.css** et **tablist.js**. **tablist.html** affiche un formulaire pour pouvoir générer à volonté un tableau trié d'entiers et la liste des sommes obtenues sur chaque ligne, comme illustré ci-dessous. Le formulaire comprend les champs suivants :

- un champ texte pour saisir le nombre de colonnes souhaitées pour le tableau ;
- un champ texte pour saisir le nombre de lignes souhaitées ;
- deux boutons radio pour choisir la couleur de fond du tableau : orange ou violet ;
- un menu déroulant pour choisir l'alignement du texte des cellules du tableau : centré, à gauche, ou à droite ;
- un bouton pour générer ou actualiser le tableau et la liste ;
- un bouton pour effacer le tableau et la liste s'ils sont affichés.

FIGURE 1 – Vue au chargement

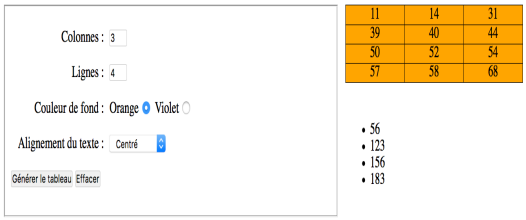


FIGURE 2 – Génération

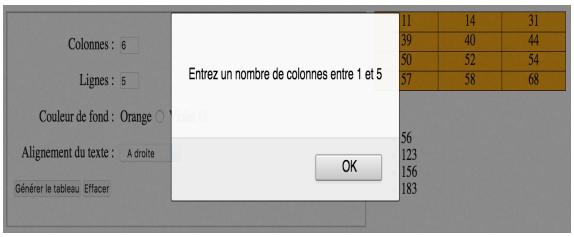


FIGURE 3 – Saisie incorrecte

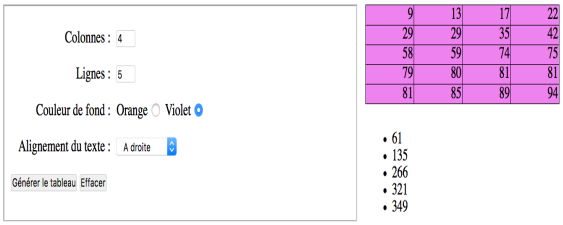
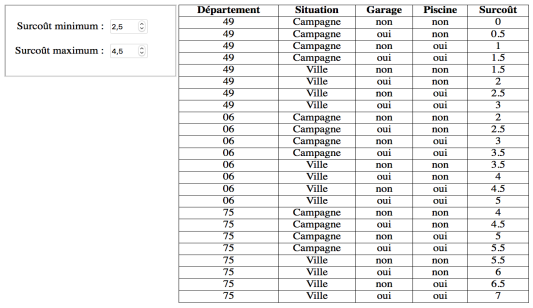


FIGURE 4 – Actualisation

Complétez le fichier **tablist.js**, sans modifier les fichiers HTML et CSS, afin d'obtenir le comportement suivant :

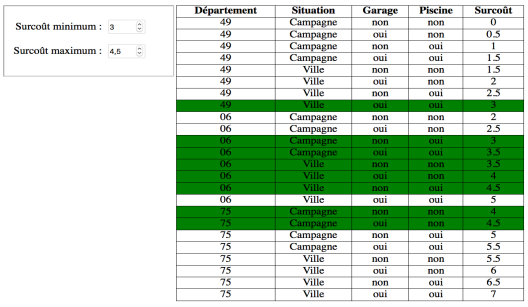
- Un clic sur le bouton `Générer le tableau` :
  1. Affiche une alerte et ne fait rien d'autre si le nombre de colonnes ou de lignes saisi est un entier en dehors de la plage $\{1, \ldots, 5\}$ (voir Figure 3).
  2. Sinon, remplace le contenu de l'élément d'identifiant `myTab` par un tableau HTML dont la taille et le style satisfont aux valeurs des champs du formulaire, et dont les éléments sont des entiers tirés aléatoirement entre 0 et 100 (potentiellement égaux) qui apparaissent en ordre croissant de gauche à droite et de haut en bas ; et remplace le contenu de l'élément d'identifiant `myList` par une liste HTML non ordonnée dont le i-ème élément est la somme des nombres de la i-ème ligne du tableau.
- Un clic sur le bouton `Effacer` :
  1. Ne fait rien si aucun tableau n'est affiché.
  2. Sinon, efface le tableau et la liste.
  3. Dans tous les cas, ne réinitialise aucun des champs du formulaire.

Exercice 2 - L'archive **annexes.tgz** contient 3 fichiers pour cet exercice : **maison.html**, **maison.css** et **maison.js**. **maison.html** contient un formulaire pour identifier des maisons dont le (sur)coût est situé dans une plage fixée par l'utilisateur. La colonne `Surcoût` du tableau ne contient initialement aucune valeur et l'objectif est de compléter le fichier **maison.js**, sans modifier les fichiers HTML et CSS, afin d'obtenir le comportement illustré ci-dessous.



Département	Situation	Garage	Piscine	Surcoût
49	Campagne	non	non	0
49	Campagne	oui	non	0.5
49	Campagne	non	oui	1
49	Campagne	oui	oui	1.5
49	Ville	non	non	1.5
49	Ville	oui	non	2
49	Ville	non	oui	2.5
49	Ville	oui	oui	3
06	Campagne	non	non	2
06	Campagne	oui	non	2.5
06	Campagne	non	oui	3
06	Campagne	oui	oui	3.5
06	Ville	non	non	3.5
06	Ville	oui	non	4
06	Ville	non	oui	4.5
06	Ville	oui	oui	5
75	Campagne	non	non	4
75	Campagne	oui	non	4.5
75	Campagne	non	oui	5
75	Campagne	oui	oui	5.5
75	Ville	non	non	5.5
75	Ville	oui	non	6
75	Ville	non	oui	6.5
75	Ville	oui	oui	7

FIGURE 5 – Vue au chargement



Département	Situation	Garage	Piscine	Surcoût
49	Campagne	non	non	0
49	Campagne	oui	non	0.5
49	Campagne	non	oui	1
49	Campagne	oui	oui	1.5
49	Ville	non	non	1.5
49	Ville	oui	non	2
49	Ville	non	oui	2.5
49	Ville	oui	oui	3
06	Campagne	non	non	2
06	Campagne	oui	non	2.5
06	Campagne	non	oui	3
06	Campagne	oui	oui	3.5
06	Ville	non	non	3.5
06	Ville	oui	non	4
06	Ville	non	oui	4.5
06	Ville	oui	oui	5
75	Campagne	non	non	4
75	Campagne	oui	non	4.5
75	Campagne	non	oui	5
75	Campagne	oui	oui	5.5
75	Ville	non	non	5.5
75	Ville	oui	non	6
75	Ville	non	oui	6.5
75	Ville	oui	oui	7

FIGURE 6 – Maisons de surcoût entre 3 et 4.5

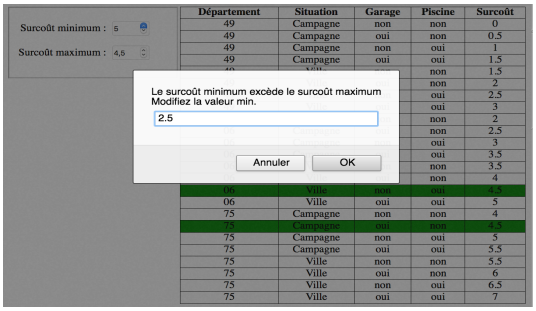
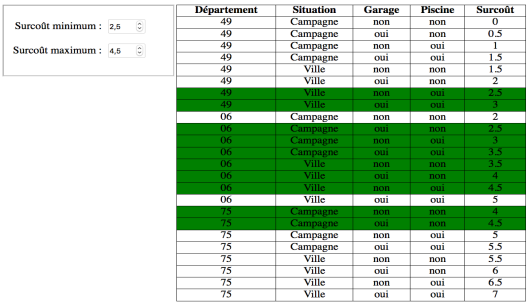


FIGURE 7 – Correction de saisie



Département	Situation	Garage	Piscine	Surcoût
49	Campagne	non	non	0
49	Campagne	oui	non	0.5
49	Campagne	non	oui	1
49	Campagne	oui	oui	1.5
49	Ville	non	non	1.5
49	Ville	oui	non	2
49	Ville	non	oui	2.5
49	Ville	oui	oui	3
06	Campagne	non	non	2
06	Campagne	oui	non	2.5
06	Campagne	non	oui	3
06	Campagne	oui	oui	3.5
06	Ville	non	non	3.5
06	Ville	oui	non	4
06	Ville	non	oui	4.5
06	Ville	oui	oui	5
75	Campagne	non	non	4
75	Campagne	oui	non	4.5
75	Campagne	non	oui	5
75	Campagne	oui	oui	5.5
75	Ville	non	non	5.5
75	Ville	oui	non	6
75	Ville	non	oui	6.5
75	Ville	oui	oui	7

FIGURE 8 – Actualisation

Chaque maison se caractérise par :

- son numéro de département pris dans l'ensemble $\{49, 6, 75\}$,
- sa situation géographique : `campagne` ou `ville`,
- ses extensions : avec ou sans `garage`, et avec ou sans `piscine`,
- son "surcoût" qui est exprimé comme un pourcentage par rapport au coût d'une maison sans extensions et située en campagne dans le département 49. Le surcoût se calcule selon la formule

$$surcout = D + S + G + P$$

où

— $D = 0$ pour le département "49", $D = 2$ pour le "6", et $D = 4$ pour le "75",

— $S = 0$ pour la campagne et $S = 1.5$ pour la ville,

— $G = 0$ si la maison est sans garage et $G = 0.5$ sinon, et

— $P = 0$ si la maison est sans piscine et $P = 1$ sinon.

La figure 5 donne les valeurs de surcoût pour les 24 types de maisons possibles.

Le fichier **maison.js** déclare le tableau des numéros de départements, le tableau des situations géographiques, et le tableau des surcoûts associés aux extensions. Il fournit aussi un constructeur `Localisation` qui modélise une paire département-situation.

Question 1. Ajoutez une méthode `toText()` au constructeur `Localisation` pour que le code

```
var l6c = new Localisation(6, 'campagne');
console.log(l6c.toText());
```

affiche dans la console

```
Département : 6
Situation : campagne
```

Question 2. Modifiez le prototype du constructeur `Localisation` pour que le code

```
var lDefaut = new Localisation();
console.log(lDefaut.toText());
```

affiche dans la console

```
Département : 49
Situation : campagne
```

Question 3. Définir un constructeur `Maison` pour que le code

```
var m75vgp = new Maison(new Localisation(75, 'ville'), [ 'garage', 'piscine' ]);
console.log(m75vgp.toText());
```

affiche dans la console

```
Département : 75
Situation : ville
Extensions : garage, piscine
Surcoût : 7
```

Le constructeur `Maison` devra :

(1) vérifier que son premier argument est une instance de `Localisation` et dans le cas contraire afficher un avertissement à la console,

(2) vérifier que son deuxième argument est un tableau représentant un sous-ensemble, potentiellement vide, de `{'garage','piscine'}` et dans le cas contraire afficher un avertissement à la console,

(3) posséder une méthode `surcout()` qui calcule et renvoie le surcout de la maison selon la formule précédente,

(4) posséder une méthode `toText()` de formattage des données comme illustré ci-dessus.

Question 4. Modifiez le prototype du constructeur `Maison` pour que le code

```
var mDefaut = new Maison();
console.log(mDefaut.toText());
```

affiche dans la console

```
Département : 49
Situation : campagne
Extensions :
Surcoût : 0
```

Question 5. Complétez **maison.js** afin d'insérer au chargement de **maison.html** le surcoût de chaque maison dans la cellule correspondante du tableau HTML (voir Figure 5). Vous pouvez utiliser le tableau `maisons` d'instances de `Maison` qui est défini dans le fichier **maison.js** et qui représente les 24 types de maisons possibles.

Question 6. Le formulaire de **maison.html** contient deux champs de type `number` permettant de fixer un surcoût minimum et un surcoût maximum. Complétez **maison.js** de sorte qu'à chaque changement de valeur de l'un des

champs, les lignes du tableau correspondant aux maisons de surcoût compris dans la plage choisie apparaissent sur fond vert, ou alors sur fond blanc (voir Figure 6).

Question 7. Vérifiez qu'à chaque changement de valeur de l'un des deux champs, le surcoût minimum choisi n'excède pas le surcoût maximum choisi. Dans le cas contraire, afficher une boîte de dialogue permettant à l'utilisateur de modifier son champ jusqu'à résolution du problème (voir Figures 7 et 8).