Programmation

JS-Crypto

Sammy Dieleman

prog@amstram.be

v15-2024

JS-Crypto

Nous allons travailler sur les chaînes de caractères, dans le but de créer un programme basique de cryptographie.

Commencez toujours par écrire les commentaires d'intention et puis le code correspondant. Les commentaires, leur qualité et leur pertinence, sont partie intégrante de votre cote finale.

Le but étant de manipuler des chaînes de caractères, l'utilisation des fonctions et propriétés des String en JavaScript **doit se limiter à** :

```
length
charAt()
charCodeAt() // à partir de 8, en mode avancé
```

Respectez toutes les consignes à la lettre.

JS-Crypto

Workspace

1) Creer un private repository sur github, nommer le **js_crypto**
2) Telecharger le fichier `js_crypto.htm` depuis le github du cours
3) Dans le même répertoire, créer un fichier JavaScript que vous appellerez `crypto_lib.js` (suffixé par votre nom, comme pour le fichier HTML)

4) Adapter le fichier HTML pour qu'il charge le bon fichier `.js`,
5) Charger manuellement le fichier HTML dans votre navigateur.
6) Implémenter les fonctions demandées, dans `crypto_lib.js`

JS-Crypto

Méthodologie

Le fichier `js_crypto.htm` contient des tests qui valideront votre programmation au fur et mesure. Vous devez donc, à la fin, avoir géré toutes les erreurs affichée dans la console.

Les erreurs des tests "Avancés" sont secondaires, et doivent donc être traitées si vous avez le temps.

JS-Crypto

Partie 1 : What are words worth ?

Déclarer et implémenter une fonction `what_are_words_worth()` qui demande une entrée à l'utilisateur, calcule sa longueur (nombre de caractère qui la compose) et puis qui demande à l'utilisateur de confirmer le nombre de lettres de ce mot.

La fonction renvoie le booléen `true` si la longueur est validée par l'utilisateur, sinon elle renvoie `false`

Pour lire une entrée de l'utilisateur :

```
let valeur_entree = prompt(message);
```

Pour demander une confirmation :

```
confirm(message);
```

Ex : si le mot introduit est "Alan Turing", le message de confirmation sera

```
Votre entrée "Alan Turing" comporte 11 caractère(s)
```

JS-Crypto

Partie 2 : *Count Wordula*

Déclarer et implémenter une fonction `count_words(string_of_words)` qui prend en paramètre une phrase et qui retourne le nombre de mots dans cette phrase. Pour se faire, on suppose que les mots de la phrases ne sont séparés que par des espaces et ne contiennent aucun caractères de ponctuation.

Ex :

```
counter = count_words('Turing believes machines think');
console.log(counter) ;  // affiche 4
```

JS-Crypto

Partie 3 : *Count Wordula, 2nd of his name*

Déclarer et implémenter une fonction

`count_words_by(string_of_words, letter).`

Elle est similaire à la fonction `count_words(string_of_words)` mais le 2e paramètre, optionel, permet de préciser le séparateur des mots dans une phrase.

Ex :

```
counter = count_words_by('Turing believes machines think');
console.log(counter) ;  // affiche 4

counter = count_words_by('Turing believes machines think', ' ');
console.log(counter) ;  // affiche 4

counter = count_words_by('Turing believes machines think','i');
console.log(counter) ;  // affiche 5
```

JS-Crypto

Partie 4: ***des chiffres & des lettres : voyelles***

Déclarer et implémenter une fonction `count_vowels(string_of_words)` qui prend en paramètre une phrase et retourne le nombre de voyelles contenues dans cette phrase.

Cette fonction doit faire appel à une fonction `is_a_vowel(letter)`, que vous devez aussi déclarer et implémenter. Cette fonction prend en paramètre une lettre unique et renvoie `true` s'il s'agit d'une voyelle, sinon `false`.

Utiliser le tableau global `vowels` déjà présent dans votre fichier HTML *(merci, de rien)*

Ex :

```
count_vowels('Turing believes machines think');   //renvoie 10
count_vowels('zrtpmlkjhgfdsqe');                  //renvoie 1
count_vowels('oui');                              //renvoie 3
count_vowels('lmpt');                             //renvoie 0
```

JS-Crypto

Partie 5: ***des chiffres & des lettres : consonnes***

Déclarer et implémenter une fonction `count_consonants(string_of_words)` qui prend en paramètre une phrase et retourne le nombre de consonnes contenues dans cette phrase.

Pour rappel : les phrases ne contiennent que des lettres et des espaces et aucun signe de ponctuation.

Cette fonction ne peut pas parcourir votre chaîne de caractères, et doit s'écrire sur une seule ligne.

Utiliser le tableau global `vowels` déjà présent dans votre fichier HTML *(merci, de rien)*

Ex :

```
count_consonants('Turing lies with men');      //renvoie 11
```

JS-Crypto

Partie 6: *Search & Destroy*

Déclarer et implémenter une fonction `remove(string_of_words, caracter)` qui prend en paramètres une phrase et un caractère a supprimer dans cette phrase.

La fonction retourne la phrase sans les occurrences du caractère a supprimer.

Ex :

```
altered_string = remove('therefore machines do not think', 'e'));
console.log(altered_string) ;
// affiche 'thrfor machins do not think'
```

JS-Crypto

Partie 7: *Caedite eos. Novit enim Dominus qui sunt eius*

Déclarer et implémenter une fonction `remove_many(string_of_words, caracters)` qui prend en paramètres une phrase et une série de caractères a supprimer dans cette phrase. La fonction retourne la phrase sans les occurrences des caractères a supprimer.

Ex :

```
remove_many(‘Therefore machines do not think’, ‘dumber’));
// renvoie ‘Thfo achins o not think’

remove_many('Turing lies with men','good people read good books');
// renvoie 'Tuiniwithmn'
```

JS-Crypto

Partie 8: *Caesar (1/3)*

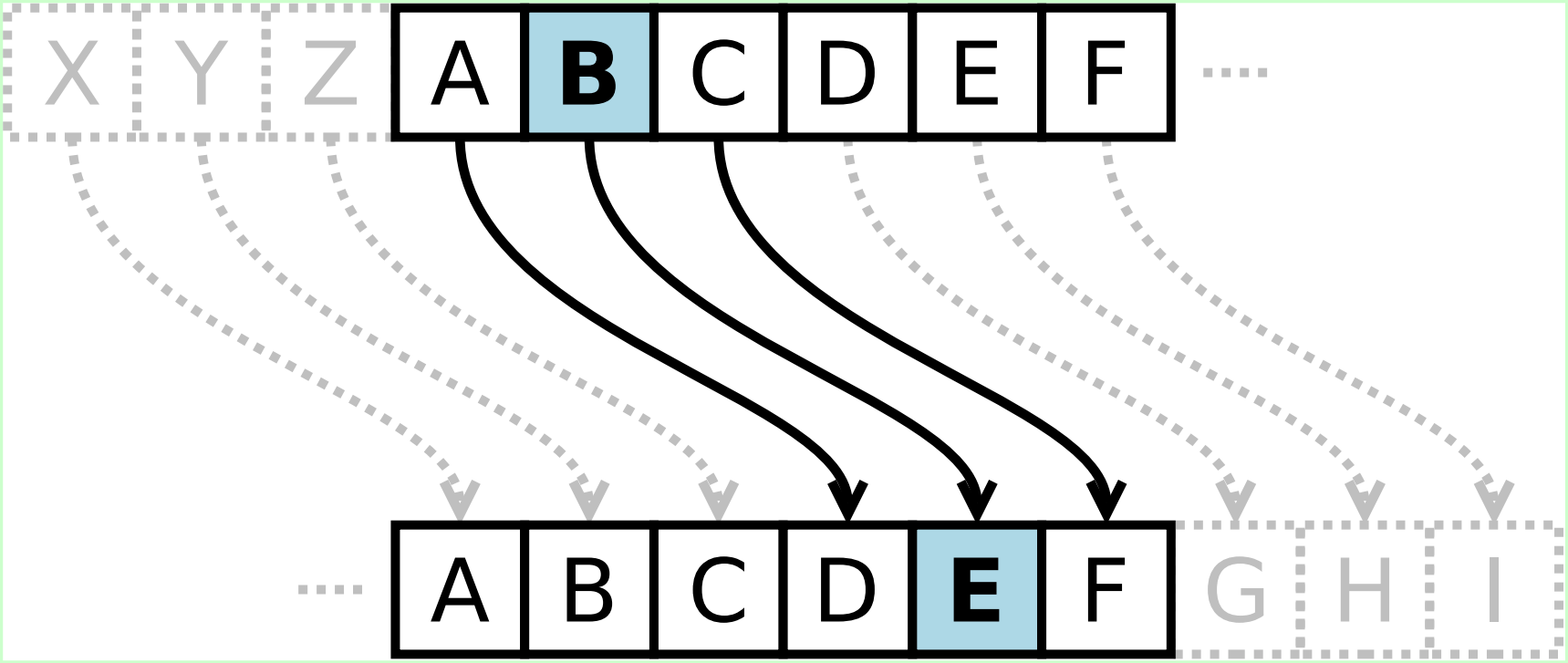
Déclarer et implémenter une fonction `crypto(a_string, a_number)` qui prend en paramètre une chaine de caractère et un nombre ; et retourne une version cryptée selon le principe suivant:

> *Le chiffre de César est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes.*
>
> *Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite), on reprend au début.*

JS-Crypto

Partie 8: ***Caesar (2/3)***

Par exemple, avec un décalage de 3 vers la droite, A est remplacé par D, B devient E, et ainsi jusqu'à W qui devient Z, puis X devient A etc. Il s'agit d'une permutation circulaire de l'alphabet



JS-Crypto

Partie 8: *Caesar (3/3)*

Ex :

```
to_crypt = 'it is possible to invent a single machine which can be
used to compute any computable sequence' ;

crypted = crypto(to_crypt, 3);
console.log(crypted);
```

affiche

```
lw lv srvvleoh wr lqyhqw d vlqjoh pdfklqh zklfk fdq eh xvhg wr
frpsxwh dqb frpsxwdeoh vhtxhqfh
```

JS-Crypto

Partie 9: *Décrypter*

Déclarer et implémenter une fonction `decrypt(a_string, a_number)` qui prend en paramètre une chaine de caractère et un nombre ; et retourne une version décryptée selon le principe précédemment exposé

Ex :

```
decrypt(te td azddtmwp ez tygpye l dtyrwp xlnstyp hstns nly mp
fdpo ez nzxafep lyj nzxafelmwp dpbfpynp', 11);
```

Renvoie

```
it is possible to invent a single machine which can be used to
compute any computable sequence
```

Cette fonction s'écrit sur une seule ligne

JS-Crypto

Partie 10: *Enigma*

Déclarer et implémenter une fonction `enigma(crypted_string)` qui prend un paramètre une chaîne de caractère cryptée avec le chiffrement césar et qui vous aide à trouver la clé de chiffrement.

Cette fonction s'écrit en 3 lignes (sans compter les { })

Quelle est la clé pour cette phrase :

```
Lzw akgdslwv esf vgwk fgl vwnwdgh sfq aflwddwulmsd hgowj. Al ak
fwuwkksjq xgj zae lg tw aeewjkwv af sf wfnajgfewfl gx glzwj ewf,
ozgkw lwuzfaimwk zw stkgjtk vmjafy lzw xajkl lowflq qwsjk gx zak
daxw. Zw esq lzwf hwjzshk vg s dalldw jwkwsjuz gx zak gof sfv
escw s nwjq xwo vakugnwjawk ozauz sjw hskkwv gf lg glzwj ewf.
Xjge lzak hgafl gx nawo lzw kwsjuz xgj fwo lwuzfaimwk emkl tw
jwysjvwv sk usjjawv gml tq lzw zmesf ugeemfalq sk s ozgdw, jslzwj
lzsf tq afvanavmsdk
```

Fin