Eric Schrafstetter

92 challenges

JavaScript

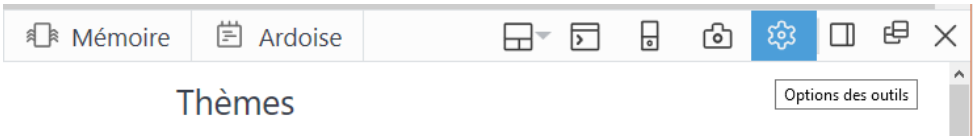
Niveau facile à avancé

www.codewars.com/users/EricSchraf

MODE D'EMPLOI

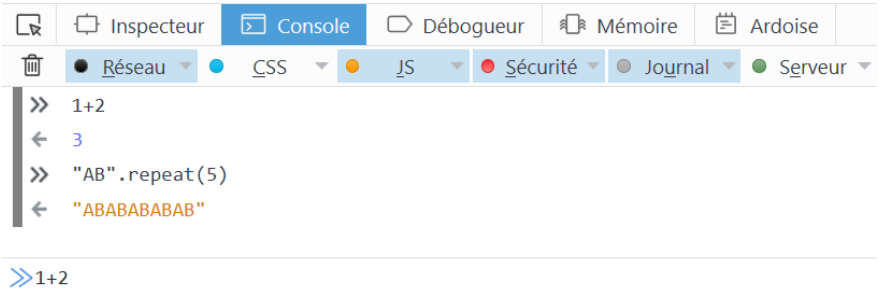
▪ DE QUOI AVEZ-VOUS BESOIN ?

- Uniquement un navigateur (Firefox ou Chrome de préférence).
- Lancez le navigateur puis touche **F12**.
- A droite **Options des outils** puis **Outils de développement par défaut** → cochez **Ardoise**



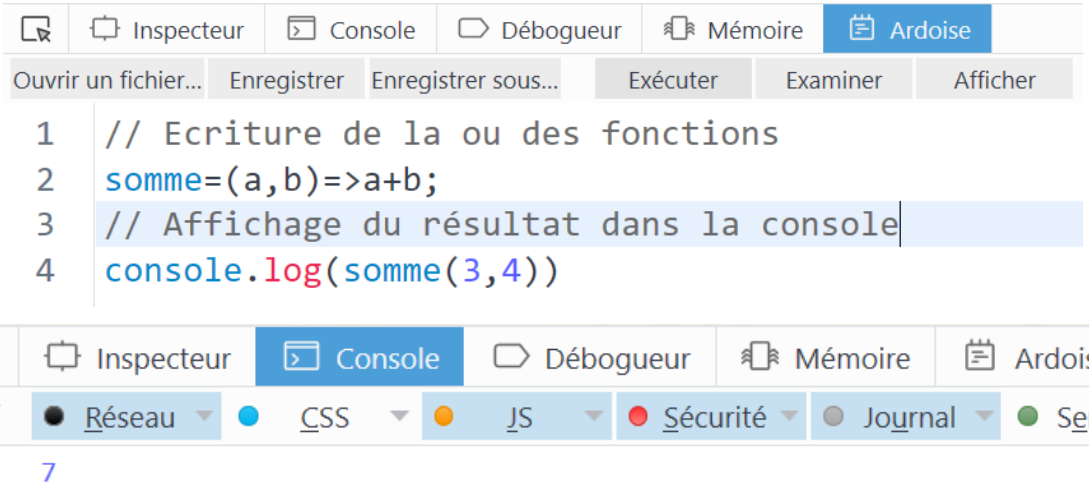
▪ TEST AVEC LA CONSOLE

- Cliquez sur Console puis tapez **1+2** Entrée.



▪ ÉCRIRE UN PROGRAMME AVEC L'ARDOISE

- Cliquez sur Ardoise puis tapez votre programme (généralement une fonction)
- Sous le programme tapez **console.log(nom_de_la_fonction(parametres))**
- Cliquez sur **Exécuter** et regardez le résultat dans la console



CHALLENGES CODESWARS.COM

#1 - Nombre de personnes dans le bus (8 kyu)

Un bus se déplace en ville, il prend et/ou dépose certaines personnes à chaque arrêt.
Vous recevez une liste d'entiers. Chaque élément comporte le nombre de personnes qui montent dans le bus (le premier élément) et le nombre de personnes qui en sortent (le deuxième élément).
Le 2e nombre du premier élément de la liste vaut toujours 0 car le bus est vide en arrivant au premier arrêt de bus. Votre tâche consiste à renvoyer le nombre de personnes encore dans le bus après le dernier arrêt.

```
number([[10,0],[3,5],[5,8]]) → 5
number([[3,0],[9,1],[4,10],[12,2],[6,1],[7,10]]) → 17
number([[3,0],[9,1],[4,8],[12,2],[6,1],[7,8]]) → 21
```

#2 - Nombre de moutons (8 kyu)

Nous avons besoin d'une fonction qui compte le nombre de moutons présents dans un tableau donné (**true** signifie que le mouton est présent). Par exemple vous devrez trouver 17 moutons pour la liste suivante :

```
[true,  true,  true,  false,
true,  true,  true,  true ,
true,  false, true,  false,
true,  false, false, true ,
true,  true,  true,  true ,
false, false, true,  true]
```

N'oubliez pas de vérifier les mauvaises valeurs comme null / undefined

#3 - Enlever le premier et le dernier caractère d'une chaîne (8 kyu)

En entrée on vous donne une chaîne de caractères, en sortie vous devez avoir la même phrase sans le premier ni le dernier caractère.

```
removeChar("Ceci est une phrase") → "eci est une phras"
```

#4 - Doubler les lettres (8 kyu)

A partir d'une chaîne de caractères, renvoyez une chaîne dans laquelle chaque caractère (sensible à la casse) est répété une fois.

```
doubleChar("String") → "SSttrriinngg"
doubleChar("Hello World") → "HHeellllloo  WWoorrlldd"
doubleChar("1234!_ ") → "11223344!!__  "
```

#5 - Accumulation de lettres (7 kyu)

Ecrire une fonction **accum** telle que :

```
accum("abcd")    →  "A-Bb-Ccc-Dddd"
```

```
accum("RqaEzty") → "R-Qq-Aaa-Eeee-Zzzzz-Ttttttt-Yyyyyyy"
accum("cwAt") → "C-Ww-Aaa-Tttt"
```

Remarquez les majuscules au début de chacun des blocs, le nombre de lettres croissant et la séparation avec des tirets.

#6 - Code PIN (7 kyu)

Les distributeurs automatiques permettent de taper des codes PIN, ces derniers ne peuvent contenir que 4 chiffres ou exactement 6 chiffres. Si la fonction est transmise est un code PIN valide, renvoyez **true**, sinon renvoyez **false**.

```
ValidatePIN ("1234") → true
ValidatePIN ("12345") → false
ValidatePIN ("a234") → false
```

#7 - Mettre chaque 1ere lettre des mots en majuscule (7 kyu)

Ajoutez une méthode **toJadenCase** à **String.prototype** qui permettra de mettre en majuscules chaque première lettre des mots d'une phrase donnée.

```
"Ceci est une phrase".toJadenCase() → "Ceci Est Une Phrase"
```

#8 - Nombre de voyelles (7 kyu)

Renvoyez le nombre de voyelles (a, e, i, o, et u) dans une chaîne donnée.

```
getCount("Ceci est une phrase") → 7
```

#9 - Meeting (7 kyu)

Des développeurs se sont inscrits pour assister à la prochaine réunion de codage que vous organisez. Votre tâche consiste à renvoyer un objet qui comprend le nombre d'options alimentaires sélectionnées par les développeurs sur le formulaire d'inscription. Par exemple, compte tenu du tableau de saisie suivant :

```
Var list1 = [
  {FirstName: 'Noah', lastName: 'M.', pays: 'Suisse', continent: 'Europe', age: 19, langue: 'C'
    Repas: 'végétarien'},
  {FirstName: 'Anna', lastName: 'R.', pays: 'Liechtenstein', continent: 'Europe', age: 52, langue:
'JavaScript',
    Repas: 'standard'},
  {FirstName: 'Ramona', lastName: 'R.', pays: 'Paraguay', continent: 'Amériques', age: 29, langue: 'Ruby',
    Repas: 'vegan'},
  {FirstName: 'George', lastName: 'B.', pays: 'Angleterre', continent: 'Europe', age: 81, langue: 'C'
    Repas: 'végétarien'},
];
```

Votre fonction doit renvoyer l'objet suivant (l'ordre des propriétés n'a pas d'importance):

```
{Végétarien: 2, standard: 1, végétalien: 1}
```

#10 - Compteur (7 kyu)

Créez une fonction qui, à partir d'un nombre donné sous la forme d'une chaîne de caractères, donnera :

```
counterEffect("1250") → [[0,1],[0,1,2],[0,1,2,3,4,5],[0]]
counterEffect("0050") → [[0],[0],[0,1,2,3,4,5],[0]]
counterEffect("0000") → [[0],[0],[0],[0]]
```

#11 - Le plus long nombre (7 kyu)

A partir d'une liste de nombres, trouvez celui avec le plus de chiffres.

Si deux nombres dans le tableau ont le même nombre de chiffres, renvoyez le premier de la liste.

```
findLongest([1, 10, 100]) → 100
findLongest([9000, 8, 800]) → 9000
findLongest([8, 900, 500]) → 900
```

#12 - Le mot le plus court (7 kyu)

On vous donne une chaîne de mots, renvoyez la longueur du ou des mots les plus courts.

La chaîne ne sera jamais vide et vous ne devez pas tenir compte des types de données (nombres, lettres etc.).

```
findShort("bitcoin take over the world maybe who knows perhaps") → 3
```

#13 - RVB vers niveaux de gris (7 kyu)

Un tableau de taille N x M représente les pixels d'une image. Chaque cellule de ce tableau contient un tableau de taille 3 avec les informations de couleur du pixel : [R, G, B]

Convertissez l'image couleur, en une image moyenne en niveaux de gris.

Le tableau [R, G, B] contient des nombres entiers entre 0 et 255 pour chaque couleur.

Pour transformer un pixel de couleur en un pixel en niveaux de gris, utilisez la valeur moyenne des valeurs de ce pixel : P = [R, G, B] => [(R + G + B) / 3, (R + G + B) / 3, (R + G + B) / 3]

Remarque : les valeurs pour le pixel doivent être entières, donc trouvez l'entier le plus proche.

Exemple

Voici un exemple d'image 2x2:

```
[
 [[123, 231, 12], [56, 43, 124]],
 [[78, 152, 76], [64, 132, 200]]
]
```

Voici l'image attendue après transformation :

```
[
 [[122, 122, 122], [74, 74, 74]],
 [[102, 102, 102], [132, 132, 132]]
]
```

#14 - Carré des nombres (7 kyu)

On vous demande de mettre au carré chaque chiffre d'un nombre.

Par exemple, si nous exécutons la fonction avec 9119, nous obtiendrons 811181.

#15 - Carte crédit – 4 derniers chiffres (7 kyu)

Habituellement, lorsque vous achetez quelque chose, on vous demande si votre numéro de carte de crédit, votre numéro de téléphone ou votre réponse à votre question la plus secrète est toujours correct. Cependant, comme quelqu'un pourrait regarder votre épaule, vous ne voulez pas que cela s'affiche sur votre écran. Au lieu de cela, nous le masquerons.

Votre tâche est d'écrire une fonction **maskify**, qui modifie tous les caractères en '#' sauf les quatre derniers.

```
maskify('4556364607935616') → '############5616'
maskify('1') → '1'
maskify('11111') → '#1111'
```

#16 - Nombres les plus grands et plus petits d'une liste (7 kyu)

Vous recevez une chaîne de nombres séparés par des espaces et vous devez renvoyer le nombre le plus grand et le plus petit.

```
highAndLow("1 2 3 4 5") → "5 1"
highAndLow("1 2 -3 4 5") → "5 -3"
highAndLow("1 9 3 4 -5") → "9 -5"
```

#17 - ADN – Remplacer A par T, C par G et réciproquement (7 kyu)

L'acide désoxyribonucléique (ADN) est un produit chimique trouvé dans le noyau des cellules et porte les « instructions » pour le développement et le fonctionnement des organismes vivants.

Dans les codes ADN, les symboles "A" et "T" sont complémentaires l'un de l'autre, comme "C" et "G". Ecrire une fonction qui à partir d'une chaine ADN donne son complémentaire.

```
DNAStrand ("ATTGC") → "TAACG"
DNAStrand ("GTAT") → "CATA"
```

#18 - Nombres vampires (7 kyu)

Notre définition d'un nombre vampire peut être décrite comme suit :

6 * 21 = 126

Les chiffres 6, 1 et 2 sont présents dans le produit et le résultat, c'est un nombre vampire.

10 * 11 = 110

110 n'est pas un numéro vampire car il y a trois 1 dans le terme de gauche mais seulement deux 1 dans le produit

```
vampire_test (21,6) → true
vampire_test (204,615) → true  (204 * 615 = 125460)
vampire_test (30, -51) → true ( 30 * -51 = -1530)
vampire_test (-246, -510) → false  (-246 * -510 = 125460)
vampire_test (2947050,8469153) → true
```

#19 - Divisible par ? (7 kyu)

Créer une fonction qui vérifie si le premier argument **n** est divisible par tous les autres arguments.

```
IsDivisible (6,1,3) → true // car 6 est divisible par 1 et 3
IsDivisible (12,2) → true // car 12 est divisible par 2
IsDivisible (100,5,4,10,25,20) → true
IsDivisible (12,7) → false // parce que 12 n'est pas divisible par 7
```

#20 - Deviner un mot (7 kyu)

Un joueur doit deviner un mot dont il connait la longueur. Écrire une fonction qui, à partir du mot secret et de la proposition du joueur, retourne le nombre de lettres bien placées.

```
CountCorrectCharacters("dog", "car") → 0 (Aucune lettre)
CountCorrectCharacters("dog", "god") → 1 (Le "o" est bien placé)
CountCorrectCharacters("dog", "cog") → 2 ("o" et "g" bien placés)
CountCorrectCharacters("dog", "cod") → 1 ("o")
CountCorrectCharacters("dog", "bog") → 2 ("o" et "g")
CountCorrectCharacters("dog", "dog") → 3
```

Vous devrez vous s'assurer que le mot proposé a bien la même longueur que le mot secret, dans le cas contraire vous devrez générer une exception avec le texte « Mauvaise longueur ».

#21 - Filtrer une liste (7 kyu)

Créez une fonction qui prend une liste d'entiers ou de chaînes de caractères et renvoie une nouvelle liste en ayant filtré uniquement les nombres.

```
Filter_list ([1,2, 'a', 'b']) → [1,2]
Filter_list ([1, 'a', 'b', 0,15]) → [1,0,15]
Filter_list ([1,2, 'aasf', '3', '124', 123]) → [1,2,123]
```

#22 - Ordre des mots (6 kyu)

Votre tâche consiste à trier une chaîne donnée. Chaque mot de la chaîne contiendra un seul numéro. Ce nombre est la position que le mot devrait avoir dans le résultat final.

Remarque : Les nombres peuvent être de 1 à 9. Donc, 1 sera le premier mot (pas 0).

Si la chaîne d'entrée est vide, renvoyez une chaîne vide. Les mots dans la chaîne d'entrée ne contiennent que des nombres consécutifs valides.

```
order("is2 Thi1s T4est 3a") → "Thi1s is2 3a T4est"
order("4of Fo1r pe6ople g3ood th5e the2" → "Fo1r the2 g3ood 4of th5e pe6ople"
order("") → ""
```

#23 - Nombre de 1 en binaire (6 kyu)

Écrivez une fonction qui prend un entier (non signé) comme entrée et renvoie le nombre de bits qui sont égaux à un dans la représentation binaire de ce nombre.

Exemple : La représentation binaire de 1234 est 10011010010, donc la fonction devrait retourner 5 car il y a 5 « 1 ».

```
countBits(0) → 0
countBits(4) → 1
countBits(7) → 3
```

#24 - Distribution d'électrons (6 kyu)

Vous savez que l'idée fondamentale de la distribution d'électrons est qu'ils doivent remplir des couches jusqu'à ce qu'elles contiennent le nombre maximum d'électrons.

Règles :

Le nombre maximum d'électrons dans une couche est de $2n^2$ (n étant le niveau de la couche).

Par exemple, le nombre maximum d'électrons dans la 3^e couche est $2 * 3^2 = 18$.

Les électrons doivent d'abord remplir la couche de niveau le plus bas.

Si les électrons ont complètement rempli un niveau, les autres électrons inoccupés combleront le niveau supérieur et ainsi de suite.

```
atomicNumber (1) → [1]
atomicNumber (10) → [2, 8]
atomicNumber (11) → [2, 8, 1]
atomicNumber (47) → [2, 8, 18, 19]
```

#25 - Numéros de téléphone (6 kyu)

Écrivez une fonction qui accepte un tableau de 10 entiers (entre 0 et 9) et qui renvoie une chaîne sous la forme d'un numéro de téléphone.

```
CreatePhoneNumber ([1, 2, 3, 4, 5, 6, 7, 8, 9, 0]) → "(123) 456-7890"
```

#26 - Compression d'une phrase (6 kyu)

Supprimez les ponctuations, espaces et nombres d'une phrase donnée.

```
Hello World 2017 ! → HelloWorld
```

#27 - Mots consécutifs (6 kyu)

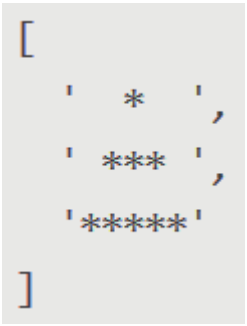
Vous recevez un tableau de chaînes de caractères et un nombre entier k. Votre tâche consiste à renvoyer la première chaîne la plus longue composée de k chaînes consécutives prises dans le tableau. Exemple :

```
longest_consec(["zone","abigail","theta","forme","libe","zas","theta","abigail"], 2) →
"abigailtheta"
```

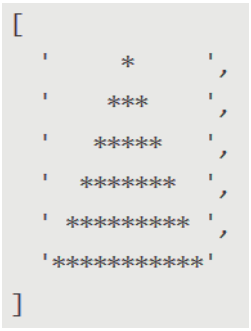
Soit n est la taille du tableau, si n = 0 ou k> n ou k <= 0 vous devez renvoyer "".

#28 - Construction d'une tour (6 kyu)

Construisez une tour à partir du nombre d'étages donné. Un bloc de la tour est représenté par *.



3 étages



6 étages

#29 - Contrariant (6 kyu)

L'enfant d'Alan peut parfois être ennuyant.

Quand Alan rentre chez lui et lui dit ce qu'il a accompli aujourd'hui, son enfant ne le croit jamais.

Comme entrée vous avez une phrase prononcée par Alan. La phrase contient la structure suivante :

```
"Today I " + [action_verb] + [object] + "."
(e.g.: "Today I played football.")
```

Votre fonction retournera la réponse de l'enfant d'Alan, avec la structure suivante :

```
"I don't think you " + [action_performed_by_alan] + " today, I think you " + ["did" OR
"didn't"] + [verb_of _action_in_present_tense] + [" it!" OR " at all!"]


(e.g.:"I don't think you played football today, I think you didn't play at all!")
```

D'autres exemples :

```
input  = "Today I played football."
output = "I don't think you played football today, I think you didn't play at all!"

input  = "Today I didn't attempt to hardcode this Kata."
output = "I don't think you didn't attempt to hardcode this Kata today, I think you did
attempt it!"

input  = "Today I didn't play football."
output = "I don't think you didn't play football today, I think you did play it!"

input  = "Today I cleaned the kitchen."
output = "I don't think you cleaned the kitchen today, I think you didn't clean at all!"
```

#30 - Parité (6 kyu)

Trouvez, parmi une liste de nombre, le seul qui n'a pas la même parité.

```
iqTest("2 4 7 8 10") → 3 // Le 3e est impair, les autres sont pairs
iqTest("1 2 1 1") → 2 // Le 2e est pair, les autres sont impairs
```

#31 - Distribution de bonbons (6 kyu)

À la maternelle, l'instituteur a donné des bonbons aux enfants. Le nombre de bonbons que chaque enfant reçoit n'est pas toujours le même. Voici par exemple le nombre de bonbons par enfant :

```
candies = [10,2,8,22,16,4,10,6,14,20]
```

L'instituteur a demandé aux enfants de faire un cercle et de jouer à un jeu : chaque enfant donne la moitié de ses bonbons à l'enfant sur sa droite (en même temps). Si le nombre de bonbons est un nombre impair, l'enseignant lui donne un bonbon supplémentaire.
On répète cette procédure jusqu'à ce que les enfants aient le même nombre de bonbons.

Vous devez renvoyer deux nombres :
1. Combien de temps va durer la distribution
2. Combien chaque enfant a-t-il de bonbons

```
candies = [ 1,2,3,4,5 ]
Distribution 1: [ 4,2,3,4,5 ]
Distribution 2: [ 5,3,3,4,5 ]
Distribution 3: [ 6,5,4,4,5 ]
Distribution 4: [ 6,6,5,4,5 ]
Distribution 5: [ 6,6,6,5,5 ]
Distribution 6: [ 6,6,6,6,6 ]

distributionOfCandy([1,2,3,4,5]) → [6,6]
```

#32 - 10 minutes de promenade (6 kyu)

Vous habitez dans une ville où toutes les routes sont disposées dans une grille parfaite. Vous êtes arrivé dix minutes trop tôt pour un rendez-vous, alors vous avez décidé de profiter de l'occasion pour faire une courte promenade. La ville fournit à ses citoyens une application sur leurs téléphones - chaque fois que vous appuyez sur le bouton, il vous envoie un ensemble de directions (par exemple ['n', 's', 'w' 'e']). Vous savez que cela vous prend une minute pour traverser un bloc de la ville, alors on vous demande de créer une fonction qui retournera **true** si la marche que vous propose l'application vous prendra exactement dix minutes et, bien sûr, vous ramène à votre point de départ. Retournez **false** sinon.

```
isValidWalk(['n','s','n','s','n','s','n','s','n','s']) → true
isValidWalk(['w','e','w','e','w','e','w','e','w','e','w','e']) → false
isValidWalk(['w']) → false
isValidWalk(['n','n','n','s','n','s','n','s','n','s']) → false
```

#33 - Likes (6 kyu)

Vous connaissez probablement le système "like" de Facebook et d'autres pages. Les gens peuvent "aimer" des articles de blog, des images ou d'autres articles. Nous voulons créer le texte qui doit être affiché à côté d'un tel élément.
Créez une fonction **like**, qui doit prendre en entrée un tableau contenant les noms des personnes qui aiment un élément. Il doit retourner le texte d'affichage comme indiqué dans les exemples ci-dessous :

```
likes [] → "no one likes this"
likes ["Peter"] → "Peter likes this"
likes ["Jacob", "Alex"] → "Jacob and Alex like this"
likes ["Max", "John", "Mark"] → "Max, John and Mark like this"
```

likes ["Alex", "Jacob", "Mark", "Max"] → "Alex, Jacob and 2 others like this"

#34 - Somme gauche = Somme droite (6 kyu)

Vous allez avoir une liste d'entiers en entrée. Votre travail est de prendre ce tableau et de trouver un indice N où la somme des nombres entiers à gauche de N est égale à la somme des entiers à droite de N. S'il n'y a pas d'index qui le ferait, retournez -1.

Par exemple :

Disons qu'on vous donne le tableau {1,2,3,4,3,2,1} :

Votre fonction renverra l'indice 3, car à la 3ème position du tableau, la somme du côté gauche de l'index ({1,2,3}) et la somme du côté droit de l'index ({3,2, 1}) tous deux égaux 6.

Regardons un autre exemple : Vous recevez le tableau {1,100,50, -51,1,1} :

Votre fonction renverra l'indice 1, car à la position 1 du tableau, la somme du côté gauche de l'index ({1}) et la somme du côté droit de l'index ({50, -51,1,1}) sont toutes les 2 égales à 1.

findEvenIndex([1,2,3,4,3,2,1]) → 3
findEvenIndex([1,100,50,-51,1,1]) → 1
findEvenIndex([1,2,3,4,5,6]) → -1
findEvenIndex([20,10,30,10,10,15,35]) → 3

#35 - Nombre divisible par 6 (6 kyu)

Une chaîne est composée de chiffres et d'un astérisque (*) qui doit être remplacé par un chiffre exactement. Trouvez toutes les options possibles pour remplacer l'astérisque pour que le résultat soit un entier divisible par 6.

"1*0" → ["120", "150", "180"].
"*1" → []
"123456789012345678901234567*0" →
["1234567890123456789012345678 00",
"1234567890123456789012345678 30",
"1234567890123456789012345678 60"]

#36 - Nombres narcissiques (6 kyu)

Un nombre narcissique est un nombre qui est la somme de ses propres chiffres, chacun élevé à la puissance du nombre de chiffres.

Par exemple, 153 (3 chiffres) : $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

Et 1634 (4 chiffres) : $1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634$

Votre code doit retourner **true** ou **false** selon que le nombre donné est narcissique ou non.

narcissistic(153) → true

#37 - Décodage Morse (6 kyu)

Vous devez écrire un décodeur simple de code Morse. Le code Morse code pour chaque lettre est une séquence de "points" et "tirets". Par exemple, la lettre A est codée comme · -, la lettre Q est codée comme - · -, et le chiffre 1 est codé comme · ---. Le code Morse est insensible à la casse, traditionnellement, les

majuscules sont utilisées. Lorsque le message est écrit en code Morse, un seul espace est utilisé pour séparer les codes de caractères et 3 espaces sont utilisés pour séparer les mots. Par exemple, le message HEY JUDE dans le code Morse est ···· · -·-· ·-- ·· -·· ·.

REMARQUE : les espaces supplémentaires avant ou après le code n'ont aucun inétrêt et doivent être ignorés.

En plus des lettres, des chiffres et de la ponctuation, il existe des codes de service spéciaux, dont le plus célèbre est le signal de détresse international SOS, qui est codé comme ··· --- ···. Ces codes spéciaux sont traités comme des caractères spéciaux uniques et sont généralement transmis sous forme de mots distincts.

Votre tâche consiste à implémenter une fonction **decodeMorse**, qui prend le code morse en tant qu'entrée et renvoi une chaîne décodée lisible par l'homme.

DecodeMorse ('.... ..-.-- .--- ..- - ...') → "HEY JUDE"

La table des codes Morse est préchargée en tant que dictionnaire, n'hésitez pas à l'utiliser.

```
MORSE_CODE = { '-.-.--': '!', '.-..-.': '"',  '...-..-': '$', '.-...': '&', '.----.':
'', '-.--.': '(', '-.--.-': ')', '.-.-.': '+', '--..--': ',', '-....-': '-', '.-.-.-':
'.', '-..-.': '/', '-----': '0', '.----': '1', '..---': '2', '...--': '3', '....-': '4',
  '.....': '5', '-....': '6', '--...': '7', '---..': '8', '----.': '9', '---...': ':',
'-.-.-.': ';', '-...-': '=', '..--..': '?', '.--.-.': '@', '.-': 'A', '-...': 'B', '-.-
.': 'C', '-..': 'D', '.': 'E', '..-.': 'F', '--.': 'G', '....': 'H', '..': 'I', '.---':
'J', '-.-': 'K', '.-..': 'L', '--': 'M', '-.': 'N', '---': 'O', '.--.': 'P', '--.-':
'Q', '.-.': 'R', '...': 'S', '-': 'T', '..-': 'U', '...-': 'V', '.--': 'W', '-..-': 'X',
'-.--': 'Y', '--..': 'Z', '..--.-': '_', '...---...': 'SOS' }
```

MORSE_CODE['.'] → "E"

#38 - Position des lettres dans l'alphabet (6 kyu)

Vous devez, à partir d'une chaîne, remplacer chaque lettre par sa position dans l'alphabet. Si quelque chose dans le texte n'est pas une lettre, ignorez-la et ne la renvoyez pas. On considère que « a » vaut 1, « b » vaut 2, etc. Par exemple :

alphabet_position("The sunset sets at twelve o' clock.") →
"20 8 5 19 21 14 19 5 20 19 5 20 19 1 20 20 23 5 12 22 5 15 3 12 15 3 11"

#39 - Nombre apparaissant un nombre impair de fois (6 kyu)

À partir d'un tableau, trouvez le nombre qui apparaît un nombre impair de fois.

Il n'y aura toujours qu'un seul entier qui apparaît un nombre impair de fois.

findOdd([20,1,-1,2,-2,3,3,5,5,1,2,4,20,4,-1,-2,5]) → 5
findOdd([1,1,2,-2,5,2,4,4,-1,-2,5]) → -1
findOdd([20,1,1,2,2,3,3,5,5,4,20,4,5]) → 5
findOdd([10]) → 10

#40 - Cryptage et décryptage d'une chaîne (6 kyu)

Partie cryptage : A partir d'une chaîne, prenez un caractère sur deux, les concaténer puis concaténer tous les autres. Répétez cela le nombre de fois précisé en paramètre.

encrypt("This is a test!", 0) → "This is a test!"

encrypt("This is a test!", 1) → "hsi etTi sats!"

encrypt("This is a test!", 2) → "s eT ashi tist!"

Partie décryptage : A partir d'une chaîne codée **n** fois, retrouvez le texte d'origine.

decrypt(" Tah itse sits!", 3) → "This is a test!"
decrypt("hskt svr neetn!Ti aai eyitrsig", 1) → "This kata is very interesting!"

#41 - Encodeur de lettres dupliquées (6 kyu)

Le but de cet exercice est de convertir une chaîne en une nouvelle chaîne où chaque caractère est '(' si ce caractère apparaît une seule fois dans la chaîne d'origine, ou ')' si ce caractère apparaissait plusieurs fois. On ne tiendra pas compte de la casse des lettres.

"din" → "((("
"recede" → "()()()"
"Success" → ")())())"
"((@" → "))(("

#42 - File d'attente cinéma (6 kyu)

Le nouveau film "Avengers" vient de sortir ! Il y a beaucoup de gens qui attendent en file à l'entrée du cinéma. Chacun d'eux a un unique billet de 100, 50 ou 25 dollars. Un billet "Avengers" coûte 25 dollars.
Vous travaillez actuellement comme commis et voulez vendre un billet à chaque personne dans cette ligne.
Pouvez-vous vendre un billet à chaque personne et donner le change si vous n'avez initialement pas d'argent et en vendant les tickets strictement dans l'ordre que les gens suivent dans la file ?
Retourner YES, si vous pouvez vendre un billet à chaque personne et donner le change. Sinon, renvoyez NO.

tickets([25, 25, 50, 50]) → "YES" // Vous récupérez 25+25 $, rendez 25$ et encore 25$
tickets([25, 100]) → "NO" // Vous récupérez 25$ mais impossible de rembourser n°2

#43 - Différence entre ensembles (6 kyu)

Garder les éléments d'un tableau **a** qui ne sont pas dans un tableau **b**.

difference([1,2],[1]) → [2]
difference([1,2,2,2,3],[2]) → [1,3]

#44 - Superposition d'intervalles (6 kyu)

On vous donne les extrémités (début et fin) d'intervalles fermés et on vous demande combien vont se superposer.

start = [8, 4, 6, 1] et end = [10, 9, 7, 2] → 2

Il y a en effet 2 intervalles qui se superposent : [4, 9] avec [6, 7] et [4, 9] avec [8,10].

start = [1, 2] et end = [3, 4] → 1
start = [1, 2] et end = [2, 4] → 1

#45 - Retournement des mots de 5 lettres et plus (6 kyu)

Écrivez une fonction qui prend une chaîne d'un ou plusieurs mots, et renvoie la même chaîne, mais avec les mots de 5 lettres ou plus inversés. Les chaînes en entrée ne comporteront que des lettres et des espaces. Les espaces seront inclus uniquement lorsque plus d'un mot est présent.

spinWords("Hey fellow warriors") → "Hey wollef sroirraw"
spinWords("This is a test") → "This is a test"
spinWords("This is another test") → "This is rehtona test"

#46 - Somme des chiffres (6 kyu)

Calculez la somme récursive de tous les chiffres d'un nombre. A partir d'un entier **n**, prenez la somme des chiffres de n. Si cette valeur a deux chiffres, continuez de réduire de cette façon jusqu'à ce qu'un nombre à un seul chiffre.

digital_root(16) → 7
digital_root(942) → 6

#47 - Lettre manquante (6 kyu)

Écrivez une fonction qui prend une série de lettres consécutives (en augmentation) comme entrée et qui renvoie la lettre manquante en sortie.
On suppose que l'on a toujours un tableau valide en entrée et exactement une lettre manquante. La longueur du tableau sera d'au moins 2.

findMissingLetter(['a','b','c','d','f']) → 'e'
findMissingLetter(['O','Q','R','S']) → 'P'

#48 - Tribonnacci (6 kyu)

Comme le nom l'indique déjà, cela fonctionne essentiellement comme les nombres de Fibonacci, mais en additionnant les 3 derniers (au lieu de 2) nombres de la suite pour générer le prochain.
Donc, si nous devons commencer notre séquence Tribonacci avec [1,1,1] (signature AKA), nous aurons cette séquence : [1,1,1,3,5,9,17,31, ...]
Mais que se passe-t-il si nous commençons avec [0,0,1] comme signature ? [0,0,1,1,2,4,7,13,24, ...]
Vous devez créer une fonction de **tribonacci** qui à partir d'un tableau et d'une signature, renvoie les n premiers éléments (signature incluse) de la séquence.

La signature contiendra toujours 3 numéros ; n sera toujours un nombre positif ; Si n == 0, renvoyez un tableau vide.

```
tribonacci([1,1,1],10) → [1,1,1,3,5,9,17,31,57,105]
tribonacci([0,0,1],10) → [0,0,1,1,2,4,7,13,24,44]
tribonacci([0,1,1],10) → [0,1,1,2,4,7,13,24,44,81]
tribonacci([1,0,0],10) → [1,0,0,1,1,2,4,7,13,24]
```

#49 - Smileys (6 kyu)

Étant donné un tableau (**arr**) comme argument, complétez la fonction **countSmileys** qui devrait renvoyer le nombre total de visages souriants.

Règles pour un visage souriant :

Chaque visage souriant doit contenir une paire d'yeux valide. Les yeux peuvent être marqués comme suit **:** ou **;**

- Un visage souriant peut avoir un nez mais pas obligatoirement. Les caractères valides pour un nez sont **-** ou **~**

- Tout visage souriant doit avoir une bouche souriante qui doit être marquée avec soit **)** soit **D**.

Exemples valides de visage souriants : **:) :D ;-D :~)**

Visages smiley invalides : **;(:> :} :]**

```
countSmileys([':)', ';(', ';}', ':-D'])       → 2
countSmileys([';D', ':-(', ':-)', ';~)'])     → 3
countSmileys([';]', ':[', ';*', ':$', ';-D']) → 1
```

#50 - Lièvre et tortue (6 kyu)

Deux tortues appelées A et B doivent faire une course. A commence avec une vitesse moyenne de 720 pieds par heure. B sait qu'elle court plus vite et de plus il n'a pas fini son chou.

Quand B commence, elle peut voir que A a une avance de 70 pieds, mais la vitesse B est de 850 pieds par heure. Combien de temps prendra-t-il B pour attraper A ?

Plus généralement : deux vitesses **v1** (vitesse de A, nombre entier> 0) et **v2** (vitesse de B, nombre entier> 0) et un écart **g** (nombre entier> 0) combien de temps prend-t-il B pour attraper A ?

Le résultat sera un tableau [h, mn, s] où h, mn, s est le temps nécessaire en heures, minutes et secondes Si v1> = v2, renvoyez **null**.

#51 - Couleurs HTML vers RGB (6 kyu)

Une chaîne de saisie représente l'une des options suivantes :

Hexadécimal à 6 chiffres : "#RRGGBB" comme "# 012345", "# 789abc", "# FFA077"

Chaque paire de chiffres représente une valeur du canal en hexadécimal: 00 à FF

Hexadécimal à 3 chiffres : "#RGB" comme "# 012", "#aaa", "# F5A"

Chaque chiffre représente une valeur de 0 à F qui se traduit 2 chiffres: 0 → 00, 1 → 11, etc.

Nom de couleur prédéfini, par exemple. "Red", "Blue", "LimeGreen" (utilisez PRESET_COLORS).

```
ParseHTMLColor ('# 80FFA0') → {r: 128, g: 255, b: 160}
ParseHTMLColor ('# 3B7') → {r: 51, g: 187, b: 119}
ParseHTMLColor ('LimeGreen') → {r: 50, g: 205, b: 50}
```

#52 - Où sont mes parents ? (6 kyu)

Les mères ont organisé une soirée de danse pour les enfants à l'école. Dans cette fête, il n'y a que des mères et leurs enfants. Tous s'amusent bien quand soudainement toutes les lumières s'éteignent. C'est la nuit sombre et personne ne peut voir. Mais en passant à proximité d'une personne vous pouvez la voir dans le noir et la téléporter où vous voulez.

Légende :

- Les lettres majuscules signifient des mères, des minuscules pour leurs enfants. Par exemple les enfants de la mère "A" sont "aaaa".

Entrée de la fonction : la chaîne contient seulement des lettres, les lettres majuscules sont uniques.

Tâche :

Placez toutes les personnes dans l'ordre alphabétique où les mères sont suivies par leurs enfants.

```
findChildren("aAbaBb") → "AaaBbb"
findChildren("beeeEBb")); → "BbbEeee"
findChildren("uwwWUeEe") → "EeeUuuWww"
```

#53 - Somme de nombres (6 kyu)

A partir d'un tableau **a** d'entiers, construisez un tableau **b** de la même longueur tel que

b [0] = a [0] + a [1] + ... + a [n - 2] + a [n - 1]

b [1] = a [1] + ... + a [n - 2] + a [n - 1]

...

b [n - 2] = a [n - 2] + a [n - 1]

b [n - 1] = a [n - 1]

Où n est la longueur de **a**.

```
suffixSums ([1, 2, 3]) → [6, 5, 3]
suffixSums ([1, 2, 3, -6]) → [0, -1, -3, -6]
suffixSums ([0, 0, 0]) → [0, 0, 0]
suffixSums ([1, 123, 23]) → [147, 146, 23]
```

#54 - Majuscules et minuscules à chaque mot (6 kyu)

Écrivez une fonction **ToWeirdCase** qui accepte une chaîne et renvoie la même chaîne avec tous les caractères aux positions paires en majuscules et tous les caractères aux positions impaires en minuscules. Le premier caractère doit donc être en majuscule.

La chaîne passée ne comportera que des caractères alphabétiques et des espaces (''). Les espaces ne seront présents que s'il existe plusieurs mots. Les mots seront séparés par un seul espace ('').

```
ToWeirdCase ("String") → "StRiNg"
ToWeirdCase ("Weird string case") → "WeIrD StRiNg CaSe"
```

#55 - Aire d'un triangle (6 kyu)

```
triangleArea(new Triangle(new Point(10, 10), new Point(40, 10), new Point(10, 50))) → 600
triangleArea(new Triangle(new Point(15, -10), new Point(40, 20), new Point(20, 50))) → 675
```

#56 - Combien d'abeilles sont dans la ruche ? (6 kyu)

Les abeilles (bee) peuvent être orientées vers le HAUT, BAS, GAUCHE ou DROIT

Les abeilles peuvent partager des parties d'autres abeilles (beeb correspond à 2 abeilles)

```
bee.bee
.e..e..
.b..eeb
```

Réponse : 5

```
bee.bee
e.e.e.e
eeb.eeb
```

Réponse : 8

```
eeeb.e..b
ee..ebe.b
eeee..bb.
.ee..beeb
e..beeb.e
b.eee..be
.e.b.eeeb
..ebee.be
beeebe...
```

Réponse : 11

La ruche peut être vide ou **null**.

#57 - Parcours d'un labyrinthe (6 kyu)

Vous avez le plan 2D d'un labyrinthe et un ensemble de directions. Votre tâche est de suivre les instructions données. Si vous atteignez le point final avant que tous vos mouvements ne se soient écoulés, vous devez retourner **Finish**. Si vous tapez dans les murs ou dépassez le bord du labyrinthe, vous devez retourner **Dead**. Si vous vous trouvez encore dans le labyrinthe après avoir fait tous les mouvements, vous devez retourner **Lost**. Le labyrinthe **maze** ressemble à :

```
maze = [[1,1,1,1,1,1,1],
        [1,0,0,0,0,0,3],
        [1,0,1,0,1,0,1],
        [0,0,1,0,0,0,1],
        [1,0,1,0,1,0,1],
        [1,0,0,0,0,0,1],
        [1,2,1,0,1,0,1]]
```

```
0 = Chemin
1 = Mur
2 = Départ
3 = Arrivée
```

Exemples :

```
mazeRunner(maze,["N","N","N","N","N","E","E","E","E","E"]) → "Finish"
mazeRunner(maze,["N","N","N","N","N","E","E","S","S","E","E","N","N","E"]) → "Finish"
mazeRunner(maze,["N","N","N","W","W"]) →"Dead"
mazeRunner(maze,["N","E","E","E","E"]) → "Lost"
```

#58 - Chez le père Noël (6 kyu)

La correspondance du Père Noël est triée et organisée par des elfes qui sont de bons travailleurs mais qui aiment faire des farces.

Leur dernière farce est de prendre le nom du cadeau qu'un enfant veut recevoir, choisir trois lettres consécutives à l'intérieur et mélanger leur ordre (c'est-à-dire qu'au moins une lettre a changé de place).

On peut donc parfois se retrouver avec le même mot, par exemple, pour le mot « doll », en prenant les lettres ' o ',' l ' et ' l ' avec le mélange : ' o 'reste à la même position et les 2 ' l ' sont échangés.

Écrivez une fonction qui calcule le nombre de triplets formés par des lettres consécutives qui peuvent rester identiques après brassage.

```
gift = "doll" → 1       // « oll » peut rester « oll » après brassage des 3 lettres
gift = "aaaaaaa" → 5    // Les 5 triplets « aaa » resteront inchangés après brassage
gift = "cat" → 0
```

#59 - Heures à partir de secondes (5 kyu)

Écrivez une fonction, qui prend un entier positif (secondes) en entrée et renvoie l'heure dans un format lisible par l'homme (HH:MM:SS)

Le temps maximum n'excède jamais 359999 (99:59:59)

```
humanReadable(0) → '00:00:00'
humanReadable(5) → '00:00:05'
humanReadable(60) → '00:01:00'
humanReadable(86399) → '23:59:59'
humanReadable(359999) → '99:59:59'
```

#60 - Hashtags (5 kyu)

L'équipe du marketing passe trop de temps à taper les hashtags. Allons les aider avec la création d'un générateur ! Si le résultat final est supérieur à 140 caractères, il doit renvoyer **false**.

- Si l'entrée est une chaîne vide, elle doit renvoyer **false**.
- Il doit commencer par un hashtag (#).
- Tous les mots doivent avoir leur première lettre en majuscule.

```
" Hello there thanks for trying my Kata" → "#HelloThereThanksForTryingMyKata"
"Hello World" => "#HelloWorld"
```

#61 - Pig Latin (5 kyu)

Déplacez la première lettre de chaque mot à la fin de celle-ci, puis ajoute «ay» à la fin du mot.

```
'Pig latin is cool' → igPay atinlay siay oolcay
```

#62 - Camel Case (5 kyu)

Écrire une fonction permettant de convertir des mots délimités par les symboles – ou _ en **Camel Case**. Le premier mot dans la sortie ne doit être mis en majuscule que si le mot d'origine était en majuscule.

```
toCamelCase('') → ''
```

```
toCamelCase("the_stealth_warrior") → "theStealthWarrior"
toCamelCase("The-Stealth-Warrior") → "TheStealthWarrior"
toCamelCase("A-B-C") → "ABC"
```

#63 - Trop c'est trop

Alice et Bob étaient en vacances. Les deux ont pris beaucoup de photos des endroits où ils se trouvaient, et maintenant ils veulent montrer à Charlie toute leur collection. Cependant, Charlie n'aime voir 40 fois une même photo. Pouvez-vous aider Alice et Bob à supprimer des nombres pour que la liste contienne chaque numéro seulement jusqu'à N fois, sans modifier l'ordre ?

```
[20, 37, 20, 21] et 1 → [20, 37, 21]
[1, 1, 3, 3, 7, 2, 2, 2, 2] et 3 → [1, 1, 3, 3, 7, 2, 2, 2]
[1, 2, 3, 1, 1, 2, 1, 2, 3, 3, 2, 4, 5, 3, 1] et 3 → [1, 2, 3, 1, 1, 2, 2, 3, 3, 4, 5]
```

#64 - Départ – Arrivée (5 kyu) A METTRE EN FORME

Vous remarquez que chaque caractère de clapet est sur une sorte de rotor et l'ordre des caractères sur chaque rotor est :

ABCDEFGHIJKLMNOPQRSTUVWXYZ?! @ # & () | <>.: = - + * / 0123456789

À partir de la gauche, tous les rotors (de l'actuel à la fin de la ligne) se voient ensemble jusqu'à ce que le caractère du rotor le plus à gauche soit correct.

Ensuite, le mécanisme avance par 1 rotor à droite ...

... RÉPÉTEZ cette procédure de rotor jusqu'à ce que toute la ligne soit mise à jour
... REPRISE cette procédure de ligne de haut en bas jusqu'à ce que l'affichage complet soit mis à jour

Exemple

Considérons un écran à rabat avec 3 rotors et une ligne 1 qui épelle actuellement CAT

Étape 1 (le rotor actuel est 1)
Flap x 1
Maintenant ligne dit DBU
Étape 2 (le rotor actuel est 2)
Flap x 13
Maintenant ligne dit DO)

Étape 3 (le rotor actuel est 3)

Flap x 27

Maintenant ligne dit DOG

Lignes // tableau de chaînes. Chaque chaîne est une ligne d'affichage de la configuration initiale

Rotors // tableau de valeurs de tableau de rotor. Chaque gamme de valeurs de rotor est appliquée à la ligne d'affichage correspondante

Résultat // tableau de chaînes. Chaque chaîne est une ligne d'affichage de la configuration finale

Before:

```
CAT
```

After:

```
DOG
```

Before:

```
+-------------------------+
| THIS IS A MULTI LINE TEST |
+-------------------------+
```

After:

```
***************************
*   DO YOU LIKE THIS KATA?  *
***************************
```

#65 - Fibonnaci (5 kyu)

Les nombres Fibonacci sont les nombres dans la séquence entière suivante (Fn) :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

tels que

F (n) = F (n-1) + F (n-2) avec F (0) = 0 et F (1) = 1.

Compte tenu d'un nombre, disons **prod** (pour le produit), nous recherchons deux nombres Fibonacci F (n) et F (n + 1) vérifiant F (n) * F (n + 1) = prod.

Votre fonction **productFib** prend un nombre entier (**prod**) et renvoie un tableau : [F (n), F (n + 1), true]

Si vous ne trouvez pas deux F (m) consécutifs en vérifiant F (m) * F (m + 1) = prod, renvoyez :

[F (m), F (m + 1), false]

Où F (m) est le plus petit tel que F (m) * F (m + 1)> prod.

productFib(714) → [21, 34, true]

productFib(800) → [34, 55, false]

#66 - Déplacement sur une carte (5 kyu)

Une personne reçoit des instructions pour aller d'un point à l'autre. Les indications sont «NORTH», «SOUTH», «WEST», «EAST». De toute évidence, "NORTH" et "SOUTH" sont opposés, "WEST" et "EAST" aussi. Aller dans une direction et revenir en sens inverse est un effort inutile. Comme il s'agit de l'ouest sauvage, avec un temps terrible et pas beaucoup d'eau, il est important de vous épargner de l'énergie, sinon vous pourriez mourir de soif ! L'idée est de traverser le désert de manière intelligente.

Les instructions données à l'homme sont, par exemple, les suivantes :

["NORD", "SUD", "SUD", "EST", "OUEST", "NORD", "OUEST"].

Vous pouvez immédiatement voir que «NORTH» et ensuite «SOUTH» s'annulent, mieux vaut rester au même endroit ! Donc, la tâche est de donner à l'homme une version simplifiée du plan. Un meilleur plan dans ce cas est tout simplement : ["OUEST"]

Autres exemples:

Dans ["NORTH", "SOUTH", "EAST", "WEST"], la direction "NORTH" + "SOUTH", mieux vaut ne rien faire.

Dans ["NORTH", "EAST", "WEST", "SOUTH", "WEST", "WEST"], "NORTH" et "SOUTH" ne sont pas directement opposés, mais ils deviennent directement opposés après la réduction de "EAST" Et "WEST" de sorte que tout le chemin est réductible à ["WEST", "WEST"].

Écrivez une fonction **dirReduc** qui prendra un ensemble de chaînes et renvoie un ensemble de chaînes avec les directions inutiles supprimées (W <-> E ou S <-> N côte à côte).

```
dirReduc(["NORTH", "SOUTH", "SOUTH", "EAST", "WEST", "NORTH", "WEST"] → ["WEST"]
dirReduc(["NORTH", "WEST", "SOUTH", "EAST"]) → ["NORTH", "WEST", "SOUTH", "EAST"]
dirReduc(["NORTH", "SOUTH", "EAST", "WEST", "EAST", "WEST"]) → []
```

#67 - Somme maxi dans un tableau (5 kyu)

Le problème consiste à trouver la somme maximale d'une sous-séquence contiguë dans un tableau ou une liste d'entiers :

MaxSéquence ([- 2, 1, -3, 4, -1, 2, 1, -5, 4])

devrait être 6: [4, -1, 2, 1]

Le cas simple est lorsque la liste est constituée uniquement de nombres positifs et la somme maximale est la somme de l'ensemble du tableau. Si la liste n'est constituée que de nombres négatifs, renvoyez 0 à la place.

La liste vide devra renvoyer 0.

#68 - Anagrammes (5 kyu)

Qu'est-ce qu'une anagramme ? Eh bien, deux mots sont des anagrammes les uns des autres si ils contiennent tous deux les mêmes lettres. Par exemple :

'Abba' et 'baab' == vrai

'Abba' et 'bbaa' == vrai

'Abba' et 'abbba' == faux

Écrivez une fonction qui trouvera tous les anagrammes d'un mot d'une liste. Vous recevrez deux entrées par mot et un tableau avec des mots. Vous devriez renvoyer un tableau de tous les anagrammes ou un tableau vide s'il n'y en a pas. Par exemple :

```
Anagrammes ('abba', ['aabb', 'abcd', 'bbaa', 'dada']) → ['aabb', 'bbaa']
Anagrams ('racer', ['crazer', 'carer', 'racar', 'caers', 'racer']) → ['carer', 'racer']
Anagrammes ('laser', ['déprimé', 'paresseux', 'lacer']) → []
```

#69 - Trous entre des nombres premiers (5 kyu)

Les nombres premiers ne sont pas régulièrement espacés. Par exemple de 2 à 3, l'écart est égal à 1. De 3 à 5, l'écart est égal à 2. De 7 à 11, il est 4. Entre 2 et 50, nous avons les paires suivantes de premiers : 3-5, 5-7, 11-13, 17-19, 29-31, 41-43

G (entier> = 2) qui indique l'écart que nous recherchons

M (entier> 2) qui donne le début de la recherche (m inclus)

N (entier> = m) qui donne la fin de la recherche (n inclus)

Dans l'exemple ci-dessus, l'intervalle (2, 3, 50) renverra [3, 5] qui est la première paire entre 3 et 50 avec un écart 2.

Donc, cette fonction devrait renvoyer la première paire de deux nombres premiers espacés d'un écart de **G** entre les **M** et **N**. Si ces nombres n'existent pas, renvoyez **null**.

Intervalle (4, 130, 200) -> [163, 167]

#70 - Somme carrés diviseurs = carré ? (5 kyu)

Les diviseurs de 42 sont : 1, 2, 3, 6, 7, 14, 21, 42. Ces diviseurs au carré sont : 1, 4, 9, 36, 49, 196, 441, 1764. La somme des diviseurs carrés est de 2500 qui est 50 * 50, un carré !

A partir de deux entiers m, n (1 <= m <= n), nous voulons trouver tous les nombres entiers entre m et n dont la somme des diviseurs au carré est elle-même un carré. 42 est un tel nombre.

Le résultat sera un tableau de tableaux, chaque élément ayant deux éléments, d'abord le nombre dont les diviseurs au carré sont un carré puis la somme des diviseurs au carré.

```
List_squared (1, 250) → [[1, 1], [42, 2500], [246, 84100]]
List_squared (42, 250) → [[42, 2500], [246, 84100]]
```

#71 - PowerSet (5 kyu)

Compte tenu d'un ensemble **nums** de nombres entiers, votre tâche est de renvoyer l'ensemble de tous les sous-ensembles possibles de **nums**.

Pour chaque nombre entier, nous pouvons choisir de le prendre ou de ne pas le prendre. L'idée ici et dans un premier temps de ne pas le prendre, puis ensuite de le prendre. Exemple :

Pour nums = [1, 2], la sortie devra être [[], [2], [1], [1, 2]], en effet :

```
Ne pas prendre l'élément 1
```

```
---- ne pas prendre l'élément 2
-------- → ajouter []
---- prendre l'élément 2
-------- → ajouter [2]
Prendre l'élément 1
---- ne pas prendre l'élément 2
-------- → ajouter [1]
---- prendre l'élément 2
-------- → ajouter [1, 2]
```

Pour `nums = [1, 2, 3]`, la sortie devrait être :

`[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]`

#72 - Parenthèses valides (5 kyu)

Écrivez une fonction appelée **validParentheses** qui prend une chaîne de parenthèses et détermine si l'ordre des parenthèses est valide. **ValidParentheses** doit renvoyer **true** si la chaîne est valide et false si elle n'est pas valide. Exemples :

```
ValidParentheses ("()") → true
ValidParentheses (") ((()))") → faux
ValidParentheses ("(") → faux
ValidParentheses ("(()) ((() () ())") → true
```

Toutes les chaînes seront non vides et ne comporteront que des parenthèses ouvertes '(' et/ou des parenthèses fermées ')'.

#73 - Nombres binaires négatifs (5 kyu)

Dans les systèmes de base négative, les nombres positifs et négatifs sont représentés sans l'utilisation d'un signe moins (ou, dans la représentation par ordinateur, un bit de signe).
Pour aider à comprendre le principe, les huit premiers chiffres (en décimale) du système Base (-2) sont:
[1, -2, 4, -8, 16, -32, 64, -128]
Exemple de conversion :
Décimal, négabinaire
6, '11010'
-6, '1110'
4, '100'
18, '10110'
-11, '110101'

#74 - Meilleur score du perdant (5 kyu)

"AL-AHLY" et "Zamalek" sont les meilleures équipes en Egypte, mais "AL-AHLY" gagne toujours les matches. Les responsables de "Zamalek" veulent savoir quel est le meilleur match qu'ils ont joué jusqu'ici.

Le meilleur match est le match qu'ils ont perdu avec la différence minimale de buts. S'il y a plus d'une correspondance avec la même différence, choisissez celle où "Zamalek" a marqué plus de buts.
Compte tenu de l'information sur tous les matchs qu'ils ont joués, retournez l'index de la meilleure correspondance. S'il y a plus d'un résultat valide, renvoyez le plus petit indice.
Pour ALAHLYGoals = [6,4] et zamalekGoals = [1,2], la sortie devrait être de 1.
Parce que 4 - 2 est inférieur à 6 - 1
Pour ALAHLYGoals = [1,2,3,4,5] et zamalekGoals = [0,1,2,3,4], la sortie devrait être 4.

#75 - Animaux écrasés (5 kyu)

Retrouvez le nom de l'animal écrasé à partir de sa photo :

```
roadKill("==========h===yyyyyy===eeee=n==a========") → "hyena"
roadKill("======pe====nnnnnn================n=n=ng====u==iiii=iii==nn============n=")
→ "penguin"
roadKill("=====r=rrr=rra=====eee======bb====b=======") → "bear"
```

La liste des animaux est donnée dans ANIMALS

```
ANIMALS=['aardvark','alligator','armadillo','antelope','baboon','bear','bobcat','butterf
ly','cat','camel','cow','chameleon','dog','dolphin','duck','dragonfly','eagle','elephant
','emu','echidna','fish','frog','flamingo','fox','goat','giraffe','gibbon','gecko','hyen
a','hippopotamus','horse','hamster','insect','impala','iguana','ibis','jackal','jaguar',
'jellyfish','kangaroo','kiwi','koala','killerwhale','lemur','leopard','llama','lion','mo
nkey','mouse','moose','meercat','numbat','newt','ostrich','otter','octopus','orangutan',
'penguin','panther','parrot','pig','quail','quokka','quoll','rat','rhinoceros','racoon',
'reindeer','rabbit','snake','squirrel','sheep','seal','turtle','tiger','turkey','tapir',
'unicorn','vampirebat','vulture','wombat','walrus','wildebeast','wallaby','yak','zebra']
```

#76 - Chaîne dans une chaîne (5 kyu)

Créez une fonction qui renvoie **true** si une partie des caractères de **str1** peut être réarrangée pour correspondre à **str2**, sinon renvoyer **false**. Par exemple :
Si **str1** est 'rkqodlw' et **str2** est 'world', la sortie devrait retourner **true**.
Si **str1** est 'cedewaraaossoqqyt' et **str2** est 'codewars' devrait retourner **true**.
Si **str1** est 'katas' et **str2** est 'steak' devrait renvoyer **false**.

Seules les minuscules seront utilisées (a-z). Aucune ponctuation ni aucun chiffre ne sera inclus.
La performance doit être considérée.

#77 - Données sur 1 octet (5 kyu)

Vous devez surveiller des installations industrielles dans le cercle arctique à l'aide de 1 à 16 caméras. En raison des coûts élevés des données, chaque commande envoyée à ces caméras est limitée à un octet. Chaque caméra dans chaque installation est initialement réglée à 30 degrés vers le bas (-30°) et pointée vers l'avant. Les caméras peuvent être commandées à distance pour les déplacer vers le haut (up) ou le

bas (down), à gauche (left) ou à droite (right), jusqu'à 45 degrés maximum dans chacune des directions. 'up' et 'right' sont considérés comme positifs, 'down' et 'left' sont considérés comme négatifs.
Les quatre premiers bits du paquet représentent l'ID de la caméra (indexées à partir de 0). Le cinquième bit indique s'il faut déplacer la caméra de cinq degrés vers le haut, le sixième bit s'il faut baisser de cinq degrés. Les septième et huitième bit indiquent s'il faut déplacer la caméra à gauche ou à droite respectivement, de cinq degrés.
Complétez ce début de programme :

```
function Network (count) {
  this.cameras = [];
  for (var i = 0; i < count; i++)
    this.cameras.push(new Camera(0, -30));
}

Network.prototype.process = function (byte) {
  // Ajustement de la caméra suivant la valeur de « byte »
}

function Camera (h, v) {
  this.horizontal = h;
  this.vertical = v;
}

Camera.prototype.move = function (h, v) {
  // Ajustement de l'angle
}
```

#78 - Triangle Pascal (4 kyu)

Écrivez une fonction qui à une profondeur (n), renvoie un tableau à une seule dimension représentant le Triangle de Pascal au n-ième niveau.

#79 - Parenthèses, accolades et crochets (4 kyu)

Écrivez une fonction appelée **valideBraces** qui prend une chaîne de caractères et détermine si l'ordre des accolades est valide. **ValidBraces** doit renvoyer **true** si la chaîne est valide et **false** si elle n'est pas valide.
Toutes les chaînes d'entrée seront non vides et ne comporteront que des parenthèses ouvertes '(', parenthèses fermées ') ', des crochets ouverts '[', des crochets fermés ']', des accolades '{' et des accolades fermées '}'.
'() {} []' Et '([{}])' sont considérés comme valides, tandis que '(}', '[(])' et '[({})] (]' sont considérés comme non valides .

```
ValidBraces ("() {} []") → true
ValidBraces ("(}") → false
ValidBraces ("[(])") → faux
ValidBraces ("([{}])") → true
```

#80 - Simplification d'un polynôme (4 kyu)

Toutes les sommes et soustractions possibles de monômes équivalents ("xy == yx") ont été effectuées, par exemple :

"cb + cba" -> "bc + abc", "2xy-yx" -> "xy", "-a + 5ab + 3a-c-2a" -> "-c + 5ab"

Tous les monômes apparaissent dans l'ordre du nombre croissant de variables, par exemple :

"-abc + 3a + 2ac" -> "3a + 2ac-abc", "xyz-xz" -> "-xz + xyz"

Si deux monômes ont le même nombre de variables, ils apparaissent dans l'ordre lexicographique, par exemple :

"a + ca-ab" -> "a-ab + ac", "xzy + zby" -> "byz + xyz"

Il n'y a pas de signe + si le premier coefficient est positif, par exemple :

"-y + x" -> "x-y", mais aucune restriction pour - : "y-x" -> "- x + y"

simplify("dc+dcba") → "cd+abcd"
simplify("2xy-yx") → "xy"
simplify("-a+5ab+3a-c-2a") → "-c+5ab"

#81 - Grandes factorielles (4 kyu)

#82 - Conversion du temps (4 kyu)

Votre tâche est d'écrire une fonction qui écrit une durée, donnée en secondes, de façon conviviale.

La fonction doit accepter un nombre entier non négatif. Si elle est nulle, elle revient simplement "now". Sinon, la durée est exprimée sous la forme d'une combinaison d'années, de jours, d'heures, de minutes et de secondes.

FormatDuration (62) → "1 minute et 2 secondes"

FormatDuration (3662) → "1 heure, 1 minute et 2 secondes"

Règles détaillées

L'expression résultante est faite de composants comme 4 secondes, 1 an, etc. En général, un nombre entier positif et l'une des unités de temps valides, séparées par un espace. L'unité de temps est utilisée au pluriel si l'entier est supérieur à 1.

Les composants sont séparés par une virgule et un espace (","). Sauf le dernier composant, qui est séparé par "et", tout comme il serait écrit en anglais.

Une année fera 365 jours et un jour 24 heures.

#83 - Chiffres romains (4 kyu)

Créez une fonction prenant un entier positif comme paramètre et renvoyant une chaîne contenant la représentation de ce nombre en chiffres romains.

En chiffres romains, 1990 s'écrit : 1000 = M, 900 = CM, 90 = XC soit **MCMXC**.

2008 s'écrit en 2000 = MM, 8 = VIII d'où **MMVIII**.

1666 utilise chaque symbole romain par ordre décroissant: MDCLXVI.

```
solution (1000) → 'M'
```

#84 - Énumération des permutations (4 kyu)

permutations('a'); // ['a']

permutations('ab'); // ['ab', 'ba']

permutations('aabb'); // ['aabb', 'abab', 'abba', 'baab', 'baba', 'bbaa']

#85 - Nombre suivant avec les mêmes chiffres (4 kyu)

Vous devez créer une fonction qui prend un nombre entier positif et renvoie le plus grand nombre suivant formé par les mêmes chiffres :

```
NextBigger (12) == 21
NextBigger (513) == 531
NextBigger (2017) == 2071
```

Si aucun nombre plus grand ne peut être composé en utilisant ces chiffres, retournez -1:

```
NextBigger (9) == - 1
NextBigger (111) == - 1
NextBigger (531) == - 1
```

#86 - Mixer 2 chaînes de caractères (4 kyu)

Étant donné deux chaînes de caractères **s1** et **s2**, nous voulons visualiser à quel point elles sont différentes. Nous ne prendrons en compte que les lettres minuscules (a à z). Tout d'abord, comptez la fréquence de chaque minuscule dans s1 et s2.

```
s1 = "A aaaa bb c"
s2 = "& aaa bbb c d"
```

s1 a 4 'a', 2 'b', 1 'c' et s2 a 3 'a', 3 'b', 1 'c', 1 'd'

Donc, le maximum pour 'a' dans **s1** et **s2** est le 4 de **s1** ; Le maximum pour 'b' est le 3 de **s2**. Dans ce qui suit, nous ne considérerons pas les lettres lorsque le maximum de leurs occurrences est inférieur ou égal à 1.

Nous pouvons reprendre les différences entre s1 et s2 dans la chaîne suivante : **1:aaaa/2:bbb**
où **1:aaaa** signifie que le maximum pour « a » est 4 avec la chaîne **s1**. De la même manière **2:bbb** signifie que le maximum est 3 pour « b » avec la chaîne **s2**.

Votre tâche est de produire une chaîne dans laquelle chaque lettre minuscule de **s1** ou **s2** apparaît autant de fois qu'elle est maximale si ce maximum est strictement supérieur à 1. Ces lettres seront préfixées par le numéro de la chaîne où elles apparaissent avec leur valeur maximale et, en cas d'égalité, le préfixe sera = :.

Dans le résultat, les sous-chaînes (une sous-chaîne est par exemple 2:nnnnn ou 1:hhh; elle contient le préfixe) seront en ordre décroissant de leur longueur et lorsqu'elles auront la même longueur triées en ordre lexicographique ascendant (lettres et chiffres); Les différents groupes seront séparés par '/'.

```
s1 = "my&friend&Paul has heavy hats! &"
s2 = "my friend John has many many friends &"
mix(s1, s2) → "2:nnnnn/1:aaaa/1:hhh/2:mmm/2:yyy/2:dd/2:ff/2:ii/2:rr/=:ee/=:ss"

s1 = "mmmmm m nnnnn y&friend&Paul has heavy hats! &"
s2 = "my frie n d Joh n has ma n y ma n y frie n ds n&"
mix(s1, s2) → "1:mmmmmm/=:nnnnnn/1:aaaa/1:hhh/2:yyy/2:dd/2:ff/2:ii/2:rr/=:ee/=:ss"

s1="Are the kids at home? aaaaa fffff"
s2="Yes they are here! aaaaa fffff"
mix(s1, s2) → "=:aaaaaa/2:eeeee/=:fffff/1:tt/2:rr/=:hh"
```

#87 - Hauteur de pluie

Étant donné un tableau de hauteurs qui contient des nombres entiers positifs ou nuls. Il représente une carte d'élévation où la largeur de chaque barre est 1. Calculez la quantité d'eau qu'elle peut piéger après la pluie.

Hauteurs = [1,0,2,1,0,1,3,2,1,2,1]

```
            H
     H      H H  H
 H   H H  H H H H H H
 1 0 2 1 0 1 3 2 1 2 1


            H
     H * * * H H * H
 H * H H * H H H H H H
 1 0 2 1 0 1 3 2 1 2 1
```

Dans ce cas on doit obtenir 6

```
TrapWater ([1,0,2,1,0,1,3,2,1,2,1]) === 6
TrapWater ([10,0,10]) === 10
TrapWater ([0,10,0]) === 0
```

#88 - Longueur d'une boucle (3 kyu)

#89 - Distance de Levensthein (3 kyu)

```
fruits = new Dictionary(['cherry', 'pineapple', 'melon', 'strawberry', 'raspberry']);
fruits.findMostSimilar('strawbery'); // must return "strawberry"
fruits.findMostSimilar('berry'); // must return "cherry"

things = new Dictionary(['stars', 'mars', 'wars', 'codec', 'codewars']);
things.findMostSimilar('coddwars'); // must return "codewars"

languages = new Dictionary(['javascript', 'java', 'ruby', 'php', 'python',
'coffeescript']);
languages.findMostSimilar('heaven'); // must return "java"
languages.findMostSimilar('javascript');
```

#90 - Rendez-vous entre plusieurs personnes (3 kyu)

Les gens d'affaires savent qu'il n'est souvent pas facile de trouver un rendez-vous. Nous voulons trouver un tel rendez-vous automatiquement. Vous recevrez les calendriers de vos interlocuteurs et une durée pour la réunion. Votre tâche consiste à trouver le plus tôt possible un créneau commun libre pendant au moins cette durée.

Personne	Réunions
A	09h00 - 11h30, 13h30 - 16h00, 16h00 - 17h30, 17h45 - 19h00
B	09h15 - 12h00, 14h00 - 16h30, 17h00 - 17h30
C	11h30 - 12h15, 15h00 - 16h30, 17h45 - 19h00

Toutes les heures dans les calendriers seront données dans le format 24h "hh: mm", le résultat doit également être dans ce format

Une réunion est représentée par son heure de début (inclusivement) et fin (exclusivement) -> si une réunion a lieu de 09h00 à 11h00, la prochaine heure de départ possible serait 11h00

Les hommes d'affaires travaillent à partir de 09h00 (inclusivement) - 19h00 (exclusivement), le rendez-vous doit commencer et se terminer dans cette fourchette

Si la réunion ne correspond pas aux annexes, renvoyer **null**

La durée de la réunion sera fournie en nombre entier en minutes

En suivant ces règles et en regardant l'exemple ci-dessus, le plus tôt possible pour une réunion de 60 minutes serait 12h15.

```
schedules=[[['09:00', '11:30'], ['13:30', '16:00'], ['16:00', '17:30'], ['17:45',
'19:00']],[['09:15', '12:00'], ['14:00', '16:30'], ['17:00', '17:30']], [['17:45',
'19:00']]];
```

getStartTime(schedules, 60) → 12:15

#91 - Chemin le plus court dans un graphe (3 kyu)

```
var roads = [
```

```
    {from: 0, to: 1, drivingTime: 5},
    {from: 0, to: 2, drivingTime: 10},
    {from: 1, to: 2, drivingTime: 10},
    {from: 1, to: 3, drivingTime: 2},
    {from: 2, to: 3, drivingTime: 2},
    {from: 2, to: 4, drivingTime: 5},
    {from: 3, to: 2, drivingTime: 2},
    {from: 3, to: 4, drivingTime: 10}
];
navigate(5, roads, 0, 4) → [0, 1, 3, 2, 4].
```

Le temps le plus court est de 5 + 2 + 2 + 5 = 14 minutes

#92 - Distance sur une sphère (3 kyu)

Nous sommes sur un grand voilier au large de la côte. Le capitaine voudrait connaître la distance entre deux points sur la carte.

Complétez la fonction afin qu'elle renvoie la distance entre deux coordonnées données. Exemples :

```
48 ° 12 '30 "N, 16 ° 22' 23" E et 23 ° 33 '0 "S, 46 ° 38' 0" W
58 ° 18 '0 "N, 134 ° 25' 0" W et 33 ° 51 '35 "S, 151 ° 12' 40" E
```

La distance devra être en kilomètres. La Terre sera considérée comme une sphère de rayon 6371 km.

Il suffira que le résultat soit précis à 10 km, plus précisément 6387 devient 6380, 643 devient 640 et 18299 devient 18290. Exemples d'entrées et de résultats attendus :

```
Distance ("48 ° 12 '30" N, 16 ° 22' 23 "E", "23 ° 33 '0" S, 46 ° 38' 0 "W") → 10130
Distance ("48 ° 12 '30" N, 16 ° 22' 23 "E", "58 ° 18 '0" N, 134 ° 25' 0 "W") → 7870
Distance ("48 ° 12 '30" N, 16 ° 22' 23 "E", "48 ° 12 '30" N, 16 ° 22' 23 "E") → Renvoie
0
```

CORRIGÉS

#1 - Nombre de personnes dans le bus (8 kyu)

- **SOLUTION 1**

```
const number = (busStops) => busStops.reduce((rem, [on, off]) => rem + on - off, 0);
```

Remarquez les alternatives :

```
[[1,2],[3,4]].map(a=>a[0]+a[1])
→ Array [ 3, 7 ]
[[1,2],[3,4]].map(([x,y])=>x+y) // Avec des parenthèses autour des [ ]
→ Array [ 3, 7 ]
```

- **SOLUTION 2**

```
var number = function (busStops) {
  var totalPeople = 0;
  for (var i = 0; i < busStops.length; i++)
       totalPeople += busStops[i][0] - busStops[i][1];
  return totalPeople;
}
```

#2 - Nombre de moutons (8 kyu)

- **SOLUTION 1**

```
function countSheeps(arrayOfSheep) {
  return arrayOfSheep.reduce((a,n)=>a+(n==true),0)
}

2+(true==true)
→ 3
2+(false==true)
→ 2
```

- **SOLUTION 2**

```
function countSheeps(arr) {
  return arr.filter(Boolean).length;
}

Boolean(false)
→ false
Boolean(true)
→ true
```

#3 - Enlever le premier et le dernier caractère d'une chaîne (8 kyu)

- **SOLUTION 1**

```
const removeChar = str => str.slice(1,-1)
```

- **SOLUTION 2**

```
const removeChar = (str) => str.replace(/^.|.$/g, '');
```

^. : Premier caractère de la chaîne

.$: Dernier caractère

```
"ABCD".match(/^.|.$/g)
→ Array [ "A", "D" ]
```

#4 - Doubler les lettres (8 kyu)

- **SOLUTION 1**

```
doubleChar=(str)=>str.replace(/./g, '$&$&')
```

- **SOLUTION 2**

```
let doubleChar=s=>s.replace(/(.)/g,"$1$1");
```

Bien voir la distinction entre :

```
"ABCD".replace(/./g, '$&$&')       // Remplace l'expression entière
→ "AABBCCDD"
"ABCD".replace(/./g, '$1$1')       // Sans les parenthèses = Pas de groupe
→ "$1$1$1$1$1$1$1$1"
"ABCD".replace(/(.)/g, '$1$1')     // Avec les parenthèses = groupe n°1
→ "AABBCCDD"
"ABCD".replace(/../g, '$&$&')
→ "ABABCDCD"
```

- **SOLUTION 3**

```
const doubleChar = (str) => str.split("").map(c => c + c).join("");
```

On peut également remplacer `str.split("")` par `[...str]`

#5 - Accumulation de lettres (7 kyu)

- **SOLUTION 1**

```
function accum(s) {
       return [...s].map((c,k)=>c.toUpperCase()+c.toLowerCase().repeat(k)).join('-')
}

"A".repeat(0)
→ ""
"A".repeat(5)
→ "AAAAA"
```

#6 - Code PIN (7 kyu)

- **SOLUTION 1**

```
function validatePIN(pin) {
  return /^(\d{4}|\d{6})$/.test(pin)
}
```

On regarde si la chaîne commence ^ par 4 chiffres `\d{4}` ou 6 chiffres `\d{6}` puis se termine `$`

#7 - Mettre chaque 1ere lettre des mots en majuscule (7 kyu)

- **SOLUTION 1**

```
String.prototype.toJadenCase = function () {
  return this.split(' ').map(m=>m[0].toUpperCase()+m.slice(1)).join(' ')
};
```

```
"ABCD".slice(1)
→ "BCD"
```

- **SOLUTION 2**

```
String.prototype.toJadenCase = function () {
  return this.replace(/(^| )(\w)/g, s=>s.toUpperCase())
};
```

Attention : `s` correspond soit à un début de phrase `^` suivi d'une lettre soit à `espace` suivi d'une lettre.

C'est cet ensemble qui est mis en majuscules.

```
"aa-bbb-c".replace(/(^|-)(\w)/g,'$1$1')   // lettres remplacées par - ou rien
→ "a--bb--"
"aa-bbb-c".match(/(^|-)(\w)/g)
→ Array [ "a", "-b", "-c" ]
"aa-bbb-c".replace(/(^|-)(\w)/g,'$2$2')   // début phrase et tirets remplacés par lettres
→ "aaabbbbcc"
```

- **SOLUTION 3**

```
String.prototype.toJadenCase = function () {
  function capitalizeFirstLetter(string) {
      return string.charAt(0).toUpperCase() + string.slice(1);
  }
  return this.split(' ').map(capitalizeFirstLetter).join(' ');
};
```

#8 - Nombre de voyelles (7 kyu)

- **SOLUTION 1**

```
function getCount(str) {
  return (str.match(/[aeiou]/g) || []).length;
}
```

JavaScript ne peut pas renvoyer la longueur de **null**.

```
"xx".match(/[aeiou]/g)
→ null
"xx".match(/[aeiou]/g).length
→ TypeError: "xx".match(...) is null [En savoir plus]
"xx".match(/[aeiou]/g) || []
→ Array [  ]
```

Remarquez aussi la différence entre :

```
null | []     // Opération binaire entre 2 booléens false et true (voir ci-dessous)
→ 0
null || []    // Opérateur « ou »
→ Array [  ]

Boolean([])
→ true
Boolean(null)
→ false
Boolean({})
→ true
Boolean("")
→ false
```

- **SOLUTION 2**

```
function getCount(str) {
  return str.replace(/[^aeiou]/gi, '').length;
}
```

On remplace toutes les lettres qui ne sont pas des voyelles par rien et on compte les lettres restantes.

#9 - Meeting (7 kyu)

- **SOLUTION 1**

```
function orderFood(list) {
  food={}
  list.map(o=>food.hasOwnProperty(o.meal) ? food[o.meal]++ : food[o.meal]=1)
  return food
}

obj={}                              // Création d'un objet vide
→ Object {  }
obj.x=10                            // Ajout propriété « x » avec valeur 10
→ 10
obj
→ Object { x: 10 }
obj.hasOwnProperty('y')             // L'objet a-t-il la propriété « y » ?
false
obj.x++                             // Incrémentation de la valeur de « x »
10
obj
Object { x: 11 }
obj['x']++                          // Autre technique d'incrémentation
11
obj
```

```
Object { x: 12 }
```

- **SOLUTION 2**

```
const orderFood = a => a.reduce( (acc,v) => ( acc[v.meal] = ( acc[v.meal] || 0 ) + 1, acc ), {} ) ;
```

```
obj.z               // La propriété « z » n'existe pas
undefined
obj.z || 0          // Renvoyer 0 quand la propriété n'existe pas
0
```

#10 - Compteur (7 kyu)

- **SOLUTION 1**

```
function counterEffect(hitCount) {
 return [...hitCount].map(n=>[...Array(+n+1).keys()])
}
```

- **SOLUTION 2**

```
function counterEffect(hitCount) {
  return [...hitCount].map( d => Array.from({length:+d+1}, (_,i) => i) )
}
```

#11 - Le plus long nombre (7 kyu)

- **SOLUTION 1**

```
function findLongest(array){
  return array.sort((a,b)=>(a.toString().length<b.toString().length) ? 1 : 0)[0]
}
```

- **SOLUTION 2**

```
const findLongest = l => l
  .reduce((a, b) => (`${b}`.length > `${a}`.length) ? b : a);
```

- **SOLUTION 3**

```
function findLongest(a) {
  let m = Math.max(...a);
  for (let i = 0; i < a.length; i++) if (m.toString().length == a[i].toString().length )
return a[i];
}
```

#12 - Le mot le plus court (7 kyu)

- **SOLUTION 1**

```
function findShort(s){
 return Math.min(...s.split(' ').map(c=>c.length))
}
```

- **SOLUTION 2**

```
function findShort(s){
  return Math.min.apply(null, s.split(' ').map(w => w.length));
}
```

- **SOLUTION 3**

```
const findShort = (s) => s
  .split(' ')
  .sort((a, b) => b.length - a.length)
  .pop()
  .length;
```

- **SOLUTION 4**

```
function findShort(s) {
  return s.split(' ').reduce((min, word) => Math.min(min, word.length), Infinity);
}
```

#13 - RVB vers niveaux de gris (7 kyu)

- **SOLUTION 1**

```
var color2grey = function (image) {
return  image.map(c=>c.map(k=>{g=Math.round((k[0]+k[1]+k[2])/3); return [g,g,g]}))
}
```

- **SOLUTION 2**

```
var color2grey = function (image) {
  return image.map( row => row.map( ([r,g,b]) => (grey =>
[grey,grey,grey])(Math.round((r+g+b)/3)) ) )
}
```

- **SOLUTION 3**

```
function color2grey (image) {
    return image.map( // Loop through each row
      a => a.map( // Loop through each pixel
        ([r, g, b]) => // Get 3 values: RGB from array(array destructuring)
          [...new Array(3)].fill( // Creating new array filled with 3 equal values
            ~~((r + g + b + 1) / 3) // Getting greyscale value and converting to
integer(~~)
          )
      )
    );
}
```

#14 - Carré des nombres (7 kyu)

- **SOLUTION 1**

```
function squareDigits(num){
  return 1*num.toString().split('').map((c)=>c*c).join('');
```

```
  // return Number(('' + num).split('').map(function (val) { return val *
val;}).join(''));
}
```

#15 - Carte crédit – 4 derniers chiffres (7 kyu)

- **SOLUTION 1**

```
function maskify(cc) {
 return ((cc.length>4) ? "#".repeat(cc.length-4) :"")+cc.slice(-4)
}
```

- **SOLUTION 2**

```
function maskify(cc) {
  return cc.slice(0, -4).replace(/./g, '#') + cc.slice(-4);
}
```

- **SOLUTION 3**

```
function maskify(cc) {
  return cc.replace(/.(?=....)/g, '#');
}
```

- **SOLUTION 4**

```
function maskify(cc) {
  return cc.replace(/.(?=.{4})/g, "#");
}
```

#16 - Nombres les plus grands et plus petits d'une liste (7 kyu)

- **SOLUTION 1**

```
function highAndLow(numbers){
  return Math.max(...numbers.split(' '))+" "+Math.min(...numbers.split(' '));
  /*  numbers = numbers.split(' ');
  return `${Math.max(...numbers)} ${Math.min(...numbers)}`;*/
}
```

- **SOLUTION 2**

```
function highAndLow(numbers){
  numbers = numbers.split(' ').map(Number);
  return Math.max.apply(0, numbers) + ' ' + Math.min.apply(0, numbers);
}
```

#17 - ADN – Remplacer A par T, C par G et réciproquement (7 kyu)

- **SOLUTION 1**

```
var pairs = {'A':'T','T':'A','C':'G','G':'C'};
function DNAStrand(dna){
return dna.split('').map((v)=> pairs[v]).join('');
}
```

- **SOLUTION 2**

```
function DNAStrand(dna) {
  return dna.replace(/./g, function(c) {
    return DNAStrand.pairs[c]
  })
}

DNAStrand.pairs = { A: 'T',  T: 'A',  C: 'G',  G: 'C',}
```

- **SOLUTION 3**

```
var pairs = {'A':'T','T':'A','C':'G','G':'C'};

function DNAStrand(dna){
  return dna.split('').map(function(v){ return pairs[v] }).join('');
}
```

- **SOLUTION 4**

```
function DNAStrand(dna) {
  return dna.split('').map(function(v) {return {A:'T', T:'A', C:'G',
G:'C'}[v];}).join('');
}
```

#18 - Nombres vampires (7 kyu)

- **SOLUTION 1**

```
comb=(s)=>[...s].sort().join('');
var vampire_test = function(a, b){
    return comb(a + '' + b)==comb((a*b).toString());
}
```

#19 - Divisible par ? (7 kyu)

- **SOLUTION 1**

```
function isDivisible(n,...r)=>[...r].filter(k=>n%k!=0).length==0
```

- **SOLUTION 2**

```
function isDivisible(firstN, ...otherN){
  return otherN.every(n => firstN % n === 0);
}
```

- **SOLUTION 3**

```
const isDivisible = (n, ...xs) => xs.every(x => n % x == 0);
```

- **SOLUTION 4**

```
const isDivisible = (m, ...ns) => !ns.some(n => m % n);
```

- **SOLUTION 5**

```
function isDivisible(n){
  return [].slice.call(arguments, 1).every(function(a){ return !(n % a) });
}
```

#20 - Deviner un mot (7 kyu)

- **SOLUTION 1**

```
function countCorrectCharacters(correctWord, guess){
  if(correctWord.length-guess.length) throw new Error('Mauvaise longueur !');
  return [...correctWord].filter((c,i)=>c===guess[i]).length;
}
```

- **SOLUTION 2**

```
function countCorrectCharacters(correctWord, guess){
  if (correctWord.length!=guess.length) throw Error("Mauvaise longueur");
  return [...correctWord].reduce((a,c,k)=>a+1*(c==guess[k]),0)
}
```

#21 - Filtrer une liste (7 kyu)

- **SOLUTION 1**

```
filter_list = (l) => l.filter((n,k)=>n===parseInt(n))
```

- **SOLUTION 2**

```
filter_list =(l) => l.filter(v => typeof v == "number")
```

#22 - }Ordre des mots (6 kyu)

- **SOLUTION 1**

```
function order(words){
  return words.split(' ').sort(function(a, b){
      return a.match(/\d/) - b.match(/\d/);
   }).join(' ');
}
```

- **SOLUTION 2**

```
function order(words) {
  return words.split(' ').sort((wordA, wordB) => wordA.match(/\d+/) >
wordB.match(/\d+/)).join(' ')
}
```

#23 - Nombre de 1 en binaire (6 kyu)

- **SOLUTION 1**

```
var countBits = function(n) {
  return n.toString(2).replace(/0/g,'').length
};
```

- **SOLUTION 2**

```
function countBits(n) {
  for(c=0;n;n>>=1)c+=n&1
  return c;
}
```

- **SOLUTION 3**

```
countBits = n => n.toString(2).split('0').join('').length;
```

#24 - Distribution d'électrons (6 kyu)

- **SOLUTION 1**

```
function atomicNumber(num){
 s=[]
 n=1
 while (num>0) {
   m=2*Math.pow(n,2);
   if (num>=m) {num-=m; s.push(m); n++;} else {s.push(num); num=0}
  }
  return s
}
```

- **SOLUTION 2**

```
function atomicNumber(num) {
  var c = 1, res = Array();
  while (num > c * c * 2) {
    res.push(c * c * 2);
    num -= c * c * 2;
    c++;
  }
  res.push(num);
  return res;
}
```

- **SOLUTION 3**

```
const atomicNumber = (n,shell=1) => n>0 ? [ Math.min( 2*shell*shell, n ),
...atomicNumber(n-2*shell*shell,shell+1) ] : [] ;
```

#25 - Numéros de téléphone (6 kyu)

- **SOLUTION 1**

```
function createPhoneNumber(numbers){
var n=numbers.join('')
  return "("+n.slice(0,3)+") "+n.slice(3,6)+"-"+n.slice(6)
}
```

- **SOLUTION 2**

```
function createPhoneNumber(numbers){
  return numbers.join('').replace(/(...)(...)(.*)/, '($1) $2-$3');
}
```

#26 - Compression d'une phrase (6 kyu)

- **SOLUTION 1**

```
sentenceCompression=s=>s.replace(/[\W\d]/g,'')
```

- **SOLUTION 2**

```
sentenceCompression=s=>s.replace(/[^A-z]/g,'')
```

- **SOLUTION 3**

```
sentenceCompression=s=>s.replace(/\d|\W/gi,'')
```

#27 - Mots consécutifs (6 kyu)

- **SOLUTION 1**

```
function longestConsec(strarr, k) {
    return (k<=0 | k>strarr.length) ? '' :
strarr.reduce((a,_,i,t)=>(a.length<t.slice(i,i+k).join('').length) ?
t.slice(i,i+k).join('') : a,'')
}
```

- **SOLUTION 2**

```
function longestConsec(strarr, k) {
    var longest = "";
    for(var i=0;k>0 && i<=strarr.length-k;i++){
      var tempArray = strarr.slice(i,i+k);
      var tempStr = tempArray.join("");
      if(tempStr.length > longest.length){
        longest = tempStr;
      }
    }
    return longest;
}
```

- **SOLUTION 3**

```
function longestConsec(strarr, k) {
  if( strarr.length==0 || k> strarr.length || k <1 ) return "";
  let lens = strarr.map( (_,i,arr) => arr.slice(i,i+k).join('').length ),
      i = lens.indexOf( Math.max(...lens) );
  return strarr.slice(i,i+k).join('')
}
```

#28 - Construction d'une tour (6 kyu)

- **SOLUTION 1**

```
function towerBuilder(nFloors) {
  return Array(nFloors).fill(0).map((_,k) =>
    " ".repeat(k)+"*".repeat(2*nFloors-1-2*k)+" ".repeat(k))
    .reverse()
}
```

- **SOLUTION 2**

```
function towerBuilder(n) {
  return Array.from({length: n}, function(v, k) {
    const spaces = ' '.repeat(n - k - 1);
    return spaces + '*'.repeat(k + k + 1) + spaces;
  });
}
```

- **SOLUTION 3**

```
function towerBuilder(n) {
  return [...Array(n)].map((_,i)=>" ".repeat(n-1-i)+"*".repeat(i*2+1)+" ".repeat(n-1-i))
}
```

#29 - Contrariant (6 kyu)

- **SOLUTION 1**

```
function AlanAnnoyingKid(input){
console.log(input)
  var r=input.split(/(?:Today I )(didn't)?\s?(\w+)\s/).filter(c=>c!="")
  var neg=r[0]==undefined
  var s="I don't think you "+(!neg ? "didn't " : "" )+r[1]+" "+r[2].slice(0,r[2].length-1)
  s+=" today, I think you "+(neg ? "didn't "+r[1].slice(0,r[1].length-2)+" at all!" : "did
"+r[1]+" it!")
  return s
}
```

- **SOLUTION 2**

```
function AlanAnnoyingKid(input){
  let action = input.substring(8, input.length - 1);
  let negation = input.indexOf("didn't") > 0;
  let verb = negation ? action.split(" ")[1] : input.match(/\w+(?=ed)/)[0];

  return "I don't think you " + action + " today, I think you " +
      (negation ? "did" : "didn't") + " " + verb + " " + (negation ? "it!" : "at all!");
```

```
}
```

- **SOLUTION 3**

```
function AlanAnnoyingKid(input) {
  const m = input.match(/^Today I ((?:didn't )?[a-z]+) ([^.]+).$/);
  if (m === null) return;
  if (m[1].startsWith("didn't")) {
    return `I don't think you ${m[1]} ${m[2]} today, I think you did ${m[1].split('
')[1]} it!`;
  } else {
    return `I don't think you ${m[1]} ${m[2]} today, I think you didn't ${m[1].slice(0,
-2)} at all!`;
  }
}
```

#30 - Parité (6 kyu)

- **SOLUTION 1**

```
function iqTest(numbers){
  var a=numbers.split(' ')
  var k=0
  while ((+a[k]+ +a[k+1])%2==0) k++
  return k+1+(k>0 | (+a[k]+ +a[k+2])%2==0)
}
```

- **SOLUTION 2**

```
function iqTest(numbers){
  numbers = numbers.split(' ')

  var evens = []
  var odds = []

  for (var i = 0; i < numbers.length; i++) {
    if (numbers[i] & 1) {
      odds.push(i + 1)
    } else {
      evens.push(i + 1)
    }
  }

  return evens.length === 1 ? evens[0] : odds[0]
}
```

- **SOLUTION 3**

```
function iqTest(numbers){
  var nums = numbers.split(" ").map(x => x % 2);
  var sum = nums.reduce((a,b) => a + b);
  var target = sum > 1 ? 0 : 1;

  return nums.indexOf(target) + 1;
}
```

- **SOLUTION 4**

```
function iqTest(numbers){
  var m = numbers.match(/[02468]\b.*[02468]\b/.test(numbers) ? /\d*[13579]\b/ :
/\d*[02468]\b/)[0];
  return numbers.split(' ').indexOf(m) + 1;
}
```

- **SOLUTION 5**

```
const iqTest = numbers => {
  numbers = numbers.replace(/(\d+\s)|(\d+$)/g, c => c % 2);
  return numbers.indexOf(numbers.match(/0/g).length > 1 ? '1' : '0') + 1;
}
```

#31 - Distribution de bonbons (6 kyu)

- **SOLUTION 1**

```
int=n=>(n+1)/2|0
distributionOfCandy= (candies) => {
  var round=0
  while (!candies.every(n=>n==candies[0])){
    round++
    candies=candies.map((b,k)=>int(b)+int(candies[(k>0) ? k-1 : candies.length-1]))
  }
  return [round,candies[0]]
}
```

- **SOLUTION 2**

```
function distributionOfCandy(candies){
  var count = 0;
  var gives = [];
  while ((candies.findIndex(v => v!=candies[0])) != -1) {
    candies.forEach((v,i,candies) => v%2 ? candies[i]++ : 0);
    gives = candies.map(v => v/2);
    candies = gives.map((v,i,gives) => v+gives[i-1]);
    candies[0] = gives[0] + gives[gives.length-1];
    count++;
  }
  return [count,candies[0]];
}
```

- **SOLUTION 3**

```
function distributionOfCandy(cs) {
  const n = cs.length;
  var k = 0;
  while (!cs.every(c => c == cs[0])) {
    const ps = cs.map(c => (c >> 1) + (c & 1));
    for (let i = 0; i < n; ++i) cs[i] += ps[(i + 1) % n] - (cs[i] >> 1);
    ++k;
  }
  return [k, cs[0]];
}
```

#32 - 10 minutes de promenade (6 kyu)

- **SOLUTION 1 (CF CARTE AU TRÉSOR)**

```
function isValidWalk(walk) {
  var opposite = { "s":"n", "n":"s", "w":"e", "e":"w"}
  return (walk.reduce(function (a, b, i) {
    opposite[a.slice(-1)] === b ? a.pop() : a.push(b);
    return a
  }, []).length==0 & walk.length==10)
}
```

- **SOLUTION 2**

```
function isValidWalk(walk) {
  var dx = 0
  var dy = 0
  var dt = walk.length

  for (var i = 0; i < walk.length; i++) {
    switch (walk[i]) {
      case 'n': dy--; break
      case 's': dy++; break
      case 'w': dx--; break
      case 'e': dx++; break
    }
  }

  return dt === 10 && dx === 0 && dy === 0
}
```

- **SOLUTION 3**

```
function isValidWalk(walk) {
  function count(val) {
    return walk.filter(function(a){return a==val;}).length;
  }
  return walk.length==10 && count('n')==count('s') && count('w')==count('e');
}
```

- **SOLUTION 4**

```
function isValidWalk(walk) {
  return walk.length == 10 && !walk.reduce(function(w,step){ return w + {"n":-
1,"s":1,"e":99,"w":-99}[step]},0)
}
```

#33 - Likes (6 kyu)

- **SOLUTION 1**

```
function likes(names) {
  names = names  [];
  switch(names.length){
    case 0: return 'no one likes this'; break;
    case 1: return names[0] + ' likes this'; break;
    case 2: return names[0] + ' and ' + names[1] + ' like this'; break;
```

```
    case 3: return names[0] + ', ' + names[1] + ' and ' + names[2] + ' like this';
break;
    default: return names[0] + ', ' + names[1] + ' and ' + (names.length - 2) + ' others
like this';
  }
}
```

- **SOLUTION 2**

```
function likes (names) {
  var templates = [
    'no one likes this',
    '{name} likes this',
    '{name} and {name} like this',
    '{name}, {name} and {name} like this',
    '{name}, {name} and {n} others like this'
  ];
  var idx = Math.min(names.length, 4);

  return templates[idx].replace(/{name}|{n}/g, function (val) {
    return val === '{name}' ? names.shift() : names.length;
  });
}
```

- **SOLUTION 3**

```
function likes(names) {
  return {
    0: 'no one likes this',
    1: `${names[0]} likes this`,
    2: `${names[0]} and ${names[1]} like this`,
    3: `${names[0]}, ${names[1]} and ${names[2]} like this`,
    4: `${names[0]}, ${names[1]} and ${names.length - 2} others like this`,
  }[Math.min(4, names.length)]
}
```

- **SOLUTION 4**

```
function likes (names) {
  var format = {
    0: "no one likes this",
    1: "{0} likes this",
    2: "{0} and {1} like this",
    3: "{0}, {1} and {2} like this"
  }[names.length] || "{0}, {1} and {n} others like this";

  return format.replace(/{([\dn])}/g, function (_, n) {
    return n == 'n' ? names.splice(2).length : names[parseInt(n, 10)];
  });
}
```

- **SOLUTION 5**

```
function likes(names) {
  names.length === 0 && (names = ["no one"]);
  let [a, b, c, ...others] = names;
  switch (names.length) {
    case 1: return `${a} likes this`;
    case 2: return `${a} and ${b} like this`;
```

```
    case 3: return `${a}, ${b} and ${c} like this`;
    default: return `${a}, ${b} and ${others.length + 1} others like this`;
  }
}
```

#34 - Somme gauche = Somme droite (6 kyu)

- **SOLUTION 1**

```
function findEvenIndex(arr)
{
  sum=(arg)=>arg.reduce((a,v)=>a+v);
  l=arr.length-1;
  for (k=1; k<l; k++) {
    if (sum(arr.slice(0,k))==sum(arr.slice(k+1))) {
      return k;
    }
  }
  return -1;
}
```

- **SOLUTION 2**

```
function findEvenIndex(arr)
{
  for(var i=1; i<arr.length-1; i++) {
    if(arr.slice(0, i).reduce((a, b) =>  a+b) === arr.slice(i+1).reduce((a, b) =>  a+b))
{
      return i;
    }
  }
  return -1;
}
```

#35 - Nombre divisible par 6 (6 kyu)

- **SOLUTION 1**

```
function isDivisibleBy6(s) {
  var v=[...s].reverse().reduce((a,k,i)=>(k!="*") ? a+(3*(i>0)+1)*k : a,0)
  var coeff=3*(s.slice(-1)!="*")+1
  return [...Array(10).keys()].filter(n=>(coeff*n+v)%6==0).map(n=>s.replace('*',n))
}
```

- **SOLUTION 2**

```
const isDivisibleBy6=s=>(r=>(r==1 ? [2,5,8] : r ? [1,4,7] :
[0,3,6,9]).map(n=>s.replace(/\*/g,n)))([...s].reduce((a,b)=>b=='*' ? a : a+
+b,0)%3).filter(s=>!"13579".includes(s.slice(-1)));
```

- **SOLUTION 3**

```
function isDivisibleBy6(s) {
  return ["0123456789"].map( d => { let n=s.replace(/\*/g,d); return isDivBy2(n) &&
isDivBy3(n) ? n : 0 } ).filter(x=>x)
}
```

```
const isDivBy2 = s => "02468".includes(s.slice(-1));
const isDivBy3 = s => [...s].reduce( (t,v) => t+ +v, 0)%3==0;
```

#36 - Nombres narcissiques (6 kyu)

- **SOLUTION 1**

```
function narcissistic( value ) {
  var v=(''+value).split('')
  return v.reduce((a,n)=>a+Math.pow(1*n,v.length),0)==value
}
```

- **SOLUTION 2**

```
function narcissistic( value ) {
  var str = value + '';
  var sum = 0;
  for (var i = 0, len = str.length; i < len; i++){
    sum += Math.pow(+str.substr(i, 1), len);
  }
  return (sum === value);
}
```

#37 - Décodage Morse (6 kyu)

- **SOLUTION 1**

```
decodeMorse = function(morseCode){
  var t=morseCode.split(' ').map((c)=>(c!="") ? MORSE_CODE[c] : " ").join('');
  //t=t.replace(/SOS/g,'S O S');
  while (t.indexOf('  ')>-1) t=t.replace('  ',' ');
  while (t[0]===' ') t=t.slice(1);
  while (t.slice(-1)===' ') t=t.slice(0,t.length-1)
  return t
}
```

- **SOLUTION 2**

```
decodeMorse = function(morseCode){
  function decodeMorseLetter(letter) {
    return MORSE_CODE[letter];
  }
  function decodeMorseWord(word) {
    return word.split(' ').map(decodeMorseLetter).join('');
  }
  return morseCode.trim().split('   ').map(decodeMorseWord).join(' ');
}
```

- **SOLUTION 3**

```
decodeMorse = function(mc) {
  return mc.trim().split('   ').map(function(v) {return v.split(' ').map(function(w)
{return MORSE_CODE[w];}).join('');}).join(' ');
};
```

- **SOLUTION 4**

```
decodeMorse = function(morseCode){
  return morseCode.trim().split(' ').map(a => MORSE_CODE[a] || '
').join('').replace(/\s+/g, ' ');
}
```

- **SOLUTION 5**

```
decodeMorse = function(morseCode){
  return morseCode
    .trim()
    .split(/  | /)
    .map( (code) => MORSE_CODE[code] || ' ')
    .join('');
}
```

#38 - Position des lettres dans l'alphabet (6 kyu)

- **SOLUTION 1**

```
function alphabetPosition(text) {
  return text.toUpperCase().split('').reduce((a,c)=>a+=(c>="A" && c<="Z") ?
c.charCodeAt(0)-64+" " : "","").slice(0,-1);
}
```

- **SOLUTION 2**

```
function alphabetPosition(text) {
  var result = "";
  for (var i = 0; i < text.length; i++){
    var code = text.toUpperCase().charCodeAt(i)
    if (code > 64 && code < 91) result += (code - 64) + " ";
  }

  return result.slice(0, result.length-1);
}
```

- **SOLUTION 3**

```
function alphabetPosition(text) {
  return text
    .toUpperCase()
    .match(/[a-z]/gi)
    .map( (c) => c.charCodeAt() - 64)
    .join(' ');
}
```

- **SOLUTION 4**

```
function alphabetPosition(text) {
  return text
    .toLowerCase()
    .replace(/[^a-z]/g,'')
    .replace(/./g, c => c + " ")
    .replace(/[a-z]/g, c => c.charCodeAt(0) - 96)
    .trim();
```

```
}
```

#39 - Nombre apparaissant un nombre impair de fois (6 kyu)

- **SOLUTION 1**

```
function findOdd(A) {
  k=0;
  while (A.count(A[k])%2==0) k++
  return A[k];
}
```

- **SOLUTION 2**

```
function findOdd(A) {
  var A=A.sort();
  return A.filter(n=>(A.lastIndexOf(n)-A.indexOf(n))%2 ==0)[0];
}
```

- **SOLUTION 3**

```
const findOdd = (xs) => xs.reduce((a, b) => a ^ b);
```

- **SOLUTION 4**

```
function findOdd(A) {
  return A.reduce(function(c,v){return c^v;},0);
}
```

- **SOLUTION 5**

```
const findOdd = arr => arr.reduce(
  (pv, cv) => arr.filter(inv => inv === cv).length % 2 === 1 ? cv : pv);
```

- **SOLUTION 6**

```
function findOdd(arr) {
  return arr.find((item, index) => arr.filter(el => el == item).length % 2)
}
```

#40 - Cryptage avec une lettre sur 2 (6 kyu)

- **SOLUTION 1**

```
encrypt=(t, n) => {
 if (t==null) return null
 while(n-->0) t=t.replace(/.(.)?/g,'$1')+t.replace(/(.).?/g,'$1')
 return t
}

decrypt=(e, n) => {
 if (e==null) return null
 var l=e.length/2|0
 while(n-->0) e=[...e].map((_,k)=>(k%2==0) ? e[l+k/2] : e[(k-1)/2]).join('')
 return e
}
```

- **SOLUTION 2**

```
function encrypt(text, n) {
  for (let i = 0; i < n; i++) {
    text = text && text.replace(/.(.|$)/g, '$1') + text.replace(/(.)./g, '$1')
  }
  return text
}

function decrypt(text, n) {
  let l = text && text.length / 2 | 0
  for (let i = 0; i < n; i++) {
    text = text.slice(l).replace(/./g, (_, i) => _ + (i < l ? text[i] : ''))
  }
  return text
}
```

#41 - Encodeur de lettres dupliquées (6 kyu)

- **SOLUTION 1**

```
function duplicateEncode(word){
  return word
    .toLowerCase()
    .split('')
    .map( function (a, i, w) {
      return w.indexOf(a) == w.lastIndexOf(a) ? '(' : ')'
    })
    .join('');
}
```

- **SOLUTION 2**

```
function duplicateEncode(word) {
  word = word.toLowerCase();
  return word.replace(/./g, m => word.indexOf(m) == word.lastIndexOf(m) ? '(' : ')');
}
```

- **SOLUTION 3**

```
function duplicateEncode(word){
    return word.toLowerCase().replace(/./g, function(match) { return
word.toLowerCase().split(match).length > 2 ? ')' : '(' ;});
}
```

- **SOLUTION 4**

```
function duplicateEncode(word){
  var wArr = [... word.toLowerCase()], counter = {}, result = '';

  for(var i in wArr)
    counter[wArr[i]] = (counter[wArr[i]]+1) || 1;

  for(var j=0; j < wArr.length; j++)
        result += counter[wArr[j]] > 1 ? ')' : '(';

  return (result);
}
```

#42 - File d'attente cinéma (6 kyu)

- **SOLUTION 1**

```
function tickets(peopleInLine){
  var [n25, n50, n100] = [0, 0, 0];
  for (var i = 0; i < peopleInLine.length; i++) {
    switch(peopleInLine[i]) {
      case 25: n25++; break;
      case 50: n50++; n25--; break;
      case 100: n100++; n25--;
        if (n50) n50--; else n25 -= 2; break;
    }
    if ([n25, n50, n100].some(v => v < 0)) return 'NO';
  }
  return 'YES';
}
```

- **SOLUTION 2**

```
function Clerk(name) {
  this.name = name;

  this.money = {
    25 : 0,
    50 : 0,
    100: 0
  };

  this.sell = function(element, index, array) {
    this.money[element]++;

    switch (element) {
      case 25:
        return true;
      case 50:
        this.money[25]--;
        break;
      case 100:
        this.money[50] ? this.money[50]-- : this.money[25] -= 2;
        this.money[25]--;
        break;
    }
    return this.money[25] >= 0;
  };
}

function tickets(peopleInLine){
  var vasya = new Clerk("Vasya");
  return peopleInLine.every(vasya.sell.bind(vasya)) ? "YES" : "NO";
}
```

#43 - Différence entre ensembles (6 kyu)

- **SOLUTION 1**

```
function array_diff(a, b) {
```

```
  return a.filter(function(x) { return b.indexOf(x) == -1; });
}
```

- **SOLUTION 2**

```
function array_diff(a, b) {
  return a.filter(e => !b.includes(e));
}
```

- **SOLUTION 3**

```
array_diff = require("lodash").difference;
```

#44 - Superposition d'intervalles (6 kyu)

- **SOLUTION 1**

```
lineIntersections= (start, end) =>
  start.reduce((t,v,k)=>
    t+start.slice(k+1).reduce((a,u,i)=>
      a+1*(v<=end.slice(k+1)[i] & u<=end[k]),0),0)
```

- **SOLUTION 2**

```
function lineIntersections(start, end) {
  for (var c=0,i=0;i<start.length-1;i++)
     for (var j=i+1;j<start.length;j++)
        if (start[i]<=end[j]&&end[i]>=start[j]) c++;
  return c
}
```

#45 - Retournement des mots de 5 lettres et plus (6 kyu)

- **SOLUTION 1**

```
function spinWords(words){
  return words.split(' ').map(m=>(m.length>4) ? m.split('').reverse().join('') :
m).join(' ');
}
```

- **SOLUTION 2**

```
function spinWords(string){
  return string.replace(/\w{5,}/g, function(w) { return w.split('').reverse().join('')
})
}
```

- **SOLUTION 3**

```
var spinWords = function(string){
  var arr = string.split(' ')
  arr.forEach((x, indx) => x.length >= 5 ? arr[indx] = x.split('').reverse().join('') :
x)
  return arr.join(' ')
```

```
}
```

#46 - Somme des chiffres (6 kyu)

- **SOLUTION 1**

```
function digital_root(n) {
  t=(''+n).split('');
  while (t.length>1) t=t.reduce((a,v)=>a+ +v,0).toString().split('');
  return +t[0]
}
```

- **SOLUTION 2**

```
function digital_root(n) {
  return (n - 1) % 9 + 1;
}
```

#47 - Lettre manquante (6 kyu)

- **SOLUTION 1**

```
function findMissingLetter(array)
{
  p=array[0].charCodeAt(0)
  for (i=1;i<array.length;i++){
    if (array[i].charCodeAt(0)==p+1) {p++} else {return String.fromCharCode(p+1)}
  }
}
```

- **SOLUTION 2**

```
function findMissingLetter(array) {
  const charArray = array.map(x => x.charCodeAt());
  return String.fromCharCode(charArray.find((x, index) => {return x + 1 !==
charArray[index + 1]}) + 1);;
}
```

#48 - Tribonnacci (6 kyu)

- **SOLUTION 1**

```
function tribonacci(signature,n){
  a=[];
  Array(n).fill(0).map((x,k)=>a.push((k<3) ? signature[k] : a.slice(-
3).reduce((b,n)=>b+n)))
  return a
}
```

- **SOLUTION 2**

```
function tribonacci(signature,n) {
  const result = signature.slice(0, n);
  while (result.length < n) {
```

```
    result[result.length] = result.slice(-3).reduce((p,c) => p + c, 0);
  }
  return result;
}
```

- **SOLUTION 3**

```
function tribonacci(s,n){
  var arr = [];
  for(var i=0; i<n; i++) {
    arr.push((i<3) ? s[i] : arr[i-1]+arr[i-2]+arr[i-3]);
  }
  return arr;
}
```

- **SOLUTION 4**

```
function tribonacci(signature, n) {
  while(signature.length < n) {
    signature.push(signature.slice(-3).reduce(sum));
  }
  return signature.slice(0, n);
}

function sum(a, b) { return a + b }
```

#49 - Smileys (6 kyu)

- **SOLUTION 1**

```
function countSmileys(arr) {
 return arr.reduce((a,c)=>a+=(c.match(/^(:|;)(-|~)?(\)|D)/g)==null) ? 0 : 1,0)
}
```

- **SOLUTION 2**

```
const countSmileys = ss => ss.reduce((a, s) => a + /^[:;][-~]?[D)]$/.test(s), 0);
```

- **SOLUTION 3**

```
function countSmileys(arr) {
  return arr.filter(x => /^[:;][-~]?[)D]$/.test(x)).length;
}
```

- **SOLUTION 4**

```
countSmileys=arr=> arr.filter(v => /(:|;)(-|~)?(\)|D)/.test(v)).length;
```

#50 - Lièvre et tortue (6 kyu)

- **SOLUTION 1**

```
function race(v1, v2, g) {
    if (v1>=v2) {return null}
    else {
      t=3600*g/(v2-v1)
      console.log(v1,v2,g,t)
      return [(t/3600)|0, ((t%3600)/60)|0, (t%60)|0]
    }
}
```

- **SOLUTION 2**

```
race=(v1,v2,g)=>(t=g/(v2-v1),t<0?null:[t,t*60%60,t*3600%60].map(Math.floor))
```

#51 - Couleurs HTML vers RGB (6 kyu)

- **SOLUTION 1**

```
function parseHTMLColor(color) {
  var c=(color[0]=="#") ? color : PRESET_COLORS[color.toLowerCase()]
  var [R,G,B]= c.match(/^#(..?)(..?)(..?)/).slice(1).map(s=>(s.length==1) ?
parseInt(s+s,16) : parseInt(s,16))
  return { 'r': R, 'g': G, 'b': B}
}
```

- **SOLUTION 2**

```
function parseHTMLColor(color) {
  var key = color.toLowerCase();
  var rgb = (PRESET_COLORS[key] || key).slice(1);

  if (rgb.length === 3)
    rgb = rgb.replace(/./g, '$&$&');

  var val = parseInt(rgb, 16);

  return {
    r: val / 65536 | 0,
    g: (val / 256 | 0) % 256,
    b: val % 256
  }
}
```

- **SOLUTION 3**

```
function parseHTMLColor(c) {
  if (!c.match("#")) c = PRESET_COLORS[c.toLowerCase()];

  c = c.replace('#', '');

  if (c.length < 6) c = c.replace(/(.)/g, "$1$1");

  return {
```

```
    r: parseInt(c.substring(0, 2), 16),
    g: parseInt(c.substring(2, 4), 16),
    b: parseInt(c.substring(4, 6), 16)
  };
}
```

#52 - Où sont mes parents ? (6 kyu)

- **SOLUTION 1**

```
function findChildren(dancingBrigade){
  return dancingBrigade.toLowerCase().split('').sort().map((v,k,a)=>v!=a[k-1] ?
v.toUpperCase() : v).join('')
};
```

- **SOLUTION 2**

```
function findChildren(dancingBrigade){
  return dancingBrigade.split("")
                       .sort((a,b)=>a.localeCompare(b,"kf",{caseFirst:"upper"}))
                       .join("");

};
```

- **SOLUTION 3**

```
const findChildren = dancingBrigade =>
  dancingBrigade
    .split('')
    .sort((a, b) => a.toLowerCase().localeCompare(b.toLowerCase()) ||
b.localeCompare(a))
    .join('')
```

#53 - Somme de nombres (6 kyu)

- **SOLUTION 1**

```
function suffixSums(arr) {
  s=[]
  arr.reverse().reduce((a,v)=>{s.push(a+v); return a+v},0)
  return s.reverse()
}
```

- **SOLUTION 2**

```
function suffixSums(a) {
  let res = new Array(a.length), s = 0;
  for (let i = a.length - 1; i >= 0; i--)
    res[i] = s += a[i];
  return res;
}
```

#54 - Majuscules et minuscules à chaque mot (6 kyu)

- **SOLUTION 1**

```
majmin=m=>[...m].map((c,k)=>(k%2==0) ? c.toUpperCase() : c.toLowerCase()).join('')
function toWeirdCase(string){
  return string.split(' ').map(m=>majmin(m)).join(' ')
}
```

- **SOLUTION 2**

```
function toWeirdCase(string){
  return string.split(' ').map(function(word){
    return word.split('').map(function(letter, index){
      return index % 2 == 0 ? letter.toUpperCase() : letter.toLowerCase()
    }).join('');
  }).join(' ');
}
```

- **SOLUTION 3**

```
function toWeirdCase(string){
  return string.replace(/(\w{1,2})/g,(m)=>m[0].toUpperCase()+m.slice(1))
}
```

- **SOLUTION 4**

```
const toWeirdCase = str =>
  str.replace(/\b\w*\b/g, w =>
    w.replace(/\w/g, (c, k) =>
      c[k % 2 === 0 ? 'toUpperCase' : 'toLowerCase']()));
```

#55 - Aire d'un triangle (6 kyu)

- **SOLUTION 1**

```
dist=(M,N)=>Math.sqrt((M.x-N.x)*(M.x-N.x)+(M.y-N.y)*(M.y-N.y))
function triangleArea(triangle){
  c=dist(triangle.a,triangle.b)
  b=dist(triangle.a,triangle.c)
  a=dist(triangle.b,triangle.c)
  s=(a+b+c)/2
  console.log(a,b,c,s)
  return Math.sqrt(s*(s-a)*(s-b)*(s-c))
}
```

- **SOLUTION 2**

```
function triangleArea(t) {
  let a = new Point(t.a.x - t.c.x, t.a.y - t.c.y)
  ,   b = new Point(t.b.x - t.c.x, t.b.y - t.c.y);

  return Math.abs(a.x * b.y - a.y * b.x) / 2;
}
```

#56 - Combien d'abeilles sont dans la ruche ? (6 kyu)

- **SOLUTION 1**

```
transpose = m => m[0].map((_,i) => m.map(x => x[i]))
find=(l,str)=>(l.join('').match(new RegExp(str, 'g'))||[]).length
count=h=>h.reduce((a,l)=>a+find(l,'bee')+find(l,'eeb'),0)

howManyBees = function(hive) {
 return (hive==null || hive.length==0) ? 0 : count(hive)+count(transpose(hive))
}
```

- **SOLUTION 2**

```
howManyBees = function(hive) {
  if (!hive || !hive.length)
    return 0;

  let flip = hive[0].map((_, i) => hive.map(row => row[i]));
  let text = hive.concat(flip).map(row => row.join('')).join('|');
  let bees = text.match(/(?=bee|eeb)/g) || [];

  return bees.length;
}
```

#57 - Parcours d'un labyrinthe (6 kyu)

- **SOLUTION 1 (UTILISATION DE LODASH)**

```
function mazeRunner(maze, directions) {
  var dir={"N":[-1,0], "S":[1,0], "E":[0,1], "W":[0,-1]}
  var arr=_.flatten(maze)
  var l = maze.length
  var x=arr.indexOf(2)/l|0
  var y=arr.indexOf(2)%l
  for (var k in directions) {
    x+=dir[directions[k]][0]
    y+=dir[directions[k]][1]
    if (x<0 | x>=l | y<0 | y>=l) return "Dead"
    if (maze[x][y]==1) return "Dead"
    if (maze[x][y]==3) break
  }
  return (maze[x][y]!=3) ? "Lost" : "Finish"
}
```

- **SOLUTION 2**

```
function mazeRunner(maze, directions) {
  var size=maze.length,i=-1,j=-1,di={N:-1,S:1,E:0,W:0},dj={W:-1,E:1,N:0,S:0}
  while(!maze[++i].includes(2));while(maze[i][++j]!=2);
  for(var s of directions){
    i+=di[s],j+=dj[s]
    if(i<0||j<0||i>=size||j>=size||maze[i][j]==1) return "Dead"
    if(maze[i][j]==3) return "Finish"
  }
  return "Lost"
```

```
}
```

#58 - Chez le père Noël (6 kyu)

- **SOLUTION 1**

```
function giftSafety(gift) {
  return [...gift]
  .reduce((a,c,k)=>
    (k<gift.length-2) ?
      a+(c==gift[k+1]|c==gift[k+2]|gift[k+1]==gift[k+2]) : a
  , 0)
}
```

- **SOLUTION 2**

```
function giftSafety(gift) {
  var a = 0;
  for (var i = 0; i < gift.length - 2; i++) {
    if (/(\w).?\1/.test(gift[i] + gift[i+1] + gift[i+2])) a++
  }
  return a
}
```

- **SOLUTION 3**

```
function giftSafety(gift) {
  var r=0
  for(var i=0;i<gift.length-2;i++)
    if(new Set(gift.slice(i,i+3)).size<3) r++
  return r
}
```

- **SOLUTION 4**

```
function giftSafety(gift) {
  safety = gift.match(/(?=(.)\1.|.(.)\2|(.).\3)/g);
  return safety === null ? 0 : safety.length;
}
```

#59 - Heures à partir de secondes (5 kyu)

- **SOLUTION 1**

```
function humanReadable(seconds) {
 return [3600,60,1].map(n=>{t=seconds/n|0; seconds %=n; return (t<10) ? "0"+t : t}).join(':')
}
```

- **SOLUTION 2**

```
p=n=>`0${n}`.slice(-2),humanReadable=(s)=>(m=s/60|0,p(m/60|0)+':'+p(m%60)+':'+p(s%60))
```

#60 - Hashtags (5 kyu)

- **SOLUTION 1**

```
function generateHashtag (str) {
  return str.length > 140 || str === '' ? false :
    "#"+str.replace(/(^| )(\w)/g, s=>s.toUpperCase()).split(' ').join('')
}
```

- **SOLUTION 2**

```
function generateHashtag (str) {
  return str.length > 140 || str === '' ? false :
    '#' + str.split(' ').map(capitalize).join('');
}

function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}
```

#61 - Pig Latin (5 kyu)

- **SOLUTION 1**

```
function pigIt(str){
  return str.split(' ').map(s=>s.slice(1)+s[0]+"ay").join(' ')
}
```

- **SOLUTION 2**

```
function pigIt(str){
  return str.replace(/(\w)(\w*)(\s|$)/g, "\$2\$1ay\$3")
}
```

- **SOLUTION 3**

```
function pigIt(str){
  return str.replace(/\b(\w)(\w+)\b/ig,"$2$1ay");
}
```

#62 - Camel Case (5 kyu)

- **SOLUTION 1**

```
function toCamelCase(str){
 return str.replace(/([-_](\w))/g, c=>c.toUpperCase()[1])
}
```

- **SOLUTION 2**

```
function toCamelCase(str){
  return str.replace(/[-_](.)/g, (_, c) => c.toUpperCase());
}
```

#63 - Trop c'est trop

- **SOLUTION 1**

```
function deleteNth(arr,x){
  return arr.filter((n,k)=>arr.slice(0,k+1).reduce((a,v)=>a+(v==n),0)<=x)
}
```

- **SOLUTION 2**

```
function deleteNth(arr,x) {
  var cache = {};
  return arr.filter(function(n) {
    cache[n] = (cache[n]||0) + 1;
    return cache[n] <= x;
  });
}
```

- **SOLUTION 3**

```
function deleteNth(arr,x){
  var count = {};
  return arr.filter(function(a){
    count[a] = ~~count[a]+1;
    return count[a]<=x;
  })
}
```

- **SOLUTION 4**

```
function deleteNth(arr, x) {
  return arr.filter(
    (e, i) => arr.slice(0, i).filter(e2 => e2 == e).length < x
  );
}
```

- **SOLUTION 5**

```
deleteNth=(a,n,r={})=>a.filter(m=>(r[m]=1+(r[m]||0),r[m]<=n))
```

#64 - Départ – Arrivée (5 kyu)

- **SOLUTION 1**

```
var flapDisplay = function (lines, rotors) {
  rotors.forEach((r, m) =>
    r.forEach((n, i) =>
      lines[m] = lines[m].split('').map((c, j) => (j >= i) ?
ALPHABET[(ALPHABET.indexOf(c) + n) % ALPHABET.length] : c).join('')
   )
  )
  return lines
}
```

- **SOLUTION 2**

```
var flipChar = function(char, flips) {
  return ALPHABET[(ALPHABET.indexOf(char) + flips) % ALPHABET.length];
}

var flapDisplay = function(lines, rotors) {
  return rotors.map(function(rotor, i) {
    var sum = 0;
    return rotor.map(function(flips, j) {
      return flipChar(lines[i][j], sum += flips);
    }).join('');
  });
}
```

#65 - Fibonnaci (5 kyu)

- **SOLUTION 1**

```
function productFib(prod){
  let [a, b] = [0, 1];
  while(a * b < prod) [a, b] = [b, a + b];
  return [a, b, a * b === prod];
}
```

- **SOLUTION 2**

```
function productFib(prod){
  var n = 0;
  var nPlus = 1;
  while(n*nPlus < prod) {
    nPlus = n + nPlus;
    n = nPlus - n;
  }
  return [n, nPlus, n*nPlus===prod];
}
```

#66 - Déplacement sur une carte (5 kyu)

- **SOLUTION 1**

```
function dirReduc(arr){
 s=[];
 for (d of arr){
  switch (d) {
  case 'NORTH': if (s.slice(-1)[0]=='SOUTH') {s.pop()} else {s.push('NORTH')}; break;
  case 'SOUTH': if (s.slice(-1)[0]=='NORTH') {s.pop()} else {s.push('SOUTH')}; break;
  case 'EAST': if (s.slice(-1)[0]=='WEST') {s.pop()} else {s.push('EAST')}; break;
  case 'WEST': if (s.slice(-1)[0]=='EAST') {s.pop()} else {s.push('WEST')}; break;
  }
 }
 return s
}
```

- **SOLUTION 2**

```
function dirReduc(plan) {
```

```
  var opposite = {
    'NORTH': 'SOUTH', 'EAST': 'WEST', 'SOUTH': 'NORTH', 'WEST': 'EAST'};
  return plan.reduce(function(dirs, dir){
      if (dirs[dirs.length - 1] === opposite[dir])
        dirs.pop();
      else
        dirs.push(dir);
      return dirs;
    }, []);
}
```

- **SOLUTION 3**

```
function dirReduc(arr) {
  var str = arr.join(''), pattern = /NORTHSOUTH|EASTWEST|SOUTHNORTH|WESTEAST/;
  while (pattern.test(str)) str = str.replace(pattern,'');
  return str.match(/(NORTH|SOUTH|EAST|WEST)/g)||[];
}
```

- **SOLUTION 4**

```
function isOppo(dir1,dir2) {
    if (dir1 + dir2 === 'SOUTHNORTH') return true;
    if (dir1 + dir2 === 'NORTHSOUTH') return true;
    if (dir1 + dir2 === 'EASTWEST') return true;
    if (dir1 + dir2 === 'WESTEAST') return true;
    return false;
}

function dirReduc(arr){
  var len = arr.length
  for (var i = 0; i < len - 1; i++) {
    if (isOppo(arr[i], arr[i+1])) {
      arr.splice(i,2);
      return dirReduc(arr);
    }
  }
  return arr;
}
```

- **SOLUTION 5**

```
function dirReduc(arr){
  var opposite = { "SOUTH":"NORTH", "NORTH":"SOUTH", "WEST":"EAST", "EAST":"WEST"}
  return arr.reduce(function (a, b, i) {
    opposite[a.slice(-1)] === b ? a.pop() : a.push(b)
    return a
  }, [])
}
```

- **SOLUTION 6**

```
function dirReduc(arr){
  return arr.reverse().reduce(function (memo, v) {
    return memo.length && ['NORTHSOUTH', 'SOUTHNORTH', 'EASTWEST', 'WESTEAST'].indexOf(v
+ memo[0]) >= 0 ? memo.slice(1) : [v].concat(memo)
  }, [])
}
```

#67 - Somme maxi dans un tableau (5 kyu)

- **SOLUTION 1**

```
var maxSequence = function(arr){
  var min = 0, ans = 0, i, sum = 0;
  for (i = 0; i < arr.length; ++i) {
    sum += arr[i];
    min = Math.min(sum, min);
    ans = Math.max(ans, sum - min);
  }
  return ans;
}
```

- **SOLUTION 2**

```
var maxSequence = function(arr){
  var max = 0;
  var cur = 0;
  arr.forEach(function(i){
    cur = Math.max(0, cur + i);
    max = Math.max(max, cur);
  });
  return max;
}
```

- **SOLUTION 3**

```
var maxSequence = function(arr){
      var max = 0;
      for (var i = 0; i < arr.length; i++) {
        for (var j = arr.length; j > i; j--) {
          var total = arr.slice(i,j).reduce(function(a, b){ return a + b; });
          if (max < total) max = total
        }
      }
      return max;
    }
```

#68 - Anagrammes (5 kyu)

- **SOLUTION 1**

```
ordre=(m)=>[...m].sort().join('');
function anagrams(word, words) {
  return words.filter(m=>ordre(m)==ordre(word))
}
```

- **SOLUTION 2**

```
String.prototype.sort = function() {
  return this.split("").sort().join("");
};

function anagrams(word, words) {
  return words.filter(function(x) {
      return x.sort() === word.sort();
```

```
  });
}
```

#69 - Trous entre des nombres premiers (5 kyu)

- **SOLUTION 1**

```
function isPrime(number) {
    var start = 2;
    while (start <= Math.sqrt(number)) {
        if (number % start++ < 1) return false;
    }
    return number > 1;
}
function gap(g, m, n) {
    d=m
    while (d<n) {
      if (isPrime(d)) {
        f=d+1
        while (((f-d)<g) && !isPrime(f) && (f<=n)) {f++}
        if ((f==d+g) && isPrime(f) && (f<=n)) {return [d,f]; break;}
      }
      d++
    }
    return null
}
```

- **SOLUTION 2**

```
function gap(g, m, n) {
    var lastPrime = 0;
    var isPrime = function(x) {
      for (var i=2; i*i<=x; i++) { if (x % i == 0) return false; } return true;
    }

    for(var i = m; i <= n; i++)
        if(isPrime(i)) {
            if(i - lastPrime == g) return [lastPrime, i];
            else lastPrime = i;
        }

    return null;
}
```

- **SOLUTION 3**

```
function isPrime(n) {
   if (isNaN(n) || !isFinite(n) || n%1 || n<2) return false;
   var m = Math.sqrt(n);
   for (var i=2;i<=m;i++) if (n%i==0) return false;
   return true;
}


function gap(g, m, n) {
  let mx = 0;
  for (m, n; m < n; m++) {
    if (isPrime(m)) {
      if (m - mx === g) return [mx, m];
```

```
      mx = m;
    }
  }
  return null;
}
```

- **SOLUTION 4**

```
function gap(g, m, n) {
  let isP = (x, e) => e <= Math.sqrt(x) ? x % e === 0 ? false : isP(x,e+1) : true
  let c = []

  for (let i=m ; i < n ; i++) {
    if (c.length > 1 && c[c.length-1] - c[c.length-2] === g) return c.slice(-2)
    if (i >= n-1) return null
    isP(i, 2) ? c.push(i) : c.slice(-1)
  }
}
```

#70 - Somme carrés diviseurs = carré ? (5 kyu)

- **SOLUTION 1**

```
r=(i,j) => Array(j-i+1).fill(0).map((v,k)=>i+k)
function listSquared(m, n) {
  l=[]
    r(m,n).forEach(p=>{
      s=r(1,Math.sqrt(p)|0).reduce((a,k)=>(p%k==0) ? a+k*k+(p/k)*(p/k) : a,0);
      if (Math.sqrt(p)%1==0) {s-=p}
      if (Math.sqrt(s)%1==0) {l.push([p,s])}
      })
  return l
}
```

- **SOLUTION 2**

```
function listSquared(m, n) {
  var arr = [];
  for (var i = m; i <= n; i++){
    var temp = 0;
    for (var j = 1; j <= i; j++) {
      if ( i % j == 0) temp += j*j;
    };
    if ( Math.sqrt(temp) % 1 == 0) arr.push([i, temp]);
  };
  return arr;
}
```

- **SOLUTION 3**

```
function listSquared(m, n) {
  const divs = _ => [...Array(~~Math.sqrt(_))].reduce((a, b, i) => {
    i++
    return _ % i ? a : a + Math.pow(i, 2) + Math.pow(i == _ / i ? 0 : _ / i, 2)
  }, 0)
  const res = []
  for (let i = m; i <= n; i++) {
    let d = divs(i)
```

```
    if (Math.sqrt(d) % 1 == 0) res.push([i, d])
  }
  return res
}
```

- **SOLUTION 4**

```
function listSquared(m, n) {
  var winners = [];
  for(m; m<=n; m++){
    var divisors = getDivisors(m);
    var sum = divisors.map(n=>n*n).reduce((a,b)=>a+b,0);
    if(Math.sqrt(sum)%1===0) winners.push([m,sum]);
  }
  return winners;
}

function getDivisors(num){
  var arr = [];
  for(var i=1; i<=num; i++){
    if(num%i===0) arr.push(i);
  }
  return arr;
}
```

#71 - PowerSet (5 kyu)

- **SOLUTION 1**

```
function powerset(nums) {
  var s=[]
  var l=nums.length
  for (var k=Math.pow(2,l)-1; k>=0;k--){
    var t=[]
    var bin=k.toString(2)
    bin = "0".repeat(l-bin.length)+bin
    bin.split('').forEach((n,i)=>n%2==0 ? t.push(nums[i]) : '')
    s.push(t)
  }
  return s
}
```

- **SOLUTION 2**

```
const powerset=(n,c=[[]])=>(p=>n.length==1 ? p :
       powerset(n.slice(0,n.length-1),p))(c.concat(c.map(e=>[n[n.length-1]].concat(e))));
```

- **SOLUTION 3**

```
function powerset(nums) {
  return nums.reduceRight(function (car, xs) {
             return car.concat(car.map(function (car) {
                 return [xs].concat(car);
             }));
         }, [[]]);
}
```

- **SOLUTION 4**

```
function powerset(nums) {
  let next = [[]];
  for (let i = nums.length - 1; i >= 0; i--)
    next = next.concat(next.map(s => [nums[i]].concat(s)));
  return next;
}
```

- **SOLUTION 5**

```
function powerset(nums) {
  const resLength = 1 << nums.length;
  let res = Array.from({length: resLength}, () => []);
  for (let i = 0; i < resLength; i++)
    for (let j = 0, m = 1 << nums.length - 1; j < nums.length; j++, m >>= 1)
      if (i & m)
        res[i].push(nums[j]);
  return res;
}
```

#72 - Parenthèses valides (5 kyu)

- **SOLUTION 1**

```
function validParentheses(parens){
return [...parens].reduce((a,c)=> (a>=0) ? a+(c=='(')-(c==')') : -1 , 0) == 0
}
```

- **SOLUTION 2**

```
function validParentheses(parens){
  var re = /\(\)/;
  while (re.test(parens)) parens = parens.replace(re, "");
  return !parens;
}
```

#73 - Nombres binaires négatifs (5 kyu)

- **SOLUTION 1**

```
function intToNegabinary(i) {
  s=''
  k=-2
  while(i!=0) {s=((i%k==0) ? '0' : '1')+s; i+=k*s[0]/2; k*=-2; }
  return (s=='') ? '0' : s;
}

function negabinaryToInt(s) {
  return [...s].reverse().reduce((a,v,k)=>a+v*Math.pow(-2,k),0);
}
```

- **SOLUTION 2**

```
const BASE = -2;
```

```
const negabinaryToInt = (a,b=1/BASE) => a.length ? a.slice(-
1)*(b*=BASE)+negabinaryToInt(a.slice(0,-1),b) : 0 ;
const Schroeppel2 = 0xAAAAAAAA;
const intToNegabinary = i => ( ( i + Schroeppel2 ) ^ Schroeppel2 ).toString(2);
```

- **SOLUTION 3**

```
function intToNegabinary( number ) {
    var Schroeppel4 = 0xAAAAAAAA;
    // Convert to NegaQuaternary String
    return ( ( number + Schroeppel4 ) ^ Schroeppel4 ).toString(2);
}
function negabinaryToInt(s) {
  return [...s].reduce((i,d)=>+d-i*2, 0);
}
```

#74 - Meilleur score du perdant (5 kyu)

- **SOLUTION 1**

```
function bestMatch(ALAHLYGoals, zamalekGoals) {
al=ALAHLYGoals;
za=zamalekGoals;
a=[0,0]
for (k=0;k<al.length;k++){
  if (al[k]-za[k]<al[a[0]]-za[a[0]]) {
    a=[k,al[k]-za[k]]
  } else if ((al[k]-za[k]==al[a[0]]-za[a[0]]) && (za[k]>za[a[0]])) {
    a=[k,a[1]]
  }
}
return a[0]

}
```

- **SOLUTION 2**

```
bestMatch=(a, b)=> b.map((e,i)=>[a[i]-e,e,i]).sort((c,d)=>c[0]<d[0]?-
1:c[0]>d[0]?1:c[1]>d[1]?-1:c[1]<d[1]?1:c[2]-d[2])[0][2];
```

- **SOLUTION 3**

```
function bestMatch(ALAHLYGoals, zamalekGoals) {
  var best = {
    zScoreIndex : 0,
    lowestdiff : ALAHLYGoals[0] - zamalekGoals[0]
  };
  for (i = 0; i < ALAHLYGoals.length; i++)  {
    var diff = ALAHLYGoals[i] - zamalekGoals[i];
    if (diff < best.lowestdiff)  {
      best.lowestdiff = diff;
      best.zScoreIndex = i;
    } else  {
      if (diff == best.lowestdiff && zamalekGoals[i] >  zamalekGoals[best.zScoreIndex])
{
      best.zScoreIndex = i;
      }
```

```
    }
  }
  return best.zScoreIndex;
}
```

- **SOLUTION 4**

```
function bestMatch(a, z) {
  let min=-Infinity,match={index:0,goal:0};
  a.forEach(function(e,i){
    if(z[i]-e>=min){
      if(z[i]-e!=min||z[i]>match.goal){
      match.index=i;
      match.goal=z[i];
      min=z[i]-e;
      }
    }
  });
  return match.index;
}
```

#75 - Animaux écrasés (5 kyu)

- **SOLUTION 1**

```
ANIMALS2=ANIMALS.map(anim=>[...anim].reduce((a,c)=>(c==a.slice(-1)) ? a : a+c, ""))

possible=(p,a)=>{
  var anim=ANIMALS[ANIMALS2.indexOf(a)]
  var photo=[...p.replace(/=/g,'')]
  for (c of anim) {
    if (photo.indexOf(c)>=0) photo[photo.indexOf(c)]=""
    else return false
  }
  return true
}

var roadKill = function(photo) {
  var anim1 = [...photo].reduce((a,c)=>(c==a.slice(-1) | c=="=") ? a : a+c,"")
  var anim2=[...anim1].reverse().join('')
  if (ANIMALS2.includes(anim1)) return (possible(photo,anim1)) ?
ANIMALS[ANIMALS2.indexOf(anim1)] : "??"
  else if (ANIMALS2.includes(anim2)) return (possible(photo,anim2)) ?
ANIMALS[ANIMALS2.indexOf(anim2)] : "??"
  else return "??"
}
```

- **SOLUTION 2**

```
var roadKill = function(photo) {
  if (photo.match(/[^a-z=]/g) || !photo) { return '??' }

  var found = photo.replace(/[^a-z]/g, '')

  for (var j of ANIMALS) {
    if ( compare(j, found) ) { return j }
  }
  return "??"
}
```

```
function compare(w1, w2) {
  for (var j of w1) {
    try { if (
             w2.match(new RegExp(j, 'g')).length <
             w1.match(new RegExp(j, 'g')).length
             )
             { return false }
    }
    catch (TypeError) { return false }
  }
  w1 = process(w1); w2 = process(w2)
  return w1==w2 || w1.split('').reverse().join('')==w2
}

function process(word) {
  return word.replace(/(.)(\1)+/g, s=> s[0])
}
```

#76 - Chaîne dans une chaîne (5 kyu)

- **SOLUTION 1 (PAS OPTIMALE EN VITESSE)**

```
function scramble(str1, str2) {
 var s1=[...str1].sort().join('')
 var s2="("+[...str2].sort().join(').*(')+")"
 return s1.match(new RegExp(s2, ''))!=null
}
```

- **SOLUTION 2**

```
function scramble(str1, str2) {
 var s1=[...str1].sort().join('')
 var s2=[...str2].sort().join('')
 var l2=str2.length
 var k=0
 var i=0
 while (i<l2) {
       t=s1.slice(k).indexOf(s2[i]);
       if (t>=0) {i++; k+=t+1} else {return false}
}
 return true
}
```

- **SOLUTION 3**

```
function scramble(str1, str2) {
  var map={};
  for(var i in str1) {
    map[str1[i]] ? map[str1[i]]++ : map[str1[i]]=1;
  }
  for(var i in str2) {
    if(!map[str2[i]]) return false;
    map[str2[i]]--;
  }
  return true;
}
```

- **SOLUTION 4**

```
function scramble(str1, str2) {
  let alph = str1.split('').reduce((p, c) => {return p[c] = (p[c] || 0) + 1, p;}, {});
  return str2.split('').every(v => alph[v]-- > 0);
}
```

#77 - Données sur 1 octet (5 kyu)

- **SOLUTION 1**

```
function Network(count) {
  this.cameras = [];
  for (var i = 0; i < count; i++)
  this.cameras.push(new Camera(0, - 30));
}
Network.prototype.process = function (byte) {
  var cam = byte & 15
  var ud = 5 * (((byte >> 4) & 1) - ((byte >> 5) & 1))
  var lr = 5 * (((byte >> 7) & 1) - ((byte >> 6) & 1))
  this.cameras[cam].move(lr, ud)
}
function Camera(h, v) {
  this.horizontal = h;
  this.vertical = v;
}
Camera.prototype.move = function (h, v) {
  var horiz = this.horizontal + h
  var vert = this.vertical + v
  this.horizontal = (Math.abs(horiz) > 45) ? Math.sign(horiz) * 45 : horiz
  this.vertical = (Math.abs(vert) > 45) ? Math.sign(vert) * 45 : vert
}
```

- **SOLUTION 2**

```
function Network (count) {
  this.cameras = [];
  for (var i = 0; i < count; i++) {
    this.cameras.push(new Camera(0, -30));
  }
}

Network.prototype.process = function(byte) {
  var camera = this.cameras[byte & 7];
  if (!camera) return;

  camera.move(
    5 * (((byte >> 7) & 1) - ((byte >> 6) & 1)),
    5 * (((byte >> 4) & 1) - ((byte >> 5) & 1))
  );
};

function Camera (h, v) {
  this.horizontal = h;
  this.vertical = v;
}

Camera.prototype.move = function(h, v) {
  this.horizontal = Math.max(-45, Math.min(45, this.horizontal + h));
```

```
  this.vertical = Math.max(-75, Math.min(15, this.vertical + v));
};
```

#78 - Triangle Pascal (4 kyu)

- **SOLUTION 1**

```
function pascalsTriangle(n) {
n--
s=[1]
p=[1]
while(n--) {
  p=p.map((v,k,a)=>a[k+1] ? v+a[k+1] : 1);
  p.unshift(1)
  s=s.concat(p)
}
return s
}
```

- **SOLUTION 2**

```
function pascalsTriangle(n) {
  var pascal = [];
  var idx = 0;

  for ( var i = 0; i < n; i++ ) {
    idx = pascal.length - i;
    for ( var j = 0; j < i+1; j++ ) {
      if ( j === 0 || j === i ) {
        pascal.push(1);
      } else {
        pascal.push( pascal[ idx+j ] + pascal[ idx+j-1 ] );
      }
    }
  }

  return pascal;
}
```

- **SOLUTION 3**

```
function pascalsTriangle(n) {
  for (var a = [[]], i = 0; i < n && (a[i] = []); ++i)
    for (var j = 0; j <= i / 2; j++)
      a[i][j] = a[i][i - j] = j ? a[i - 1][j] + a[i - 1][j - 1] : 1;
  return a.reduce(function(c, p){return c.concat(p)});
}
```

#79 - Parenthèses, accolades et crochets (4 kyu)

- **SOLUTION 1**

```
function validBraces(braces){
  opposite = {'{': '}', '(': ')', '[': ']'};
  s=[]
  for (c of braces){
    if ("({[".includes(c)) {s.push(c)}
```

```
    else if (c==opposite[s.slice(-1)[0]]) {s.pop()}
    else return false;
  }
```

- **SOLUTION 2**

```
function validBraces(braces){
 while(/\(\)|\[\]|\{\}/g.test(braces)){braces = braces.replace(/\(\)|\[\]|\{\}/g,"")}
 return !braces.length;
}
```

- **SOLUTION 3**

```
function validBraces(braces){
  while(braces.indexOf("{}") != -1 || braces.indexOf("()") != -1 || braces.indexOf("[]")
!= -1){
    braces = braces.replace("{}", "").replace("()", "").replace("[]", "");
  }
  return braces.length == 0;
}
```

- **SOLUTION 4**

```
function validBraces(str){
  var prev = "";
  while (str.length != prev.length) {
            prev = str;
            str = str
                .replace("()", "")
                .replace("[]", "")
                .replace("{}", "");
    }
    return (str.length === 0);
}
```

#80 - Simplification d'un polynôme (4 kyu)

- **SOLUTION 1**

```
tri=s=>[...s].sort().join('')
signe=n=>(n<0) ? "" : "+"
alpha=(a,b)=>(a.length<b.length) ? -1 : ((a.length==b.length) && (a<=b)) ? -1 : 1
function simplify(poly){
       elt={}
       s=''
       e=poly.split(/([-+]?\d*)(\w+)+/g).filter(c=>c!="")
       for (i in e) {
              if (typeof e[i]=='string'){
                     if (isNaN(parseInt(e[i]))) {
                            if (e[i]=="+") e[i]=1
                            else if (e[i]=="-") e[i]=-1
                            else e[i]=tri(e[i])
                            } else {
                            e[i]=parseInt(e[i])
                     }
              }
       }
       if (isNaN(e[0])) e.unshift(1)
```

```
        for (i=0;i<e.length;i+=2) {
                if (elt.hasOwnProperty(e[i+1])) elt[e[i+1]]+=e[i]
                else elt[e[i+1]]=e[i]
        }
        ordre=Object.keys(elt).sort(alpha)
        ordre.forEach((c,k)=> {
                if (elt[c]==1) s+=(s.length==0) ? c : "+"+c
                else if (elt[c]==-1) s+="-"+c
                else if (elt[c]!=0) s+=signe(elt[c])+elt[c]+c

        })
        return (s[0]=="+") ? s.slice(1) : s
}
```

- **SOLUTION 2**

```
function simplify(poly){
  var h = {};
  poly.match(/[+-]?[^+-]+/g).forEach(x => {
    var m = x.match(/[a-z]+/)[0],
        k = x.replace(m, '');
    m = m.split('').sort().join('');
    k = Number(/\d/.test(k) ? k : k+'1');
    h[m] = (h[m]||0) + k;
  });
  return Object.keys(h)
    .filter(m => h[m])
    .sort((x, y) => x.length - y.length || (x < y ? -1 : 1))
    .map((m, i) => ({
      sign : h[m] < 0 ? '-' : i > 0 ? '+' : '',
      k : Math.abs(h[m]),
      m : m
    }))
    .map(o => o.sign + (o.k == 1 ? '' : o.k) + o.m).join('');
}
```

- **SOLUTION 3**

```
function simplify(poly){
  return [computeTerms, reduceTerms, termsToString]
    .reduce((result, fn) => fn(result), poly)
}

function computeTerms(poly) {
  return poly.replace(/[-+]/g, '|$&')
    .split('|')
    .filter(e => e)
    .map(term => {
      const arr = [...term]
      const sign = arr[0] === '-' ? -1 : 1
      const abs = Number(arr.filter(c => /\d/.test(c)).join('') || 1)
      return {
        value: sign * abs,
        expr: arr.filter(c => /[a-z]/g.test(c)).sort().join('')
      }
    })
}

function reduceTerms(terms) {
  return terms
```

```
    .reduce((map, term) => map.set(term.expr, (map.get(term.expr) || 0) + term.value),
new Map())
}

function termsToString(terms) {
  return [...terms]
    .sort((t1, t2) => cmpExpr(t1[0], t2[0]))
    .map(tupleToTerm)
    .filter(e => e)
    .join('')
    .replace(/^[+]/, '')
}

function tupleToTerm(tuple) {
  const sign = tuple[1] < 0 ? '-' : '+'
  const value = Math.abs(tuple[1])
  const num = value === 1 ? '' : value.toString()
  return value === 0 ? '' : sign + num + tuple[0]
}

function cmpExpr(a, b) {
  const lngCmp = a.length - b.length
  const abcCmp = a > b ? 1 : -1
  return lngCmp !== 0 ? lngCmp : abcCmp
}
```

- **SOLUTION 4**

```
function simplify(poly){
  let result = '';
  let reg = /(\+\w*)|(-\w*)/g;
  let myArray;
  let obj = {};
  if(poly[0] !== '-' && poly[0] !== '+') {
    poly = '+' + poly;
  }
  while ((myArray = reg.exec(poly)) !== null) {
    time = myArray[0].match(/(\+\d*)|(-\d*)/g)[0];
    if(time === '+') time = '+1';
    if(time === '-') time = '-1';
    item = myArray[0].match(/([a-z]+$)/g)[0].split('').sort().join('');
    obj[item] = (obj[item]||0) + (+time);
  }
  var keys = Object.keys(obj).sort((i, j)=> {
    return i.length - j.length === 0 ? i<j ? -1 : 1 : i.length - j.length;
  });
  for(var i = 0; i<keys.length; i++){
    var num = obj[keys[i]]
    if(num == 0){
      continue;
    }
    num = num < 0 ? (num == -1 ?  '-' : num) : (num == 1 ?  '+' : '+' + num);
    result += num + keys[i];
  }
  return result.replace(/^\+/, '');
}
```

#81 - Grandes factorielles (4 kyu)

- **SOLUTION 1**

```
function mul(num1, num2) {
  num1 = `${num1}`.split('').reverse();
  num2 = `${num2}`.split('').reverse();
  const d = Array(num1.length + num2.length).fill(0);

  for (let i = 0; i < num1.length; i++) {
    for (let j = 0; j < num2.length; j++) {
      d[i + j] += parseInt(num1[i]) * parseInt(num2[j]);
    }
  }

  const numbers = [];
  for(let i = 0; i < d.length; i++){
    const number = d[i] % 10;
    const carry = Math.floor(d[i] / 10);
    if (i + 1 < d.length){
      d[i + 1] += carry;
    }
    numbers.push(number);
  }

  return numbers.reverse().join('').replace(/^0*/, '');
}

function factorial(n){
  s='1'
  for (i=1; i<=n;i++) s=mul(i,s)
  return s
}
```

- **SOLUTION 2**

```
function factorial(n) {
  var res = [1];
  for (var i = 2; i <= n; ++i) {
    var c = 0;
    for (var j = 0; j < res.length || c !== 0; ++j) {
      c += (res[j] || 0) * i;
      res[j] = c % 10;
      c = Math.floor(c / 10);
    }
  }
  return res.reverse().join("");
}
```

- **SOLUTION 3**

```
function factorial(n){
  let result = '1'
  while (n > 1)
    result = multiply(result, n--)
  return result
}

function multiply(str, x) {
```

```
  const resultDigits = []
  let carry = 0

  str.split('').reverse().forEach(char => {
    let intermediate = Number(char) * x + carry
    resultDigits.unshift(intermediate % 10)
    carry = Math.floor(intermediate / 10)
  })
  if (carry > 0)
    resultDigits.unshift(carry)

  return resultDigits.join('')
```

#82 - Conversion du temps (4 kyu)

- **SOLUTION 1**

```
function formatDuration (seconds) {
  var lettres=['year','day','hour','minute','second']
  var codes = [31536000,86400,3600,60,1];
  var t = [0,0,0,0,0];
  for (i in codes) {
    while ( seconds >= codes[i] ) {
      t[i] += 1;
      seconds -= codes[i];
    }
  }
  t=t.map((v,k)=>(v==0) ? '' : v+' '+lettres[k]+((v>1) ? 's' : '')).filter(n=>n!="")
  return (t.length>1) ? t.slice(0,t.length-1).join(', ')+" and "+t.slice(-1) :
((t.length==0) ? 'now' :t.slice(-1)+'');
}
```

- **SOLUTION 2**

```
function formatDuration (seconds) {
  var time = { year: 31536000, day: 86400, hour: 3600, minute: 60, second: 1 },
      res = [];

  if (seconds === 0) return 'now';

  for (var key in time) {
    if (seconds >= time[key]) {
      var val = Math.floor(seconds/time[key]);
      res.push(val += val > 1 ? ' ' + key + 's' : ' ' + key);
      seconds = seconds % time[key];
    }
  }

  return res.length > 1 ? res.join(', ').replace(/,([^,]*)$/,' and'+'$1') : res[0]
}
```

- **SOLUTION 3**

```
function formatDuration (seconds) {
  if(!seconds)return "now";
  var strout = "";
  var s = seconds%60;
  seconds = (seconds-s)/60;
  var m = seconds%60;
```

```
  seconds = (seconds-m)/60;
  var h = seconds%24;
  seconds = (seconds-h)/24;
  var d = seconds%365;
  seconds = (seconds-d)/365;
  var y = seconds;

  var english=[];
  if(y)english.push(y+" year"+(y>1?'s':''));
  if(d)english.push(d+" day"+(d>1?'s':''));
  if(h)english.push(h+" hour"+(h>1?'s':''));
  if(m)english.push(m+" minute"+(m>1?'s':''));
  if(s)english.push(s+" second"+(s>1?'s':''));

  return english.join(", ").replace(/,([^,]*)$/," and$1");

}
```

#83 - Chiffres romains (4 kyu)

- **SOLUTION 1**

```
function solution(number){
var  roman = {M:1000,CM:900, D:500,CD:400,C:100,XC:90,L:50,XL:40,X:10,IX:9,V:5,IV:4,I:1 }

var ans = '';
while(number>0){
    for(a in roman){
        if(roman[a]<=number){ ans += a; number-=roman[a]; break;}

    }
}
return ans;
}
```

- **SOLUTION 2**

```
function solution(number)
{
  var result   = '',
  decimals = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1],
  roman    = ['M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I'];

  decimals.map(function (value, index) {
    while (number >= value) {
      result += roman[index];
      number -= value;
    }
  });

  return result;
}
```

#84 - Énumération des permutations (4 kyu)

- **SOLUTION 1**

```
Next = perm => {
```

```
  var idxEnd = perm.length - 1
  var last = perm[idxEnd]
  var idxCur = idxEnd - 1
  while (idxCur > - 1) {
    val = perm[idxCur]
    if (val < last) break
    last = val
    idxCur -= 1
  }
  if (idxCur < 0) return []
  out = perm.slice(0, idxCur)
  idxPivot = idxCur
  idxCur++
  while (idxCur <= idxEnd & perm[idxCur] > val) idxCur++
  idxFirst = idxCur - 1
  out.push(perm[idxFirst])
  idxCur = idxEnd
  while (idxCur > idxPivot) {
    out.push((idxCur == idxFirst) ? val : perm[idxCur])
    idxCur--
  }
  return out
}


function permutations(string) {
  var s=[]
  var arr=[...string].map(c=>c.charCodeAt(0)).sort((a,b)=>a-b)
  while (arr.length>=1) {
    s.push(arr.map(n=>String.fromCharCode(n)).join(''))
    arr=Next(arr)
  }
  return s
}
```

- **SOLUTION 2**

```
function permutations(string) {
  var arr = string.split(''), tmp = arr.slice(), heads = [], out = [];
  if(string.length == 1) return [string];
  arr.forEach(function(v, i, arr) {
    if(heads.indexOf(v) == -1) {
      heads.push(v);
      tmp.splice(tmp.indexOf(v), 1);
      permutations(tmp.join('')).forEach(function(w) {out.push(v + w);});
      tmp.push(v);
    }
  });
  return out;
}
```

- **SOLUTION 3**

```
function permutations(string) {
  return (string.length == 1) ? [string] : string.split('').map(
     (e, i) => permutations(string.slice(0,i) + string.slice(i+1)).map((e2) => e+e2)
  ).reduce((r,e) => r.concat(e)).sort().filter((e,i,a) => (i==0) || a[i-1] != e);
}
```

- **SOLUTION 4**

```
function permutations(str) {
 return (str.length <= 1) ? [str] :
    Array.from(new Set(
     str.split('')
        .map((char, i) => permutations(str.substr(0, i) + str.substr(i + 1)).map(p => char + p))
        .reduce((r, x) => r.concat(x), [])
     ));
}
```

- **SOLUTION 5**

```
const unique = xs => [ ...new Set(xs) ]
const concat = (a, b) => [ ...a, ...b ]
const drop = i => xs => [ ...xs.slice(0, i), ...xs.slice(i + 1) ]

const permute = (x, i, xs) =>
  combinations(drop(i)(xs)).map(y => x + y)

const combinations = s =>
  s.length === 1 ? [ s ] : [ ...s ].map(permute).reduce(concat)

const permutations = s => unique(combinations(s))
```

#85 - Nombre suivant avec les mêmes chiffres (4 kyu)

- **SOLUTION 1 (MÊME CODE QUE ÉNUMÉRATION DES PERMUTATIONS)**

```
Next = perm => {
  var idxEnd = perm.length - 1
  var last = perm[idxEnd]
  var idxCur = idxEnd - 1
  while (idxCur > - 1) {
    val = perm[idxCur]
    if (val < last) break
    last = val
    idxCur -= 1
  }
  if (idxCur < 0) return -1
  out = perm.slice(0, idxCur)
  idxPivot = idxCur
  idxCur++
  while (idxCur <= idxEnd & perm[idxCur] > val) idxCur++
  idxFirst = idxCur - 1
  out.push(perm[idxFirst])
  idxCur = idxEnd
  while (idxCur > idxPivot) {
    out.push((idxCur == idxFirst) ? val : perm[idxCur])
    idxCur--
  }
  return Number(out.join(''))
}

function nextBigger(n){
  return Next([...n.toString()].map(c=>+c))
}
```

- **SOLUTION 2**

```
const sortedDigits = n => { let arr = n.toString().split(''); arr.sort((a, b) => b - a);
return arr; };

function nextBigger(n){

  let arr = sortedDigits(n);
  let max = parseInt(arr.join(''), 10);

  for(var i = n + 1; i <= max; i++){
    if(sortedDigits(i).every((x, j) => x === arr[j])){
      return i;
    }
  }

  return -1;
}
```

- **SOLUTION 3**

```
function nextBigger(n){
  var arr = n.toString().split("").reverse();
  var i = arr.findIndex((d, _i) => d < arr[_i-1]);
  if (i === -1) { return -1; }
  var subarr = arr.slice(0, i);
  var j = subarr.findIndex((d) => d > arr[i]);
  subarr.splice(j, 1, arr[i]);
  return parseInt(arr.slice(i+1).reverse().concat(arr[j]).concat(subarr).join(""));
}
```

#86 - Mixer 2 chaînes de caractères (4 kyu)

- **SOLUTION 1**

```
obj = (n, s) => {
  var o = {};
  [...s.replace(/[^a-z]/g, '')].map(c => o.hasOwnProperty(c) ? o[c].v++ : o[c] =
{'idx':n, 'v':1})
  return o
}

sort = obj => {
  var sortable = [];
  for (var key in obj) sortable.push([key,obj[key].v, obj[key].idx]);
  return sortable
    .filter(n=>n[1]>1)
    .sort((a, b) => {
      if (a[1]!=b[1]) return b[1]-a[1];
      if (a[2]!=b[2]) return 2*(a[2]>b[2])-1;
      if (a[0]!=b[0]) return 2*(a[0]>b[0])-1;
      return 0
    })
    .map(n=>n[2]+":"+n[0].repeat(n[1])).join('/');
}
mix = (s1, s2) => {
  var o1 = obj('1',s1)
  var o2 = obj('2',s2)
  for (var key in o2) {
```

```
    if (o1.hasOwnProperty(key)) {
      if (o1[key].v<o2[key].v) {o1[key].v=o2[key].v; o1[key].idx='2';}
      else if (o1[key].v==o2[key].v) {o1[key].idx='='}
    } else o1[key]={'idx':'2', 'v': o2[key].v}
  }
  return sort(o1)
}
```

- **SOLUTION 2**

```
function mix(s1, s2) {
  var counter = s => s.replace(/[^a-z]/g,'').split('').sort().reduce((x,y)=> (x[y] = 1 +
(x[y]||0), x),{});
  s1 = counter(s1); s2 = counter(s2);
  var res = [], keys = new Set(Object.keys(s1).concat(Object.keys(s2)));
  keys.forEach(key => {
    var c1 = s1[key]||0, c2 = s2[key]||0, count = Math.max(c1, c2);
    if (count>1) {
      var from = [1, '=', 2][Math.sign(c2-c1)+1];
      var str = [...Array(count)].map(_=>key).join('');
      res.push(from+':'+str);
    }
  });
  return res.sort((x, y) => y.length - x.length || (x < y ? -1 : 1)).join('/');
}
```

#87 - Hauteur de pluie

- **SOLUTION 1**

```
function rainVolume(towers) {
w=0
while (towers.length>1){
  towers=towers.join(',').replace(/^(0,)+|(,0)+$/g,'')
  w+=(towers.match(/,0/g) || []).length
  towers=towers.split(',').map(n=>(n>0) ? n-1 : n)
}
return w
}
```

- **SOLUTION 2**

```
function trapWater(heights){
  let sum = 0;
  for(let i = 1; i < heights.length-1; i++) {
    let lm = Math.max(...heights.slice(0,i)), rm = Math.max(...heights.slice(i));
    let diff = Math.min(lm, rm) < heights[i] ? 0 : Math.min(lm,rm) - heights[i];
    sum += diff;
  }
  return sum;
}
```

- **SOLUTION 3**

```
function trapWater(heights){
  var h=heights.slice(0).sort((a,b)=>b-a), c=0;v=0;
```

```
  for (var i=0; i<heights.length; i++){
      h.splice(h.indexOf(heights[i]),1);
      if (heights[i]>=c) c=Math.min(heights[i],h[0]);
      else v+=c-heights[i];
  }
  return v;
}
```

#88 - Longueur d'une boucle (3 kyu)

- **SOLUTION 1**

```
function loop_size(node){
  var A = node;
  var B = node.getNext();
  while (A != B) {
    A = A.getNext();
    B = B.getNext().getNext();
  }
  var count = 0;
  do {
    count++;
    B = B.getNext();
  }
  while (A != B);
  return count;
}
```

- **SOLUTION 2**

```
function loop_size(node){
  var nodes = [], n = node;

  while (nodes.indexOf(n) === -1) {
    nodes.push(n);
    n = n.getNext();
  }

  return nodes.length - nodes.indexOf(n);
}
```

#89 - Distance de Levensthein (3 kyu)

- **SOLUTION 1**

```
function Dictionary(words) {
  this.words = words;
}

var Levenshtein=(c1,c2)=>{
  var l1=c1.length
  var l2=c2.length
  var d=Array(l1+1).fill(0).map(n=>Array(l2+1).fill(0))
  for (i=0;i<=l1;i++) d[i][0]=i
  for (j=0;j<=l2;j++) d[0][j]=j
  for (i=1;i<=l1;i++) {
```

```
    for (j=1;j<=l2;j++){
      var cout=1*(c1[i-1]!=c2[j-1])
      d[i][j]=Math.min(d[i-1][j]+1,d[i][j-1]+1,d[i-1][j-1]+cout)
    }
  }
  return d[l1][l2]
}

Dictionary.prototype.findMostSimilar = function(term) {
  dist=this.words.map(m=>Levenshtein(m,term))
  return this.words[dist.indexOf(Math.min(...dist))]
}
```

- **SOLUTION 2**

```
function Dictionary(words) {
  this.words = words;
}

Dictionary.prototype.findMostSimilar = function(term) {
  var levenshtein = function(word) {
    if (word === term) {return 0}
    if (term.length === 0) {return word.length}
    if (word.length === 0) {return term.length}
    var v0 = new Array(term.length + 1);
    var v1 = new Array(term.length + 1);
    for (var i=0; i<v0.length; i++) { v0[i] = i; }
    for (var i=0; i<word.length; i++) {
      v1[0] = i+1;
      for (var j=0; j<term.length; j++) {
        var cost = word.charAt(i) === term.charAt(j) ? 0 : 1;
        v1[j+1] = Math.min(v1[j]+1, v0[j+1]+1, v0[j]+cost);
      }
      var tmp = v0;
      v0 = v1;
      v1 = tmp;
    }
    return v0[term.length];
  }
  return this.words.sort(function(a,b){return levenshtein(a)-levenshtein(b)})[0];
}
```

#90 - Rendez-vous entre plusieurs personnes (3 kyu)

- **SOLUTION 1**

```
freeTime=p=> {
  if (p.length>0) {
    var times=p.join(';').match(/(\d+)/g);   // Only numbers
    var free=[]
    for (i=1; i<times.length; i+=2) free.push(1*times[i]+60*times[i-1]) // Times in minutes
    free=free.filter((m,k)=>(m!=free[k+1] && m!=free[k-1])) // 10:00-11:00 & 11:00-13:00 => 10:00-
13:00
    if (free[0]>540) free.unshift(540); else free.shift() // 540='9:00'
    if (free.slice(-1)<1140) free.push(1140); else free.pop() // 1140='19:00'
    return free
  } else { return []}
}

inter=(f1,f2)=>{
```

```
  var commun=[]
  for (i=0; i<f1.length; i+=2) {
    for (j=0; j<f2.length; j+=2) {
      if (f1[i]<f2[j+1] && f1[i+1]>f2[j]) commun.push(Math.max(f1[i],f2[j]),
Math.min(f1[i+1],f2[j+1]))
      if (f2[j]>f1[i+1]) break
    }
  }
  return commun
}

heure=minute=>{h=(minute/60|0); m=minute%60; return ((h<10) ? "0":"")+h+":"+((m<10) ? "0":"")+m}

function getStartTime(schedules, duration) {
  var arr=schedules.map(p=>freeTime(p));
  var possible=arr.reduce((a,i)=>inter(a,i),arr[0])
  for (i=0; i<possible.length; i+=2) {
     if ((possible[i+1]-possible[i])>=duration) return heure(possible[i])
    }
  return null
}
```

- **SOLUTION 2**

```
function getStartTime(schedules, duration) {
  function pad(num) {
    var s = num.toString();
    if (s.length == 1) { s = '0' + s; }
    return s;
  }
  schedules = schedules.map(function(sched) {return sched.map(
    function(times) {return times.map(function(t) {
      var parse = t.match(/^(\d+):(\d+)$/);
      return parseInt(parse[1])*60 + parseInt(parse[2]);
    })}
  )});
  var scan = 540; // 09:00
  while (scan + duration <= 1140) { // 19:00
    for (var b = 0; b < schedules.length; b++) {
      if (schedules[b].length == 0) { continue; }
      if (schedules[b][0][0] < scan + duration) {
        scan = Math.max(scan, schedules[b][0][1]);
        schedules[b].shift();
        break;
      }
    }
    if (b == schedules.length) {
      return pad(Math.floor(scan / 60)) + ':' + pad(scan % 60);
    }
  }
  return null;
}
```

- **SOLUTION 3**

```
function getStartTime(schedules, duration) {
  function toMinutes(s) {
    return s.split(':').reduce(function(h, m) {
      return parseInt(h) * 60 + parseInt(m);
    });
  }
  return schedules.reduce(function(p, n) {
```

```
    return p.concat(n);
  }, [['00:00', '09:00'], ['19:00', '24:00']]).sort().reduce(function(p, n) {
    if (!p.start && toMinutes(p.last) + duration <= toMinutes(n[0])) {
      p.start = p.last;
    }
    p.last = p.last < n[1] ? n[1] : p.last;
    return p;
  }, {last: '00:00', start: null}).start;
}
```

#91 - Chemin le plus court dans un graphe (3 kyu)

- **SOLUTION 1**

```
function navigate(numberOfIntersections, roads, start, finish) {
  if (start == finish) {
    return [start]
  } else {
    // Bellman-Ford
    var d = Array(numberOfIntersections).fill(Number.MAX_VALUE)
    d[start] = 0
    for (var k = 1; k < numberOfIntersections; k++) {
      roads.forEach(a => d[a.to] = Math.min(d[a.to], d[a.from] + a.drivingTime))
    }
    var s = [finish]
    preced = finish
    while (preced!=start) {
      var preced = roads.filter(a => (a.to == preced & d[a.from] + a.drivingTime ==
d[preced]))
      if (preced.length > 0) {
        preced = preced[0].from;
        s.push(preced)
      }
      else {
        return null
      }
    }
    return s.reverse()
  }
}
```

- **SOLUTION 2**

```
function navigate(numberOfIntersections, roads, start, finish) {
  let paths = [], minLen = -1;
  (function traverse(path, pathLen) {
    if (minLen >= 0 && minLen < pathLen) return;
    if (path.length > numberOfIntersections) return;

    const curr = path[path.length - 1];
    if (curr == finish) { paths.push(path); minLen = pathLen; }

    const dirs = roads.filter(r => r.from == curr)
      .sort((a, b) => a.drivingTime - b.drivingTime);
    for (let nxt of dirs) traverse(path.concat(nxt.to), pathLen + nxt.drivingTime)
  }).call(null, [start], 0);

  return paths[paths.length - 1];
}
```

#92 - Distance sur une sphère (3 kyu)

- **SOLUTION 1**

```
NSEW=arr=>arr.match(/[NSEW]/g).map(o=>(o=='N' | o=='E') ? 1 : -1)
coord=arr=>arr.split(',').map(c=>c.match(/\d\d?/g).reduce((a,v,k)=>a+v/Math.pow(60,k),0)
)
sphere=x=>coord(x).map((c,k)=>c*NSEW(x)[k]*Math.PI/180)
function distance(coord1, coord2) {
  var c1=sphere(coord1)
  var c2=sphere(coord2)
  var
dist=6371*Math.acos(Math.sin(c1[0])*Math.sin(c2[0])+Math.cos(c1[0])*Math.cos(c2[0])*Math
.cos(c2[1]-c1[1]))
  return 10*(dist/10|0)
}
```

- **SOLUTION 2**

```
function distance(coord1, coord2){
    function parseCoord(coord){
        var r = /(\d+)° (\d+)' (\d+)" ([NS]), (\d+)° (\d+)' (\d+)" ([EW])/.exec(coord);

        var lat = (parseInt(r[1]) + parseInt(r[2])/60 + parseInt(r[3])/3600) *
(r[4]=="S" ? -1 : 1);
        var lon = (parseInt(r[5]) + parseInt(r[6])/60 + parseInt(r[7])/3600) *
(r[8]=="W" ? -1 : 1);

        return {'lat': deg2rad(lat), 'lon': deg2rad(lon)};
    }

    function deg2rad(deg){
        return deg * Math.PI / 180;
    }

    coord1 = parseCoord(coord1);
    coord2 = parseCoord(coord2);

    var dlon = coord2.lon - coord1.lon,
        dlat = coord2.lat - coord1.lat;

    var a = Math.pow(Math.sin(dlat/2), 2) + Math.cos(coord1.lat) * Math.cos(coord2.lat)
* Math.pow(Math.sin(dlon/2), 2),
        c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a)),
        d = 6371 * c;

    return Math.floor(d/10) * 10;
}
```

- **SOLUTION 3**

```
function degreesToRadians(deg) {
  deg = deg.split(/[°'"] /g);
  deg = (~~deg[0] + deg[1] / 60 + deg[2] / 3600) * (deg[3] == 'N' || deg[3] == 'E' ? 1 :
-1);
  return deg * Math.PI / 180;
}
function distance(coord1, coord2) {
```

```
  var latlon1 = coord1.split(', '), latlon2 = coord2.split(', '), lat1 =
degreesToRadians(latlon1[0]),
      lon1 = degreesToRadians(latlon1[1]), lat2 = degreesToRadians(latlon2[0]), lon2 =
degreesToRadians(latlon2[1]),
      diffLat = lat2 - lat1, diffLon = lon2 - lon1, sinDiffLat = Math.sin(diffLat / 2),
sinDiffLon = Math.sin(diffLon / 2),
      a = sinDiffLat * sinDiffLat + Math.cos(lat1) * Math.cos(lat2) * sinDiffLon *
sinDiffLon;
  return ~~(6371 * 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a)) / 10) * 10;
}
```

- **SOLUTION 4**

```
function distance(coord1, coord2) {
  function rad(degrees) {
    var arr = degrees.split(' ');
    function n() { var s = arr.shift(); return s.substr(0, s.length - 1); }
    return {N: 1, E: 1, S: -1, W: -1}[arr.pop()] * (n()/1 + n()/60 + n()/3600) * Math.PI
/ 180;
  }
  var a = coord1.split(', '), b = coord2.split(', '),
      lat1 = rad(a[0]), lat2 = rad(b[0]),
      lon1 = rad(a[1]), lon2 = rad(b[1]),
      sinproduct = Math.sin(lat1) * Math.sin(lat2),
      cosproduct = Math.cos(lat1) * Math.cos(lat2) * Math.cos(lon2 - lon1),
      deltasigma = Math.acos(sinproduct + cosproduct);
  return 10 * Math.floor(637.1 * deltasigma);
}
```