JAVASCRIPT

Étapes d'une fonction de génération

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision

Sommaire

A Introduction

Cette activité réalisée en classe permet de **montrer comment un code évolue**.

Elle ne permet pas d'apprendre à programmer, directement. Il y a donc beaucoup de copier/coller à faire.

A.1 Prérequis

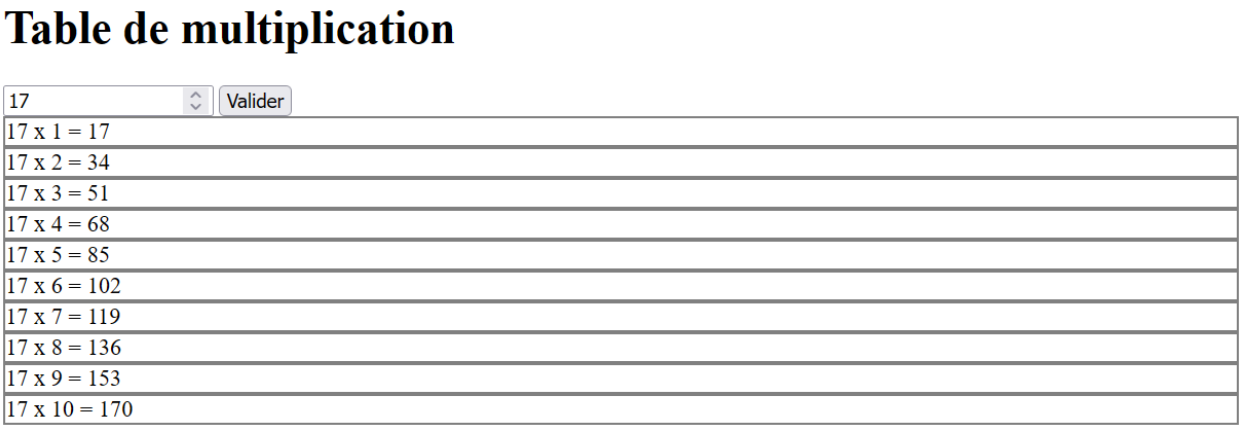
Pour réaliser cette activité il faut :

- être capable d'utiliser les boucles (répétitions)
- savoir manipuler les variables
- connaître les instructions pour lire et écrire dans une page HTML
  - document.getElementById()
  - .innerHTML
- Connaître le HTML (balises et notamment l'utilisation des ID)

A.2 Objectif

L'objectif de cette activité est de créer un code de manière modulaire :

- Initialement, on affichera la table de 5
- En intermédiaire, on affichera la table souhaitée (nombre modifiable)
- Finalement, on utilisera des champs de saisie pour choisir la table et sa longueur



B Phase initiale

Il faut d'abord déterminer l'algorithme du code, c'est-à-dire, ce qu'il doit faire.

B.1.1 Algorithme

La première phrase qui peut venir à l'esprit est trompeuse, car on a tendance à généraliser.

"**Le programme doit afficher la table de multiplication par 5**" n'est pas un algorithme, car la phrase n'est pas assez précise pour un ordinateur. Décomposons :

- **doit afficher** : utilise-t-on un console.log() ou bien affiche-t-on le résultat dans la page HTML ?
    - On sait que l'utilisateur ne peut pas voir la console. Il faut donc utiliser les instructions d'affichage en HTML de JavaScript, soit objet.innerHTML (et il faut donc utiliser document.getElementById() pour trouver l'objet dans la page et le donner à JavaScript)
- **La table de multiplication** : ça n'existe pas comme ça (il n'y a pas une instruction pour créer une table de multiplication)
    - Une table s'écrit
      5 x 1 = 5, puis
      5 x 2 = 10, puis
      5 x3 = 15 et "*ainsi de suite*" ► le terme suggère une répétition, donc une boucle
- **par 5** : on applique le choix 5 sur la table, ce qui signifie qu'on pourrait faire évoluer ce choix par la suite ? Choisir un autre nombre ?
    - On peut envisager plusieurs tables et donc, écrire
      nb x 1 = quelquechose
      nb x 2 = quelquechose
      … ► utilisation de variables (let nb = 5 par exemple)

Par la suite, on va tenter d'écrire l'algorithme :

```
nb ← 5
pour i allant de 1 à 10
afficher nb + " x " + i + " = " + nb*i
fin
```

Il manque la récupération de l'endroit où afficher avec JavaScript, l'algorithme ressemble alors à :

```
JSMult ← objet balise id "JSMult"
nb ← 5
pour i allant de 1 à 10
afficher nb + " x " + i + " = " + nb*i
fin
```

B.1.2 Code

Avec l'algorithme et notre connaissance des instructions JavaScript, on peut maintenant déduire le code.

JSMult.js

```
let JSMult = document.getElementById("JSMult")
let nb = 5
for (let i=1 ; i<11 ; i++) {
    JSMult.innerHTML = nb + " x " + i + " = " + nb*i
}
```

Et le code HTML est plutôt standard :

JSMult.html

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Table Multiplication</title>
</head>
<body>
    <h1>Table de multiplication</h1>
    <div id="JSMult">Hello</div>
    <script src="JSMult.js"></script>
</body>
</html>
```

B.1.3 Bogue une seule ligne

Lors de l'exécution de ce code, il n'y a qu'une ligne qui s'affiche, pour une raison simple : nous expliquons à JavaScript d'écrire à un endroit précis, le résultat... si nous avions une caméra très haute fréquence, nous aurions vu qu'il écrit 5 x 1 = 5 et qu'il écrit de suite après au même endroit 5 x 2 = 10 et ainsi de suite...

B.1.4 Correction du bogue

En fait, nous désirons que JavaScript n'écrase pas le résultat précédent à chaque écriture : il faut,

1. soit reprendre ce qui est précédemment écrit et y ajouter la nouvelle ligne
2. soit écrire toutes les lignes dans une zone de stockage et écrire cette zone dans la page HTML

B.1.4.a Solution 1

On lit le texte contenu à l'endroit où on veut écrire et on y ajoute notre texte

JSMult.js

```
let JSMult = document.getElementById("JSMult")
let nb = 5
for (let i=1 ; i<11 ; i++) {
    JSMult.innerHTML = JSMult.innerHTML + nb + " x " + i + " = " + nb*i
}
```

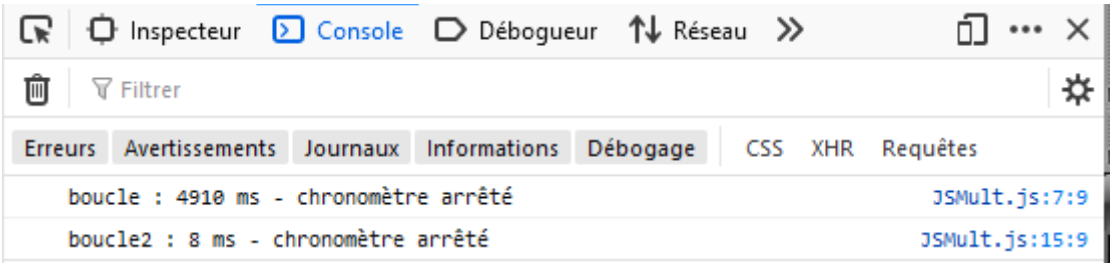
B.1.4.b Solution 2

On crée une variable qui stocke le texte à chaque itération (passage dans la boucle) et à la sortie de la boucle, on écrit cette variable dans la localisation de la page HTML

JSMult.js

```
let JSMult = document.getElementById("JSMult")
let nb = 5
let txtResult = ""
for (let i=1 ; i<11 ; i++) {
    txtResult = txtResult + nb + " x " + i + " = " + nb*i
}
JSMult.innerHTML = txtResult
```

Cette solution – bien qu'un peu plus longue – est **extrêmement** plus rapide, parce que l'écriture dans une page HTML prend du temps. Voici les résultats pour 10000 itérations (comptage de 1 à 10000).



🚩 pour "chronométrer" une instruction ou une fonction, on utilise **console.time('nom_du_chrono')** avant l'élément à chronométrer et **console.timeEnd('nom_du_chrono')** juste après.

B.1.5 Bogue écriture à la suite

Un dernier problème est à régler : l'écriture n'est pas d'une ligne par opération. C'est normal car HTML ne connaît pas le retour à la ligne. Soit nous insérons le code "
" à chaque ligne, soit nous utilisons une balise " " et " " pour délimiter une ligne.

B.1.6 Correction écriture à la suite

Nous allons en profiter pour ajouter un peu de style :

JSMult.js

```
let JSMult = document.getElementById("JSMult")
let nb = 5
let txtResult = ""
for (let i=1 ; i<11 ; i++) {
    txtResult = txtResult + "<div class='cadre'>" + nb + " x " + i + " = " + nb*i + "</div>"
}
JSMult.innerHTML = txtResult
```

JSMult.html

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Table Multiplication</title>
    <style>
        .cadre { border-style: solid; border-color: #BBBBBB;}
    </style>
</head>
<body>
    <h1>Table de multiplication</h1>
    <div id="JSMult">Hello</div>
    <script src="JSMult.js"></script>
</body>
</html>
```

Table de multiplication

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

B.2 Phase intermédiaire

Dans cette phase, nous allons ajouter un champ de saisie pour permettre a l'utilisateur de choisir sa table de multiplication. Cependant, pour pouvoir réaliser cela, nous aurons besoin :

- de champ de saisie HTML (<input type="number">)
- d'ajouter un événement (.addEventListener("input", fonction))
- de transformer notre code pour avoir une fonction

B.2.1 Ajout du champ de saisie

Dans le code HTML, il faut ajouter un champ de saisie de type "number".

Inutile d'ajouter un bouton : comme il n'y a qu'un champ modifié, on peut utiliser une autre propriété que 'click', c'est 'input' (en HTML c'est 'oninput')

JSMult.html

```
<h1>Table de multiplication</h1>
<input type="number" id="nb" name="nb" value="5">
<hr>
<div id="JSMult">Hello</div>
```

B.2.2 Modification en fonction du code JavaScript

Globalement, il suffit de placer les éléments suivants autour du code JavaScript :

- ouverture ►function multiplication(nb) {
- fermeture ►}

Le code serait à peu près celui-là :

```
function tableMultiplication(nb) {
    let JSMult = document.getElementById("JSMult")
    let txtResult = ""
    for (let i=1 ; i<11 ; i++) {
        txtResult=txtResult+"<div class='cadre'>"+nb+" x "+i+" = "+nb*i+"</div>"
    }
    JSMult.innerHTML = txtResult
}
```

Une fonction étant la préparation d'un bloc de code, il faut l'appeler pour qu'il s'exécute. On ajoute donc l'instruction d'appel de la fonction après sa déclaration :

```
tableMultiplication(5)
```

Le code fonctionne comme avant, mais le champ de saisie n'est pas reconnu : il manque la création de l'événement.

Si vous rencontrez un problème, utilisez [F12] ou [CTRL]+[shift]+[i] pour afficher la console de débogage.

B.2.3 Ajout de l'événement

La méthode addElementById(type, fonction) permet d'appliquer un événement sur un objet. Ici, l'objet est le champ de saisie. Les paramètres de cette méthode sont :

- type ► click, dblClick, change, input...
- fonction ► le nom d'une fonction JavaScript – sans les parenthèses – pour indiquer son nom mais ne pas l'exécuter immédiatement.

Le tableau d'événements suivant permet de retrouver l'équivalence avec la méthode qui consiste à écrire l'événement dans le code HTML (*ce fonctionnement est désormais obsolète mais peut être rencontré dans des codes un peu anciens*) :

Par exemple (**ne pas copier dans le code**) :

```
<input type="text" id="saisie" oninput="fonction()">
```

HTML	JavaScript	Action
onfocus, onblur	focus, blur	Lorsqu'un élément récupère ou perd le focus (le curseur)
onclick, ondblclick	click, dblclick	Lorsqu'un élément est cliqué ou double-cliqué
onchange, oninput	change, input	Lorsqu'un élément est modifié (état = change, saisie = input)
onkeypress	keypress	Récupération du numéro de touche appuyée
onkeyup, onkeydown	keyup, keydown	Appui ou relâchement d'une touche du clavier
onmousedown onmouseenter onmouseleave onmousemove onmouseout onmouseover onmouseup onmousewheel	mousedown mouseenter mouseleave mousemove mouseout mouseover mouseup mousewheel	(à vérifier) Tous les événements liés à la souris

La modification sur le code est la suivante, à la fin du code JavaScript :

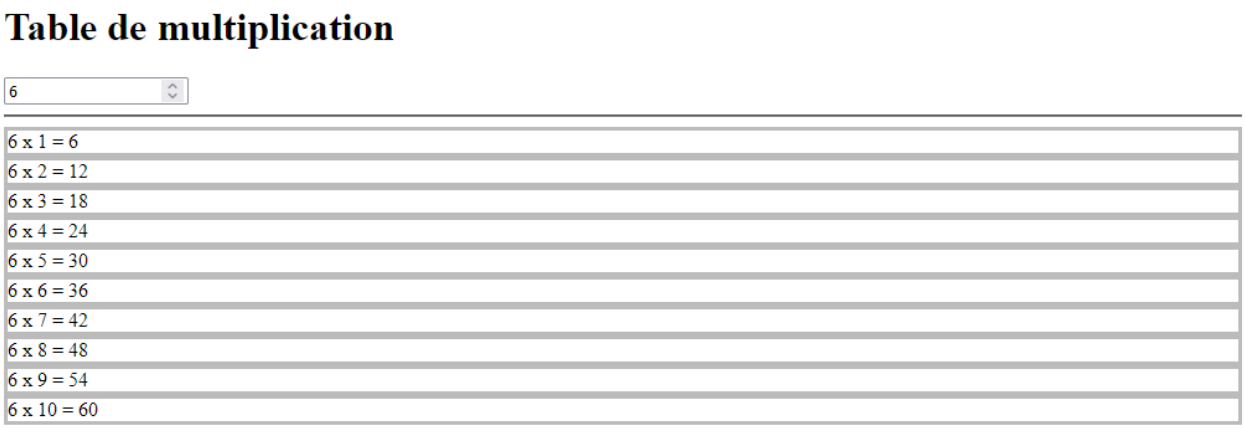
```
let saisieNb = document.getElementById("nb")
saisieNb.addEventListener("input", tableMultiplication)
```

et en début de fonction (à l'intérieur)

```
if (typeof(nb) == 'object') {
    nb = document.getElementById("nb").value
}
```

Cette partie permet d'appeler la fonction avec un nombre (fonctionnement normal), soit avec l'objet lors d'un événement.

Si tout se passe bien, vous obtiendrez le fonctionnement avec l'affichage suivant à chaque changement de valeur numérique :



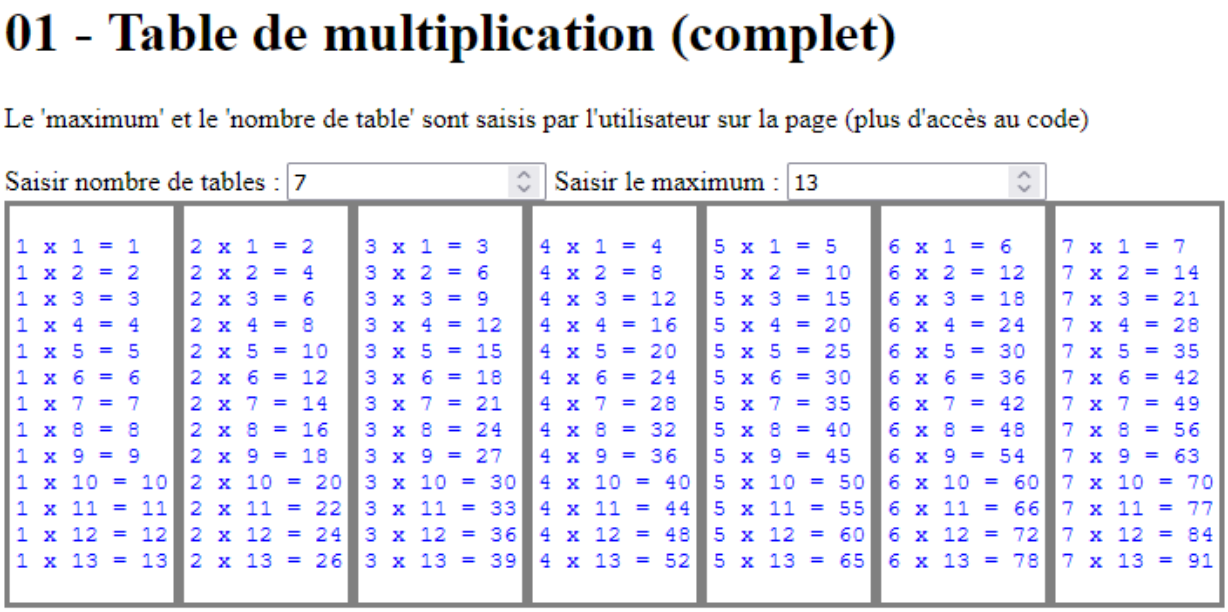
L'intérêt de cette méthode, c'est qu'on récupère la valeur de l'objet cliqué directement.

B.3 Phase finale

Si dans un cahier des charges, le client souhaite définir aussi le nombre de table à afficher, il faut casser une grande partie de notre code pour se rapprocher de la demande.

💡 Il est normal de modifier un code qui fonctionne, en cassant certaines parties et en reconstruisant des parties différentes.

Imaginons donc la demande du client :



01 - Table de multiplication (complet)

Le 'maximum' et le 'nombre de table' sont saisis par l'utilisateur sur la page (plus d'accès au code)

Saisir nombre de tables : 7 Saisir le maximum : 13

1 x 1 = 1 1 x 2 = 2 1 x 3 = 3 1 x 4 = 4 1 x 5 = 5 1 x 6 = 6 1 x 7 = 7 1 x 8 = 8 1 x 9 = 9 1 x 10 = 10 1 x 11 = 11 1 x 12 = 12 1 x 13 = 13	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12 2 x 7 = 14 2 x 8 = 16 2 x 9 = 18 2 x 10 = 20 2 x 11 = 22 2 x 12 = 24 2 x 13 = 26	3 x 1 = 3 3 x 2 = 6 3 x 3 = 9 3 x 4 = 12 3 x 5 = 15 3 x 6 = 18 3 x 7 = 21 3 x 8 = 24 3 x 9 = 27 3 x 10 = 30 3 x 11 = 33 3 x 12 = 36 3 x 13 = 39	4 x 1 = 4 4 x 2 = 8 4 x 3 = 12 4 x 4 = 16 4 x 5 = 20 4 x 6 = 24 4 x 7 = 28 4 x 8 = 32 4 x 9 = 36 4 x 10 = 40 4 x 11 = 44 4 x 12 = 48 4 x 13 = 52	5 x 1 = 5 5 x 2 = 10 5 x 3 = 15 5 x 4 = 20 5 x 5 = 25 5 x 6 = 30 5 x 7 = 35 5 x 8 = 40 5 x 9 = 45 5 x 10 = 50 5 x 11 = 55 5 x 12 = 60 5 x 13 = 65	6 x 1 = 6 6 x 2 = 12 6 x 3 = 18 6 x 4 = 24 6 x 5 = 30 6 x 6 = 36 6 x 7 = 42 6 x 8 = 48 6 x 9 = 54 6 x 10 = 60 6 x 11 = 66 6 x 12 = 72 6 x 13 = 78	7 x 1 = 7 7 x 2 = 14 7 x 3 = 21 7 x 4 = 28 7 x 5 = 35 7 x 6 = 42 7 x 7 = 49 7 x 8 = 56 7 x 9 = 63 7 x 10 = 70 7 x 11 = 77 7 x 12 = 84 7 x 13 = 91

B.3.1 Analyse des changements

Voici ce que l'on constate :

- il n'y a plus de cadre pour chaque multiplication mais seulement un cadre par table. On peut en déduire que cela touche le code CSS.
- Il y a deux champs : nombre de tables et nombre maximum. On peut déduire qu'il faut afficher les tables de 1 à N, ce qui implique une boucle. À l'intérieur d'une table il y a une boucle qui compte de 1 à MAX pour calculer le résultat. Nous avons déjà le code pour une table… il faut créer le code pour appeler ce code plusieurs fois.
- Dès qu'on modifie une valeur dans un champ de saisie, la page est modifiée, il y a donc deux événements (un par champ)

B.3.2 Propositions de changements

En algorithmie, il faut imaginer que notre fonction tableMultiplication sera appelée plusieurs fois par une autre fonction qui sera appelée par les événements.

```
Pour table de 1 à nb
    Pour nombre de 1 à maximum
        afficher multiplication
    fin Pour
fin Pour
```

On appelle ce genre de code, une boucle imbriquée : il y a en effet, une boucle à l'intérieur d'une autre.

B.3.3 Codage JavaScript

Le plus simple est de créer une fonction qui dessine une seule table, puis de modifier le code de la fonction qui est appelée par les événements :

JSMult.js

```
function calculTable(numTable, maximum) {
    let texte = "<div class='cadre'><pre>"
    // Dessiner une table de multiplication de 1 à Maximum
    for (let nombre = 1; nombre <= maximum; nombre = nombre + 1){
        texte=texte+numTable+ " x "+nombre+" = "+numTable*nombre+"\n"
    }
    texte = texte + "</pre></div>"
    return texte  // renvoie le résultat à celui qui appelle la fonction
}

function tableMultiplication(nb, maximum=10) {
    if (typeof(nb) == 'object') {
        nb = saisieNb.value
        maximum = saisieMx.value
    }
    let txtResult = ""
    let JSMult = document.getElementById("JSMult")
    for (let nbTable=1; nbTable<nb; nbTable++) {
        txtResult = txtResult + calculTable(nbTable, maximum)
    }
    JSMult.innerHTML = txtResult
}
```

Dans la première fonction, nous avons changé le fonctionnement en utilisant la balise <pre> </pre> pour indiquer à HTML qu'il ne doit pas interpréter le texte à l'intérieur de celles-ci. La balise n'est utilisée qu'une fois en début de texte et fermée en fin de texte.

🚩🚩🚩Important, le texte (le résultat du travail de la fonction) est renvoyé à l'endroit où la fonction a été appelée. L'instruction **return** sert à ça.

Dans la deuxième fonction, nous appelons la première fonction à l'intérieur de la boucle, le reste est presque similaire.

Il ne faut pas oublier d'ajouter l'écoute de l'événement 'input' sur chaque saisie, après les fonctions :

```
let saisieNb = document.getElementById("nb")
let saisieMx = document.getElementById("maximum")
saisieNb.addEventListener("input", tableMultiplication)
saisieMx.addEventListener("input", tableMultiplication)
```

B.3.4 Codage HTML

Il s'agit d'ajouter le champ de saisie maximum et de modifier le CSS comme ceci :

JSMult.html

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Table Multiplication</title>
    <style>
        .cadre {
            border-style: solid;
            border-color: #BBBBBB;
            display: inline-block;
            width: 120px;
        }
    </style>
</head>
<body>
    <h1>Table de multiplication</h1>
    <input type="number" id="nb" name="nb" value="5">
    <input type="number" id="maximum" name="maximum" value="10">
    <hr>
    <div id="JSMult">Hello</div>
    <script src="JSMult.js"></script>
</body>
</html>
```

La partie CSS est intéressante, car elle permet de juxtaposer les blocs DIV (normalement, ils ne peuvent pas être les uns à côté des autres), c'est l'instruction **display: inline-block**;

Voici le résultat :

Table de multiplication

21 10

1 x 1 = 1 1 x 2 = 2 1 x 3 = 3 1 x 4 = 4 1 x 5 = 5 1 x 6 = 6 1 x 7 = 7 1 x 8 = 8 1 x 9 = 9 1 x 10 = 10	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12 2 x 7 = 14 2 x 8 = 16 2 x 9 = 18 2 x 10 = 20	3 x 1 = 3 3 x 2 = 6 3 x 3 = 9 3 x 4 = 12 3 x 5 = 15 3 x 6 = 18 3 x 7 = 21 3 x 8 = 24 3 x 9 = 27 3 x 10 = 30	4 x 1 = 4 4 x 2 = 8 4 x 3 = 12 4 x 4 = 16 4 x 5 = 20 4 x 6 = 24 4 x 7 = 28 4 x 8 = 32 4 x 9 = 36 4 x 10 = 40	5 x 1 = 5 5 x 2 = 10 5 x 3 = 15 5 x 4 = 20 5 x 5 = 25 5 x 6 = 30 5 x 7 = 35 5 x 8 = 40 5 x 9 = 45 5 x 10 = 50	6 x 1 = 6 6 x 2 = 12 6 x 3 = 18 6 x 4 = 24 6 x 5 = 30 6 x 6 = 36 6 x 7 = 42 6 x 8 = 48 6 x 9 = 54 6 x 10 = 60	7 x 1 = 7 7 x 2 = 14 7 x 3 = 21 7 x 4 = 28 7 x 5 = 35 7 x 6 = 42 7 x 7 = 49 7 x 8 = 56 7 x 9 = 63 7 x 10 = 70	8 x 1 = 8 8 x 2 = 16 8 x 3 = 24 8 x 4 = 32 8 x 5 = 40 8 x 6 = 48 8 x 7 = 56 8 x 8 = 64 8 x 9 = 72 8 x 10 = 80	9 x 1 = 9 9 x 2 = 18 9 x 3 = 27 9 x 4 = 36 9 x 5 = 45 9 x 6 = 54 9 x 7 = 63 9 x 8 = 72 9 x 9 = 81 9 x 10 = 90	10 x 1 = 10 10 x 2 = 20 10 x 3 = 30 10 x 4 = 40 10 x 5 = 50 10 x 6 = 60 10 x 7 = 70 10 x 8 = 80 10 x 9 = 90 10 x 10 = 100
11 x 1 = 11 11 x 2 = 22 11 x 3 = 33 11 x 4 = 44 11 x 5 = 55 11 x 6 = 66 11 x 7 = 77 11 x 8 = 88 11 x 9 = 99 11 x 10 = 110	12 x 1 = 12 12 x 2 = 24 12 x 3 = 36 12 x 4 = 48 12 x 5 = 60 12 x 6 = 72 12 x 7 = 84 12 x 8 = 96 12 x 9 = 108 12 x 10 = 120	13 x 1 = 13 13 x 2 = 26 13 x 3 = 39 13 x 4 = 52 13 x 5 = 65 13 x 6 = 78 13 x 7 = 91 13 x 8 = 104 13 x 9 = 117 13 x 10 = 130	14 x 1 = 14 14 x 2 = 28 14 x 3 = 42 14 x 4 = 56 14 x 5 = 70 14 x 6 = 84 14 x 7 = 98 14 x 8 = 112 14 x 9 = 126 14 x 10 = 140	15 x 1 = 15 15 x 2 = 30 15 x 3 = 45 15 x 4 = 60 15 x 5 = 75 15 x 6 = 90 15 x 7 = 105 15 x 8 = 120 15 x 9 = 135 15 x 10 = 150	16 x 1 = 16 16 x 2 = 32 16 x 3 = 48 16 x 4 = 64 16 x 5 = 80 16 x 6 = 96 16 x 7 = 112 16 x 8 = 128 16 x 9 = 144 16 x 10 = 160	17 x 1 = 17 17 x 2 = 34 17 x 3 = 51 17 x 4 = 68 17 x 5 = 85 17 x 6 = 102 17 x 7 = 119 17 x 8 = 136 17 x 9 = 153 17 x 10 = 170	18 x 1 = 18 18 x 2 = 36 18 x 3 = 54 18 x 4 = 72 18 x 5 = 90 18 x 6 = 108 18 x 7 = 126 18 x 8 = 144 18 x 9 = 162 18 x 10 = 180	19 x 1 = 19 19 x 2 = 38 19 x 3 = 57 19 x 4 = 76 19 x 5 = 95 19 x 6 = 114 19 x 7 = 133 19 x 8 = 152 19 x 9 = 171 19 x 10 = 190	20 x 1 = 20 20 x 2 = 40 20 x 3 = 60 20 x 4 = 80 20 x 5 = 100 20 x 6 = 120 20 x 7 = 140 20 x 8 = 160 20 x 9 = 180 20 x 10 = 200

B.4 Sécurisation du code

Le code fonctionne, il faut cependant le commenter et le sécuriser.

B.4.1 Affichage pour valeurs négatives ou nulles

Par exemple, l'utilisateur peut mettre les champs à 0 et l'affichage ne signifie plus rien.

La première idée est de sécuriser le champ <input> avec les attributs min et max :

JSMult.html

```
<input type="number" id="nb" name="nb" value="5" min=1 max=100>
<input type="number" id="maximum" name="maximum" value="10" min=1 max=20>
```

Cela fonctionne lorsqu'on utilise la souris.

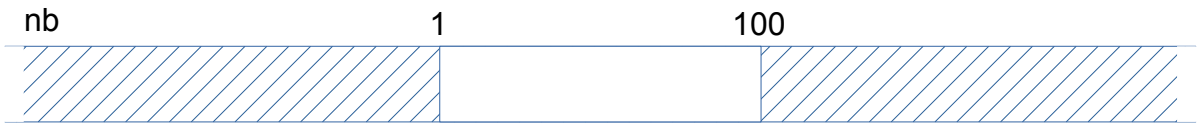
Table de multiplication

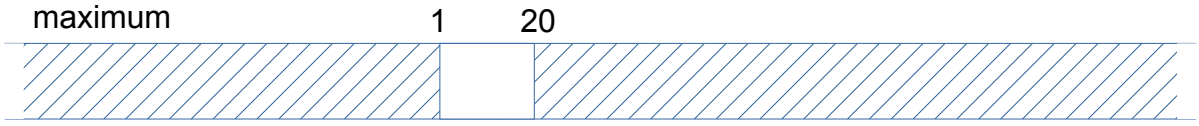
100 1

1 x 1 = 1	2 x 1 = 2	3 x 1 = 3	4 x 1 = 4	5 x 1 = 5	6 x 1 = 6	7 x 1 = 7	8 x 1 = 8	9 x 1 = 9	10 x 1 = 10
11 x 1 = 11	12 x 1 = 12	13 x 1 = 13	14 x 1 = 14	15 x 1 = 15	16 x 1 = 16	17 x 1 = 17	18 x 1 = 18	19 x 1 = 19	20 x 1 = 20
21 x 1 = 21	22 x 1 = 22	23 x 1 = 23	24 x 1 = 24	25 x 1 = 25	26 x 1 = 26	27 x 1 = 27	28 x 1 = 28	29 x 1 = 29	30 x 1 = 30
31 x 1 = 31	32 x 1 = 32	33 x 1 = 33	34 x 1 = 34	35 x 1 = 35	36 x 1 = 36	37 x 1 = 37	38 x 1 = 38	39 x 1 = 39	40 x 1 = 40
41 x 1 = 41	42 x 1 = 42	43 x 1 = 43	44 x 1 = 44	45 x 1 = 45	46 x 1 = 46	47 x 1 = 47	48 x 1 = 48	49 x 1 = 49	50 x 1 = 50
51 x 1 = 51	52 x 1 = 52	53 x 1 = 53	54 x 1 = 54	55 x 1 = 55	56 x 1 = 56	57 x 1 = 57	58 x 1 = 58	59 x 1 = 59	60 x 1 = 60
61 x 1 = 61	62 x 1 = 62	63 x 1 = 63	64 x 1 = 64	65 x 1 = 65	66 x 1 = 66	67 x 1 = 67	68 x 1 = 68	69 x 1 = 69	70 x 1 = 70
71 x 1 = 71	72 x 1 = 72	73 x 1 = 73	74 x 1 = 74	75 x 1 = 75	76 x 1 = 76	77 x 1 = 77	78 x 1 = 78	79 x 1 = 79	80 x 1 = 80
81 x 1 = 81	82 x 1 = 82	83 x 1 = 83	84 x 1 = 84	85 x 1 = 85	86 x 1 = 86	87 x 1 = 87	88 x 1 = 88	89 x 1 = 89	90 x 1 = 90
91 x 1 = 91	92 x 1 = 92	93 x 1 = 93	94 x 1 = 94	95 x 1 = 95	96 x 1 = 96	97 x 1 = 97	98 x 1 = 98	99 x 1 = 99	

Mais l'utilisateur peut saisir manuellement des valeurs fausses. Il faut donc intervenir aussi sur le code JavaScript. Il ne faut exécuter la fonction, que si les valeurs sont dans l'intervalle. Pour cela, nous utiliserons les conditions.

En algorithme, la condition N°1 porte sur le nombre Nb et la condition N°2 sur le nombre maximum





Ainsi, nb est compris entre 1 et 100, ce qui s'écrit avec deux parties :

```
Si (nb > 0 et nb < 101) alors
    exécuter la fonction tableMultiplication
fin si
```

Mais comme il faut également tester maximum, nous allons avoir une condition imbriquée (comme pour la répétition dans le chapitre précédent).

```
function tableMultiplication(nb, maximum=10) {
    if (typeof(nb) == 'object') {
        nb = saisieNb.value
        maximum = saisieMx.value
    }

    if (nb>0 && nb<101 && maximum>1 && maximum<21) {
        let txtResult = ""
        let JSMult = document.getElementById("JSMult")
        for (let nbTable=1; nbTable<nb; nbTable++) {
            txtResult = txtResult + calculTable(nbTable, maximum)
        }
        JSMult.innerHTML = txtResult
    } else {
        JSMult.innerHTML = "Erreur : les valeurs sont hors de l'intervalle"
    }
}
```

Résultat :