(cc) BY-NC-SA

LYCÉE ARISTIDE BERGÈS
SIO

Exploration

133 - Code de César

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision
2022-11-13	Création

(cc) BY-NC-SA

Sommaire

(cc) BY-NC-SA

A Introduction

La programmation permet de simuler des éléments du monde réel, c'est la raison pour laquelle on trouve l'informatique dans tous les métiers.

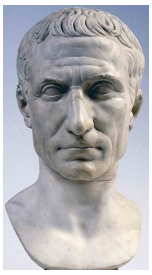
Pour coder une situation réelle et la transformer en application, il y a un travail d'adaptation que l'on appelle l'algorithmie, puis un travail de création dans un langage de programmation et enfin un travail de sécurisation.

À partir d'une situation réelle, nous allons créer une application qui reprend le fonctionnement et nous l'adapterons à l'outil informatique.

A.1 Présentation

Dans l'empire romain, transmettre un message en temps de guerre peut s'avérer dangereux : les messagers étaient souvent interceptés, torturés et éventuellement remplacés pour modifier les ordres. Ces attaques sont assimilables à l'attaque informatique appelée l'homme du milieu.

Il existe une technique utilisée par l'empereur César, consistant à décaler les lettres : A vaut C, B vaut D, etc.



Ainsi la phrase CESAR ATTAQUE CE SOIR devient EGUCT CVVCSWG EG UQKT.

Bien que ce code paraisse désuet et facile à comprendre (codage par substitution), il fût utilisé en 1915 par l'armée russe, les autres systèmes étant jugés trop complexes pour leurs troupes.

A.2 Prérequis

Pour réaliser cette exploration, vous devez connaître :

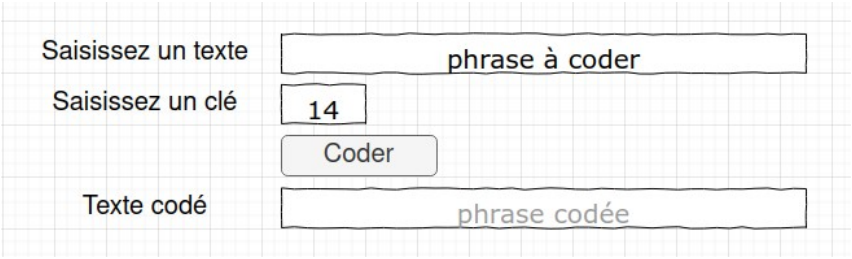
- Les itérations (notamment la boucle for)
- Les conditions
- Les fonctions
- Les formulaires HTML (input)
- Les conditions
- Les modulos (reste de division entière)

B Algorithme

L'algorithme doit décrire le fonctionnement du code. Un langage de programmation simple ne pourra convertir toute la phrase avec une fonction existante, c'est donc au programmeur de décrire toutes les étapes à suivre.

B.1 Maquette

La première chose à faire est d'imaginer les interactions avec l'utilisateur d'un tel système. Il voudra saisir sa phrase et sa clé (si tout le monde utilise la même clé, ce n'est plus un secret). Voici un exemple d'interface :



Le fait de créer cette maquette sans avoir codé, nous permet déjà de savoir ce dont nous aurons besoin dans le code HTML :

- Balises <label> et <input>
- Balises pour contenir les éléments et notamment une balise qui contiendra toute l'interface

B.2 Algorithme général

Contrairement aux codes étudiés en Python en Mathématiques pour l'informatique, JavaScript n'exécutera une action qu'après un clic sur un bouton (par contre, l'initialisation des variables peut se faire avant).

- Récupération des variables d'emplacement des champs HTML
  - ex : let emplacement bouton "Coder" = requête de sélection sur (identifiant du bouton Coder)
- Déclaration de la fonction de codage
  - ex : fonction Coder() contient { … }
- Création d'un événement associé au bouton "Coder"
  - ex. Sur le bouton "Coder", Ajoute l'écoute d'un événement : au "clic" exécuter la fonction Coder.

Il manque l'algorithme de la fonction Coder.

(cc) BY-NC-SA

B.3 Algorithme fonction "Coder"

La fonction Coder doit réaliser les actions suivantes :

- Récupérer le contenu du champ "Phrase à coder"
- Lire **chaque** caractère et…
  - Trouver la valeur ASCII de ce caractère
  - Ajouter la clé à cette valeur
  - Convertir cette nouvelle valeur ASCII en caractère
  - Ajouter ce nouveau caractère à une variable "Phrase codée"
- Écrire le résultat dans l'emplacement du champ "Phrase codée"

Nous verrons plus tard qu'il faudra imposer des restrictions et quelques corrections supplémentaires.

C Premier code

Nous allons créer un seul fichier HTML qui contiendra le CSS et le JavaScript, car l'ensemble n'est pas très volumineux.

code_cesar.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Code César</title>
    <style>
    </style>
</head>
<body>
    <div>
    </div>

    <script>
    </script>

</body>
</html>
```

C.1 Insérer les éléments du formulaire

À l'intérieur de la balise `<div>`, insérez le code suivant :

```
        <div>
            <label for="phrase">Saisie</label>
            <input type="text" id="phrase" size="30" maxlength="40" placeholder="Tapez ici une phrase à
coder...">
        </div>
        <div>
            <label for="cle">Clé</label>
            <input type="number" size="2" min="-9" max="9" value="2" id="cle">
        </div>
        <div>
            <input type="button" id="btnCoder" value="Coder">
            <input type="button" id="btnAider" value="Aider">
        </div>
        <div>
            <label for="txtCode">Résultat</label>
            <input type="text" id="txtCode" size="30" disabled>
        </div>
```

C.1.1 input

La balise `<input>` supporte plusieurs propriétés qui sont expliquées sur le site w3school. Recherchez particulièrement les propriétés `placeholder`, `maxlength`, `min` et `max`.

C.1.2 Label

Cette balise permet d'associer un texte avec un champ du formulaire : un clic sur le label active automatiquement le champ (focus). La propriété `for="..."` doit contenir l'identifiant du champ.

(cc) BY-NC-SA

C.1.3 Div

Notez l'usage intensif des divisions, qui permettent plus de souplesse et favorise ici, le placement des éléments les uns sous les autres.

C.2 Insérer le code JavaScript

Le code JavaScript consiste en une partie déclaration (qui s'exécute au moment de la lecture du code par le moteur de rendu JavaScript) et une partie réaction aux événements.

La fonction Coder est déclarée mais pas exécutée, tant qu'on ne l'associe pas à un événement.

```
console.log("Hello")

// Déclaration des variables et des fonctions
let phrase=document.querySelector("#phrase")
let cle=document.querySelector("#cle")
let btnCoder=document.querySelector("#btnCoder")
let txtCode=document.querySelector("#txtCode")
// Pour mémoire char.charCodeAt(0) et String.fromCharCode(ASCIICode)

function Coder() {
    let phraseACoder = phrase.value
    console.log(phrase.placeholder)     // sur un objet, on peut afficher ses propriétés
    let phraseCodee=""                  // déclaration d'une variable vide pour créer le code César
    let c
    for (let index = 0; index < phraseACoder.length; index++) {
        c = phraseACoder[index].charCodeAt(0)+cle.value*1   // cle.value est un texte à convertir
        phraseCodee = phraseCodee + String.fromCharCode(c)
        console.log(c, String.fromCharCode(c))
    }
    txtCode.value = phraseCodee
}

// Création des événements
btnCoder.addEventListener("click", Coder)
```

C.2.1 querySelector

Cette fonction recherche un objet dans le DOM correspondant soit à un identifiant (exemple, `#phrase`), soit une classe (exemple, `.sport`) soit un élément HTML (exemple, `h1`). La fonction renvoie le premier élément rencontré. Voir également W3school querySelector. Pour retourner une liste contenant tous les éléments, on utilisera plutôt querySelectorAll.

Enfin, pour retrouver un identifiant, il est plus rapide d'utiliser la fonction getElementById.

C.2.2 addEventListener

C'est la fonction qui va associer un événement et une fonction. L'événement devant être lié à un objet, c'est l'objet `btnCoder` qui est utilisé. Vous trouverez plus d'aide sur le site w3school addEventListener. Elle est placée à la fin du code, lorsque tous les éléments ont été déclarés.

C.2.3 charCodeAt et fromCharCode

Ces deux fonctions sont l'inverse l'une de l'autre.

La fonction charCodeAt(x) prend le caractère à la position x et le converti en une valeur numérique qui représente son code ASCII. Par exemple 'A' a pour valeur 65.

La fonction fromCharCode(code) appartient à la librairie String et convertit le code en un caractère ASCII. Par exemple, String.fromCharCode(32) renvoie un espace.

Il est impossible ou très difficile de connaître toutes les fonctions disponibles dans JavaScript et ses librairies. Cela ne doit pas vous décourager : vous devez au contraire faire des recherches sur Internet et apprendre sur des sites comme w3schools ou OpenClassrooms. Vous pouvez également tester les exemples dans mes cours.

C.3 Test du programme

Si tout se passe bien, vous obtiendrez le résultat suivant :

Vous pouvez également copier le résultat dans le champ de saisie et vérifier qu'en changeant la clé par -2, vous décodez bien la phrase codée.

C.4 Critique du programme

Notre programme est loin d'être parfait, nous allons classer les problèmes et proposer des solutions :

1. **Problème** : Les espaces sont convertis alors qu'ils ne devraient pas.
    1. **Solution** : ne pas convertir le caractère si son code ASCII est 32
2. **Problème** : Les conversions peuvent envoyer des caractères bizarres (non alphabétiques)
    1. **Solution** : utiliser un modulo 26 sur des phrases uniquement en lettres capitales
3. **Problèmes** : les champs de saisie ne sont pas alignés
    1. **Solution** : ajouter un peu de CSS

BY-NC-SA

D Amélioration du code

Le programmeur vérifie toujours son code pour pouvoir l'améliorer : c'est une partie que l'algorithmie ne peut pas valider, car elle s'affranchit des contraintes des machines et des langages de programmation.

Sauvegardez votre code sous une nouvelle version (par exemple code_cesar_v2.html). Ainsi vous pourrez comparer la progression et retravailler chaque partie indépendamment. En cas de bogue, vous pourrez toujours reprendre la version 1.

D.1 Conversion des espaces

Pour éviter la conversion des espaces, nous allons introduire une condition. Au choix, tester le caractère avant la conversion en valeur numérique (on recherche un espace) ou après la conversion (on recherche la valeur 32). Dans ce cas, on n'ajoute pas la clé, pour tous les autres cas, on ajoute la valeur de la clé.

D.1.1 Modification

Dans la boucle for, remplacer la ligne

```
c = phraseACoder[index].charCodeAt(0)+cle.value*1
```

Par

```
c = phraseACoder[index].charCodeAt(0)
if (c != 32) {
    c = c+cle.value*1
}
```

D.1.2 Vérification

En saisissant la chaîne CESAR ATTAQUE CE SOIR, le programme propose la réponse : EGUCT CVVCSWG EG UQKT.

D.2 Caractères spéciaux

Nous pouvons gérer le problème des caractères en lettres capitales avec une fonction JavaScript qui s'appelle toUpperCase() et qui s'applique sur une chaîne.

La conversion ne devrait porter que sur les caractères ASCII de 'A' à 'Z' et le programmeur doit demander à son client que faire pour la valeur 'Z' + 2 ? Dans notre cas, pour éviter les caractères spéciaux, il faut revenir au début de l'alphabet : un 'Z' + 2 donnera un 'B'.

A	B	C	...			...	X	Y	Z
C	D	E	...			...	Z	A	B

Pour réussir cela, il faut utiliser la fonction mathématique de modulo :

Ainsi, si le code ASCII de 'Z' était 26 (attention, ce n'est pas le cas), la conversion donnerait le nombre 28. Grâce au modulo, 28 % 26 = 2, qui serait le 'B'.

Dans la réalité, il suffit d'abord soustraire la valeur du premier caractère ('A' = 65), effectuer le modulo et rajouter 65 pour que l'opération fonctionne. Le nombre 65 est un *offset* (en français, *décalage*).

D.2.1 Modification

Corriger le code

```
let phraseACoder = phrase.value
```

par

```
let phraseACoder = phrase.value.toUpperCase()
```

puis modifier le code

```
if (c != 32) {
    c = c+cle.value*1
    phraseCodee = phraseCodee + String.fromCharCode((c-65)%26+65)
} else {
    phraseCodee = phraseCodee + String.fromCharCode(c)
}
```

Enfin, pour être plus propre, convertir le champ de saisie en lettres capitales aussi en ajoutant :

```
txtCode.value = phraseCodee
phrase.value = phraseACoder
```

D.2.2 Vérification

La phrase Cesar attaque a zero heure donne bien EGUCT CVVCSWG C BGTQ JGWTG (et le champ de saisie initial passe bien à CESAR ATTAQUE A ZERO HEURE)

BY-NC-SA

D.3 Esthétique CSS

La mise en place d'un peu de CSS va permettre d'aligner les champs et rendre le formulaire plus agréable à utiliser.

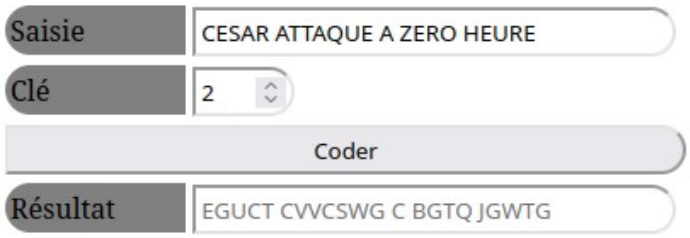
D.3.1 Modification

Dans la section <style> ajouter le code CSS suivant :

```
#formulaire {
    width: 400px;
}
label {
    display: inline-block;
    min-width: 100px;
    background-color: gray;
    margin: 5px 0px 0px 0px;
    padding: 3px;
    border-radius: 20px 0px 0px 20px;
}
input {
    border-radius: 0px 20px 20px 0px;
    margin: 5px 0px 0px 0px;
    padding: 3px;
}
#btnCoder {
    width: 100%;
}
```

D.3.2 Vérification

L'affichage devrait être un peu plus sympathique



Sauvegardez votre code sous une nouvelle version (par exemple code_cesar_v3.html). Cette version contient les modifications utiles pour faire fonctionner l'application avec un utilisateur sympa. La prochaine version va augmenter la sécurité et la qualité du code.

E Sécurisation du code

Jusqu'à maintenant, l'utilisateur était plutôt sympathique et savait ce qu'il devait faire.

Mais que se passe-t-il s'il utilise des symboles ou des chiffres dans la phrase à coder ?

Saisissez la phrase suivante : César attaque à 0h00, cette nuit ! (utilisez bien les accents et la ponctuation). Le résultat semble fonctionner et renvoie EIUCT CVVCSWG Z 2J22. EGVVG PWKV =

Si nous saisissons la phrase à décoder et repassons dans le programme, il y a quand même un problème :

Le programme a pour résultat : CGSAR ATTAQUE X 0H00, CETTE NUIT ;

Il y a donc deux possibilités : **informer** l'utilisateur sur les règles d'utilisation de notre programme (une aide) et le **blocage** s'il ne respecte pas cette règle.

E.1 Information

Il existe plusieurs moyens de créer de l'aide : une page spécifique, les info-bulles ou… un didacticiel.

Le moyen le plus facile serait une page spécifique (avec un lien `<a href="aide.html">aide</a>`), toutefois, les didacticiels sont à la mode en ce moment, et il existe des frameworks (des structures ou librairies) qui permettent de le faire très facilement.

E.1.1 Modification

Nous allons ajouter un bouton Aide à côté du bouton Coder, dans la même division :

```
<input type="button" id="btnCoder" value="Coder">
<input type="button" id="btnAider" value="Aider">
```

Nous allons ajouter le framework intro.JS disponible sur Internet ainsi que la feuille de style CSS associée. Pour le script, il faut ajouter la ligne suivante au-dessus de notre script :

```
<script src="https://cdn.jsdelivr.net/npm/intro.js@6.0.0/intro.min.js"></script>
<script>
    // notre code.........
```

Il s'agit d'une source dite CDN (Content Delivery Network). Plutôt que de télécharger le fichier localement dans votre répertoire de travail, le navigateur le télécharge directement depuis Internet. Le fichier sera mis en cache du navigateur pour de futures utilisations. Il sera supprimé du cache après quelques jours.

De même, nous allons ajouter la feuille de style à l'intérieur de notre balise `<style>`, de préférence au début :

```
<style>
    @import url("https://cdn.jsdelivr.net/npm/intro.js@6.0.0/introjs.css");
```

(cc) BY-NC-SA

Enfin, en étudiant la documentation, nous allons utiliser la propriété `data-intro="texte d'aide"` dans nos balises `<div>`.

Pour la balise `<div>` qui contient le champ de saisie de la phrase :

```
<div data-intro="Saisir ici une phrase sans accents ni symboles de ponctuation">
    <label...>
    <input...>
```

Pour la balise `<div>` qui contient la clé (je ne remets pas tout le code) :

```
<div data-intro="La clé symétrique : positive, elle permet de coder la phrase. Négative, elle permet de le décoder.">
```

Pour les deux boutons :

```
<input type="button" id="btnCoder" value="Coder" data-intro="Lance le codage">
<input type="button" id="btnAider" value="Aider" data-intro="Affiche ce tutoriel">
```

Et enfin pour le résultat :

```
<div data-intro="Le résultat s'affichera ici, en lettres capitales">
```

Pour que ce code puisse fonctionner, nous allons créer associer une fonction à ce bouton d'aide dans la partie JavaScript. La fonction :

```
function Aider() {
    introJs().start()
}
```

Et l'association au bouton :

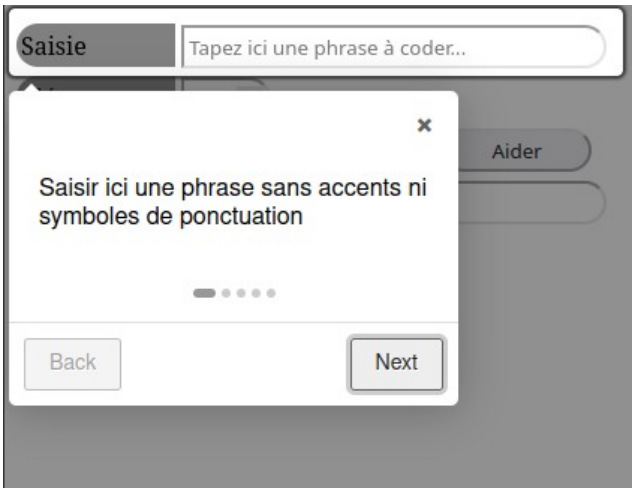
```
btnAider.addEventListener("click", Aider)
```

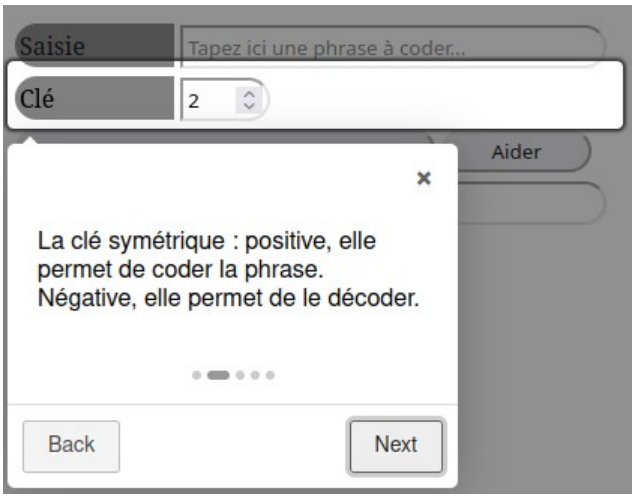
Sauvegardez votre résultat.

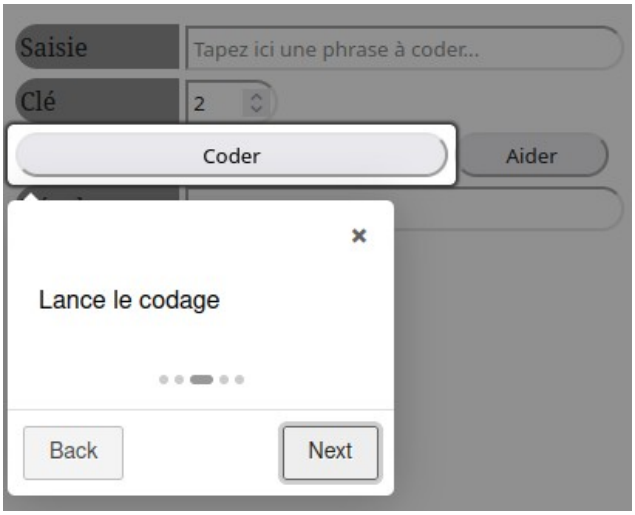
Vous devez créer l'association vers la fin du code, sous l'association du bouton btnCoder. À l'inverse, la fonction Aider() doit se trouver juste après la fonction Coder(). En cas de problème, lisez les messages d'erreurs dans la console de débogage (touches `Ctrl`+`Shift`+`I`).

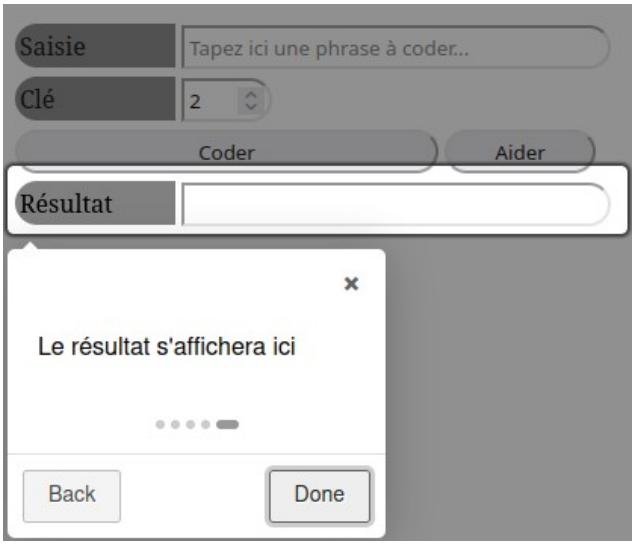
E.1.2 Vérification

Lors d'un clic sur le bouton d'aide, un tutoriel devrait apparaître.









(cc) BY-NC-SA

E.2 Blocage

L'idéal serait un blocage à la saisie, mais la nature événementielle ne permet pas facilement ce traitement. Nous allons donc effectuer ce traitement lors du clic pour la conversion : on peut effectuer la conversion, si tous les caractères sont compris entre les intervalles suivants :

- A à Z (65 à 90)
- 0 à 9 (48 à 57)
- Espace (32)

Pour cela, il faut lire chaque caractère et s'il ne répond à aucun intervalle, renvoyer un résultat faux, sinon continuer (et à la fin, renvoyer un résultat vrai). Voici la fonction à écrire :

```
function Verifier(phrase) {
    let c=0
    for (let index=0; index < phrase.length; index++) {
        c = phrase.charCodeAt(index)
        if ((c == 32) || (c > 64 && c < 91) || (c > 47 && c < 58)) {
            // c'est bon on continue
        } else {
            return false
        }
    }
    return true
}
```

Comme vous pouvez le constater, la condition est relativement complexe : pour rappel, l'usage de `||` signifie OU et l'usage de `&&` signifie ET.

Notez que l'ensemble de la condition est entre parenthèses (en bleu dans le code ci-dessus).

Il faut maintenant modifier la fonction Coder() comme suit :

```
function Coder() {
    let phraseACoder = phrase.value.toUpperCase()
    if (Verifier(phraseACoder)==true) {

     // ancien code de la fonction


    } else {
        alert("Erreur de saisie : vérifiez vos symboles.")
    }
}
```

Le plus simple est de ré-écrire la fonction Coder() (faire un copier/coller du code ci-dessus sans effacer l'ancienne fonction) puis de faire un `Ctrl`+`X` du contenu de l'ancienne fonction Coder() et enfin, d'effacer l'ancienne fonction Coder() qui doit être vide.

Si vous rencontrez des problèmes, utilisez la fonction **Undo** (`Ctrl`+`Z`) pour revenir à l'état précédent.

E.2.1 Vérification

Vous devez utiliser un jeu de test pour valider votre saisie. Un jeu de test est un ensemble de données qui feront fonctionner le code différemment. Ici, nous allons utiliser les tests suivants :

Saisir Cesar a z 0 9

Le résultat sera EGUCT C B 2 ;

Problème : on constate que le chiffre 9 est codé par un symbole. Cela posera un problème pour le décodage.

Solution : on peut accepter les symboles si la clé est négative.

La condition devient :

```
if (Verifier(phraseACoder)==true || cle.value*1 < 0) {
```

Problème : le modulo à l'envers (en partant de 65) pose ne retrouve pas la phrase originale (ici, on a EGUCT C B 2 ; qui donne CESAR A @ 0 9)

Solution : une nouvelle aventure pour le développeur commence…

E.3 Séparation en fichiers et commentaires

Nous l'avons vu, il est préférable de séparer le code JavaScript du fichier HTML et de même pour le code CSS.

Créez deux fichiers `code_cesar.css` et `code_cesar.js` puis copier le contenu de la section `<style>` dans le fichier CSS et le contenu de la section `<script>` dans le fichier JavaScript.

Ajoutez les lignes suivantes :

```
<style>
    @import url('code_cesar.css');
```

et

```
<script src='code_cesar.js'></script>
```

Enfin, ajoutez un entête de commentaires dans le fichier JavaScript avec l'auteur, la date, le rôle du programme…

(cc) BY-NC-SA

F Conclusion

Le codage n'est jamais fini : un code aussi simple que le codage par substitution de César peut entraîner de nombreux rebondissements. Par exemple :

- Est-il possible de lancer le tutoriel au premier lancement du programme (en appelant la fonction les associations d'événement) ?
- Est-il possible de tester automatiquement toutes les combinaisons pour décoder une phrase (26 possibilités) ?
- Est-il possible (avec l'instruction JavaScript `navigator.clipboard.writeText(texte)`) d'envoyer directement le résultat du codage dans le presse-papier ? Voir w3schools…
- Est-il possible de ne pas utiliser une fenêtre modale (bloquante) comme message d'erreur, mais un nouveau champ vide (et un peu de CSS pour le style) ? Ou bien, par exemple, teindre en rouge le champ de saisie en cas d'erreur ?
- Peut-on proposer plusieurs méthodes de chiffrement (par exemple, avec une clé "361" on décale le premier caractère de 3, puis le deuxième de 6 puis le troisième de 1 et on recommence, le quatrième de 3, etc.) ?

Essayez de réaliser une ou plusieurs améliorations du code