HEI Madagascar : 30 shades of PROG1

Manipulation de chaînes de caractères

1. **Compter les chiffres dans une chaîne**
    - Écrire une fonction qui retourne le nombre de chiffres présents dans une chaîne de caractères.
    - Exemple : l'entrée "test123" retourne 3
2. **Suppression des espaces**
    - Écrire une fonction qui retire tous les espaces d'une chaîne de caractères.
    - Exemple : l'entrée "a b c" retourne "abc"
3. **Vérification d'une chaîne numérique**
    - Écrire une fonction qui compte combien de chiffres composent un string.
    - Exemples : "123" retourne 3, et "a1c" retourne 1;
    - Indice : isNaN(x) permet de vérifier si *x n'est pas un nombre*.
4. **Somme des chiffres et concaténation des lettres**
    - Écrire une fonction qui retourne un tableau contenant la somme des chiffres contenus dans une chaîne ainsi que la concaténation des chaînes de caractère séparées par des espaces.
    - Exemple : l'entrée [1, "hello", 2, "world"] retourne [3, "hello world"]
5. **Reproduction de `slice`**
    - Écrire une fonction qui reproduit le comportement de `slice()` en JavaScript.
6. **Inversion de chaque mot d'une phrase**
    - Écrire une fonction qui inverse chaque mot d'une phrase sans changer leur ordre.
    - exemple : l'entrée "this is a test" retourne "siht si a tset"
7. **Alternance des lettres de deux chaînes**
    - Écrire une fonction qui alterne les lettres de deux chaînes de même longueur.
    - Exemple : "jri" et "adn" donne "jardin"
8. **Reproduction de `splice` (effacement)**

- Écrire une fonction qui imite la méthode `splice()` pour supprimer un certain nombre d'éléments des éléments d'un tableau à un index donné.

9. **Reproduction de `splice` (ajout)**
    - Écrire une fonction qui imite la méthode `splice()` pour ajouter un ou plusieurs éléments à un tableau à un index spécifique.
10. **Tronquer une chaîne et ajouter `...`**
    - Écrire une fonction `trim` qui tronque une chaîne après un certain nombre de caractères et ajoute `...` si nécessaire.
    - Exemple : quand on trim "hello" à 2, cela donne "he…"

Transformations et formatage

11. **Deviner la fonction `accum`**
    - Écrire une fonction `accum` qui répète chaque lettre d'une chaîne selon sa position et sépare les segments par `-`.
    - Exemple : accum("abcd") retourne A-Bb-Ccc-Dddd
12. **Séparation des mots en Camel Case**
    - Écrire une fonction qui insère un espace devant chaque majuscule d'une chaîne en Camel Case.
    - Exemple : l'entrée "myAwesomeCode" donne "my Awesome Code".
13. **Décodage d'un message caché**
    - Écrire une fonction qui extrait un message caché d'un tableau en prenant le `i`-ème caractère du `i`-ème mot.
    - Exemple : "Yet we rise" donne : "Yes"; "He cannot patch passwords" donne "Hats"
14. **Transformation en Pascal Case**
    - Écrire une fonction qui transforme une phrase en Pascal Case.
    - Pour rappel voici un mot en Pascal Case : *ImportantVariable*. Il ressemble à un camel case mais la première lettre est en majuscule.

Manipulation de tableaux

15. **Suppression du plus petit élément d'un tableau**
    - Écrire une fonction qui supprime le plus petit élément d'un tableau sans modifier l'original.
    - S'il y a plusieurs éléments qui ont la valeur minimale, on n'effacer que celle qui apparaît en premier dans le tableau selon leur index.
    - Exemples : l'entrée [1, 2, 3] retourne [2, 3], l'entrée [1, 2, 1, 4, 5] retourne [2, 1, 4, 5]
16. **Somme de deux tableaux de taille égales**
    - Écrire une fonction qui additionne deux matrices, représentées sous la forme de tableaux multidimensionnels.
    - En somme, la somme des matrices, quand elles ont la même taille, se résume à additionner les éléments aux mêmes positions (voir les exemples ci-dessous pour des exemples) pour donner une nouvelle matrice.

$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} + \begin{pmatrix} 4 & 1 \\ 5 & 3 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 6 & 6 \end{pmatrix}$$

$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{bmatrix} + \begin{bmatrix} 5 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 6 & 4 & 3 \\ 5 & 6 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{bmatrix} - \begin{bmatrix} 5 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix} = \begin{bmatrix} -4 & 0 & -3 \\ 3 & 0 & -5 \end{bmatrix}$$

17. **Différence entre deux tableaux**
    - Écrire une fonction qui retourne les éléments d'un premier tableau qui ne sont pas dans un autre tableau.
    - Exemple : pour les entrées [1,2,3] et [1, 3] retourne [2], pour les entrées [1, 5, 10] et [ ] retourne [1, 5, 10]. Si l'entrée est un tableau vide, on retourne un tableau vide ;

18. **Tri mixte (nombres et chaînes)**
    - Écrire une fonction qui trie un tableau en plaçant les nombres en premier, puis les chaînes de caractères.
    - Exemple : pour l'entrée [1, "a", 2, "d", "c", 3, 4] retourne [1, 2, 3, 4, "a", "d", "c".
    - Vous devez garder l'ordre de passage des chiffres et des chaines de caractère.

Structures et Algorithmes

19. **Somme des diagonales d'une matrice carrée**
    - Écrire une fonction qui calcule la somme des deux diagonales d'une matrice carrée (le nombre de lignes est égal au nombre de colonnes) ;
    - exemple : pour l'entrée [[1, 2, 3], [4, 5, 6], [7, 8, 9]] la somme des diagonales est 1 + 5 + 9 pour l'une, et 3 + 5 + 7 pour l'autre. Ce qui donne 30.
20. **Vérification des parenthèses équilibrées**
    - Écrire une fonction qui vérifie si une chaîne de parenthèses est bien formée
    - Exemples : ")" retourne false; "( )" retourne true, "( ))" retourne false, et " ((( )))" retourne true.
21. **Vérification de réarrangement possible (`scramble`)**
    - Écrire une fonction qui détermine si l'on peut réarranger les lettres d'une première chaîne pour obtenir une deuxième chaîne fournie en paramètre.
    - Exemples :
        i. scramble('rkqodlw', 'world') ==> True
        ii. scramble('cedewaraaossoqqyt', 'codewars') ==> True
        iii. scramble('katas', 'steak') ==> False
22. **Vérification de pangramme**
    - Écrire une fonction qui vérifie si une phrase contient toutes les lettres de l'alphabet au moins une fois.
    - Exemple : "The quick brown fox jumps over the lazy dog" retourne true, et "This is not a pangram." retourne false.
23. **Traitement du caractère `#` comme suppression**

- Écrire une fonction qui interprète # comme une suppression du caractère précédent.

24. Duplicate encoder

- Le but de cet exercice est de convertir une chaîne de caractères en une nouvelle chaîne où chaque caractère de la nouvelle chaîne est « (» si ce caractère n'apparaît qu'une seule fois dans la chaîne originale, ou «) » si ce caractère apparaît plus d'une fois dans la chaîne originale.
- Ne tenez pas compte de la casse lorsque vous déterminez si un caractère est dupliqué.
- Exemples :
  i. « din » => »(((»
  ii. « recede » => »()()() »
  iii. « Success » => »)())()) »
  iv. « ((@ » => »))((»

Exercices ludiques

25. **Jeu du pendu**
    - Implémenter le jeu du pendu avec un ensemble de mots aléatoires qu'il faudra deviner lettre par lettre.
26. **Alerte aux moutons (identifier le loup)**
    - Écrire une fonction qui détecte la position du loup dans un tableau et affiche l'alerte appropriée.
27. **Génération de liens HTML**
    - Écrire une fonction qui génère du HTML à partir d'un tableau d'objets. Chaque objet contenant l'url a mettre en href, et le texte du lien.
    - Exemple : Pour l'entrée [{ url: "http://www.google.com", text: "10^100" }], la fonction retournera 10^100 .
    - Exemple 2 : Pour l'entrée [{ url: "#q", text: "query" }, { url: "#a", text: "ans" }], la fonction retournera queryans

---

Battle dev

28. Battle dev niveau 2 : Pré carré

La ferme a désormais bien plus d'allure ! Vous pouvez vous mettre au travail. Vous voulez profiter de ce début d'hiver pour refaire les haies autour de vos parcelles. Vous souhaitez déterminer quelle sera la longueur totale de haie que vous aurez à planter.

En regardant le plan vos parcelles semblent plutôt ordonnées : chacune d'elle est un carré, et le côté sud de chaque parcelle est aligné. Il ne vous manque que la liste des différentes largeurs de chaque parcelle.

A partir de cette liste de largeurs, écrivez un programme qui renvoie la distance de haie que vous aurez à planter.

Données

Entrée

Ligne 1 : un entier N (1 <= N <= 10) représentant le nombre de parcelles Ligne 2 : N entiers compris entre 1 et 100 séparés par des espaces, représentant dans l'ordre la largeur de chaque parcelle

Sortie

Un entier représentant la longueur totale de haie qu'il est nécessaire de planter pour délimiter toutes les parcelles.

Attention : veillez à ne planter qu'une seule fois la haie là où les parcelles se touchent.

Exemples

Exemple 1

2

1 2

Votre programme devra renvoyer : 11

Exemple 2

4

3 4 6 1

Votre programme devra renvoyer : 48

Exemple 3

1

5

Votre programme devra renvoyer : 20

29. **Battle dev niveau 2 : Jeu de cartes**

Objectif

En balade à Las Vegas, vous tombez sur un stand avec un jeu simple : le croupier mélange le jeu de carte (limité aux cartes de 1 à 9) et en tire une au hasard. Avant le tirage, si vous aviez annoncé le bon chiffre, vous remportez huit fois votre mise, sinon vous perdez votre argent.

Vous savez que les gains potentiels sur ce type de jeu sont très faibles mais votre ami, n'est pas du même avis. Il se lance ! Quitte à le regarder jouer, vous lui conseillez de jouer les chiffres qui tombent souvent consécutivement (il n'y a pas vraiment d'explication pour cette méthode, mais bon, cela fait au moins une stratégie).

Vous étudiez donc les tirages passés qui sont affichés sur un écran au-dessus de la table pour déterminer la longueur de la plus grande série où un chiffre est répété.

Données

Entrée

Ligne 1 : un entier N compris entre 1 et 10 000 correspondant au nombre de tirages affichés dans l'historique.

Lignes 2 à N+1 : un chiffre entre 1 et 9 correspondant au numéro de la carte tirée.

Sortie

Un nombre correspondant à la longueur de la plus grande série où un chiffre est répété.

Exemple

Pour l'entrée :

10

5

3

3

4

9

9

9

9

9

6

La sortie sera 5, car la plus longue série où un chiffre s'est répété est lorsque le 9 est tombé 5 fois de suite.

30. **Battle dev niveau 3 : 25 heures sur 25**

Objectif

Difficile de trouver un moment libre dans l'agenda de tous vos collègues pour organiser vos réunions d'équipe ! Vous décidez de leur demander de vous communiquer les créneaux où ils sont indisponibles. En utilisant, ces données, votre objectif est de trouver un créneau de 60 minutes consécutives qui conviendra à tout le monde pendant la semaine à venir.

Données

Entrée

Ligne 1 : un entier N compris entre 1 et 1000 représentant le nombre de créneaux impossibles pour la réunion

Lignes 2 à N + 1 : un créneau impossible pour un collègue au format jour hh:mm-hh:mm. Le jour est au format ISO : 1 = lundi, 2 = mardi, etc. Les minutes de début et de fin sont incluses dans l'indisponibilité. Les horaires de travail sont du lundi au vendredi de 8:00 à 17:59. Les créneaux impossibles sont inclus dans les horaires de travail de votre entreprise.

Sortie

Une ligne au format jour hh:mm-hh:mm représentant l'horaire de réunion choisi. Il doit être :

- pendant les horaires de travail, sans les dépasser

- d'une durée d'exactement 60 minutes

- Les minutes de début et de fin sont incluses dans l'horaire donc une réunion de 08:00 à 8:59 ou de 9:20 à 10:19 font exactement 60 minutes

- n'être en intersection avec aucun créneau impossible d'aucun collègue

- il est garanti qu'il existe au moins une solution. S'il en existe plusieurs, vous pouvez donner n'importe quel horaire valide.

Exemple

Pour l'entrée :

5

1 08:45-12:59

3 11:09-11:28

5 09:26-09:56

5 16:15-16:34

3 08:40-10:12

Une solution possible est :

1 13:00-13:59.

En effet, le premier jour il n'y a qu'un seul créneau impossible de 08:45 à 12:59. En faisant par exemple commencer la réunion à 13:00 et en la terminant à 13:59, elle n'aura aucune intersection avec les créneaux impossibles. Il existe par ailleurs de nombreuses autres solutions par exemple n'importe quel intervalle de 60 minutes durant les horaires de travail du jour 2.