

Eric Schrafstetter

160 challenges

JavaScript

Niveau facile à avancé

Version du 29 août 2017

A yellow square containing the letters 'JS' in a bold, black, sans-serif font.

Mode d'emploi..... 9

Quelques pistes..... 10

- #1 - Programmation fonctionnelle : map, filter, reduce 10
- #2 - Expressions régulières : match, test, every, replace, RegExp 11

Challenges Codeswars.com 13

- #1 - Nombre de personnes dans le bus (8 kyu)..... 13
- #2 - Nombre de moutons (8 kyu) 13
- #3 - Enlever le premier et le dernier caractère d'une chaîne (8 kyu) 13
- #4 - Nettoyage de chaînes (8 kyu)..... 13
- #5 - Doubler les lettres (8 kyu) 14
- #6 - Supprimer les doublons (8 kyu)..... 14
- #7 - Joueur suivant (8 kyu) 14
- #8 - Pourboire (8 kyu)..... 14
- #9 - Accumulation de lettres (7 kyu) 15
- #10 - Compter en Arara (7 kyu)..... 15
- #11 - Code PIN (7 kyu) 15
- #12 - Mettre chaque 1ere lettre des mots en majuscule (7 kyu)..... 15
- #13 - Nombre de voyelles (7 kyu)..... 15
- #14 - Meeting (7 kyu) 16
- #15 - Compteur (7 kyu)..... 16
- #16 - Le plus long nombre (7 kyu) 16
- #17 - Le mot le plus court (7 kyu)..... 16
- #18 - Mot le plus long (7 kyu)..... 16
- #19 - RVB vers niveaux de gris (7 kyu)..... 17
- #20 - Carré des nombres (7 kyu) 17
- #21 - Carte crédit – 4 derniers chiffres (7 kyu)..... 17
- #22 - Carte crédit – date d'expiration (7 kyu) 17
- #23 - Nombres les plus grands et plus petits d'une liste (7 kyu)..... 18
- #24 - Combien sont plus petits que moi ? (7 kyu)..... 18
- #25 - ADN – Remplacer A par T, C par G et réciproquement (7 kyu)..... 18
- #26 - Nombres vampires (7 kyu) 18
- #27 - Couleurs et associations (7 kyu)..... 18
- #28 - Jason (7 kyu)..... 19
- #29 - Scoutisme (7 kyu) 19
- #30 - La guerre des lettres (7 kyu)..... 19
- #31 - Nouvelle guerre des lettres (7 kyu) 20
- #32 - Chaîne cool (7 kyu) 20
- #33 - Divisible par ? (7 kyu) 20
- #34 - Deviner un mot (7 kyu)..... 21
- #35 - Filtrer une liste (7 kyu) 21
- #36 - Nombre de X et de O (7 kyu)..... 21
- #37 - Moutons perdus (7 kyu) 21
- #38 - Points au tennis (7 kyu) 22
- #39 - Qu'est-ce qui vient après ? (7 kyu)..... 22
- #40 - Les martins-chasseurs (7 kyu) 22
- #41 - « g », la lettre heureuse (7 kyu) 22
- #42 - Ascenseur ou pas ? (7 kyu)..... 23

#43 - Tatouages (7 kyu)	24
#44 - Coupons de réduction (7 kyu)	24
#45 - Taco Bell (7 kyu)	24
#46 - Différence entre 2 collections (7 kyu)	25
#47 - Nombre du milieu (7 kyu).....	25
#48 - Fonction à partir d'une chaîne (7 kyu)	25
#49 - Écran de mobile (7 kyu).....	26
#50 - Jumeaux (7 kyu).....	26
#51 - Distribution d'or (7 kyu)	26
#52 - Médailles (7 kyu)	27
#53 - Shushis (7 kyu).....	27
#54 - Monts et vallées (7 kyu)	28
#55 - Ordre des mots (6 kyu).....	28
#56 - Nombre de 1 en binaire (6 kyu)	28
#57 - Distribution d'électrons (6 kyu).....	28
#58 - Numéros de téléphone (6 kyu).....	29
#59 - Touches d'un piano (6 kyu)	29
#60 - Compression d'une phrase (6 kyu).....	30
#61 - Mots consécutifs (6 kyu)	30
#62 - Construction d'une tour (6 kyu)	30
#63 - Notation Polonaise Inverse (6 kyu)	30
#64 - Contrariant (6 kyu)	31
#65 - Parité (6 kyu)	31
#66 - Distribution de bonbons (6 kyu).....	31
#67 - 10 minutes de promenade (6 kyu)	32
#68 - Likes (6 kyu).....	32
#69 - Trier mes animaux (6 kyu).....	32
#70 - Les rats sourds de Hamelin (6 kyu)	33
#71 - Somme gauche = Somme droite (6 kyu)	33
#72 - Nombre divisible par 6 (6 kyu)	34
#73 - Nombres narcissiques (6 kyu)	34
#74 - Dactylographe (6 kyu)	34
#75 - Décodage Morse (6 kyu)	35
#76 - Position des lettres dans l'alphabet (6 kyu)	35
#77 - Discours politique (6 kyu).....	35
#78 - Nombre apparaissant un nombre impair de fois (6 kyu)	36
#79 - Cryptage et décryptage d'une chaîne (6 kyu)	36
#80 - Encodeur de lettres dupliquées (6 kyu)	36
#81 - File d'attente cinéma (6 kyu).....	37
#82 - Les séniors (6 kyu)	37
#83 - Diversité des langages (6 kyu).....	37
#84 - Différence entre ensembles (6 kyu).....	38
#85 - Superposition d'intervalles (6 kyu).....	38
#86 - Retournement des mots de 5 lettres et plus (6 kyu)	38
#87 - Liste de courses (6 kyu)	38
#88 - Somme des chiffres (6 kyu)	39
#89 - Lettre manquante (6 kyu).....	39
#90 - Nombre unique (6 kyu)	39
#91 - Tribonacci (6 kyu)	39
#92 - Smileys (6 kyu).....	40
#93 - Répétition de lettres (6 kyu).....	40
#94 - Michaël (6 kyu).....	40

#95 - Lièvre et tortue (6 kyu).....	41
#96 - Détection de cycles (6 kyu)	41
#97 - Couleurs HTML vers RGB (6 kyu).....	42
#98 - Où sont mes parents ? (6 kyu)	42
#99 - Somme de nombres (6 kyu)	43
#100 - Majuscules et minuscules à chaque mot (6 kyu)	43
#101 - Codes secrets par mobile (6 kyu)	43
#102 - Substitution de lettres (6 kyu).....	43
#103 - Aire d'un triangle (6 kyu).....	44
#104 - Combien d'abeilles sont dans la ruche ? (6 kyu)	44
#105 - Parcours d'un labyrinthe (6 kyu)	44
#106 - Chez le père Noël (6 kyu)	45
#107 - Faux sites web (6 kyu)	45
#108 - Trop c'est trop (6 kyu)	46
#109 - Cellules cancéreuses (6 kyu)	46
#110 - Gendarmes et voleurs (6 kyu)	47
#111 - Types de données (6 kyu).....	47
#112 - Lettres alternées ? (6 kyu).....	47
#113 - Langage Tick (6 kyu)	48
#114 - Longueur de la clé (6 kyu)	48
#115 - Exercices sur les nœuds (6 kyu et 7 kyu).....	48
#116 - Somme max dans un arbre (6kyu)	50
#117 - Heures à partir de secondes (5 kyu).....	51
#118 - Hashtags (5 kyu)	51
#119 - Pig Latin (5 kyu)	51
#120 - Camel Case (5 kyu)	51
#121 - Départ – Arrivée (5 kyu)	51
#122 - Fibonnaci (5 kyu)	52
#123 - Déplacement sur une carte (5 kyu)	52
#124 - Un problème de poids (5 kyu)	53
#125 - Somme maxi dans un tableau (5 kyu)	53
#126 - Anagrammes (5 kyu)	53
#127 - Trous entre des nombres premiers (5 kyu).....	54
#128 - Somme carrés diviseurs = carré ? (5 kyu).....	54
#129 - PowerSet (5 kyu)	54
#130 - Parenthèses valides (5 kyu).....	55
#131 - Combinaisons téléphoniques (5 kyu)	55
#132 - Nombres binaires négatifs (5 kyu)	56
#133 - Meilleur score du perdant (5 kyu).....	56
#134 - Animaux écrasés (5 kyu).....	56
#135 - Hunger Games au zoo (5 kyu)	57
#136 - Chaîne dans une chaîne (5 kyu)	57
#137 - banana (5 kyu).....	58
#138 - Données sur 1 octet (5 kyu)	58
#139 - Hauteur de pluie (5 kyu).....	59
#140 - Vecteurs (5 kyu).....	59
#141 - NSA et espionnage (5 kyu)	61
#142 - Matrices infinies (5 kyu)	62
#143 - Triangle Pascal (4 kyu).....	63
#144 - Écriture d'intervalles (4 kyu)	63
#145 - Somme d'intervalles (4 kyu).....	63
#146 - Parenthèses, accolades et crochets (4 kyu)	63

#147 - Simplification d'un polynôme (4 kyu)	64
#148 - Grandes factorielles (4 kyu)	64
#149 - Conversion du temps (4 kyu)	64
#150 - Chiffres romains (4 kyu)	65
#151 - Énumération des permutations (4 kyu)	65
#152 - Nombre suivant avec les mêmes chiffres (4 kyu)	65
#153 - Mixer 2 chaînes de caractères (4 kyu).....	65
#154 - Serpent (4 kyu)	66
#155 - Nombre binaire multiple de 3	67
#156 - Longueur d'une boucle (3 kyu).....	67
#157 - Distance de Levenstein (3 kyu)	67
#158 - Rendez-vous entre plusieurs personnes (3 kyu)	68
#159 - Chemin le plus court dans un graphe (3 kyu).....	69
#160 - Distance sur une sphère (3 kyu).....	69

Corrigés 70

#1 - Nombre de personnes dans le bus (8 kyu).....	70
#2 - Nombre de moutons (8 kyu)	70
#3 - Enlever le premier et le dernier caractère d'une chaîne (8 kyu)	71
#4 - Nettoyage de chaînes (8 kyu).....	71
#5 - Doubler les lettres (8 kyu)	71
#6 - Supprimer les doublons (8 kyu).....	72
#7 - Joueur suivant (8 kyu)	72
#8 - Pourboire (8 kyu).....	72
#9 - Accumulation de lettres (7 kyu)	73
#10 - Compter en Arara (7 kyu).....	73
#11 - Code PIN (7 kyu)	74
#12 - Mettre chaque 1ere lettre des mots en majuscule (7 kyu).....	74
#13 - Nombre de voyelles (7 kyu).....	75
#14 - Meeting (7 kyu)	75
#15 - Compteur (7 kyu).....	76
#16 - Le plus long nombre (7 kyu)	77
#17 - Le mot le plus court (7 kyu).....	77
#18 - Mot le plus long (7 kyu).....	78
#19 - RVB vers niveaux de gris (7 kyu).....	78
#20 - Carré des nombres (7 kyu)	79
#21 - Carte crédit – 4 derniers chiffres (7 kyu).....	79
#22 - Carte crédit – date d'expiration (7 kyu)	80
#23 - Nombres les plus grands et plus petits d'une liste (7 kyu).....	80
#24 - Combien sont plus petits que moi ? (7 kyu).....	81
#25 - ADN – Remplacer A par T, C par G et réciproquement (7 kyu).....	81
#26 - Nombres vampires (7 kyu)	82
#27 - Couleurs et associations (7 kyu).....	82
#28 - Jason (7 kyu).....	83
#29 - Scoutisme (7 kyu)	83
#30 - La guerre des lettres (7 kyu).....	85
#31 - Nouvelle guerre des lettres (7 kyu).....	85
#32 - Chaîne cool (7 kyu)	85
#33 - Divisible par ? (7 kyu)	86
#34 - Deviner un mot (7 kyu).....	87
#35 - Filtrer une liste (7 kyu)	87

#36 - Nombre de X et de O (7 kyu).....	88
#37 - Moutons perdus (7 kyu)	88
#38 - Points au tennis (7 kyu).....	89
#39 - Qu'est-ce qui vient après ? (7 kyu).....	89
#40 - Les martins-chasseurs (7 kyu)	89
#41 - « g », la lettre heureuse (7 kyu)	90
#42 - Ascenseur ou pas ? (7 kyu).....	91
#43 - Tatouages (7 kyu)	91
#44 - Coupons de réduction (7 kyu)	92
#45 - Taco Bell (7 kyu)	92
#46 - Différence entre 2 collections (7 kyu)	93
#47 - Nombre du milieu (7 kyu).....	93
#48 - Fonction à partir d'une chaîne (7 kyu)	94
#49 - Écran de mobile (7 kyu).....	94
#50 - Jumeaux (7 kyu).....	95
#51 - Distribution d'or (7 kyu)	95
#52 - Médailles (7 kyu)	96
#53 - Shushis (7 kyu).....	96
#54 - Monts et vallées (7 kyu)	96
#55 - Ordre des mots (6 kyu).....	97
#56 - Nombre de 1 en binaire (6 kyu)	97
#57 - Distribution d'électrons (6 kyu).....	98
#58 - Numéros de téléphone (6 kyu).....	99
#59 - Touches d'un piano (6 kyu)	99
#60 - Compression d'une phrase (6 kyu).....	99
#61 - Mots consécutifs (6 kyu)	100
#62 - Construction d'une tour (6 kyu)	100
#63 - Notation Polonaise Inverse (6 kyu)	101
#64 - Contrariant (6 kyu)	102
#65 - Parité (6 kyu)	102
#66 - Distribution de bonbons (6 kyu).....	103
#67 - 10 minutes de promenade (6 kyu)	104
#68 - Likes (6 kyu).....	105
#69 - Trier mes animaux (6 kyu).....	106
#70 - Les rats sourds de Hamelin (6 kyu)	107
#71 - Somme gauche = Somme droite (6 kyu)	107
#72 - Nombre divisible par 6 (6 kyu)	108
#73 - Nombres narcissiques (6 kyu)	108
#74 - Dactylographe (6 kyu)	109
#75 - Décodage Morse (6 kyu)	110
#76 - Position des lettres dans l'alphabet (6 kyu)	110
#77 - Discours politique (6 kyu).....	111
#78 - Nombre apparaissant un nombre impair de fois (6 kyu)	112
#79 - Cryptage avec une lettre sur 2 (6 kyu)	112
#80 - Encodeur de lettres dupliquées (6 kyu)	113
#81 - File d'attente cinéma (6 kyu).....	114
#82 - Les séniors (6 kyu)	115
#83 - Diversité des langages (6 kyu)	115
#84 - Différence entre ensembles (6 kyu)	116
#85 - Superposition d'intervalles (6 kyu).....	116
#86 - Retournement des mots de 5 lettres et plus (6 kyu)	116
#87 - Liste de courses (6 kyu)	117

#88 - Somme des chiffres (6 kyu)	117
#89 - Lettre manquante (6 kyu).....	118
#90 - Nombre unique (6 kyu)	118
#91 - Tribonnacci (6 kyu)	119
#92 - Smileys (6 kyu).....	119
#93 - Répétition de lettres (6 kyu).....	120
#94 - Michaël (6 kyu).....	120
#95 - Lièvre et tortue (6 kyu).....	121
#96 - Détection de cycles (6 kyu)	121
#97 - Couleurs HTML vers RGB (6 kyu).....	121
#98 - Où sont mes parents ? (6 kyu)	122
#99 - Somme de nombres (6 kyu)	123
#100 - Majuscules et minuscules à chaque mot (6 kyu)	123
#101 - Codes secrets par mobile (6 kyu)	124
#102 - Substitution de lettres (6 kyu).....	124
#103 - Aire d'un triangle (6 kyu).....	125
#104 - Combien d'abeilles sont dans la ruche ? (6 kyu)	125
#105 - Parcours d'un labyrinthe (6 kyu)	126
#106 - Chez le père Noël (6 kyu)	126
#107 - Faux sites web (6 kyu)	127
#108 - Trop c'est trop (6 kyu)	128
#109 - Cellules cancéreuses (6 kyu)	129
#110 - Gendarmes et voleurs (6 kyu)	129
#111 - Types de données (6 kyu).....	130
#112 - Lettres alternées ? (6 kyu).....	130
#113 - Langage Tick (6 kyu)	131
#114 - Longueur de la clé (6 kyu)	131
#115 - Exercices sur les nœuds (6 kyu et 7 kyu).....	132
#116 - Somme max dans un arbre (6kyu)	134
#117 - Heures à partir de secondes (5 kyu).....	134
#118 - Hashtags (5 kyu)	134
#119 - Pig Latin (5 kyu)	135
#120 - Camel Case (5 kyu)	135
#121 - Départ – Arrivée (5 kyu)	136
#122 - Fibonnaci (5 kyu)	136
#123 - Déplacement sur une carte (5 kyu).....	137
#124 - Un problème de poids (5 kyu).....	138
#125 - Somme maxi dans un tableau (5 kyu)	138
#126 - Anagrammes (5 kyu)	139
#127 - Trous entre des nombres premiers (5 kyu).....	139
#128 - Somme carrés diviseurs = carré ? (5 kyu).....	141
#129 - PowerSet (5 kyu)	142
#130 - Parenthèses valides (5 kyu).....	143
#131 - Combinaisons téléphoniques (5 kyu)	143
#132 - Nombres binaires négatifs (5 kyu)	143
#133 - Meilleur score du perdant (5 kyu).....	144
#134 - Animaux écrasés (5 kyu).....	145
#135 - Hunger Games au zoo (5 kyu)	146
#136 - Chaîne dans une chaîne (5 kyu)	147
#137 - banana (5 kyu).....	148
#138 - Données sur 1 octet (5 kyu)	149
#139 - Hauteur de pluie (5 kyu).....	150

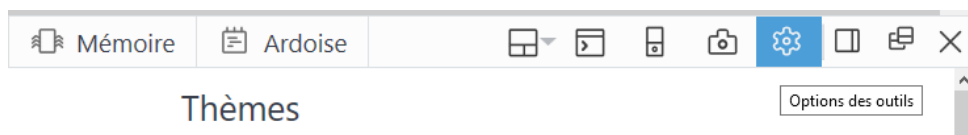
#140 - Vecteurs (5 kyu).....	150
#141 - NSA et espionnage (5 kyu)	151
#142 - Matrices infinies (5 kyu)	153
#143 - Triangle Pascal (4 kyu).....	153
#144 - Écriture d'intervalles (4 kyu)	154
#145 - Somme d'intervalles (4 kyu).....	154
#146 - Parenthèses, accolades et crochets (4 kyu)	155
#147 - Simplification d'un polynôme (4 kyu)	156
#148 - Grandes factorielles (4 kyu)	158
#149 - Conversion du temps (4 kyu)	159
#150 - Chiffres romains (4 kyu)	160
#151 - Énumération des permutations (4 kyu)	161
#152 - Nombre suivant avec les mêmes chiffres (4 kyu)	162
#153 - Mixer 2 chaînes de caractères (4 kyu).....	164
#154 - Serpent (4 kyu)	165
#155 - Nombre binaire multiple de 3	165
#156 - Longueur d'une boucle (3 kyu).....	166
#157 - Distance de Levenstein (3 kyu)	166
#158 - Rendez-vous entre plusieurs personnes (3 kyu)	167
#159 - Chemin le plus court dans un graphe (3 kyu).....	169
#160 - Distance sur une sphère (3 kyu).....	169

Illustrations : <https://openclipart.org> et dessins de l'auteur

MODE D'EMPLOI

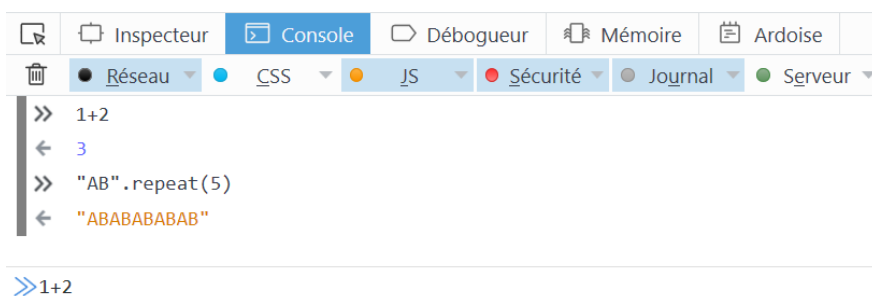
DE QUOI AVEZ-VOUS BESOIN ?

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



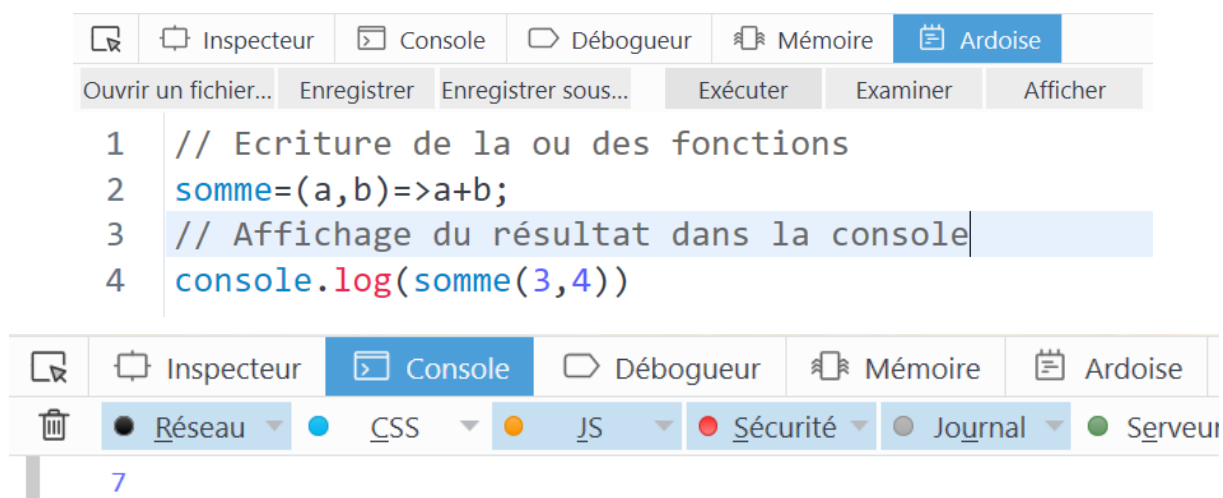
TEST AVEC LA CONSOLE

- 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(paramètres))**
- Cliquez sur **Exécuter** et regardez le résultat dans la console



QUELQUES PISTES

#1 - Programmation fonctionnelle : map, filter, reduce

MAP

Exemple : nous voulons convertir **tous les éléments d'un tableau** écrit en degrés Fahrenheit vers des degrés Celsius. Pour cela, nous allons appliquer la formule $C = \frac{5}{9}(F - 32)$ en utilisant **map** :

```
var fahrenheit = [ 45, 32, 0, 50, 75, 80, 120, 99 ]
var celcius = fahrenheit.map((elem)=> Math.round((elem - 32) * 5 / 9))
celcius → [ 7, 0, -18, 10, 24, 27, 49, 37 ]
```

Paramètres de `map` : `array.map((elem, index, array)=> ... , thisArg)`

Exemple : remplacer les positions impaires du tableau Fahrenheit par des « * ». Pour cela nous avons besoin de la position de l'élément (index).

```
[ 45, 32, 0, 50, 75, 80, 120, 99 ].map((v,k)=>(k%2==0) ? v : '*')
→ Array [ 45, "*", 0, "*", 75, "*", 120, "*" ]
```

FILTER

Exemples : récupérer les nombres pairs du tableau Fahrenheit puis récupérer les éléments qui sont aux positions paires.

```
fahrenheit.filter(n=>n%2==0) // On garde si le nombre est divisible par 2
→ Array [ 32, 0, 50, 80, 120 ]
fahrenheit.filter((n,k)=>k%2==0) // On garde si l'index est divisible par 2
→ Array [ 45, 0, 75, 120 ]
```

On peut bien entendu enchaîner **map** et **filter**, par exemple **ne garder que les éléments** de Fahrenheit supérieurs à 50 puis les convertir en Celsius :

```
fahrenheit.filter(v=>v>50).map(v=>(v-32)*5/9)
→ Array [ 23.88888888888889, 26.666666666666668, 48.888888888888886, 37.22222222222222 ]
```

Pour récupérer à la fois la valeur en Fahrenheit et la valeur en Celsius, on peut penser à :

```
fahrenheit.filter(v=>v>50).map((v,k)=>[fahrenheit[k], (v-32)*5/9])
→ Array [ Array[2], Array[2], Array[2], Array[2] ]
→ [[75, 23.88], [80, 26.66] ... ]
```

Cela fonctionne mais c'est une mauvaise idée car on utilise à l'intérieur de **filter** un élément (**fahrenheit**) qui lui est extérieur. Il vaut mieux écrire :

```
fahrenheit.filter(v=>v>50).map((v,k,arr)=>[arr[k], (v-32)*5/9])
→ Array [ Array[2], Array[2], Array[2], Array[2] ]
```

REDUCE

Exemple classique : faire la somme des éléments de [1,2,3]

```
[1,2,3].reduce((a,v)=>a+v,0)
→ 6
```

L'accumulateur « a » est initialement à 0 puis prend les valeurs « a+v » où « v » parcourt le tableau.

```
[1,2,3].reduce((a,v)=>v,0)
→ 3
```

L'accumulateur vaut 0 puis 1 puis 2 et enfin 3 (il est en fait écrasé à chaque fois).

Exemple : Faire la somme des nombres pairs du tableau Fahrenheit.

```
fahrenheit = [ 45, 32, 0, 50, 75, 80, 120, 99 ]
fahrenheit.filter(v=>v%2==0).reduce((a,v)=>a+v,0) // On filtre puis somme
→ 282
fahrenheit.reduce((a,v)=>(v%2==0) ? a+v : a,0) // On fait le test dans reduce
→ 282
```

Il est important de voir qu'il faut récupérer l'accumulateur à chaque tour. Par exemple, si on veut récupérer les positions de la lettre « a » dans "salut JavaScript", on pourrait penser à :

```
[..."salut JavaScript"].reduce((a,c,k)=>c=='a' ? a.push(k) : a ,[])
→ TypeError: a.push is not a function
```

Cela ne fonctionne pas car `a.push(k)` ajoute bien `k` au tableau mais nous perdons l'accumulateur, plus précisément :

```
a=[1,2] // si 'a' est un tableau
→ Array [ 1, 2 ]
a.push(9) // on ajoute 9 au tableau
→ 3 // en retour nous n'avons pas un tableau mais un entier (d'où l'erreur)
```

Une solution serait d'utiliser `concat` :

```
a=[1,2]
→ Array [ 1, 2 ]
a.concat(9)
→ Array [ 1, 2, 9 ] // on a bien le tableau en retour
```

L'autre possibilité est cette écriture :

```
[..."salut JavaScript"].reduce((a,c,k)=>c=='a' ? (a.push(k),a) : a ,[])
→ Array [ 1, 7, 9 ]
```

#2 - Expressions régulières : match, test, every, replace, RegExp

```
"sAlut JavA5cript".match(/\d/) // Trouver un nombre d'un chiffre dans la chaîne
→ Array [ "1" ]
"sAlut JavA5cript".match(/\d/g) // Recherche globale
→ Array [ "1", "5" ]
```

Exemple : comptez le nombre total d'animaux dans la phrase "Il y a 15 vaches, 3 cochons et 6 chèvres"

```
"Il y a 15 vaches, 3 cochons et 6 chèvres".match(/\d/g)
→ Array [ "1", "5", "3", "6" ] // Recherche des nombres à 1 chiffre uniquement !
"Il y a 15 vaches, 3 cochons et 6 chèvres".match(/\d+/g)
→ Array [ "15", "3", "6" ] // Là c'est ok
"Il y a 15 vaches, 3 cochons et 6 chèvres".match(/\d+/g).reduce((a,v)=>a+v,0)
→ "01536" // Concaténation de chaînes de caractères...
"Il y a 15 vaches, 3 cochons et 6 chèvres".match(/\d+/g).reduce((a,v)=>a+ +v,0)
→ 24 // notez le +v qui permet de convertir la chaîne en nombre
```

```

/j/i.test("sAlut JavaScript") // Y a-t-il un « j » (minuscule ou majuscule) ?
→ true
/d/.test("sAlut JavaScript") // Y a-t-il un nombre ?
→ false
/d/.test("sAlut JavA5cript")
→ true

```

Exemple : Est-ce qu'un mot est écrit entièrement en majuscule ?

```

/[A-Z]/g.test("JAVASCRIPT") // Il y a bien un caractère qui est en majuscule
→ true // donc vrai...
/[A-Z]/g.test("JAVASCRIPt") // idem même si quelque part il y a une minuscule...
→ true // donc le test n'est pas concluant !
 /^[^A-Z]/g.test("JAVASCRIPT") // Y-a-t-il une minuscule ?
→ false
!/[^A-Z]/g.test("JAVASCRIPT") // Négation de « il n'y a pas de majuscule »
→ true // cette fois cela fonctionne

```

```

[..."JAVASCRIPT"].every(c=>/[A-Z]/.test(c)) // Autre technique
→ true // On teste si chaque lettre est une majuscule

```

Évidemment, on pouvait aussi faire :

```

"JAVASCRIPT"==="JAVASCRIPT".toUpperCase()
→ true

```

Exemple : est-ce que tous les nombres du tableau Fahrenheit sont inférieurs à 100 ? Plusieurs techniques :

```

fahrenheit.every(v=>v<100) // Est-ce qu'ils sont tous <100 ?
→ false
fahrenheit.filter(v=>v>=100).length==0 // on filtre ceux plus grands que 100
→ false // et on les compte...
fahrenheit.some(v=>v>=100) // Est-ce qu'un au moins est >=100 ?
→ true

```

Exemple : récupérez les mots d'une liste qui contiennent au moins une majuscule

```

["abc", "abC", "ABC", "cb"].filter(v=>/[A-Z]/.test(v))
→ Array [ "abC", "ABC" ]

```

Exemple : récupérez les mots d'une liste qui contiennent au moins 2 lettres consécutives identiques (en majuscule ou minuscule)

```

["abc", "abBC", "ABC", "Ccb"].filter(v=>/(.)\1/i.test(v))
→ Array [ "abBC", "Ccb" ]

```

Exemple avancé : Des personnes voient des ballons de différentes couleurs. On veut récupérer les nombres maximums d'observations pour chacune des couleurs (par exemple si Jean a vu 12 rouges et Claude 5 rouges, on gardera le fait qu'il y a au moins 12 rouges)

```

var description=["Jean voit 12 rouges, 7 bleus and 8 verts","Max lui voit 5 jaunes, 9
verts, 2 jaunes et 6 bleus","Claude a également vu 5 rouges et 1 gris"]
obj={}
description.map(d=>d.replace(/(\d+) (\w+)s?/g, (_,n,p)=>obj[p] = Math.max(obj[p]||0,+n)))
console.log(obj)
→ Object { rouge: 12, bleu: 7, vert: 9, jaune: 5, gris: 1 }

```

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 2^e 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.

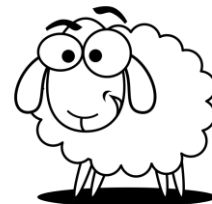
```
number([[10,0],[3,5],[5,8]]) → 5
number([[3,0],[9,1],[4,10],[12,2],[6,1],[7,10]]) → 17
number([[3,0],[9,1],[4,8],[12,2],[6,1],[7,8]]) → 21
```



#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 - Nettoyage de chaînes (8 kyu)

Votre patron a décidé d'économiser de l'argent en achetant un logiciel de reconnaissance de caractères pour numériser d'anciens romans dans votre base de données. Il semble que les mots soient bien saisis, mais vous remarquez rapidement qu'il y a aussi des nombres dans des endroits aléatoires dans le texte :

```
stringClean('! !') → '! !'
stringClean('123456789') → ''
stringClean("E3at m2e2!!") → "(Eat me!!)"
stringClean("Wh7y can't we3 buly the goo0d software3? #cheapskates3") → "Why can't we buy the good software? #cheapskates"
```

Vos collaborateurs vous demandent une solution pour supprimer tous les nombres. Votre programme prendra en entrée une chaîne, devra nettoyer tous les caractères numériques, et renverra une chaîne avec les espaces, caractères et caractères spéciaux ~#\$%^&!@*():; ",., tous intacts.

#5 - 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") → "SSttrriinnngg"  
doubleChar("Hello World") → "HHeellllloo WWoorrlldd"  
doubleChar("1234!_ ") → "11223344!!__ "
```

#6 - Supprimer les doublons (8 kyu)

A partir d'une liste de nombres, supprimez les doublons et renvoyez la liste simplifiée sous forme ordonnée.

```
removeDuplicates([1,1,2,4,5,2,1,2,3,5,5,5])  
→ Array [ 1, 2, 3, 4, 5 ]
```

#7 - Joueur suivant (8 kyu)

Deux joueurs - **black** et **white** - jouent à un jeu qui se compose de plusieurs tours. Si un joueur gagne un tour, il commencera également le prochain. S'il perd, c'est l'autre joueur qui commencera. A partir de la couleur du joueur actuel et du résultat du tour (**true** ou **false**), déterminez qui débutera le prochain tour.

```
whoseMove("black",false) → "white"  
whoseMove("white",true) → "white"  
whoseMove("white",false) → "black"
```

#8 - Pourboire (8 kyu)

Écrivez une fonction qui calcule combien vous devez donner de pourboire en fonction du montant total de la facture et du service. Vous devez tenir compte des notes suivantes:

- Terrible: 0%
- Poor: 5%
- Good: 10%
- Great: 15%
- Excellent: 20%

L'avis sur le service doit être insensible à la casse. Si un avis non reconnu est entré, vous devez renvoyer "Rating not recognised".

Parce que vous êtes une personne sympathique, vous arrondissez toujours à la valeur supérieure, quel que soit le service.

```
calculateTip(20, "ExcellEnt") → 4 // Excellent service, vous donnez 20% de 20 = 4  
calculateTip(26.95, "goOd") → 3 // Bon service, 10% de 26.95 arrondi à 3  
calculateTip(20, "hi") → "Rating not recognised"
```

#9 - Accumulation de lettres (7 kyu)

Ecrire une fonction **accum** telle que :

```
accum("abcd") → "A-Bb-Ccc-Dddd"  
accum("RqaEzty") → "R-Qq-Aaa-Eeee-Zzzzz-Tttttt-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.

#10 - Compter en Arara (7 kyu)

La tribu Arara compte de la façon suivante :

```
1 = anane  
2 = adak  
3 = adak anane  
4 = adak adak  
5 = adak adak anane  
6 = adak adak adak  
7 = adak adak adak anane  
8 = adak adak adak adak
```

Ecrire une fonction qui transforme un nombre en langage Arara.

```
countArara(1) → "anane"  
countArara(3) → "adak anane"  
countArara(8) → "adak adak adak adak"
```

#11 - 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
```



#12 - 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"
```

#13 - 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
```

#14 - 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, vegan: 1}
```

#15 - Compteur (7 kyu)

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

```
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]]
```

#16 - 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
```

#17 - 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
```

#18 - Mot le plus long (7 kyu)

On vous donne une chaîne de mots, renvoyez le mot le plus long. S'il y en a plusieurs, prendre le dernier.

```
longestWord('a b c d e fgh') → "fgh"  
longestWord('one two three') → "three"  
longestWord('red blue grey') → "grey"
```


#19 - 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 : $P = [R, G, B] \Rightarrow [(R + G + B) / 3, (R + G + B) / 3, (R + G + B) / 3]$

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]]
]
```

#20 - 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.

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

Habituellement, lorsque vous achetez quelque chose, on vous demande votre numéro de carte de crédit, votre numéro de téléphone ou votre réponse à une question secrète. Cependant, comme quelqu'un pourrait regarder par-dessus votre épaule, vous ne voulez pas que cela s'affiche sur votre écran. Votre tâche consiste à écrire une fonction **maskify**, qui modifie tous les caractères en '#' sauf les quatre derniers.

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

#22 - Carte crédit – date d'expiration (7 kyu)

Vous recevrez une chaîne de caractères comme entrée. Il aura le mois (2 chiffres) et l'année (2 ou 4 chiffres). Ceux-ci sont séparés par un caractère ("-", "/" ou peut-être plusieurs espaces). Par exemple :

```
02/21
02/21
02 / 2021
02-2021
```

Votre tâche consiste à écrire une fonction qui renvoie **true** ou **false** suivant que la carte est encore valide ou non.

Remarque : si la carte le mois courant, renvoyez **true**.

#23 - 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"
```

#24 - Combien sont plus petits que moi ? (7 kyu)

Écrire une fonction **smaller(arr)** qui donne le nombre de nombres à droite de **arr[i]** qui lui sont inférieurs.

Par exemple :

```
smaller([5, 4, 3, 2, 1]) === [4, 3, 2, 1, 0] // 4 nombres plus petits à droite de 5  
smaller([1, 2, 0]) === [1, 1, 0] // Un plus petit que 1 (0), un plus petit que 2 (0)
```

#25 - 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 chaîne ADN donne son complémentaire.



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

#26 - 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
```

#27 - Couleurs et associations (7 kyu)

La couleur joue un rôle important dans nos vies. La plupart d'entre nous aiment une couleur mieux qu'une autre. Les spécialistes pensent que certaines couleurs ont des significations psychologiques.

Vous recevez en entrée un tableau composé d'une couleur et de son association. La fonction que vous devez écrire doit renvoyer la couleur en tant que « clé » et l'association comme sa « valeur ».

```
colourAssociation([["white", "goodness"], ["blue", "tranquility"]])
→ [{white:"goodness"},{blue:"tranquility"}]
colourAssociation([["red", "energy"],["yellow", "creativity"],["brown" ,
"friendly"],["green", "growth"]])
→ [{red: "energy"},{yellow: "creativity"}, {brown: "friendly"},{green: "growth"}]
```

#28 - Jason (7 kyu)

C'est vendredi 13 et Jason est prêt pour son premier meurtre !

Créez une fonction **killcount**, qui accepte deux arguments : un tableau de couples d'éléments (le nom d'une personne et son intelligence, par exemple ["Tchad", 2]) et un entier représentant l'intelligence de Jason.

```
var counselors = [{"Tchad", 2}, {"Tommy", 9}]
var jason = 7
```

Votre fonction doit renvoyer les noms de toutes les personnes qui ont une intelligence inférieure à Jason et donc qui peuvent être tuées par lui.

```
killcount([['Tiffany',4],['Jack',6],['Megan',7],['Tyler',3]],6) → ['Tiffany', 'Tyler']
```

#29 - Scoutisme (7 kyu)

Le GA-DE-RY-PO-LU-KI est un codage de substitution utilisé dans le scoutisme pour chiffrer les messages. Le cryptage est basé sur une clé courte et facile à retenir. La clé la plus fréquemment utilisée est "GA-DE-RY-PO-LU-KI".

```
G => A
g => a
a => g
A => G
D => E
etc.
```

Les lettres qui ne figurent pas sur la liste restent dans le texte chiffré sans modifications.

```
encode("Ala has a cat") → 'Gug hgs g cgt'
encode("Ala has a cat") → "Gug hgs g cgt"
decode("Gug hgs g cgt") → "Ala has a cat"
encode("ABCD") → "GBCE"
encode("gaderypoluki") → "agedyropulik"
```

#30 - La guerre des lettres (7 kyu)

Il y a deux groupes de lettres hostiles que nous appellerons les lettres de gauche (par exemple « w ») et les lettres de droite (par exemple « m »).

Vous devez écrire une fonction qui accepte une chaîne de combat et donne en retour le gagnant. Lorsque le côté gauche gagne, retournez **Left side wins!** (Le côté gauche gagne !), lorsque c'est le côté droit **Right side wins!** (Le côté droit gagne !), en cas d'égalité retournez **Let's fight again!** (battons-nous encore !)

Voici les lettres et leurs pouvoirs

Côté gauche	Côté droit
w - 4	m - 4
p - 3	q - 3
b - 2	d - 2
s - 1	z - 1

Les autres lettres n'ont aucun pouvoir.

Exemples de combats

```
alphabetWar("z");           → Right side wins!
alphabetWar("zmqmwpbs");    → Let's fight again!
alphabetWar("zzzzs");       → Right side wins!
alphabetWar("wwwwwz");      → Left side wins!
```

#31 - Nouvelle guerre des lettres (7 kyu)

La guerre continue entre les lettres ! Aidez-nous à déterminer quel groupe est plus puissant. Pour cela, créez une fonction qui accepte 2 paramètres et renvoyez celle qui est plus forte. Chaque lettre a son propre pouvoir :

```
A = 1, B = 2, ... Y = 25, Z = 26
a = 0,5, b = 1, ... y = 12,5, z = 13
```

Seuls les lettres alphabétiques peuvent participer à une bataille.

Le mot dont la puissance totale (a + b + c + ...) est la plus grande gagne.

Si les puissances sont égales, renvoyer « Tie ! »

Exemples

```
battle("One", "Two") → "Two"
battle("One", "Neo") → "One"
battle("One", "neO") → "Tie!"
battle("Foo", "BAR") → "Tie!"
battle("Four", "Five") → "Four"
```

#32 - Chaîne cool (7 kyu)

Disons qu'une chaîne est **cool** si elle est formée uniquement par des lettres latines et que l'on n'a jamais deux minuscules ou deux majuscules à des positions adjacentes. Par exemple :

```
coolString("aAaAaAa") → true
coolString("tTzXmLkG") → true
coolString("976") → false
coolString("aBC") → false
```

Écrire une fonction qui à partir d'une chaîne, vérifie si elle est **cool** ou non.

#33 - Divisible par ? (7 kyu)

Créez 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
```

#34 - Deviner un mot (7 kyu)

Un joueur doit deviner un mot dont il connaît 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 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 ».

#35 - 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]
```

#36 - Nombre de X et de O (7 kyu)

Vérifiez si une chaîne a le même nombre de 'x' et 'o'. Vous devez renvoyer un booléen et le résultat doit être insensible à la casse.

```
XO('xo') → true
XO("xxOo") → true
XO("xxxm") → false
XO("Oo") → false
XO("oom") → false
XO("abcdefghijklmnopqrstuvwxyZ") → true
```

#37 - Moutons perdus (7 kyu)

Chaque semaine (vendredi et samedi soir), un fermier et son fils comptent les moutons revenus dans la cour de leur ferme. Ils comptent les moutons revenus le vendredi soir et ceux qui reviennent le samedi (Un mouton qui rentre le vendredi ne repart pas le samedi dans la colline).

Le fermier connaît bien sûr le nombre de moutons qu'il a en tout.

Votre objectif est de calculer la quantité de moutons perdus (non revenus) après les 2 soirées.

```
lostSheep([1,2],[3,4],15) → 5 //1+2=3 revenus vendredi, 7 samedi, il en manque 5
lostSheep([3,1,2],[4,5],21) → 6
lostSheep([5,1,4],[5,4],29) → 10
lostSheep([11,23,3,4,15],[7,14,9,21,15],300) → 178
```

#38 - Points au tennis (7 kyu)

Votre ami vous a invité à regarder un match de tennis. Vous vous ennuyez dès le premier jeu et commencez à chercher quelque chose pour vous divertir. En regardant le tableau des scores, vous vous rendez compte que vous ne savez même pas combien de points ont été gagnés depuis le début du jeu et vous vous décidez à le trouver. Le tableau ci-dessous donnent les points au tennis :

zéro (« love » en anglais) : pour aucun point marqué dans le jeu ;

15 : pour un point marqué ;

30 : pour deux points marqués ;

40 : pour trois points marqués.

```
tennisGamePoints("15-40") → 4 // 1er joueur = 1 pt et 2e joueur = 3 pts
tennisGamePoints("30-all") → 4 // 2 pts pour chaque joueur
tennisGamePoints("love-30") → 2 // 1er joueur = 0 pt et 2e joueur = 2 pts
tennisGamePoints("15-30") → 3 // 1er joueur = 1 pt et 2e joueur = 2 pts
```



#39 - Qu'est-ce qui vient après ? (7 kyu)

Vous recevrez deux entrées : une chaîne de caractères et une lettre. Renvoyez le caractère alphabétique après chaque instance de la lettre voulue (insensible à la casse).

S'il y a un nombre, une ponctuation ou un trait de soulignement après la lettre, il ne doit pas être renvoyé.

```
comes_after("Pirates say arrrrrrrrr.", 'r') → 'arrrrrrrr'
comes_after("Free coffee for all office workers!", 'F') → 'rfeofi'
comes_after("king kUnta is the sickest rap song ever kNown k!", 'k') → 'iUeN'
comes_after("p8tice makes pottery p0rfect!", 'p') → 'o'
comes_after("d8u d._ rly 2dls", 'D') → ''
comes_after("nothing to be found here", 'z') → ''
```

#40 - Les martins-chasseurs (7 kyu)

Une famille de kookaburras (martin-chasseur) est dans mon jardin. Je ne peux pas les voir tous, mais je peux les entendre ! Le truc pour compter les kookaburras est d'écouter attentivement :

- Les mâles font HaHaHa ...

- Les femelles font hahaha ...

Et ils alternent toujours mâles / femmes

```
kookaCounter("") → 0
kookaCounter("hahahahaha") → 1
kookaCounter("hahahahahaHaHaHa") → 2
kookaCounter("HaHaHahahaHaHa") → 3
```



#41 - « g », la lettre heureuse (7 kyu)

Nous dirons que "g" est une lettre heureuse dans une chaîne donnée, s'il y a un autre "g" immédiatement à droite ou à gauche de celle-ci.

Pour str = "gg0gg3gg0gg", la sortie doit être **true**

Pour str = "gog", la sortie doit être **false**.

```
gHappy("ggg") → true
gHappy("gggg") → true
gHappy("umwho cia q6z onb kbs") → true
gHappy("ggg ggg g ggg") → false
gHappy("good grief") → false
```

#42 - Ascenseur ou pas ? (7 kyu)

John vit au nième étage d'un immeuble. Chaque matin, il descend le plus rapidement possible pour aller à son travail qu'il adore. Il a deux manières de descendre : marcher ou prendre l'ascenseur.

Lorsque John utilise l'ascenseur, il suit les étapes suivantes

1. Attendre l'ascenseur qui va aller de l'étage m à l'étage n où il habite
2. Attendre que la porte de l'ascenseur s'ouvre et entrer
3. Attendre qu'elle se referme
4. Attendre que l'ascenseur descende à l'étage 1
5. Attendre que la porte s'ouvre et sortir

(Les temps d'entrée / sortie de l'ascenseur seront ignorés)

On vous donne les éléments suivants

n : nombre entier. Le niveau où habite de John

m : nombre entier. Le niveau où est l'ascenseur

Vitesses : un ensemble d'entiers. Il contient quatre entiers $[a, b, c, d]$

a : Les secondes requises lorsque l'ascenseur monte ou descend d'un étage

b : Les secondes nécessaires pour que la porte s'ouvre

c : Les secondes nécessaires pour que la porte se ferme

d : Les secondes nécessaires pour que John descende un étage à pied

Aidez John à calculer le temps le plus court arriver au niveau 1.

Exemples

- Pour $n = 5$, $m = 6$ et vitesses = $[1,2,3,10]$, la sortie devrait être 12.

En effet, avec l'ascenseur il faudra : $1 + 2 + 3 + 4 + 2 = 12$ (1s pour que l'ascenseur arrive, 2s pour l'ouverture, 3s pour la fermeture, 4s pour descendre et 2s pour l'ouverture). A pieds il lui faudrait $4 \times 10 = 40$ s.

- Pour $n = 1$, $m = 6$ et vitesses = $[1,2,3,10]$, la sortie devrait être 0.

John est déjà à 1 étage, donc il part directement de chez lui...

- Pour $n = 5$, $m = 4$ et vitesses = $[2,3,4,5]$, la sortie devrait être de 20.

John descend en marchant : $5 \times 4 = 20$

c = cheese
g = guacamole
s = salsa

De plus, peu importe les ingrédients, il y aura toujours une coquille (shell)

Voici quelques exemples :

```
tacofy("") → ['shell', 'shell']  
tacofy("a") → ['shell', 'beef', 'shell']  
tacofy("ggg") → ['shell', 'guacamole', 'guacamole', 'guacamole', 'shell']  
tacofy("ogl") → ['shell', 'beef', 'guacamole', 'lettuce', 'shell']  
tacofy("yjdkpwrzto") → ['shell', 'tomato', 'beef', 'shell']
```

#46 - Différence entre 2 collections (7 kyu)

Trouvez les différences entre 2 listes de caractères. Il s'agit des caractères présents dans une liste et pas dans l'autre, et réciproquement. Renvoyez la liste de ces éléments par ordre alphabétique. Attention, les listes peuvent contenir plusieurs fois les mêmes caractères.

```
a = ["a", "b", "z", "d", "e", "d"];  
b = ["a", "b", "j", "j"];  
diff(a, b) → ["d", "e", "j", "z"]
```

#47 - Nombre du milieu (7 kyu)

On vous donne une liste de 3 nombres, trouvez le rang de celui qui est au milieu des 2 autres.

```
gimme([2, 3, 1]) → 0
```

En effet, le chiffre 2 (de rang 0) est bien entre 1 et 3

```
gimme([5, 10, 14]) → 1
```

En effet, le nombre 10 (de rang 1) est bien entre 5 et 14

#48 - Fonction à partir d'une chaîne (7 kyu)

Créez une fonction `runYourString` qui à partir d'un argument et d'un objet contenant un paramètre et une fonction écrite sous la forme d'une chaîne de caractères, renvoie le résultat voulu. Par exemple :

```
var arg = 4, // argument  
    obj = {  
      param: 'num', // nom du paramètre dans la fonction ci-dessous  
      func: 'return Math.sqrt(num)' // description de la fonction dans une chaîne  
    };  
  
runYourString(arg, obj) // on doit trouver 2 qui est la racine carrée de 4  
→ 2
```

Voici d'autres exemples :

```
runYourString(123, { param: 'a', func: 'return a' }) → 123  
runYourString(10, { param: 'a', func: 'return a === 10' }) → true  
runYourString(123, { param: 'radius', func: 'return (radius > 0) ? Math.round((Math.PI *  
Math.pow(radius, 2))*100)/100 : false;' }) → 47529.16
```

#49 - Écran de mobile (7 kyu)

Vous souvenez-vous des anciens claviers des mobiles ? Vous souvenez-vous également de l'inconvénient pour écrire ? Eh bien, ici, vous devez calculer le nombre de touches que vous devez taper pour écrire un mot spécifique. Voici la disposition :

1	2 abc	3 def
4 ghi	5 jkl	6mno
7 pqrs	8 tuv	9wxyz
*	0	#

Exemples

```
mobileKeyboard("123") → 3 (1+1+1)
mobileKeyboard("abc") → 9 (2+3+4) // Taper 2 fois pour obtenir « a »
mobileKeyboard("codewars") → 26 (4+4+2+3+2+2+4+5) // 4 fois pour le « c »
```

Ne pas oublier les touches spéciales * et #.

#50 - Jumeaux (7 kyu)

Agent 47, vous avez une nouvelle tâche ! Parmi les citoyens de la ville sont cachés 2 jumeaux très dangereux. Votre tâche est de les identifier et d'éliminer !

A partir d'un ensemble d'entiers, votre tâche est de trouver les deux mêmes nombres et de renvoyer l'un d'entre eux, par exemple dans le tableau [2, 3, 6, 34, 7, 8, 2] la réponse est 2.

S'il n'y a pas de jumeaux dans la ville - renvoyez **null**.

```
elimination([2,5,34,1,22,1]) → 1
elimination([2,2,34,1,22]) → 2
elimination([2,5,34,1,22]) → null
```

#51 - Distribution d'or (7 kyu)

Sur le terrain, deux chercheurs d'or A et B ont trouvé de l'or en même temps. Ils ont décidé d'utiliser des règles simples pour la distribution :

Ils divisent l'or en n piles qu'ils mettent en ligne.

La quantité de chaque pile et l'ordre des piles sont aléatoires.

Ils comparent la quantité d'or sur la pile à l'extrême gauche avec celle à l'extrême droite.

Ils choisissent toujours la plus grande pile. C'est-à-dire que si par exemple la gauche est 1 et la droite est 2, ils prendront 2.

Si les deux côtés sont égaux, ils prennent la pile de gauche.

Étant donné un nombre entier de piles, et supposons que A prend toujours en premier, calculez le montant final obtenu par chacun. Ce sera un tableau à deux éléments [montant pour A, montant pour B].

Exemple

Pour la distribution `gold`= [4,2,9,5,2,7], la sortie devrait être [14, 15]. En effet :

Le tas de plus à gauche est 4, le tas le plus à droite est 7, A choisi le plus grand → 7

Maintenant, le tas de plus à gauche est 4, le tas les plus à droite est 2, B choisit le plus grand → 4

Maintenant, le tas de plus à gauche est 2, le tas les plus à droite est 2, A choisi le plus grand → 2 (7+2=9)

Maintenant, le tas de plus à gauche est 9, le tas les plus à droite est 2, B choisit le plus grand → 9 (4+9=13)

Maintenant, le tas de plus à gauche est 5, le tas les plus à droite est 2, A choisi le plus grand → 5 (9+5=14)

Il reste 2 qui va pour B → 2+13=15

Deux autres exemples

```
distributionOf([4,7,2,9,5,2]) → [11,18]
```

```
distributionOf([10,1000,2,1]) → [12,1001]
```

#52 - Médailles (7 kyu)

On vous donne le temps d'achèvement d'un challenge par un joueur ainsi que les temps maxi pour obtenir une médaille d'or (**Gold**), d'argent (**Silver**) ou de bronze (**Bronze**). Ecrire une fonction qui renvoie la médaille obtenue ou **None** si le temps était trop long pour gagner quelque chose.

```
evilCodeMedal("00:30:00", "00:15:00", "00:45:00", "01:15:00") → "Silver"
```

```
evilCodeMedal("01:15:00", "00:15:00", "00:45:00", "01:15:00") → "None"
```

```
evilCodeMedal("00:00:01", "00:00:10", "00:01:40", "01:00:00") → "Gold"
```

```
evilCodeMedal("90:00:01", "60:00:02", "70:00:03", "80:00:04") → "None"
```

```
evilCodeMedal("03:15:00", "03:15:00", "03:15:01", "03:15:02") → "Silver"
```

```
evilCodeMedal("99:59:58", "99:59:57", "99:59:58", "99:59:59") → "Bronze"
```

#53 - Shushis (7 kyu)

Sam a ouvert un nouveau restaurant à sushis. Il utilise une technologie de reconnaissance visuelle qui permet d'enregistrer le nombre et la couleur des assiettes de sushis prises par un client. Par exemple, si un client a mangé 3 assiettes rouges, l'ordinateur renverra la chaîne 'rrr'.

Actuellement, Sam ne sert que des sushis sur des assiettes rouges ou transparentes pour les condiments tels que le gingembre et le wasabi - l'ordinateur les note comme un espace ('rrr r' désigne 4 assiettes de sushis et une assiette de condiments).

Sam souhaiterait votre aide pour calculer le montant total qu'un client doit payer lorsqu'il demande la facture. Le calcul est le suivant :

Plaques rouges de sushi ('r') : 2 \$ chacune, mais si un client mange 5 assiettes, la 5ème est gratuite.

Condiments (' ') : gratuit.

```
totalBill('rr') → 4
```

```
totalBill('rr rrr') → 8
```

```
totalBill('rr rrr rrr rr') → 16
```

```
totalBill('rrrrrrrrrrrrrrrrrrrr rr r') → 34
```

```
totalBill('') → 0
```

#54 - Monts et vallées (7 kyu)

A partir d'une liste de nombres, trouvez les monts et les vallées. Un mont est un nombre plus grand que les 3 qui sont à sa gauche et les 3 à sa droite. De même, une vallée est un nombre plus petit que les 3 de chacun de ses côtés. Un exemple :

```
[10,20,30,40,30,20,10,11,12,13,14,15,16,15,14,13]
[      40  ,  10      ,      16      ]
```

40 est un mont car plus grand que les 3 à gauche (10,20,30) et plus grand que les 3 à droite (30,20,10)

De même pour 10 et 16 qui sont des vallées.

```
peakAndValley([10,20,30,40,30,20,10,11,12,13,14,15,16,15,14,13]) → [40,10,16]
```

```
peakAndValley([50,84,49,47,80,87,87,53,76,30,10]) → [47]
```

```
peakAndValley([45,94,41,76,29,96,28,13,84,69,25]) → [96,13]
```

```
peakAndValley([1,16,63,78,53,78,42,39,46,88,49,96,58,82]) → [39]
```

```
peakAndValley([49,97,76,56,96,88,65,20,14,93,32]) → []
```

#55 - Ordre des mots (6 kyu)

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

Remarque : Les nombres peuvent être entre 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("") → ""
```

#56 - 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
```

#57 - 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]
```

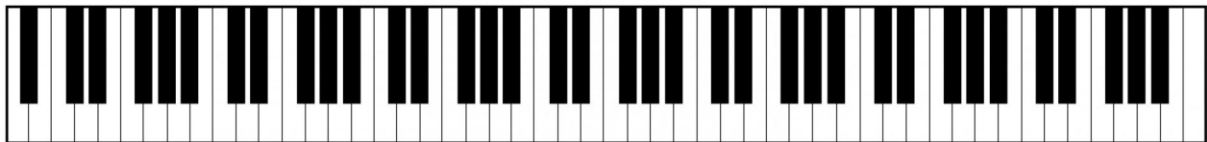
#58 - Numéros de téléphone (6 kyu)

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"
```

#59 - Touches d'un piano (6 kyu)

Voici les 88 touches d'un piano :



La touche n°1 complètement à gauche est blanche, la 2^e noire, la 3^e et 4^e blanches, la 5^e noire etc.

La touche n°88 complètement à droite est blanche et la 89^e sera pour nous à nouveau la touche n°1.

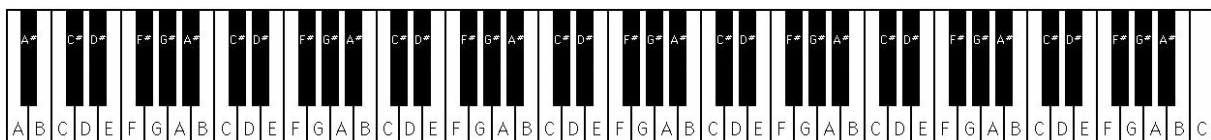
Vous jouez en tapant successivement sur les touches 1 puis 2 puis 3 etc.

Compte tenu du numéro sur lequel vous vous êtes arrêté, serez-vous sur une touche noire ou sur une touche blanche ? Si vous vous êtes arrêté à 92, vous êtes allé au bout des touches 1 à 88, puis vous avez continué sur la n°1, de sorte que vous serez sur la quatrième touche qui est blanche.

Votre fonction recevra un nombre entier quelconque et devra renvoyer la chaîne **black** ou **white** .

```
black_or_white_key(1) → "white"
black_or_white_key(12) → "black"
black_or_white_key(42) → "white"
black_or_white_key(100) → "black"
black_or_white_key(2017) → "white"
```

Reprendre le programme précédent en donnant cette fois le nom de la touche :



"A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"

```
which_note(1) → "A"
which_note(12) → "G#"
which_note(42) → "D"
which_note(100) → "G#"
which_note(2017) → "F"
```

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

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

Hello World 2017 ! → HelloWorld

#61 - Mots consécutifs (6 kyu)

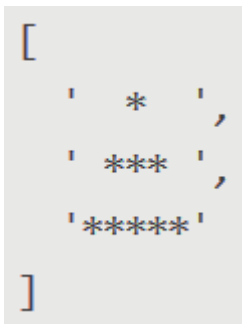
Vous recevez un tableau de chaînes de caractères et un nombre entier k. Votre tâche consiste à renvoyer la première chaîne la plus longue composée de k chaînes consécutives prises dans le tableau. Exemple :

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

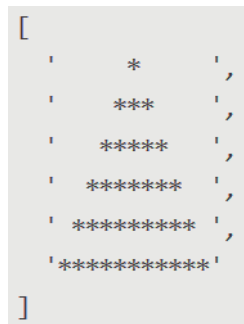
Soit n est la taille du tableau, si $n = 0$ ou $k > n$ ou $k \leq 0$ vous devez renvoyer "".

#62 - 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

#63 - Notation Polonaise Inverse (6 kyu)

RPN : https://fr.wikipedia.org/wiki/Notation_polonaise_inverse



Calculatrice HP 355S en mode RPN

L'expression « $3 \times (4 + 7)$ » peut s'écrire en Notation Polonaise Inverse (NPI ou RPN) sous la forme « 4 {Ent} 7 + 3 × » ({Ent} pour la touche Entrée). Les données en entrée seront séparées par des espaces.

```
solvePostfix("2 3 +") → 5 // 2+3=5
solvePostfix("2 8 -") → -6 // 2-8=-6
solvePostfix("4 2 /") → 2 // 4/2=2
solvePostfix("10 5 / 7 + 3 ^ 10 -") → 719 // (10/5 + 7)^3-10=719
solvePostfix("8 3 4 ^ +") → 89 // 8+3^4=89
```

#64 - 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!"
```

#65 - 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
```

#66 - Distribution de bonbons (6 kyu)

À la maternelle, l'institutrice 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'institutrice 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 pour que le partage tombe juste.



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 de bonbons à la fin du jeu

```
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]
```

#67 - 10 minutes de promenade (6 kyu)

Vous habitez dans une ville où tous les blocs de maisons 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']) → false
isValidWalk(['w']) → false // Ne dure pas 10' et pas de retour
isValidWalk(['n','n','n','s','n','s','n','s','n','s']) → false // Pas de retour
```

#68 - 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"
```



#69 - Trier mes animaux (6 kyu)

Considérons la classe suivante :

```
Var Animal = {
  Nom: "Chat",
  NumberOfLegs: 4
}
```


Écrivez une fonction qui accepte une liste d'objets de type **Animal** et renvoie une nouvelle liste. Cette nouvelle liste devra classer les animaux par nombre de jambes puis par ordre alphabétique du nom.

Si **null** est passé, la fonction doit renvoyer **null**. De même, si la liste est vide, la fonction doit renvoyer une liste vide.

```
sortAnimal([ { name: "Cat", numberOfLegs: 4 }, { name: "Snake", numberOfLegs: 0 },  
{ name: "Dog", numberOfLegs: 4 }, { name: "Pig", numberOfLegs: 4 },  
{ name: "Human", numberOfLegs: 2 }, { name: "Bird", numberOfLegs: 2 } ])
```

```
→ [ { name: 'Snake', numberOfLegs: 0 }, { name: 'Bird', numberOfLegs: 2 }, { name:  
'Human', numberOfLegs: 2 }, { name: 'Cat', numberOfLegs: 4 }, { name: 'Dog',  
numberOfLegs: 4 }, { name: 'Pig', numberOfLegs: 4 } ]
```

#70 - Les rats sourds de Hamelin (6 kyu)

Un joueur de flûte a proposé à la ville de Hamelin de la débarrasser de ses rats. Malheureusement certains de ces rats sont sourds et vont dans la mauvaise direction ! Voici les notations :

P : joueur de flûte

O~ : rat allant vers la gauche

~O : rat allant vers la droite

Exemples :

~O~O~O~O P → Aucun rat sourd

P O~ O~ ~O O~ → 1 rat sourd

~O~O~O~OP~O~OO~ → 2 rats sourds

Ecrire une fonction qui admet en paramètre les rats et le joueur de flûte et renvoie le nombre de rats sourds.

```
countDeafRats("~O~O~O~O P") → 0
```

```
countDeafRats("P O~ O~ ~O O~") → 1
```

```
countDeafRats("~O~O~O~OP~O~OO~") → 2
```

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

Vous avez 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 fonctionne, retournez -1.

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}) sont tous deux égaux à 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
```

#72 - 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" → []
"1234567890123456789012345678*0" →
["123456789012345678901234567800",
"123456789012345678901234567830",
"123456789012345678901234567860"]
```

#73 - 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
```

#74 - Dactylographe (6 kyu)

John est un dactylographe. Il a l'habitude de taper : il n'utilise jamais la touche **Shift** pour changer de boîtier mais uniquement **Caps Lock**. À partir d'une chaîne de caractères, votre tâche consiste à compter combien de fois le clavier a été utilisé par John.

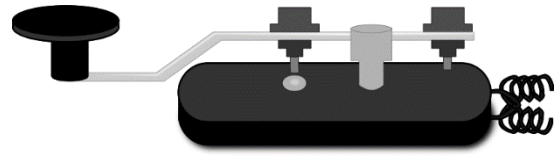
Vous pouvez supposer qu'au début le témoin **Caps Lock** n'est pas allumé.

```
typist("a") → 1
typist("aa") → 2
typist("A") → 2 // Caps Lock + « a »
typist("AA") → 3
typist("aA") → 3 // « a » + Caps Lock + « a »
typist("Aa") → 4
typist("BeiJingDaXueDongMen") → 31
typist("AAAAaaBBBBbbABAB") → 21
typist("AmericanRAILWAY") → 18
typist("AaAaAa") → 12
typist("DFjfkdaB") → 11
```



#75 - 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 intérêt 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 `··· --- ···`. 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"
```

#76 - 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"
```

#77 - Discours politique (6 kyu)

Vous devez créer une fonction pour détecter si un discours politique est bien de votre président ou non. Nous savons que les discours du président ont beaucoup de voyelles supplémentaires.

L'idée est de calculer le nombre de voyelles répétées plus d'une fois de suite divisé par le nombre total de voyelles du discours. Plus ce nombre sera élevé, plus il sera probable que le discours soit bien du président.

Exemples

```
trumpDetector("I will build a huge wall") → 0 // pas de voyelle répétée
trumpDetector("HUUUUUGEEEE WAAAAALL") → 4 // 4 U en trop + 3E +5A = 12 / 3 voyelles
trumpDetector("MEXICAAAAAANS GOOOO HOOME") → 2.5 // 7A+3O+2O+3E=15 / 6 voyelles
trumpDetector("America NUUUUKEEEE Oooobaaaamaaaa") → 1.89
trumpDetector("listen migrants: IIII KIIIDD YOOUUU NOOOOOTT") → 1.56
```

#78 - 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
```

#79 - 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!"
```

#80 - 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" → ")()())"
"(( @" → ")()("
```

#81 - 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 le faire 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 le n°2`



#82 - Les séniors (6 kyu)

Créer une fonction qui donne la liste des programmeurs les plus âgées, par exemple si :

```
var list = [
  { firstName: 'Gabriel', lastName: 'X.', country: 'Monaco', continent: 'Europe', age:
49, language: 'PHP' },
  { firstName: 'Odval', lastName: 'F.', country: 'Mongolia', continent: 'Asia', age: 38,
language: 'Python' },
  { firstName: 'Emilija', lastName: 'S.', country: 'Lithuania', continent: 'Europe',
age: 19, language: 'Python' },
  { firstName: 'Sou', lastName: 'B.', country: 'Japan', continent: 'Asia', age: 49,
language: 'PHP' },
];

findSenior(list) // va renvoyer ceux qui ont 49 ans
→ [
  { firstName: 'Gabriel', lastName: 'X.', country: 'Monaco', continent: 'Europe', age:
49, language: 'PHP' },
  { firstName: 'Sou', lastName: 'B.', country: 'Japan', continent: 'Asia', age: 49,
language: 'PHP' },
]
```

#83 - Diversité des langages (6 kyu)

Trois langages de programmation sont représentés : Python, Ruby et JavaScript. Renvoyez **true** si dans la liste des programmeurs, aucun langage n'est représenté plus de 2 fois par rapport à un autre et **false** sinon. Par exemple, s'il y a 6 programmeurs Python et 2 en Ruby, Python est 3 fois plus représenté que Ruby et on renverra donc **false**. Autre exemple :

```
var list = [
  { firstName: 'Daniel', lastName: 'J.', country: 'Aruba', continent: 'Americas', age:
42, language: 'Python' },
```

```

    { firstName: 'Kseniya', lastName: 'T.', country: 'Belarus', continent: 'Europe', age:
22, language: 'Ruby' },
    { firstName: 'Jayden', lastName: 'P.', country: 'Jamaica', continent: 'Americas', age:
18, language: 'JavaScript' },
    { firstName: 'Joao', lastName: 'D.', country: 'Portugal', continent: 'Europe', age:
25, language: 'JavaScript' }
  ];
isLanguageDiverse(list) → true

```

#84 - 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]

```

#85 - 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

```

#86 - 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"

```

#87 - Liste de courses (6 kyu)

Calculez le coût d'une liste d'achats à l'aide d'une fonction. Cette dernière prend en arguments une liste de listes, par exemple:

```
shoppingListCost([["Orange Juice", 2],["Chocolate", 4],["Pears", 8]])
```

Le liste d'achats comprend dans chaque sous-liste le nom et la quantité de chaque élément acheté, par exemple ici 2 jus d'oranges, 4 chocolats, 8 poires.

A cela s'ajoute une variable globale épicerie (**groceries**) dont voici le contenu fixe :

```
var groceries={ 'Orange Juice': { price: 1.5, discount: 10, bogof: false },
```

```
Chocolate: { price: 2, discount: 0, bogof: true },
Sweetcorn: { price: 4, discount: 20, bogof: true },
Apples: { price: 6, discount: 0, bogof: false },
Pears: { price: 2, discount: 50, bogof: false }
```

Cet objet contient le prix de l'article, la réduction actuellement appliquée et précise s'il est en «1 acheté - 1 gratuit» (bogof : buy one get one free).

Retournez le coût de la liste de courses avec 2 décimales. Si la liste d'entrée ne contient aucun élément, renvoyez zéro.

Vous pouvez supposer que toutes les listes sont valides et contiennent des éléments inclus dans l'épicerie. Chaque élément n'apparaîtra qu'une seule fois.

```
shoppingListCost(["Chocolate", 3],["Apples", 8],["Orange Juice", 15],["Pears",1])
→ 73.25 // Choco= 2 (1 gratuit)*2 + Apples= 8*6 + Orange= 15*1.5*0.9 + Pears= 1*2*0.5
shoppingListCost(["Sweetcorn", 12],["Pears", 6],["Apples", 5]) → 55.2
shoppingListCost(["Pears", 4],["Chocolate", 87],["Sweetcorn", 3]) → 98.4
shoppingListCost(["Orange Juice", 100]) → 135
```

#88 - 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
```

#89 - 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'
```

#90 - Nombre unique (6 kyu)

On vous donne en entrée une liste d'au moins 3 nombres dont un seul est unique. Le retrouver !

```
findUniq([ 1, 1, 1, 2, 1, 1 ]) → 2
findUniq([ 0, 0, 0.55, 0, 0 ]) → 0.55
```

#91 - 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 commençons notre séquence Tribonacci avec [1,1,1] (signature), 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 **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]
```

#92 - Smileys (6 kyu)

Étant donné un tableau (**arr**) donné en argument, écrire une fonction **countSmileys** qui renverra le nombre total de visages souriants.

Règles pour un visage souriant :

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

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

Tout visage souriant doit avoir une bouche souriante qui doit être soit) soit D.

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

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

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



#93 - Répétition de lettres (6 kyu)

Ecrire une fonction qui compte le nombre de lettres ou chiffres distincts qui apparaissent plus d'une fois dans une chaîne de caractères. Le paramètre d'entrée contient seulement des lettres (majuscules et minuscules) ou des chiffres.

Exemples :

```
"abcde" → 0 # pas de caractères multiples
"aabbcd" → 2 # 'a' et 'b'
"aabBcde" → 2 # 'a' et 'b'
"aA11" → 2 # 'a' et '1'
```

```
duplicateCount("") → 0
duplicateCount("abcde") → 0
duplicateCount("aabbcd") → 2
```

#94 - Michaël (6 kyu)

A partir d'un texte, par exemple :


```
inputText = "Michael, how are you? - Cool, how is John Williamns and Michael Jordan? I don't know but Michael Johnson is fine. Michael do you still score points with LeBron James, Michael Green AKA Star and Michael Wood?";
```

Obtenez tous les noms de famille des Michaël. Le résultat devrait être:

```
["Jordan", "Johnson", "Green", "Wood"]
```

Remarques :

- Le prénom sera toujours **Michael** avec majuscule 'M'.
- Il y aura toujours un caractère d'espace entre 'Michael' et le nom de famille.
- Le nom de famille sera toujours un mot, en commençant par une lettre majuscule et en continuant avec des minuscules.
- Il y aura toujours au moins un 'Michael' avec un nom de famille valide dans le texte de saisie.

```
getMichaelLastName(inputText) → ["Jordan", "Johnson", "Green", "Wood"]
```

#95 - 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**.

#96 - Détection de cycles (6 kyu)

#97 - Pour une version plus abstraite, voir #155 - Nombre binaire multiple de 3

▪ SOLUTION 1

```
multipleOf3Regex = /^(1(01*0)*1|0)*$/  
Longueur d'une boucle (3 kyu) page 165.
```

On considère une suite $x_{n+1} = f(x_n)$ avec une valeur initiale x_0 . Lorsqu'il existe $i \neq j$ tels que $x_i = x_j$ la suite est périodique.

Par exemple dans la suite : 2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ... Le cycle est 6, 3, 1 et se répétera forcément puisque $f(1) = 6$, $f(6) = 3$ et $f(3) = 1$.

Soit μ la position de la première valeur du cycle et λ la longueur du cycle.

Dans notre exemple $\mu = 2$ (position du 6) et $\lambda = 3$ (longueur entre deux 6 consécutifs).

Construire une fonction qui renverra $[\mu, \lambda]$ à partir d'une séquence donnée sous la forme de liste. Tout le tableau sera supposé valide, c'est-à-dire associé à une fonction déterministe f . Cela signifie que la séquence bouclera lorsqu'une valeur sera atteinte une seconde fois. (Vous traitez deux cas : pas de répétition comme dans $[1,2,3,4]$ ou une répétition comme $[1,2,1,2]$, il n'y aura pas de cas hybrides comme $[1,2,1,4]$). Lorsqu'il n'y a pas de répétition, retournez $[\]$.

```
cycle([2,3,4,2,3,4]) → [0,3]
cycle([1,2,3,4]) → [ ]
```

#98 - 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", "turquoise" (utilisez PRESET_COLORS).

PRESET_COLORS=

```
{aliceblue: '#f0f8ff', black: '#000000', blanchenalmond: '#ffebcd', blue: '#0000ff', blueviolet: '#8a2be2', brown: '#a52a2a', burlywood: '#deb887', cadetblue: '#5f9ea0', chartreuse: '#7fff00', chocolate: '#d2691e', coral: '#ff7f50', cornflowerblue: '#6495ed', cornsilk: '#fff8dc', crimson: '#dc143c', cyan: '#00ffff', darkblue: '#00008b', gold: '#ffd700', goldenrod: '#daa520', gray: '#808080', grey: '#808080', green: '#008000', greenyellow: '#adff2f', honeydew: '#f0fff0', linen: '#fafae6', magenta: '#ff00ff', maroon: '#800000', navy: '#000080', oldlace: '#fdf5e6', olive: '#808000', olivedrab: '#6b8e23', orange: '#ffa500', orangered: '#ff4500', orchid: '#da70d6', red: '#ff0000', rosybrown: '#bc8f8f', royalblue: '#4169e1', sandybrown: '#f4a460', seagreen: '#2e8b57', seashell: '#fff5ee', sienna: '#a0522d', silver: '#c0c0c0', skyblue: '#87ceeb', slateblue: '#6a5acd', tomato: '#ff6347', turquoise: '#40e0d0', violet: '#ee82ee', wheat: '#f5deb3', white: '#ffffff', whitesmoke: '#f5f5f5', yellow: '#ffff00', yellowgreen: '#9acd32'}
```

```
ParseHTMLColor ('# 80FFA0') → {r: 128, g: 255, b: 160}
```

```
ParseHTMLColor ('# 3B7') → {r: 51, g: 187, b: 119}
```

```
ParseHTMLColor ('LimeGreen') → {r: 50, g: 205, b: 50}
```

#99 - 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'amuse 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, les minuscules pour leurs enfants. Par exemple les enfants de la mère "A" sont "aaaa".

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

Tâche :

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

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

#100 - Somme de nombres (6 kyu)

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

$$b[0] = a[0] + a[1] + \dots + a[n-2] + a[n-1]$$
$$b[1] = a[1] + \dots + a[n-2] + a[n-1]$$

...

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

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]
```

#101 - 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"
```

#102 - Codes secrets par mobile (6 kyu)



Pour verrouiller et déverrouiller les pièces de sa maison ainsi que les différents appareils électroniques, monsieur Safety utilise son téléphone en tapant un code spécifique. Voici quelques-uns de ses codes secrets :

```
unlock("Nokia") → 66542  
unlock("Voiture") → 8648873  
unlock("Porte") → 76783
```

Une fois que vous aurez compris le principe, piratez son système en créant une fonction qui trouvera les bons mots de passe pour chaque objet ou pièce.

#103 - Substitution de lettres (6 kyu)

Créez une fonction (sans paramètre) qui renvoie un objet dans lequel toutes les clés seront les lettres de l'alphabet en minuscules et par ordre alphabétique. La valeur de chaque clé sera également une lettre minuscule, qui devra être sélectionnée au hasard. Parce que c'est aléatoire, il est possible que la lettre originale et la lettre substituée soient identiques. Voici des exemples :

```
randomSub()
→ Object { a: "i", b: "c", c: "h", d: "d", e: "y", f: "z", g: "j", h: "w", i: "x", j:
"v", 16 de plus... }
randomSub()
→ Object { a: "x", b: "o", c: "y", d: "d", e: "h", f: "m", g: "r", h: "j", i: "b", j:
"w", 16 de plus... }
```

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

Les objets **Point** ont des attributs **x** et **y**. Les objets **Triangle** ont les attributs **a**, **b**, **c** décrivant leurs coins, chacun d'eux étant un point.

Écrivez une fonction qui calcule l'aire d'un triangle donné (avec 6 décimales).

```
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
```

#105 - 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 donc à 2 abeilles)

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



Réponse : 5

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

Réponse : 8

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



Réponse : 11

La ruche peut être vide ou **null**.

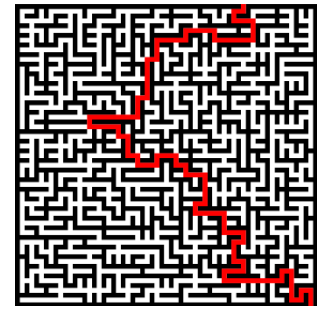
#106 - Parcours d'un labyrinthe (6 kyu)

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"

```

#107 - 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 = "aaaaaa" → 5 // Les 5 triplets « aaa » resteront inchangés après brassage
gift = "cat" → 0

```

#108 - Faux sites web (6 kyu)

John veut créer un site Web et donc lui donner un nom. L'idée de John est de trouver un site célèbre puis modifiez un caractère.

Vous devez coder la fonction **goodName**, qui accepte un nom en paramètre, c'est le nom du site célèbre en suivant les règles suivantes :

1) Si le nom du site Web contient "o", remplacez-le en "0" comme ça

```
codewars.com → c0dewars.com
```

2) Si le nom du site Web contient "l", remplacez-le en "1" comme ça :

```
leetcode.com → 1eetcode.com
```

3) Si le nom du site possède une sous-chaîne de deux caractères consécutifs identiques, l'un d'entre eux sera supprimé comme ça :

```
leetcode.com → letcode.com
```

```
facebook.com → facebok.com
```

```
fighter2000.com → fighter200.com
```

4) Le nom de domaine de niveau supérieur ne doit pas être modifié, tel que .com .net .org etc.

5) John veut voir toutes les possibilités.

6) Le tableau renvoyé doit être trié (en ordre Unicode). Si le nom du site ne peut pas changer de caractère, renvoyez un tableau vide.

```
goodName("codewars.com") → ["c0dewars.com"]
goodName("microsoft.com") → ["micr0soft.com", "microso0ft.com"]
goodName("leetcode.com") → ["1eetcode.com", "leetc0de.com", "letcode.com"]
goodName("goodlink.com") → ["g0odlink.com", "go0dlink.com", "godlink.com", "goodlink.com"]
goodName("fighter2000.com") → ["fighter200.com"]
goodName("pex4fun.com") → []
goodName("xqoomjlieggggg.cn") → ["xq0omjlieggggg.cn", "xqo0mjlieggggg.cn",
"xqomjlieggggg.cn", "xqoomjlieggggg.cn", "xqoomjlieggggg.cn"]
```

#109 - Trop c'est trop (6 kyu)

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]
```

#110 - Cellules cancéreuses (6 kyu)

Votre tâche est d'écrire une fonction qui coupe des cellules cancéreuses du corps.

Les cellules cancéreuses sont divisées en deux types :

- Étape avancée, écrite avec la lettre **C** en majuscule
- Étape initiale, écrite avec la lettre **c** en minuscule

Le reste des cellules est divisée comme suit :

- Cellule normale, écrite avec une lettre minuscule
- Cellule importante, écrite avec une lettre majuscule

Conditions :

- Les cellules importantes ne doivent pas être ôtées.
- Les cellules cancéreuses en état avancé doivent être découpées avec des cellules adjacentes si celles-ci peuvent être ôtées

.

En entrée vous avez une chaîne (représentant un corps), supprimez les caractères "cancéreux" (selon les règles décrites) et renvoyez le corps guéri.

```
cutCancerCells('acb') → 'ab'
cutCancerCells('aCb') → ''
cutCancerCells('acCb') → 'a'
```

```

cutCancerCells('acCcb') → 'ab'
cutCancerCells('ab') → 'ab'
cutCancerCells('aCZ') → 'Z'
cutCancerCells('BCE') → 'BE'
cutCancerCells('sjCmwOqC') → 'sw0'

```

#111 - Gendarmes et voleurs (6 kyu)

Des personnes forment une file d'attente devant un distributeur de billets. Certaines personnes doivent retirer de l'argent, ceux sont des gens ordinaires; d'autres veulent voler de l'argent, ceux sont des voleurs et enfin les autres sont des policiers. Ils cherchent les voleurs.

Étant donné une file d'attente au format chaîne, comme ceci: "X1X#2X#XX". Où # représente une personne ordinaire; "X" un voleur et les chiffres 1-9 représente le policier. La valeur numérique représente la zone de surveillance du gendarme. Par exemple, 1 signifie qu'il peut voir 1 personne devant lui et 1 personne à l'arrière.

Tous les voleurs dans la ligne de vue de la police seront capturés !

Votre tâche est de calculer le nombre de voleurs capturés.

Exemples

Pour la file d'attente = "X1X#2X#XX", la sortie devrait être de 3.

```

X1X#2X#XX
^ ^ ^ <--- Ces 3 voleurs seront capturés!

```

Pour la file d'attente = "X5X#3X###XXXX##1#X1X", la sortie devrait être 5.

```

X5X#3X###XXXX##1#X1X
^ ^ ^           ^ ^ <--- Ces 5 voleurs seront capturés!

```

```

catchThief("X1X#2X#XX") → 3
catchThief("X5X#3X###XXXX##1#X1X") → 5
catchThief("X#X1#X9XX") → 5

```

#112 - Types de données (6 kyu)

Vous recevez une chaîne de chiffres, de lettres et/ou d'espaces.

La fonction `dataTypes` doit renvoyer un tableau contenant trois types de données JavaScript suivants: **string**, **number**, **boolean**. Chaque ensemble de caractères qui ne contient pas de nombres ou d'espaces doit être considéré comme une seule chaîne. Les exceptions sont les valeurs **true** et **false** qui doivent être évaluées en "boolean". Exemples:

```

dataTypes("You are number 1") → ['string', 'string', 'string', 'number']
dataTypes("Youarenumber1") → ['string', 'number']
dataTypes("123gjet") → ['number', 'string']
dataTypes("truestring1") → ['boolean', 'string', 'number']
dataTypes("FalsetruefalseABCDEFGHIJKLMNOpqrstuvwxyz0123456789false") →
['boolean', 'boolean', 'boolean', 'string', 'number', 'boolean']

```

#113 - Lettres alternées ? (6 kyu)

Ecrire une fonction qui permet de savoir si les voyelles (aeiou) et les consonnes sont alternées dans un mot.

```
isAlt("amazon") → true
isAlt("apple") → false
isAlt("banana") → true
isAlt("a") → true
isAlt("b") → true
```

#114 - Langage Tick (6 kyu)

Tick est un langage de programmation exotique créé en 2017. Il a seulement 4 commandes et une bande de mémoire infinie. Il est également fait pour ignorer les autres caractères sans commandement.

Commandes

- > : Déplacez le sélecteur vers la cellule suivante du ruban
- < : Déplacer le sélecteur vers la cellule précédente du ruban
- + : Augmenter la cellule mémoire de 1. Tronquer de sorte que $255 + 1 = 0$
- * : Ajoutez la valeur Ascii de la cellule de mémoire au flux de sortie

Exemple de programme :

```
interpreter('+++++*>+++++
+++++*>+++++*>+++++
+++++*>+++++*>+++++
+++++*>+++++*>+++++
+++++*<<<*>>>+++++
+++++*<<<*>>>+++++
+++++*>+++++*')
→ Hello World !
```

#115 - Longueur de la clé (6 kyu)

Pour coder du texte on utilise une clé, par exemple « 123 » en la répétant, par exemple :

```
bonjour
1231231 // répétition de la clé 123
cqkqkqxs
```

On vous demande de retrouver la longueur minimale de la clé connaissant ses répétitions, par exemple :

```
findTheKey("123412341234123412")
→ 1234 // 1234 est la plus courte séquence qui se répète dans la chaîne
findTheKey("123")
→ 123
findTheKey("1231")
→ 123
findTheKey("123124")
→ 123124
findTheKey("111111")
→ 1
```


#116 - Exercices sur les nœuds (6 kyu et 7 kyu)

Considérons le nœud suivant :

```
function Node(data, next = null) {
  this.data = data;
  this.next = next;
}
```

A partir de ce nœud nous pouvons créer par exemple la liste : 1→2→3

```
var head=new Node(1, new Node(2, new Node(3)))
```

Dans tous les cas ci-dessous, prévoir aussi le cas où `head` est `null`

- **LONGUEUR**

Créez une fonction **length** qui renvoie la longueur d'une liste, par exemple :

```
length(head)
→ 3
```

- **INDEXOF**

Créez une fonction **indexOf** qui recherche le 1^{er} élément égal à une valeur donnée, par exemple :

```
indexOf(head,2) // la valeur 2 est en position 1 de la liste
→ 1
```

- **LASTINDEXOF**

Créez une fonction **lastIndexOf** qui recherche le dernier élément égal à une valeur donnée.

- **COUNTIF**

Créez une fonction **countIf** qui compte le nombre d'éléments vérifiant une condition. Par exemple avec la liste 1→2→3 et la fonction booléenne `x=> x>=2` on doit obtenir 2 car « 2 » et « 3 » vérifient la condition.

- **FILTER**

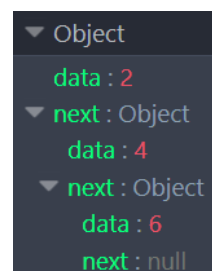
Créez une fonction **filter** qui à partir d'une liste et d'une fonction booléenne donne la liste vérifiant la condition. Par exemple avec la liste 1→2→3 et la fonction `x=> x>=2` on doit obtenir la liste 2→3

- **MAP**

Soit maintenant une fonction, par exemple multiplier par 2 : `var f=x => x * 2`

Ecrire une fonction `map(head, f)` qui renvoie la liste des nœuds où chaque élément (data) a été multiplié par 2. Dans notre cas on aura donc :

```
map(head, f)
→ Object { data: 2, next: Object }
```



Avec la structure ci-contre →

La fonction `map` doit pouvoir s'appliquer à des nœuds contenant tous types de données.

▪ REDUCE

En entrée on vous donne une liste, une fonction à 2 paramètres et une valeur initiale. Par exemple :

1→2→3→4, avec la fonction $(acc, curr) \Rightarrow acc * curr$ et la valeur initiale 1, on obtient la valeur réduite 24

Voici d'autres exemples :

```
reduce(null, (a, b) => a + b, 0) → 0
reduce(new Node(1, new Node(2, new Node(3))), (a, b) => a + b, 0) → 6
reduce(new Node(1, new Node(2, new Node(3, new Node(4)))), (a, b) => a * b, 1) → 24
reduce(new Node('a', new Node('b', new Node('c'))), (a, b) => a + b, '') → 'abc'
```

▪ FUSION DE 2 LISTES ORDONNÉES

Considérons 2 listes ordonnées, par exemple :

```
var l1=new Node(2, new Node(5, new Node(7)))
var l2=new Node(3, new Node(4, new Node(6, new Node(10))))
```

Créez une fonction qui renvoie une seule liste ordonnée contenant l'ensemble des valeurs :

2→3→4→5→6→7→10

```
mergeTwoLists(l1,l2)
→ Object { data: 2, next: Object }
```

#117 - Somme max dans un arbre (6kyu)

Le nœud d'un arbre est défini par :

```
var TreeNode = function(value, left, right) {
  this.value = value;
  this.left = left;
  this.right = right;
};
```

Ecrire une fonction qui renvoie la somme maximale des branches, par exemple si l'arbre est :

```

  17
 /  \
3    -10
/    /  \
2   16   1
      /
     13
```

On doit trouver 23 qui correspond à la branche [17,-10,16]. Autre exemple :

```
var root = new TreeNode(5, new TreeNode(-22, new TreeNode(9), new TreeNode(50)), new
TreeNode(11, new TreeNode(9), new TreeNode(2)));
maxSum(root) → 33
```

```

  / \
-22  11
 / \ / \
 9 50 9  2

```

#118 - 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'

```

#119 - 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"

```

#120 - Pig Latin (5 kyu)

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

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

#121 - 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"

```

#122 - Départ - Arrivée (5 kyu)

Vous remarquez que chaque caractère des clapets est sur une sorte de rotor où l'ordre est :

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

À partir de la gauche, tous les rotors tournent ensemble jusqu'à ce que le caractère du rotor soit correct.

Ensuite, le mécanisme passe au second rotor ...

... RÉPÉTEZ la procédure pour ce rotor (en faisant tourner également tous les rotors à sa droite)

... CONTINUER jusqu'à ce que l'affichage soit celui voulu.



Exemple : Considérons un affichage de 3 rotors et 1 ligne avec les lettres CAT. On veut afficher DOG :

Étape 1 : Tourner 1 fois le rotor n°1 et ceux à droite (donc tous les rotors) → DBU

Étape 2 : Tourner 13 fois le rotors n°2 et ceux à droite (donc rotor n°3) → DO

Étape 3 : Tourner 27 fois le rotor n°3 (il n'y en a plus à droite) → DOG

Vous avez en entrée :

Lines : tableau de chaînes. Chaque chaîne est une ligne d'affichage de la configuration initiale

Rotors : tableau des valeurs des rotors à appliquer

Exemples :

```
flapDisplay(["CODE"], [[20,20,28,0]]) → ["WARS"]
flapDisplay(["HELLO "], [[15,49,50,48,43,13]]) → ["WORLD!"]
flapDisplay([ '+-----+', '| THIS IS A MULTI LINE TEST |', '+-----+
-----+' ] [ [ 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 53 ], [ 8, 46, 7, 12, 30, 1, 4, 16, 34, 52, 32, 13, 11, 48, 3, 14, 4, 24, 16, 13, 3, 47, 22, 26, 50, 13, 52, 47, 8 ], [ 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 53 ] ]) →
[ '*****', '* DO YOU LIKE THIS KATA? *', '*****' ]
```

#123 - 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 $F(1) = 1$.

A partir d'un nombre, disons **prod** (pour le produit), nous recherchons deux nombres de 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]

#124 - 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"], 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"]) → []
```

#125 - Un problème de poids (5 kyu)

Mon ami John et moi-même sommes membres du "Fat to Fit Club (FFC)". John s'inquiète parce que chaque mois, une liste avec le poids des membres est publiée et chaque mois, il est le dernier sur la liste, ce qui signifie qu'il est le plus lourd.

Je suis celui qui établit la liste, alors je lui ai dit: "Ne t'inquiète plus, je modifierai l'ordre de la liste". Le poids d'un nombre sera à partir de maintenant la somme de ses chiffres.

Par exemple 99 aura "poids" 18, 100 aura "poids" 1 donc dans la liste 100 sera avant 99. Compte tenu d'une chaîne avec les poids des membres du FFC dans l'ordre normal, pouvez-vous donner cette chaîne gérée par le "poids" de ces chiffres ?

```
orderWeight("103 123 4444 99 2000") → "2000 103 123 4444 99"
orderWeight("2000 10003 1234000 44444444 9999 11 11 22 123") → "11 11 2000 10003 22 123 1234000 44444444 9999"
```

#126 - 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 :

```
MaxSequence ([- 2, 1, -3, 4, -1, 2, 1, -5, 4]) → 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.

#127 - Anagrammes (5 kyu)

Qu'est-ce qu'une anagramme ? Eh bien, deux mots sont des anagrammes les uns des autres s'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, un mot et un tableau avec des mots. Vous devez renvoyer un tableau de tous les anagrammes ou un tableau vide s'il n'y en a pas. Par exemple :

```
anagrams('abba', ['aabb', 'abcd', 'bbaa', 'dada']) => ['aabb', 'bbaa']
```

```
anagrams('racer', ['crazer', 'carer', 'racar', 'caers', 'racer']) => ['carer', 'racer']
```

```
anagrams('laser', ['lazing', 'lazy', 'lacer']) => []
```

#128 - 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 devra 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]
```

#129 - 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]]
```

#130 - 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]]
```

#131 - 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( "()(())") → false
validParentheses( "(" ) → false
validParentheses( "(())((())())") → true
```

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

#132 - Combinaisons téléphoniques (5 kyu)

A partir d'une chaîne de nombres, retournez toutes les combinaisons possibles de lettres.

Par exemple :

```
letterCombinations("23")
→ Array [ "ad", "bd", "cd", "ae", "be", "ce", "af", "bf", "cf" ]
```



Les chiffres utilisés seront uniquement ceux entre 2 et 9.

#133 - 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 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' // -2 - 8 + 16 = 6
-6, '1110' // -2 + 4 -8 = -6
4, '100'
18, '10110' // -2 +4 + 16 = 18
-11, '110101' // 1 + 4 + 16 -32 = -11
```

#134 - Meilleur score du perdant (5 kyu)

"AL-AHLY" et "Zamalek" sont les meilleures équipes en Egypte, mais "AL-AHLY" gagne toujours les matchs.

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.

#135 - Animaux écrasés (5 kyu)

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

```
roadKill("====h====yyyyy====eee=n=a====") → "hyena"
roadKill("====pe====nnnnn====n=nng====u====iii=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','butterfly','cat','camel','cow','chameleon','dog','dolphin','duck','dragonfly','eagle','elephant','emu','echidna','fish','frog','flamingo','fox','goat','giraffe','gibbon','gecko','hyena','hippopotamus','horse','hamster','insect','impala','iguana','ibis','jackal','jaguar','jellyfish','kangaroo','kiwi','koala','killerwhale','lemur','leopard','llama','lion','monkey','mouse','moose','meercat','numbat','newt','ostrich','otter','octopus','orangutan','penguin','panther','parrot','pig','quail','quokka','quoll','rat','rhinoceros','raccoon','reindeer','rabbit','snake','squirrel','sheep','seal','turtle','tiger','turkey','tapir','unicorn','vampirebat','vulture','wombat','walrus','wildebeast','wallaby','yak','zebra']
```

#136 - Hunger Games au zoo (5 kyu)

Une panne de courant au zoo a provoqué l'ouverture de toutes les portes des cages ! Les animaux sont sortis et ils commencent à se manger entre eux ! Voici une liste d'animaux du zoo et ce qu'ils peuvent manger :

- L'antelope mange de l'herbe
- Les gros poissons mangent des petits poissons
- Les punaises mangent des feuilles
- L'ours mange des gros poissons
- L'ours mange des punaises
- L'ours mange des poules
- L'ours mange des vaches
- L'ours mange des feuilles
- L'ours mange des moutons
- Les poules mangent des punaises
- La vache mange de l'herbe
- Le renard mange des poules
- Le renard mange des moutons
- La girafe mange des feuilles
- Le lion mange de l'antelope
- Le lion mange de la vache
- Le panda mange des feuilles
- Le mouton mange de l'herbe

Traduction des différents éléments en anglais :

```
['antelope','grass','big-fish','little-fish','bug','leaves','bear','chicken','cow','sheep','fox','giraffe','lion','panda']
```

Le but est d'afficher qui mange qui jusqu'à ce que la situation soit stable.

En entrée vous avez tous les éléments qui sont dans le zoo séparés par des virgules.

En sortie vous devez obtenir une liste avec :

- Le premier élément le zoo initial
- Le dernier élément est une chaîne séparée par des virgules de ce que ressemble le zoo à la fin
- Tous les autres éléments (du 2^e à l'avant dernier) sont de la forme X mange Y décrivant ce qui s'est passé

Remarques :

- Les animaux ne peuvent manger que des éléments (animaux ou végétaux) à côté d'eux
- Les animaux mangent toujours à leur GAUCHE avant de manger à leur DROITE
- L'animal le plus à gauche mange toujours avant les autres

Les autres choses que vous pouvez trouver dans le zoo (qui ne sont pas listés ci-dessus) ne mangent rien et ne sont pas comestibles. Exemples :

```
whoEatsWho("fox,bug,chicken,grass,sheep")
```

```
→ ["fox,bug,chicken,grass,sheep", "chicken eats bug", "fox eats chicken", "sheep eats grass", "fox eats sheep", "fox" ];
```

```

whoEatsWho("fox,panda,grass,bear,cow,chicken,antelope,little-fish,fox,sheep")
→ ["bear eats cow", "bear eats chicken", "fox eats sheep", "fox,panda,grass, bear,
antelope,little-fish,fox" ]
whoEatsWho("fox,chicken,tree,chicken,bug,banana,bug,bear")
→ ["fox,chicken,tree,chicken,bug,banana,bug,bear", "fox eats chicken", "chicken eats
bug", "bear eats bug", "fox,tree,chicken,banana,bear"]

```

#137 - 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.

#138 - banana (5 kyu)

A partir d'une chaîne de lettres a, b et n, énumérez les différentes façons de faire le mot «banana» en traversant les lettres de gauche à droite. (Utilisez « -« pour indiquer une lettre non utilisée)

Entrée : bbananana

Sortie :

b-anana--	b-a--nana	-bana--na
b-anan--a	b---anana	-ban--ana
b-ana--na	-banana--	-ba--nana
b-an--ana	-banan--a	-b--anana

#139 - 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.

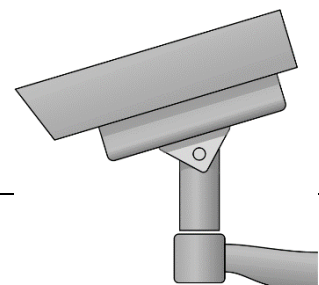
Les quatre premiers bits du paquet représentent l'ID de la caméra (indexées à partir de 0). Le cinquième bit indique s'il faut déplacer la caméra de cinq degrés vers le haut, le sixième bit s'il faut baisser de cinq degrés. Les septième et huitième bit indiquent s'il faut déplacer la caméra à gauche 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
}

```

#140 - Hauteur de pluie (5 kyu)

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
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

```

#141 - Vecteurs (5 kyu)

Déclarez et définissez une classe **Vector** qui représente un vecteur dans un espace tridimensionnel. Cette classe devrait avoir trois propriétés *i*, *j* et *k*.

La classe Vector doit avoir un constructeur de classe qui accepte exactement trois arguments, tous requis, dans l'ordre suivant: *i*, *j*, *k*, et attribuer les valeurs des arguments aux propriétés publiques *i*, *j* et *k* respectivement. Les trois arguments sont des nombres valides (entiers et ou décimaux) mais aucune vérification n'est requise de votre part.

Définissez une méthode publique **getMagnitude** qui n'accepte aucun argument et renvoie la norme du vecteur.

Les vecteurs unitaires dans les directions *x*, *y* et *z* sont communément désignés par *i*, *j* et *k* respectivement. Définissez trois méthodes publiques, statiques (c'est-à-dire directement appelées à partir de la classe elle-

même), **getI**, **getJ** et **getK**, chacune sans argument et renvoyant des vecteurs représentant i, j et k respectivement.

Définissez également (voir exemples après) :

- `add(that)` : addition de 2 vecteurs
- `multiplyByScalar(n)` : multiplication par un scalaire
- `dot(that)` : produit scalaire
- `cross(that)` : produit vectoriel
- `isParallelTo(that)` : est parallèle à...
- `isPerpendicularTo(that)` : est perpendiculaire à...
- `normalize()` : normalise le vecteur
- `isNormalized()` : true ou false suivant que le vecteur est normalisé ou non.

Exemples

```
var v = new Vector(6, 10, -3);
v.getMagnitude() → 12.041594578792296

var i = Vector.getI();
var j = Vector.getJ();
var k = Vector.getK();
console.log(i.i,i.j,i.k)
→ 1 0 0
console.log(j.i,j.j,j.k)
→ 0 1 0
console.log(k.i,k.j,k.k)
→ 0 0 1
var v = new Vector(3, 7 / 2, -3 / 2);
var s = v.add(new Vector(-27, 3, 4));
console.log(s.i,s.j,s.k)
→ -24 6.5 2.5
var v = new Vector(1 / 3, 177 / 27, -99);
var e = v.multiplyByScalar(-3 / 7);
console.log(e.i,e.j,e.k)
→ -0.14285714285714285 -2.8095238095238093 42.42857142857142
var v = new Vector(-99 / 71, 22 / 23, 45)
v.dot(new Vector(-5, 4, 7)) → 325.7979179
var a = new Vector(2, 1, 3);
var b = new Vector(4, 6, 5);
var aCrossB = a.cross(b);
console.log(aCrossB.i,aCrossB.j,aCrossB.k)
→ -13 2 8
var a = new Vector(1045 / 23, -666 / 37, 15);
var b = new Vector(161.3385037, -59124 / 925, 9854 / 185);
a.isParallelTo(b) → true
b.isParallelTo(a) → true
var c = new Vector(-3, 0, 5);
var d = new Vector(-12, 1, 20);
c.isParallelTo(d) → false
var a = new Vector(3, 4, 7);
var b = new Vector(1 / 3, 2, -9 / 7);
a.isPerpendicularTo(b) → true
var c = new Vector(1, 3, 5);
var d = new Vector(-2, -7, 4.4);
c.isPerpendicularTo(d) → false
var v = new Vector(-1, -1, 1);
```

```
var u = v.normalize();
console.log(u.i,u.j,u.k)
→ -0.5773502691896258 -0.5773502691896258 0.5773502691896258
var a = new Vector(-1 / Math.sqrt(2), 0, 1 / Math.sqrt(2));
var b = new Vector(1, 1, 1);
a.isNormalized() → true
b.isNormalized() → false
```

#142 - NSA et espionnage (5 kyu)

L'agence NSA veut espionner les conversations téléphoniques (appels ou SMS) et fait appel à vos services de programmeur. Pour ce faire, nous aurons besoin de définir des personnes (**Person**) ayant au moins les propriétés ou méthodes suivantes : **name** (pour le nom), **call** (appel) et **text** (SMS). La méthode **call** a 2 paramètres, un objet téléphone contenant le propriétaire (**owner**) et le numéro (**number**) et la personne appelée.

Par exemple

```
var dan = new Person("Dan");
var alex = new Person("Alex");
var phone = {owner : dan, number: "202-555-0199"};
dan.call(phone, alex);
```

Ici 2 personnes, Dan et Alex, un téléphone appartenant à Dan et Dan appelle Alex avec son propre téléphone.

La méthode **text** est très similaire à la méthode **call**, mais au lieu d'avoir un unique destinataire, il peut y en avoir un nombre quelconque (tous étant des personnes).

Exemple

```
var mark = new Person ("Mark");
dan.text (phone, alex, mark);
```

Dan envoie un SMS à Alex et Mark avec son propre téléphone.

La NSA vous oblige à enregistrer chaque appel téléphonique et chaque SMS que chaque personne a émis. L'objet NSA aura une méthode **log**. Cette méthode prend un paramètre : une instance de **Person**. Il renverra le journal conservé sur cette personne dans le format suivant :

```
[CALLER] called/texted [CALLEE] from [PHONE OWNER]'s phone([PHONE NUMBER])
```

Chaque enregistrement sera séparé par `\n`.

```
Dan called Erin from Dan's phone(202-555-0149)
Dan texted Anthony from Anthony's phone(202-555-0199)
Dan texted Alex from Dan's phone(202-555-0149)
```

S'il n'y a pas d'entrées pour la personne, la méthode renverra simplement « No Entries ».

La NSA ayant peur d'être accusée d'avoir espionné des civils, assurez-vous d'effacer chaque enregistrement individuel après l'avoir lu dans le journal.

Exemple complet

```

var dan = new Person("Dan");
var mark = new Person("Mark");
var phone = {owner: dan, number: '202-555-0199'};
dan.call(phone, mark);
NSA.log(dan) → 'Dan called Mark from Dan\'s phone(202-555-0199) '
var anthony = new Person("Anthony");
anthony.call(phone,dan)
NSA.log(anthony) → 'Anthony called Dan from Dan\'s phone(202-555-0199) '
var mobile = {owner: mark, number: '202-555-0166'};
mark.text(mobile,dan,anthony)
mark.call(phone,anthony)
NSA.log(mark) →
'Mark texted Dan from Mark\'s phone(202-555-0166)
  Mark texted Anthony from Mark\'s phone(202-555-0166)
  Mark called Anthony from Dan\'s phone(202-555-0166)
'
var erin = new Person("Erin");
NSA.log(erin) → 'No Entries'

```

Structure de votre programme

```

// Objet NSA
var NSA = {};

// Constructeur de personnes
var Person = function() {
  this.name;
  this.call = function(cellphone, callee) {
  }
  this.text = function(cellphone) {
  }
}

```

#143 - Matrices infinies (5 kyu)

Grace au code ci-dessous (à copier-coller), nous pouvons construire des matrices à 2 dimensions infinies où chaque élément est **false** sauf celui aux coordonnées n, m. Écrivez une fonction qui prend une telle matrice et renvoie la paire d'indices où la matrice est **true**.

```

const generate = (m,n)=>{
  var hithandler = {
    get: (_,col)=>col==n?true:false,
    set: ()=>false
  }
  var misshandler = {
    get: ()=>false, set: ()=>false
  }
  var hit = new Proxy({}, hithandler);
  var miss = new Proxy({}, misshandler);
  var rowhandler = {
    get: (_,row)=>row==m?hit:miss,
    set: ()=>false
  };
  var p = new Proxy({}, rowhandler);
  return p;
}

```

Vous pouvez utiliser `mat[j][i]` normalement pour accéder aux éléments qui seront tous faux, sauf lorsque $j = m$ et $i = n$. Cependant, les proxies se comportent tout à fait différemment des tableaux, de sorte que les méthodes standards ne fonctionneront pas.

```
var mat = generate(0,10)
findTrue(mat) → [0,10]
```

#144 - Triangle Pascal (4 kyu)

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

```
pascalsTriangle(4) == [1,1,1,1,2,1,1,3,3,1]
```

#145 - Écriture d'intervalles (4 kyu)

On veut formater une liste ordonnée d'entiers en les séparant par des virgules telles que l'on ait :

- des nombres entiers seuls

- des plages d'entiers séparés par un tiret '-'. Cette plage comprend tous les entiers dans l'intervalle, y compris les extrémités et doit s'étendre sur au moins 3 nombres. Par exemple ("12, 13, 15-17")

Écrire une fonction qui prend en entrée une liste d'entiers en ordre croissant et renvoie une chaîne correctement formatée. Exemple :

```
solution([-6, -3, -2, -1, 0, 1, 3, 4, 5, 7, 8, 9, 10, 11, 14, 15, 17, 18, 19, 20])
→ "-6,-3-1,3-5,7-11,14,15,17-20"
```

#146 - Somme d'intervalles (4 kyu)

Écrivez une fonction **sumIntervals** qui accepte en entrée une liste d'intervalles et renvoie la somme de toutes les longueurs des intervalles. Les chevauchements ne doivent être comptés qu'une seule fois.

Les intervalles sont représentés par une paire d'entiers sous la forme d'une liste. La première valeur de l'intervalle sera toujours inférieure à la seconde valeur. Exemple d'intervalle : [1, 5] est un intervalle de 1 à 5. La longueur de cet intervalle est de 4.

Liste contenant des intervalles se chevauchant :

```
[
  [1,4]
  [7, 10],
  [3, 5]
]
```

La somme des longueurs de ces intervalles est de 7. Puisque [1, 4] et [3, 5] se chevauchent, on peut traiter l'intervalle comme [1, 5], qui a une longueur de 4.

```
sumIntervals( [[1,2],[6, 10],[11, 15]] ) → 9
sumIntervals( [[1,4],[7, 10],[3, 5]] ) → 7
sumIntervals( [[1,5],[10, 20],[1, 6],[16, 19],[5, 11]] ) → 19
```

#147 - 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
```

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

Toutes les sommes et soustractions possibles de monômes équivalents (" $xy == yx$ ") seront 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"
```

#149 - Grandes factorielles (4 kyu)

Calculez $n! = n \times (n - 1) \times \dots \times 1$ pour n arbitraire, ce qui exclut la version récursive habituelle.

#150 - 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 positif ou nul. Si le nombre est nul, renvoyer 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 and 2 seconds"
formatDuration(3662) → "1 hour, 1 minute and 2 seconds"
```


Règles détaillées

L'expression résultante est faite de plusieurs 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 une espace (", "). Sauf le dernier composant, qui est séparé par " and ". Une année fera 365 jours et un jour 24 heures.

#151 - 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'

#152 - Énumération des permutations (4 kyu)

Énumérez toutes les permutations d'une chaîne en supprimant les doublons s'il y a lieu. Cela signifie que vous devez mélanger toutes les lettres de l'entrée dans toutes les ordres possibles.

`permutations('a') → ['a']`

`permutations('ab') → ['ab', 'ba']`

`permutations('aabb') → ['aabb', 'abab', 'abba', 'baab', 'baba', 'bbaa']`

`permutations('abc') → ['\abc\ ', '\acb\ ', '\bac\ ', '\bca\ ', '\cab\ ', '\cba\ ']`

#153 - 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`

#154 - 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 = "mmmmmm 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/=:nnnnn/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) → "=:aaaaa/2:eeee/=:fffff/1:tt/2:rr/=:hh"
```

#155 - Serpent (4 kyu)

À partir d'une matrice n x n, renvoyez les éléments en partant des plus extérieurs et en vous déplaçant dans le sens des aiguilles d'une montre.

```
Tableau = [[1,2,3],
            [4,5,6],
            [7,8,9]]
snail(tableau) → [1,2,3,6,9,8,7,4,5]
```

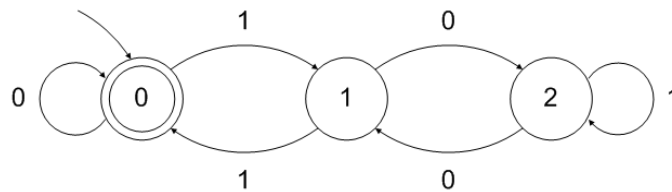
Cette image illustrera les choses plus clairement :

1	2	3
4	5	6
7	8	9

1	2	3	1
4	5	6	4
7	8	9	7
7	8	9	7

#156 - Nombre binaire multiple de 3

Écrire une expression régulière nommée **multipleOf3Regex** permettant de savoir si un nombre écrit en binaire est un multiple de 3. On pourra utiliser l'automate suivant :



Par exemple, en partant de la flèche en haut à gauche et en suivant 1 puis 1, on tombe sur l'état 0 qui est un état acceptant donc 11 (=3 en base 10) est un multiple de 3.

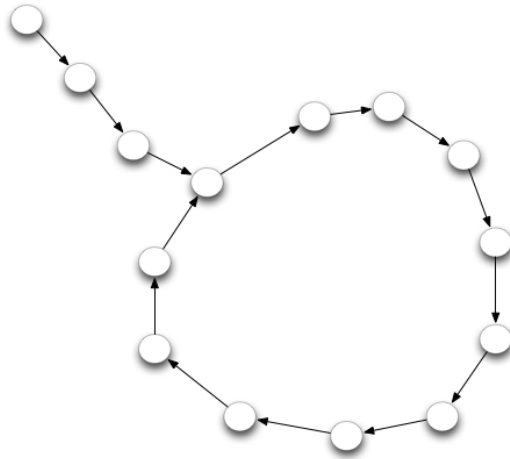
De même avec **101110100**, on passera par les états 1 puis 2 pour **10111**, ensuite 1 et 0 pour **0100**, on est donc encore sur un état acceptant. Le nombre est en fait $372 = 3 \cdot 124$.

```

>> multipleOf3Regex.test('11')
true
>> multipleOf3Regex.test((372).toString(2))
true
>> multipleOf3Regex.test((7).toString(2))
false
  
```

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

On vous donne un nœud qui est le début d'une liste. Cette liste contient toujours une queue et une boucle. Votre objectif est de déterminer la longueur de la boucle. Par exemple, sur l'image suivante, la taille de la queue est 3 et celle de la boucle est 11.



```
// Utilisez 'getNext' ou 'next' pour passer au noeud suivant
node.getNext()
node.next
```

#158 - Distance de Levenstein (3 kyu)

Référence : https://fr.wikipedia.org/wiki/Distance_de_Levenshtein

On vous donne un terme en entrée (chaîne en minuscules) et un dictionnaire (ensemble de mots connus également en minuscules). Votre tâche est de savoir quel mot est le plus similaire à celui en entrée. La similitude est décrite par le nombre minimum de lettres que vous devez ajouter, supprimer ou remplacer pour passer du mot saisi à l'un de ceux du dictionnaire. Plus le nombre de changements requis est petit, plus la similitude entre les deux mots est élevée.

Les mêmes mots sont évidemment les plus semblables. Un mot qui nécessite une seule lettre à modifier est plus similaire qu'un autre mot qui nécessite 2 (ou plus) lettres à modifier.

```
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');
```

#159 - 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 recevez 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 sera 11h00

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

Si la réunion ne correspond pas aux agendas des personnes, renvoyer **null**

La durée de la réunion sera fournie en minutes.

En suivant ces règles et en regardant l'exemple ci-dessous, 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
```

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

Votre GPS est cassé et vous devez vous rendre quelque part ! Heureusement, vous avez une carte et des compétences en programmation. Votre carte est plutôt étrange : les intersections sont étiquetées avec des nombres entiers distincts et le temps en minutes pour aller d'une intersection à une autre.

```
{from: 0, to: 1, drivingTime: 5} // il existe une route entre l'intersection 0 et 1 et il faut 5 minutes de trajet
```

On vous donne le nombre total d'intersections, le nombre total de routes, la carte (les deux étiquettes d'intersection, entiers inférieurs au nombre d'intersections, et la durée de conduite entre les deux).

S'il n'y a pas de route dans l'autre sens, c'est une voie à sens unique.

Complétez la fonction de `navigate(numberOfIntersections, roads, start, finish)` afin de renvoyer les intersections sur l'itinéraire le plus rapide entre le début et la fin.

S'il existe plusieurs itinéraires, choisissez-en un. S'il n'y a pas de possibilité, retournez **null**.

```
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]. // 5 routes, la carte et aller de 0 à 4

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

#161 - 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") → 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);
```

```
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 - Nettoyage de chaînes (8 kyu)

▪ SOLUTION 1

```
function stringClean(s) {  
  return s.replace(/\d/g, '');  
}
```

▪ SOLUTION 2

```
function stringClean(s) {  
  return s.replace(/[0-9]/g, '');  
}
```

#5 - 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]`

#6 - Supprimer les doublons (8 kyu)

▪ SOLUTION 1

```
removeDuplicates = (nums) =>
  nums.reduce((a,v)=>a.includes(v) ? a : a.concat(v), []).sort((a,b)=>a-b)
```

▪ SOLUTION 2

```
removeDuplicates = (nums) => {
  var out=[]
  for (v of nums)
    if (!out.includes(v)) out.push(v)
  return out.sort()
}
```

#7 - Joueur suivant (8 kyu)

▪ SOLUTION 1

```
whoseMove = (lastPlayer, win) => ['white', 'black'].find(c=>(c==lastPlayer)==win)
```

▪ SOLUTION 2

```
whoseMove = (lastPlayer, win) =>
  win ? lastPlayer : {black : "white", white : "black"}[lastPlayer]
```

▪ SOLUTION 3

```
function whoseMove(lastPlayer, win) {
  return win?lastPlayer:lastPlayer=="white"?"black":"white"
}
```

#8 - Pourboire (8 kyu)

▪ SOLUTION 1

```
const TIPS = {"terrible":0.0, "poor":0.05, "good":0.1, "great":0.15, "excellent":0.2};
```

```
const calculateTip = (amount, rating) => {
  rating = rating.toLowerCase();
  return rating in TIPS ? Math.ceil(TIPS[rating] * amount) : "Rating not recognised";
};
```

▪ SOLUTION 2

```
function calculateTip(amount, rating) {
  switch(rating.toLowerCase()){
    case "terrible":return 0;
    case "poor":return Math.ceil(amount * 0.05);
    case "good":return Math.ceil(amount * 0.1);
    case "great":return Math.ceil(amount * 0.15);
    case "excellent":return Math.ceil(amount * 0.2);
    default:return "Rating not recognised";
  }
}
```

▪ SOLUTION 3

```
let calculateTip = (a,r) => {
  const i = ['terrible', 'poor', 'good', 'great', 'excellent'].indexOf(r.toLowerCase());
  return i === -1 ? "Rating not recognised" : Math.ceil(i*a*.05);
}
```

▪ SOLUTION 4

```
function calculateTip(amount, rating) {
  console.log(amount,rating)
  var base = { terrible: 0, poor: 5, good: 10, great: 15, excellent: 20 }
  var total = Math.ceil(base[rating.toLowerCase()]*amount/100)
  return isNaN(total)? 'Rating not recognised' : total
}
```

#9 - 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"
```

#10 - Compter en Arara (7 kyu)

▪ SOLUTION 1

```
countArara = (n) => ("adak ".repeat(n>>1)+(n&1 ?"anane":"")).trim()
```

```
13>>1 // permet de diviser un nombre par 2 sans partie décimale
→ 6
(13).toString(2) // >>1 supprime le bit à droite
```

```

→ "1101"
(6).toString(2) // on obtient 110 qui correspond à 6
→ "110"
13&1 // vaut 0 si le nombre est pair et 1 sinon
→ 1 // On fait un ET logique entre le dernier bit de 13 et 1 → 1&1 = 1

```

▪ SOLUTION 2

```
countArara = (n) => n > 2 ? "adak " + countArara(n - 2) : n == 2 ? "adak" : "anane"
```

▪ SOLUTION 3

```

countArara = (n) => {
  if (n==1) return "anane";
  if (n==2) return "adak";
  return "adak" + " " + countArara(n-2);
}

```

#11 - 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 \$

#12 - 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(' ');  
};
```

#13 - 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.

#14 - 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++ // Incréméntation de la valeur de « x »
→ 10
obj
→ Object { x: 11 }
obj['x']++ // Autre technique d'incréméntation
→ 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

```

#15 - Compteur (7 kyu)

▪ SOLUTION 1

```

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

```

```

Array(5).keys()
→ Array Iterator {  }
[...Array(5).keys()]
→ Array [ 0, 1, 2, 3, 4 ]

```

▪ SOLUTION 2

```

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

```

```

Array.from({length:5}, (_,i)=>i*i)
→ Array [ 0, 1, 4, 9, 16 ]
[...Array(5).keys()].map(i=>i*i)
→ Array [ 0, 1, 4, 9, 16 ]

```

#16 - 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]
}
```

```
[5,10,2,18,3].sort((a,b)=>(a<b) ? -1 : 1)      // -1 = mettre avant, 1 = mettre après
→ Array [ 2, 3, 5, 10, 18 ]
[5,10,2,18,3].sort((a,b)=>a-b)                // a-b sera négatif (mettre avant) ou positif (après)
→ Array [ 2, 3, 5, 10, 18 ]
[5,10,2,18,3].sort((a,b)=>b-a)
→ Array [ 18, 10, 5, 3, 2 ]
```

▪ SOLUTION 2

```
const findLongest = 1 => 1
  .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];
}
```

```
Math.max([5,10,2,18,3])
→ NaN
Math.max(...[5,10,2,18,3])
→ 18
Math.max.apply(null, [5,10,2,18,3])
→ 18
```

#17 - 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));
}
```

```
Ou encore :
function findShort(s){
  return Math.min(...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);
}
```

```
Math.min(10,Infinity)
→ 10
Math.max(10,Infinity)
→ Infinity
```

#18 - Mot le plus long (7 kyu)

▪ SOLUTION 1

```
longestWord = (s) => s.split(' ').sort((a,b)=> a.length-b.length).slice(-1) [0]
```

▪ SOLUTION 2

```
longestWord = (s) => s.split(' ').reduce((a,w)=> w.length>=a.length ? w : a)
```

#19 - 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]}))
}
```

Ou encore :

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

```
(g=>[g,g,g]) (3)
→ Array [ 3, 3, 3 ]
```

▪ SOLUTION 3

```
color2grey = (image) =>
  image.map(a => a.map(([r,g,b]) => [...new Array(3)].fill(~((r + g + b + 1) / 3))))
```

```
~~(17/3)    // Récupérer la partie entière
→ 5
17/3|0     // Autre possibilité
→ 5
```

#20 - 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(''));
}
```

#21 - 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);
}
```

```
"ABCD".replace(/./g, '#') // Remplacer chaque caractère
→ "####"
s="ABCDEF"
→ "ABCDEF"
s.slice(0,s.length-2)
→ "ABCD"
s.slice(0,-2)           // Version plus courte
→ "ABCD"
```

▪ SOLUTION 3

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

```
"ABCDEF".match(/.(?=...)/g) // Un caractère suivi de 4 autres
→ Array [ "A", "B" ]
"ABCDEF".match(/.(?=..)/g) // Un caractère suivi de 2 autres
→ Array [ "A", "B", "C", "D" ]
```

▪ SOLUTION 4

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


#22 - Carte crédit – date d’expiration (7 kyu)

▪ SOLUTION 1

```
checkExpiryValid = (date) => {
  var [m,y]=date.split(/[^\d]/).filter(n=>n)
  y+=y+((y<100)?2000:0)
  var now=new Date()
  return (m>now.getMonth() && y==now.getFullYear()) || (y>now.getFullYear())
}
```

▪ SOLUTION 2

```
checkExpiryValid = (date) =>
  new Date(+date.substr(-2) + 2E3, +date.substr(0, 2), 0) >= new Date()
```

#23 - 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(' '));
}
```

```
function highAndLow(numbers){
  numbers = numbers.split(' ');
  return `${Math.max(...numbers)} ${Math.min(...numbers)}`;
}
highAndLow('1 9 3 4 -5')
9 -5
```

▪ SOLUTION 2

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

Attention ! Il s’agit d’un ` (AltGr+7) et pas d’un ` (apostrophe)

```
add = (a, b) => a + ' + ' + b + ' = ' + (a + b) // Concaténation classique
add2 = (a, b) => `${a} + ${b} = ${a+b}` // Utilisation de ` et de ${}
add(2,5)
→ "2 + 5 = 7"
add2(2,5)
→ "2 + 5 = 7"
```

▪ SOLUTION 3

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

#24 - Combien sont plus petits que moi ? (7 kyu)

▪ SOLUTION 1

```
function smaller(nums) {
  return nums.map((v,i)=>nums.slice(i).filter(k=>k<v).length)
}
```

#25 - 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('');
}
```

```
couleurs={'R':108, 'V':55, 'B':17}           // Création d'un objet
→ Object { R: 108, V: 55, B: 17 }
couleurs['V']                                // Accès à la valeur de V
→ 55
couleurs.V                                    // Autre technique
→ 55
Remarquez que de la même façon :
"ABCDEF".length
→ 6
"ABCDEF"['length']
→ 6
```

▪ 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', }
```

On pourrait également simplifier cette solution sous la forme :

```
DNASTrand = dna => dna.replace(/./g, c => ({A:'T', T:'A', C:'G', G:'C'})[c])
```

Remarquez l'usage des parenthèses autour de l'objet :

```
{'A':1}
→ SyntaxError: missing ; before statement [En savoir plus]
({'A':1})
```

```
→ Object { A: 1 }
({'A':1}).A
→ 1
```

▪ 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

```
const DNAStrand = dna => dna.replace(/./g, m => 'CGAT'['GCTA'.indexOf(m)]);
```

```
'CGAT'['GCTA'.indexOf('T')]
→ "A"
'GCTA'.indexOf('T')           // On cherche la position du 'T' dans 'GCTA'
→ 2
'CGAT'[2]                    // ce qui nous donne sa correspondance dans 'CGAT'
→ "A"
```

#26 - 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());
}
```

Autre écriture du même code :

```
comb=(s) => [...s].sort().join('')
vampire_test=(a, b) => comb(`${a}${b}`)==comb((a*b).toString())
```

#27 - Couleurs et associations (7 kyu)

▪ SOLUTION 1

```
const colourAssociation=(array)=>array.map(([c,asso])=>({[c]:asso}))
```

▪ SOLUTION 2

```
function colourAssociation(array) {
  return array.map(function(i) {
    var v = {};
    v[i[0]] = i[1]
    return v;
  });
}
```

#28 - Jason (7 kyu)

▪ SOLUTION 1

```
function killcount(counselors, jason){
  return counselors.reduce((a,v)=>(v[1]<jason) ? a.concat(v[0]) : a, [])
}
```

On ajoute au fur et à mesure à l'accumulateur « a » les noms v[0] des personnes qui ont une intelligence v[1] inférieure à celle de Jason

▪ SOLUTION 2

```
const killcount = (cs,j) => cs.filter(c=>c[1]<j).map(c=>c[0])
```

On filtre les personnes qui ont une intelligence c[1] inférieure à Jason puis on récupère leurs noms c[0]

▪ SOLUTION 3

```
const killcount = (cs, j) => cs.reduce((a, [s, k]) => k < j ? (a.push(s), a) : a, []);
```

Remarquez l'utilisation de la virgule dans « a.push(s), a » afin de récupérer l'accumulateur :

```
a=[]
→ Array [  ]
a.push(5)
→ 1           // On perd l'accumulateur
a.push(7)
→ 2
a
→ Array [ 5, 7 ]
a.push(9),a
→ Array [ 5, 7, 9 ] // On ajoute 9 et on récupère l'accumulateur
```

▪ SOLUTION 4

```
function killcount(counselors, jason){
  var arr = []
  for (var i = 0; i < counselors.length; i++){
    if (counselors[i][1] < jason) {
      arr.push(counselors[i][0])
    }
  }
  return arr;
}
```

Programmation procédurale...

#29 - Scoutisme (7 kyu)

▪ SOLUTION 1

```
function encode(str)
{
```

```

var code = 'GADERYPOLUKIgaderypoluki'
var s = [...str].map(c => {
  var r = code.indexOf(c)
  return (r >= 0) ? code[2 * (r / 2 | 0) + (1 - r % 2)] : c
})
return s.join('');
}
function decode(str)
{
  return encode(str);
}

```

La formule mathématique consiste « simplement » à trouver la position de la lettre de substitution. Si c'est un « G » (position 0) on doit la substituer par « A » (position 1), de même on doit échanger positions 2 et 3 etc.

```

[...Array(10).keys()].map(r=>2 * (r / 2 | 0) + (1 - r % 2))
→ Array [ 1, 0, 3, 2, 5, 4, 7, 6, 9, 8 ]

```

▪ SOLUTION 2

```

const dict = {
  G: "A", A: "G", D: "E", E: "D",
  R: "Y", Y: "R", P: "O", O: "P",
  L: "U", U: "L", K: "I", I: "K",
  g: "a", a: "g", d: "e", e: "d",
  r: "y", y: "r", p: "o", o: "p",
  l: "u", u: "l", k: "i", i: "k",
}

const encode = s => s.replace(/./g, c => dict[c] || c)
const decode = encode

```

La partie fastidieuse est la construction du dictionnaire.

▪ SOLUTION 3

```

const key = 'gaderypoluki';
const encode = str => str.replace(new RegExp(`[${key}]`, 'gi'), c => {
  const i = key.indexOf(c.toLowerCase());
  const e = i % 2 === 0 ? key[i+1] : key[i-1];
  return /[A-Z]/.test(c) ? e.toUpperCase() : e;
});
const decode = encode

```

Remarquez que le calcul de « e » est basé sur le même calcul que la solution n°1

```

[...Array(10).keys()].map(i=>i % 2 === 0 ? i+1 : i-1)
→ Array [ 1, 0, 3, 2, 5, 4, 7, 6, 9, 8 ]

```

```

/[A-Z]/.test('e')    // On teste si c'est une majuscule
false
/[A-Z]/.test('E')
true

```

#30 - La guerre des lettres (7 kyu)

▪ SOLUTION 1

```
function alphabetWar(fight) {
  let map = { w: -4, p: -3, b: -2, s: -1, m: 4, q: 3, d: 2, z: 1 };
  let result = fight.split('').reduce((a, b) => a + (map[b] || 0), 0);
  return result < 0 ? "Left" : "Right" + " side wins!" : "Let's fight again!";
}
```

▪ SOLUTION 2

```
function alphabetWar(s) {
  var a = s.split(""), l = a.reduce((s, c) => s + 1 + "sbpw".indexOf(c), 0), r =
  a.reduce((s, c) => s + 1 + "zdqm".indexOf(c), 0);
  return (l !== r ? (l > r ? "Left" : "Right") + " side wins" : "Let's fight again") +
  "!";
}
```

#31 - Nouvelle guerre des lettres (7 kyu)

▪ SOLUTION 1

```
const battle = (x, y) => [x, "Tie!", y][Math.sign(val(y) - val(x)) + 1];
const val = x => [...x].reduce((s, v) => s + ("
abcdefghijklmnopqrstuvwxy".indexOf(v.toLowerCase()) * (v > "Z" ? .5 : 1)), 0);
```

▪ SOLUTION 2

```
function battle(x, y) {
  const score = (str) => str.split('').map(v => v == v.toUpperCase() ? v.charCodeAt() - 64 :
  (v.charCodeAt() - 96) / 2).reduce((a, b) => a + b);
  return score(x) > score(y) ? x : (score(x) < score(y) ? y : 'Tie!');
}
```

▪ SOLUTION 3

```
var alpha = "abcdefghijklmnopqrstuvwxy";
power = (w) => [...w].reduce((a, c) =>
  a + (1 + alpha.indexOf(c.toLowerCase()) * (1 + (alpha.indexOf(c) == -1))), 0);
battle = (x, y) => {
  var [px, py] = [power(x), power(y)];
  return px > py ? x : (px < py ? y : "Tie!");
}
```

#32 - Chaîne cool (7 kyu)

▪ SOLUTION 1

```
function coolString(s) {
  return !/[a-z]{2,}|[A-Z]{2,}|[a-zA-Z]/.test(s)
}
```

```
}
```

On teste s'il y a au moins 2 minuscules qui se suivent `[a-z]{2,}` ou 2 majuscules ou s'il y a des caractères autres que des lettres `[^a-zA-Z]`.

▪ SOLUTION 2

```
function coolString(s) {  
  return /^[a-z]+$/.test(s) && !/([A-Z]{2})|([a-z]{2})/.test(s);  
}
```

Ici `^[a-z]+$` permet de tester s'il y a des caractères autres que des lettres (`/i` pour que le test soit indépendant de la casse)

▪ SOLUTION 3

```
function coolString(s) {  
  return !s.match(/^[^A-z]/g) && !s.match(/[A-Z]{2,}|[a-z]{2,}/g);  
}
```

3^e version pour tester s'il y a d'autres caractères que des lettres : `/[^A-z]/g`

#33 - Divisible par ? (7 kyu)

▪ SOLUTION 1

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

```
reste=(a,...r)=>r  
→ function reste()  
reste(1,2)  
→ Array [ 2 ] // On récupère le 2e paramètre dans un tableau  
reste(1,2,3,4)  
→ Array [ 2, 3, 4 ] // On récupère les paramètres
```

▪ SOLUTION 2

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

On teste si tous les restes des divisions sont nuls.

▪ SOLUTION 3

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

Même version en plus court.

▪ SOLUTION 4

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

On joue ici avec la négation. Si « m » n'est pas divisible par « n » alors $m \% n$ est différent de 0 et donc `ns.some(...)` est vrai et donc `!ns.some(...)` est faux.

```
Boolean(2)
→ true
Boolean(0)
→ false
```

▪ SOLUTION 5

```
function isDivisible(n) {
  return [].slice.call(arguments, 1).every(function(a) { return !(n % a) });
}
function reste(n) {return arguments;} // récupérer les arguments
→ undefined
reste(1,2)
→ Arguments { 0: 1, 1: 2, 2 de plus... }
reste(1,2,3,4)
→ Arguments { 0: 1, 1: 2, 2: 3, 3: 4, 2 de plus... }
```

#34 - 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+(c===guess[k]),0)
}
```

Rappelons que $0+true=1$ donc on ajoute bien 1 à chaque fois qu'une lettre est à la bonne place.

#35 - 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")
```


#36 - Nombre de X et de O (7 kyu)

▪ SOLUTION 1

```
function XO(str) {
  clean=str.toUpperCase().replace(/^[^OX]/g, '')
  return clean.replace(/O/g, "").length==clean.length/2
}
```

On enlève tout ce qui n'est pas O ou X → clean. On enlève ensuite de clean tous les O, ce qui doit correspondre exactement à la moitié de la longueur de la chaîne.

▪ SOLUTION 2

```
function XO(str) {
  let x = str.match(/x/gi);
  let o = str.match(/o/gi);
  return (x && x.length) === (o && o.length);
}
```

```
function XO(str) {
  let x = str.match(/x/gi);
  let o = str.match(/o/gi);
  return (x && x.length) === (o && o.length);
}
XO('xxOoXoXXoO') true
```

▪ SOLUTION 3

```
function XO(str) {
  return str.replace(/o/ig, '').length == str.replace(/x/ig, '').length
}
```

#37 - Moutons perdus (7 kyu)

▪ SOLUTION 1

```
function lostSheep(friday, saturday, total) {
  return friday.concat(saturday).reduce((s,l)=>s-l, total)
}
```

▪ SOLUTION 2

```
lostSheep = (f,s,t) => [...f,...s].reduce((a,v)=>a-v,t)
```

#38 - Points au tennis (7 kyu)

▪ SOLUTION 1

```
function tennisGamePoints(score) {
  var Pts={"love":0, "15": 1, "30":2, "40":3}
  var s=score.split("-")
  return Pts[s[0]] + ((s[1]=="all") ? Pts[s[0]] : Pts[s[1]])
}
```

▪ SOLUTION 2

```
function tennisGamePoints(score) {
  let scores = score.split('-');
  let points = {"love": 0, "15": 1, "30": 2, "40":3};
  return scores.reduce((p,c)=>c=="all"?p*2:p+points[c],0);
}
```

▪ SOLUTION 3

```
const tennisGamePoints = s => s.split("-").reduce((acc,v) => acc + { "love": 0, "15": 1, "30": 2, "40": 3, "all": acc }[v], 0 );
```

#39 - Qu'est-ce qui vient après ? (7 kyu)

▪ SOLUTION 1

```
function comes_after(str,l) {
  return [...str.toLowerCase()]
    .map((c,k)=>(c=l.toLowerCase() & /[a-z]/i.test(str[k+1])) ? str[k+1] : "")
    .join('')
}
```

▪ SOLUTION 2

```
function comes_after(str,l,rex=RegExp(l,"i")) {
  return [...str].filter((c,i) => i && rex.test(str[i-1]) && /[a-z]/i.test(c)).join``
}
```

▪ SOLUTION 3

```
function comes_after(str,l) {
  var regexp = new RegExp(`${l}+([a-z])`,`gi`)

  return (str.match(regexp) || []).reduce((p,c)=>p+c.slice(1),'')
}
```

#40 - Les martins-chasseurs (7 kyu)

▪ SOLUTION 1

```
var kookaCounter = function(laughing) {
  return (laughing.match(/(Ha)+|(ha)+/g) || []).length;
}
```

```
"HaHaHahahaHa".match(/(Ha)+|(ha)+/g)
→ Array [ "HaHaHa", "haha", "Ha" ]
null.length
→ TypeError: null has no properties [En savoir plus]
null || []
→ Array [ ]
(null || []).length // Permet de ne pas avoir d'erreur en cherchant la longueur
→ 0
```

▪ SOLUTION 2

```
const kookaCounter=l=>(l.match(/(ha|Ha)\1+/g) || []).length;
```

Cette solution suppose que les martins-chasseurs font au moins 2 fois « haha » ou « Haha », en effet on cherche un « ha » ou un « Ha » suivi d'au moins la même chose (\1+)

```
"HaHaHahahaHa".match(/(ha|Ha)\1+/g)
→ Array [ "HaHaHa", "haha" ] // On perd dernier « Ha » car pas suivi de « Ha »
"HaHahahaHaHa".match(/(ha|Ha)\1+/g)
→ Array [ "HaHa", "haha", "HaHa" ] // Ici c'est ok
```

▪ SOLUTION 3

```
var kookaCounter = function(stri) {
var stri = stri.replace(/a/g, "").replace(/H+/g, "H").replace(/h+/g, "h")
return stri.length
}
```

On enlève les « a » puis on regroupe les suites adjacentes de « H » en un seul et pareil pour les « h »

```
"HaHaHahahaHa".replace(/a/g, "")
→ "HHHhhH"
"HHHhhH".replace(/H+/g, "H")
→ "HhhH"
"HhhH".replace(/h+/g, "h")
→ "HhH"
```

▪ SOLUTION 4

```
var kookaCounter = function(laughing) {
return laughing.split('a').reduce((a,v,k,arr)=>a+(v!=arr[k+1]),0) -1
}
```

On sépare la chaîne en utilisant les « a » puis on compte le nombre de changements de lettres.

#41 - « g », la lettre heureuse (7 kyu)

▪ SOLUTION 1

```
const gHappy = str => str.replace(/g{2,}/g, '').indexOf('g')===-1
```

▪ SOLUTION 2

```
const gHappy = str => !/g/.test(str.replace(/g{2,}/g, ''))
```

▪ SOLUTION 3

```
const gHappy = str => str.replace(/g{2,}/g, ' ').search('g') < 0
```

L'idée des 3 solutions est de supprimer les « gg » et de regarder s'il en reste dans la chaîne.

▪ SOLUTION 4

```
const gHappy = str => !/([^g]|^)g([^g]|$)/.test(str)
```

Si on trouve une chaîne du type XgY avec X et Y différents de « g » (expression /([[^]g])g([[^]g])/) alors on a un « g » isolé. Mais dans le cas de 2 lettres cela ne fonctionnera pas :

```
"go".match(/([^g])g([^g]|$)/g)  
→ null
```

On ajoute donc les cas « début de chaîne + g + Y » ou « X + g + fin de chaîne »

#42 - Ascenseur ou pas ? (7 kyu)

▪ SOLUTION 1

```
shortestTime = (n,m,[ve,to,tc,vw]) =>  
  Math.min( (n-1)*vw, (Math.abs(m-n)+(n-1))*ve+to+tc+to )
```

▪ SOLUTION 2

```
function shortestTime(n,m,speeds){  
  var elevator = Math.abs(m-n)*speeds[0]+speeds[1]*2+speeds[2]+speeds[0]*Math.abs(n-1);  
  var walk = speeds[3]*(n-1);  
  return (elevator < walk) ? elevator : walk;  
}
```

#43 - Tatouages (7 kyu)

▪ SOLUTION 1

```
function robot(skinScan) {  
  return skinScan.map(c=>c.join('').replace(/X/g, '*').split(''))  
}
```

▪ SOLUTION 2

```
function robot(skinScan) {  
  return skinScan.map((arr) => {  
    return arr.map((c) => ({'X': '*'})[c] || c);  
  });  
}
```

▪ SOLUTION 3

```
function robot(skinScan) {  
  return skinScan.map(v => v.map(s => s === 'X' ? '*' : s));  
}
```

#44 - Coupons de réduction (7 kyu)

▪ SOLUTION 1

```
function checkCoupon(enteredCode, correctCode, currentDate, expirationDate){
  return enteredCode===correctCode && new Date(currentDate) <= new Date(expirationDate);
}
```

Ou encore :

```
checkCoupon = (EC, CC, CD, ED) => EC === CC && new Date(CD) <= new Date(ED);
```

▪ SOLUTION 2

```
function checkCoupon(enteredCode, correctCode, currentDate, expirationDate){
  return enteredCode === correctCode && Date.parse(expirationDate) >=
Date.parse(currentDate)
}
```

#45 - Taco Bell (7 kyu)

▪ SOLUTION 1

```
function tacofy(word) {
  var taco = {'#': 'beef', 't': 'tomato', 'l': 'lettuce', 'c': 'cheese', 'g':
'guacamole', 's': 'salsa'}

  return word
    .toLowerCase()
    .replace(/[aeiuo]/g, '#')
    .split('')
    .reduce((a, m) => taco.hasOwnProperty(m) ? a.concat(taco[m]) : a, ['shell'])
    .concat('shell')
}
```

▪ SOLUTION 2

```
function tacofy(word) {
  var taco = {t:'tomato',l:'lettuce',c: 'cheese',g:'guacamole', s:'salsa',a:'beef',
e:'beef',i:'beef',o:'beef',u:'beef'};
  return ['shell',...[...word].map(x => taco[x.toLowerCase()]).filter(x=>x),'shell'];
}
```

`filter` permet d'enlever toutes les valeurs indéfinies :

```
taco['z'] // N'existe pas
→ undefined
[1,2,undefined,6].filter(x=>x)
→ Array [ 1, 2, 6 ] // On enlève les undefined
```

`...word` permet d'obtenir une liste à partir de la chaîne de caractères. Inversement, `...[...word]` permet de récupérer les éléments :

```
[...[1,2,3]]
→ Array [ 1, 2, 3 ]
[[1,2,3]]
```

| → Array [Array[3]]

#46 - Différence entre 2 collections (7 kyu)

▪ SOLUTION 1

```
delta = (x,y) => x.filter(c=>!y.includes(c))
diff = (a, b) => [...new Set(delta(a,b).concat(delta(b,a)))] .sort()
```

▪ SOLUTION 2

```
const diff = (a, b) => {
  return [...new Set(a.concat(b).filter(x => a.includes(x) ^ b.includes(x)))] .sort()
}
```

▪ SOLUTION 3

```
function diff(a, b){
  let set1 = new Set(a)
  let set2 = new Set(b)
  let diff1 = a.filter(item => !set2.has(item))
  let diff2 = b.filter(item => !set1.has(item))
  return Array.from(new Set(diff1.concat(diff2))) .sort()
}
```

#47 - Nombre du milieu (7 kyu)

▪ SOLUTION 1

```
var gimme = (arr) => arr.indexOf(arr.slice().sort((a,b)=>a-b)[1])
```

Remarquez l'importance du `arr.slice()` sinon `arr.sort()` trierait réellement le tableau or nous voulons conserver l'original.

```
arr=[1,5,3,4]
→ Array [ 1, 5, 3, 4 ]
arr.sort()
→ Array [ 1, 3, 4, 5 ]
arr
→ Array [ 1, 3, 4, 5 ] // arr a été modifié !
arr=[1,5,3,4]
→ Array [ 1, 5, 3, 4 ]
arr.slice().sort()
→ Array [ 1, 3, 4, 5 ] // arr.slice() crée une copie de arr
arr
→ Array [ 1, 5, 3, 4 ] // et donc arr n'est pas modifié
```

On peut aussi faire :

```
arr.concat().sort()
→ Array [ 1, 3, 4, 5 ]
arr
→ Array [ 1, 5, 3, 4 ]
```

Ou encore :

```
[...arr].sort()
→ Array [ 1, 3, 4, 5 ]
arr
→ Array [ 1, 5, 3, 4 ]
```

Soit donc :

```
var gimme = (arr) => arr.indexOf([...arr].sort((a,b)=>a-b)[1])
```

▪ SOLUTION 2

```
gimme = ([a, b, c]) => a < b ? a > c ? 0 :
  b > c ? 2 : 1 :
  b > c ? 1 : a > c ? 2 : 0;
```

▪ SOLUTION 3

```
var gimme = (a) => 3-a.indexOf(Math.min(...a))-a.indexOf(Math.max(...a))
```

Comme 0+1+2=3, il suffit d'enlever les index du plus petit et du plus grand.

#48 - Fonction à partir d'une chaîne (7 kyu)

▪ SOLUTION 1

```
runYourString = (arg, obj) => Function(obj.param,obj.func)(arg)
```

▪ SOLUTION 2

```
function runYourString (arg, obj) {
  this[obj.param] = arg;
  return (new Function(obj.func)).call(this);
}
```

▪ SOLUTION 3

```
function runYourString (arg, obj) {
  var s = eval('(' + obj.param + ')=>{' + obj.func + '}');
  return s(arg);
}
```

#49 - Écran de mobile (7 kyu)

▪ SOLUTION 1

```
const map = ['1234567890*#', 'adgjmptw', 'behknqux', 'cfilorvy', 'sz']
mobileKeyboard = (str) =>
  [...str].reduce((s,c)=>s+map.findIndex(l=>l.includes(c)), str.length)
```

```
t=[1,2,5,4,3,5,2,2] // Le chiffre 2 est à plusieurs endroits
→ Array [ 1, 2, 5, 4, 3, 5, 2, 8 ]
t.findIndex(c=>c==2) // On recherche le 2
→ 1 // Seule la première position est renvoyée
```

Dans le cas ci-dessus on peut bien sûr utiliser `t.indexOf(2)`

▪ SOLUTION 2

```
mobileKeyboard = (str) => {
  var map=['0123456789#*', 'adgjmntw', 'behknqux', 'cfilorvy', 'sz']
  return [...str].reduce((s,c)=>s+1+map.indexOf(map.filter(w=>w.includes(c))[0]),0)
}
```

On recherche déjà la chaîne contenant la lettre avec `filter` puis on utilise `indexOf` pour retrouver sa position dans le tableau. Par exemple le « s » est dans la chaîne « sz » qui est en position 4.

#50 - Jumeaux (7 kyu)

▪ SOLUTION 1

```
const elimination = ar => ar.find((e,i) => i !== ar.lastIndexOf(e)) || null
```

▪ SOLUTION 2

```
elimination= (arr) =>arr.sort().find((n, i) => n === arr[i + 1]) || null
```

#51 - Distribution d'or (7 kyu)

▪ SOLUTION 1

```
distributionOf = (golds) =>{
  var [A,me,s]=[0,true,golds.reduce((a,v)=>a+v)]
  while(golds.length>0) {
    var g=(golds[0]>=golds.slice(-1)) ? golds.shift() : golds.pop()
    if (me) A+=g
    me=!me
  }
  return [A,s-A]
}
```

▪ SOLUTION 2

```
distributionOf = (golds) => {
  var AB=[0,0]
  var turn=0
  while(golds.length>0) AB[turn++%2]+=(golds[0]>=golds.slice(-1)) ? golds.shift() :
golds.pop()
  return AB
}
```

▪ SOLUTION 3

```
distributionOf = (g, a=0, b=g.length-1) =>
  g.reduce((p,_,i) => {
    p[i%2] += g[ b ]>g[ a ] ? b--:a++; return p } , [0,0]);
```


#52 - Médailles (7 kyu)

▪ SOLUTION 1

```
evilCodeMedal = (u, g, s, b) =>
  ["Gold", "Silver", "Bronze"][[g,s,b].findIndex(m=>u<m)] || "None"
```

▪ SOLUTION 2

```
function evilCodeMedal(userTime, gold, silver, bronze) {
  return (userTime<gold)? "Gold": ((userTime<silver)? "Silver" : ((userTime<bronze)?
"Bronze": "None"))
}
```

▪ SOLUTION 3

```
function evilCodeMedal(userTime, gold, silver, bronze) {
  if(userTime < gold) return "Gold";
  if(userTime < silver) return "Silver";
  if(userTime < bronze) return "Bronze";
  return "None";
}
```

#53 - Shushis (7 kyu)

▪ SOLUTION 1

```
function totalBill(s) {
  return s.replace(/ /g, "").replace(/r{5}/g, 'rrrr').length * 2;
}
```

▪ SOLUTION 2

```
totalBill =(str) => {
  var l = str.replace(/\s/g, '').length
  return 2*(l-(l/5 | 0))
}
```

▪ SOLUTION 3

```
var totalBill=(s,r=s.split('r').length-1)=>(r-(r/5|0))*2
```

▪ SOLUTION 4

```
function totalBill(str) {
  const plates = (str.match(/r/g)||[]).length
  return (plates*2) - Math.floor(plates/5)*2
}
```

#54 - Monts et vallées (7 kyu)

▪ SOLUTION 1

```
function peakAndValley(arr){
```

```

const check = (x, xs) => xs.every(y => y > x) || xs.every(y => y < x)
const slice = i => arr.slice(i - 3, i).concat(arr.slice(i + 1, i + 4))
return arr.filter((x, i) => i > 2 && i < arr.length - 3 && check(x, slice(i)))
}

```

- **SOLUTION 2**

```

peakAndValley = (arr) => arr.slice(3,-3).filter((v,k)=>
arr.slice(k,k+7).filter(c=>c>=v).length==1||arr.slice(k,k+7).filter(c=>c<=v).length==1)

```

- **SOLUTION 3**

```

peakAndValley = (a, b) => a.filter((e, i) =>
    i > 2 && i < a.length - 3 &&
    ((b = a.slice(i - 3, i).concat(a.slice(i + 1, i + 4))).every(i => i < e) ||
b.every(i => i > e)));

```

#55 - 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(' ');
}

```

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

- **SOLUTION 1**

```

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

```

- **SOLUTION 2**

```

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

```

Les solutions 1 et 2 convertissent le nombre en binaire :

```
(17).toString(2)
```

```
→ "10001"
```

On enlève ensuite les « 0 » par replace ou split-join et on compte les « 1 » restants.

- **SOLUTION 3**

```

function countBits(n) {

```

```

    for (c=0;n;n>>=1)c+=n&1
    return c;
}

```

Version en calcul binaire :

```

(37).toString(2)           // 37 s'écrit « 100101 » en binaire
→ "100101"
37>>1                      // On décale d'un bit vers la droite
→ 18                       // ce qui revient à diviser le nombre par 2
(18).toString(2)          // On retrouve bien « 100101 » / 2 → « 10010 »
→ "10010"
18>>1                      // On recommence en décalant à droite
→ 9
(9).toString(2)           // « 10010 » / 2 → « 1001 »
→ "1001"
On regarde à chaque fois si l'unité vaut 1.
3&1                         // 3 s'écrit « 11 » en binaire → unité = 1
→ 1
6&1                         // 6 s'écrit « 110 » → unité = 0
→ 0

```

#57 - 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) ] : [] ;

```

#58 - 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');
}
```

#59 - Touches d'un piano (6 kyu)

▪ SOLUTION 1

```
function blackOrWhiteKey(k) {
  return [1,4,6,9,11].includes((k-1)%88%12) ? "black" : "white"
}
function whichNote(k) {
  return 'A,A#,B,C,C#,D,D#,E,F,F#,G,G#'.split(',')[((k-1)%88)%12];
}
```

▪ SOLUTION 2

```
var [w, b] = ["white", "black"]
var key_wb = [w, b, w, w, b, w, b, w, w, b, w, b]
var key_notes=["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]

function blackOrWhiteKey(k) {
  return key_wb[(k - 1) % 88 % 12]
}

function whichNote(k) {
  return key_notes[(k - 1) % 88 % 12]
}
```

#60 - 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, '')
```

#61 - 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('')
}
```

#62 - 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))
}
```

```
}
```

#63 - Notation Polonaise Inverse (6 kyu)

▪ SOLUTION 1

```
function solvePostfix(pfx){
  var s=pfx.split(' ')
  var pile=[parseInt(s.shift())]
  while (s.length>0) {
    var c=s.shift()
    if (isNaN(parseInt(c))) {
      var b=pile.pop()
      var a=pile.pop()
      pile.push((c=="^") ? Math.pow(a,b) : eval("(" + a + ")" + c + "(" + b + ")"))
    } else {
      pile.push(parseInt(c))
    }
  }
  return pile[0]
}
```

▪ SOLUTION 2

```
function solvePostfix(pfx){
  var stack = [];
  pfx.split(' ').forEach(function(v) {
    stack.push( "+-*/^".indexOf(v)>-1 ? OPS[v](stack.pop(),stack.pop()) : +v )
  });
  return stack[0];
}

var OPS = {
  "+" : function(a,b){return a+b},
  "-" : function(b,a){return a-b},
  "*" : function(a,b){return a*b},
  "/" : function(b,a){return a/b},
  "^" : function(b,a){return Math.pow(a,b)}
}
```

▪ SOLUTION 3

```
function solvePostfix(pfx){
  var obj={
    '*':(a,b)=>a*b,
    '/':(a,b)=>a/b,
    '+':(a,b)=>a+b,
    '-':(a,b)=>a-b,
    '^':(a,b)=>Math.pow(a,b)
  }
  while(pfx.indexOf(' ')!==-1){
    pfx=pfx.replace(/(\-?\d+) (\-?\d+) (\+|\-|\*|\/|\^)/,function(_,a,b,c){
      return obj[c](+a,+b);
    })
  }
  return +pfx;
}
```

▪ SOLUTION 4

```
function solvePostfix(s) {
  var r=(-?[\d\.]+) (-?[\d\.]+) ([-+*\^\/^)]/
  while(r.test(s))
s=s.replace(r, (_ ,a,b,op)=>op=="^"?Math.pow(a,b):eval((a+op+b).replace("--","+")))
  return +s
}
```

#64 - 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!`;
  }
}
```

#65 - 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;
}
```

#66 - 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]];
}
```

#67 - 10 minutes de promenade (6 kyu)

▪ SOLUTION 1

```
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)
}
```

#68 - 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(/{\d}/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`;
  }
}

```

#69 - Trier mes animaux (6 kyu)

▪ SOLUTION 1

```

sortAnimal=(animal) => animal ?
  animal.sort((a,b)=>
    a.numberOfLegs==b.numberOfLegs ?
      a.name.localeCompare(b.name) :
      a.numberOfLegs-b.numberOfLegs)
  : animal

```

▪ SOLUTION 2

```

function sortAnimal(animals) {
  function sort_by_legs_then_name(a, b) {
    return a.numberOfLegs === b.numberOfLegs ? a.name.localeCompare(b.name) :
    a.numberOfLegs - b.numberOfLegs;
  };

  return animals && animals.length ? animals.sort(sort_by_legs_then_name) : animals ? []
  : null;
}

```

▪ SOLUTION 3

```
function sortAnimal(animals) {
  return !animals ? null :
    animals.slice().sort((a, b) => a.numberOfLegs - b.numberOfLegs ||
a.name.localeCompare(b.name));
}
```

#70 - Les rats sourds de Hamelin (6 kyu)

▪ SOLUTION 1

```
var countDeafRats = (town) => {
  [l,r]=town.split('P')
  return ([...l].concat([...r].reverse()).filter((c,k)=>c=="~" && k&1) || []).length
}
```

Les rats à gauche du joueur de flûtes doivent être comme ~O et ceux à droite comme O~.

On inverse l'orientation de ceux de droite et on les ajoute à ceux de gauche. L'ensemble des rats devraient donc normalement ressembler à : ~O~O~O~O~O~O~O~O~O. On compte alors combien il y a de ~ aux positions impaires (k&1).

▪ SOLUTION 2

```
var countDeafRats = function(town) {
  if(town != null){
    [left,right]=town.split('P');
    var a=left+right.split('').reverse().join('');
    var b=(a.match(/O~|~O/gi) || []).filter(v=>v=='O~').length;
    return b;
  }
  return 0;
}
```

#71 - 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;
}

```

#72 - 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;

```

#73 - 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);
}

```

#74 - Dactylographe (6 kyu)

▪ SOLUTION 1

```
function typist(s){
  return (s.replace(/(.)/g, '$1$1').match(/^[A-Z]|[a-z][A-Z]|[A-Z][a-z]/g) || []).length+s.length
}
```

On double toutes les lettres car :

```
"abAa".match(/^[A-Z]|[a-z][A-Z]|[A-Z][a-z]/g)
→ Array [ "bA" ] // On ne voit pas le changement « Aa »
"abAa".replace(/(.)/g, '$1$1') // en doublant les lettres
"aabbAAaa"
"abAa".replace(/(.)/g, '$1$1').match(/^[A-Z]|[a-z][A-Z]|[A-Z][a-z]/g)
Array [ "bA", "Aa" ] // on a bien tous les changements
```

▪ SOLUTION 2

```
function typist(s){
  var r = s.replace(/([A-Z]+)/g, "-$&-");
  if (r[r.length-1] == '-') r = r.substring(0, r.length-1);
  return r.length;
}
```

On remplace les suites adjacentes de majuscules par « - » suivi de toute l'expression (\$&) et à nouveau « - », ce qui revient à simuler la frappe de la touche Majuscule au clavier. Si la dernière lettre est une majuscule, il faut enlever 1 car on n'a pas à taper sur Majuscule à la fin de la saisie.

On peut réécrire plus simplement cette version en :

```
function typist(s){
  var r = s.replace(/([A-Z]+)/g, "-$&-");
  return r.length - ((r.slice(-1) == '-') ? 1 : 0);
}
```

▪ SOLUTION 3

```
function typist(s) {
  return s.length + ('a' + s).match(/([A-Z]+|[a-z]+)/g).length - 1;
}
```

Pour régler le cas de la majuscule initiale, on ajoute une minuscule au début puis on cherche toutes les séries de majuscules ou minuscules.

▪ SOLUTION 4

```
function typist(s) {
  var res = s.length, clPressed = false;
  for (let i = 0; i < s.length; i++) {
    if (clPressed ? s[i].toUpperCase() !== s[i] : s[i].toLowerCase() !== s[i]) {
      res++; clPressed = !clPressed;
    }
  }
  return res;
}
```

```
}
```

#75 - 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('');
}
```

#76 - 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();
}

```

#77 - Discours politique (6 kyu)

- **SOLUTION 1**

```

trumpDetector=(trumpySpeech)=>+(trumpySpeech
  .match(/a+|e+|i+|o+|u+/gi) || [])
  .reduce((a,v,_,t)=>a+(v.length-1)/t.length,0)
  .toFixed(2)

```

- **SOLUTION 2**

```

function trumpDetector(trumpySpeech){
  arr = trumpySpeech.match(/([aeiou]) (\1*)/ig)
  return +(arr.map(x => x.length-1).reduce((x,y) => x+y, 0) / arr.length).toFixed(2);
}

```


#78 - 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);
```

L'astuce ici consiste à utiliser le « ou exclusif » ($0^0=1^1=0$ et $1^0=1$), en effet :

```
3^3 // 3 s'écrit « 11 » en binaire donc 11^11 = 00
→ 0
0^5 // 5 s'écrit « 101 » donc 000^101=101
→ 5
```

Ainsi, en faisant la somme de tous les termes en utilisant le ou exclusif, tous les termes s'élimineront... sauf celui que l'on cherche.

▪ 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)
}
```

#79 - 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[1+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
}

```

#80 - 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);
}

```

#81 - 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";
  }
```

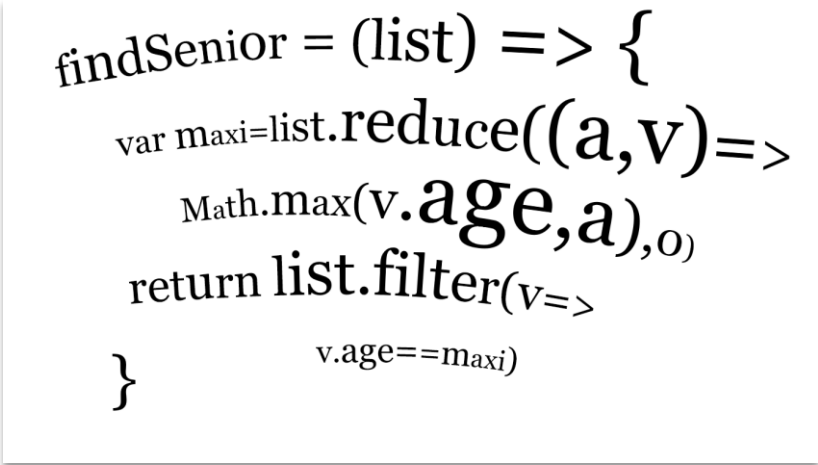
#82 - Les séniors (6 kyu)

• SOLUTION 1

```
function findSenior(list) {
  var maxAge = Math.max(...list.map(person => person.age));
  return list.filter(person => person.age === maxAge);
}
```

• SOLUTION 2

```
findSenior = (list) => {
  var maxi=list.reduce((a,v)=>Math.max(v.age,a),0)
  return list.filter(v=>v.age==maxi)
}
```



```
findSenior = (list) => {
  var maxi=list.reduce((a,v)=>
    Math.max(v.age,a),0)
  return list.filter(v=>
    v.age==maxi)
}
```

• SOLUTION 3

```
function findSenior(list) {
  var high = list.map(a => a.age).sort((a,b) => b - a)[0];
  return list.filter(a => a.age == high);
}
```

#83 - Diversité des langages (6 kyu)

• SOLUTION 1

```
function isLanguageDiverse(list) {
  var PRJ=['Python', 'Ruby', 'JavaScript'].map(l=>list.filter(v=>v.language==l).length)
  return Math.max(...PRJ)<=2*Math.min(...PRJ)
}
```

• SOLUTION 2

```
function isLanguageDiverse(list) {
```

```

var counter = {};
list.map(p => counter[p.language] = (counter[p.language] || 0) + 1);
var vals = Object.keys(counter).map(lang => counter[lang]);
return Math.max.apply(null, vals) <= 2* Math.min.apply(null, vals);
}

```

#84 - 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;

```

Lodash (<https://lodash.com/>) est une bibliothèque JS permettant de se faciliter la vie. Allez sur le site puis dans la console tapez par exemple :

```

_.range(10)
→ Array [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
_.random(1,6)
→ 4

```

#85 - 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+(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
}

```

#86 - 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('')
})
}

```

Ou encore :

```

spinWords=(s)=>s.replace(/\w{5,}/g, (w)=>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(' ')
}

```

#87 - Liste de courses (6 kyu)

- **SOLUTION 1**

```

shoppingListCost = (arr) => +arr.reduce((a, [item, qty])=>{
    var gr=groceries[item];
    return a+(gr.bogof? Math.ceil(qty/2) : qty) *gr.price*(1-gr.discount/100)
},0)
.toFixed(2)

```

- **SOLUTION 2**

```

shoppingListCost = (arr) => +arr.reduce((a, [elt, qty])=>{
    var [{price, discount, bogof}]=groceries[elt];
    return a+(bogof? Math.ceil(qty/2) : qty) *price*(1-discount/100)
},0)
.toFixed(2)

```

#88 - 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;
}

```

```
}
```

Il faut se convaincre que le reste de la division de $\overline{x00\dots0}$ par 9 est x

Imaginez par exemple $300 = 3*(99+1) = 3*99 + 3$ donc le reste sera bien 3. Le problème est quand $x = 9$ puisque le reste serait 0 au lieu de 9. L'idée est d'enlever 1 (par exemple 900 devient 899 dont le reste est 8) puis d'ajouter le 1.

#89 - 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);;
}
```

#90 - Nombre unique (6 kyu)

▪ SOLUTION 1

```
findUniq = a => a.find(v=>a.indexOf(v)==a.lastIndexOf(v))
```

▪ SOLUTION 2

```
function findUniq(arr) {
  arr.sort((a,b)=>a-b);
  return arr[0]==arr[1] ? arr.pop() : arr[0]
}
```

▪ SOLUTION 3

```
function findUniq(arr) {
  let [a,b,c] = arr.slice(0,3);
  if( a != b && a!=c ) return a;
  for( let x of arr ) if( x!=a ) return x
}
```

#91 - Tribonacci (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 }
```

#92 - 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);
```


`.test` renvoie **true** ou **false** et on utilise toujours l'astuce `0+true=1`

- **SOLUTION 3**

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

- **SOLUTION 4**

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

#93 - Répétition de lettres (6 kyu)

- **SOLUTION 1**

```
function duplicateCount(text){  
  return new Set(text.toLowerCase().match(/(.) (?=.*\1)/gi)).size  
}
```

- **SOLUTION 2**

```
function duplicateCount(text){  
  return (text.toLowerCase().split('').sort().join('').match(/([^\s])\1+/g) || []).length;  
}
```

- **SOLUTION 3**

```
duplicateCount=(text)=> {  
  var t=text.toLowerCase()  
  return [...t].reduce((a,c)=>t.indexOf(c)!=t.lastIndexOf(c) &&  
  !a.includes(c)?a+c:a,"").length  
}
```

#94 - Michaël (6 kyu)

- **SOLUTION 1**

```
getMichaelLastName = (inputText) =>  
  inputText.match(/Michael\s([A-Z]\w+)/g).map(n=>n.split(' ')[1])
```

- **SOLUTION 2**

```
function getMichaelLastName(inputText) {  
  return (inputText.match(/Michael [A-Z]\w+/g) || []).map(x => x.replace(/Michael /, ''));  
}
```

- **SOLUTION 3**

```
const getMichaelLastName = text =>  
  text  
    .match(/Michael [A-Z][a-z]+/g)  
    .map(m => m.slice(8));
```

#95 - 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))
```

#96 - Détection de cycles (6 kyu)

▪ SOLUTION 1

```
cycle = function(sequence){
  var mu_idx=sequence.findIndex((n,k,arr)=>arr.slice(k+1).indexOf(n)>-1)
  var lambda_idx=sequence.slice(mu_idx+1).indexOf(sequence[mu_idx])
  return (mu_idx>-1) ? [mu_idx, lambda_idx+1] : []
}
```

▪ SOLUTION 2

```
cycle = function (sequence) {
  var seen = {}
  for (var i=0; i<sequence.length; i++) {
    if (sequence[i] in seen)
      return [seen[sequence[i]], i-seen[sequence[i]]]
    seen[sequence[i]] = i
  }
  return []
}
```

▪ SOLUTION 3

```
cycle = function(sequence){
  for(i=0; i<sequence.length; ++i)
    for(j=i+1; j<sequence.length; ++j){
      if(sequence[i] === sequence[j]) return [i, j-i];
    }
  return []
}
```

#97 - 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)
  };
}

```

#98 - 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('')
```

#99 - 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;
}
```

#100 - 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']()));
```

#101 - Codes secrets par mobile (6 kyu)

• SOLUTION 1

```
function unlock(str)
{
  var codes=[["ABC",2],["DEF",3],["GHI",4],["JKL",5],["MNO",6],
             ["PQRS",7],["TUV",8],["WXYZ",9]]
  return [...str].reduce((a,c)=>a+codes.find(r=>r[0].indexOf(c.toUpperCase())>-1)[1], '')
}
```

• SOLUTION 2

```
unlock=(str) => {
  var iOf="indexOf"
  var codes=["ABC","DEF","GHI","JKL","MNO","PQRS","TUV","WXYZ"]
  return [...str].reduce((a,c)=>a+(2+codes[iOf](codes.find(r=>
    r[iOf](c.toUpperCase())>-1))), '')
}
```

• SOLUTION 3

```
function unlock(str)
{ let code = ['abc', 'def', 'ghi', 'jkl', 'mno', 'pqrs', 'tuv', 'wxyz'];
  let nums = '';
  str.split('').forEach((le)=>{
    code.forEach((l,i)=> {
      var re = new RegExp('[' + l + ']', 'gi');
      if(re.test(le)){nums += (i+2)}
    });
  });
  return nums;
}
```

• SOLUTION 4

```
function unlock(str) {
  var pairs = {'abc':2,'def':3,'ghi':4,'jkl':5,'mno':6,'pqrs':7,'tuv':8,'wxyz':9}
  return str.toLowerCase().split('').map(e1 => {
    for(key in pairs) {
      if(key.indexOf(e1) !== -1)
        return pairs[key];
    }
  }).join('');
}
```

#102 - Substitution de lettres (6 kyu)

• SOLUTION 1

```
var alpha="abcdefghijklmnopqrstuvwxy"
function randomSub() {
```

```

var rand=[...alpha].sort((a,b)=>Math.random()-0.5)
return [...alpha].reduce((a,c,k)=>{a[c]=rand[k]; return a},{})
}

```

▪ SOLUTION 2

```

function randomSub() {
  var obj = {};
  var alphabet = "abcdefghijklmnopqrstuvwxy";
  var value = alphabet.split("");
  for (var i in alphabet) {
    var index = Math.floor(Math.random() * value.length);
    obj[alphabet[i]] = value[index];
    value.splice(index, 1);
  }
  return obj
}

```

#103 - 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;
}

```

#104 - 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;
}
```

#105 - 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"
}
```

#106 - 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;
}
```

#107 - Faux sites web (6 kyu)

▪ SOLUTION 1

```
function goodName(name) {
  let goodNames = [], end = name.lastIndexOf(".");
  for( let i=0, char; i< end; i++ ){
    char = name[i];
    if( char=="o" ) goodNames.push( repAt(name,i,"0" ) )
    if( char=="l" ) goodNames.push( repAt(name,i,"1" ) )
    if( char==name[i+1] && (!i || char!=name[i-1]) ) goodNames.push( repAt(name,i,"") )
  }
  return goodNames.sort()
}
```

```
const repAt = ([...chars], idx, rep) =>{ chars[idx]=rep; return chars.join("") };
```

▪ SOLUTION 2

```
function goodName(name) {
  var reRules = {
    'o' : 0,
    'l' : 1,
    '(.)\\1+' : ''
  };
  var arr = [];

  for( var prop in reRules ){
    var re = new RegExp( prop + '(?=.*\\.)', 'g' );

    while ( (match = re.exec(name)) !== null ) {
```



```

    arr.push( name.substr(0, match.index) + reRules[prop] + name.substr(match.index +
1) );
  }
}

return arr.sort();
}

```

▪ SOLUTION 3

```

function goodName(name) {
  var s = []
  var adrs = name.split('.')
  var t = adrs[0].split('')
  var res = t.reduce((a, v, k) => {
    if (v == 'o') a['0'].push(k)
    if (v == 'l') a['1'].push(k)
    if (v == t[k+1]) a[''].push(k)
    return a
  }, {'0': [ ], '1': [ ], '':[ ]})
  for (c in res) {
    var preced=-2
    for (k of res[c]) {
      if ((c!='') || (c=='' && k!=preced+1)) s.push(t.slice(0, k).join('') + c +
t.slice(k + 1).join('')+ '.'+adrs[1])
      preced=k
    }
  }
  return s.sort()
}

```

#108 - Trop c'est trop (6 kyu)

▪ 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))
```

#109 - Cellules cancéreuses (6 kyu)

▪ SOLUTION 1

```
const cutCancerCells = (organism) => organism.replace(/[^A-Z]?C[^A-Z]?|c/g, '')
```

▪ SOLUTION 2

```
function cutCancerCells(o){
  return o.replace(/c|[a-z]?C[a-z]?/g, '')
}
```

#110 - Gendarmes et voleurs (6 kyu)

▪ SOLUTION 1

```
function catchThief(q){
  q=q.split``
  for(i=0;i<q.length;i++) if(!isNaN(q[i]))for(j=i-q[i];j<=+q[i]+i;j++)
    if(q[j]== 'X') q[j]='!'
  return q.filter(p=>p=='!').length
}
```

▪ SOLUTION 2

```
function catchThief(queue){
  var out=queue.split('')
  for (var i in queue) {
    if (/\\d/.test(queue[i])) {
      var l+=queue[i]
      out=out.map((c,k)=>k>=i-1 && k<=+i+1 ? "#" : c)
    }
  }
  return queue.match(/X/g).length-out.filter(c=>c=='X').length
}
```

▪ SOLUTION 3

```
function catchThief(queue){
  var watched = [...queue].fill(0);
  for(let i=0; i<queue.length; i++) if(/\\d/.test(queue[i])) {
    for(let j=Math.max(0,i-queue[i]); j<=Math.min(queue.length-1,+queue[i]+i); j++)
      watched[j]=1;
  }
}
```

```

return [...queue].filter((c,i)=>c==='X' && watched[i]).length;
}

```

#111 - Types de données (6 kyu)

▪ SOLUTION 1

```

function dataTypes (string) {
  return string.replace(/(true|false)/ig, ' $1 ').match(/[a-z]+\d+/ig).map(function(x) {
    return /^true|false$/i.test(x) ? 'boolean' : /\d/.test(x) ? 'number' : 'string';
  });
}

```

▪ SOLUTION 2

```

function dataTypes (string) {
  return string.replace(/true|false/ig, 'ÿ').match(/ÿ|\d+|^[^0-9ÿ ]+/gi).map(a=>{
    return /ÿ/.test(a) ? 'boolean' :
      /\d+/.test(a) ? 'number' : 'string';
  });
}

```

▪ SOLUTION 3

```

const trans = {"+": "boolean", "-": "number", "=": "string"}

function dataTypes (string) {
  return string.replace(/true|false/gi, '+')
    .replace(/\d+/gi, '-')
    .replace(/[a-z]+/gi, '=')
    .replace(/^[^\+\-\=]/g, '')
    .split('')
    .map(e=> trans[e])
}

```

#112 - Lettres alternées ? (6 kyu)

▪ SOLUTION 1

```

function isAlt(word) {
  return ![aeiou]{2}|^[aeiou]{2}/.test(word);
}

```

▪ SOLUTION 2

```

function isAlt(word) {
  return /^[^aeiou]?([aeiou][^aeiou])*[aeiou]?$/i.test(word);
}

```

▪ SOLUTION 3

```

const isVowel = c => /[aeiou]/.test(c);
const isAlt = ([a,b,...c]) =>
  b===undefined || isVowel(a) !== isVowel(b) && isAlt([b,...c]);

```

▪ SOLUTION 4

```
function isAlt=(word) =>
  word.split("").every((c,i)=>i==0||/[aeiou]/.test(c)!=[aeiou]/.test(word[i-1]))
```

#113 - Langage Tick (6 kyu)

▪ SOLUTION 1

```
function interpreter(tape) {
  var code = tape.match(/(\++|\*+|>+|<+)/g)
  var rub = {}
  var s = ''
  var pos = 0
  for (c of code) {
    var l=c.length
    switch (c[0]) {
      case '+': rub[pos] = ((rub[pos] || 0) + 1) %256; break;
      case '*': s += String.fromCharCode(rub[pos]).repeat(l); break;
      case '>': pos+=1; break;
      case '<': pos-=1; break;
    }
  }
  return s
}
```

▪ SOLUTION 2

```
function interpreter(tape) {
  const mem = []
  let ptr = 0
  let out = ""
  for (const op of tape) {
    switch (op) {
      case ">": ++ptr; break
      case "<": --ptr; break
      case "+": mem[ptr] = (mem[ptr] | 0) + 1 & 255; break
      case "*": out += String.fromCharCode(mem[ptr]); break
    }
  }
  return out
}
```

#114 - Longueur de la clé (6 kyu)

▪ SOLUTION 1

```
function findTheKey(code){
  var c = code.split('')
  for(let l=1; l<=c.length; l++){
    if(c.every((n,i)=>n===c[i%l])) return +c.slice(0,l).join('');
  }
}
```

▪ SOLUTION 2

```
function findTheKey(code){
  var n=1
```

```

while (code.indexOf(code.split(code.slice(0,n)).join(''))!=0) n++
return +code.slice(0,n)
}

```

- **SOLUTION 3**

```

function findTheKey(code) {
  var len = code.length
  for (let i = 1; i < len; i++) {
    var shortest = code.slice(0, i);
    if (shortest.repeat(code.length).slice(0, len) == code) return +shortest
  }
  return +code
}

```

- **SOLUTION 4**

```

findTheKey = (s) =>
+s.slice(0,1+[...s].findIndex((_,k)=>
  s.slice(0,k+1).repeat(s.length).slice(0,s.length)==s))

```

#115 - Exercices sur les nœuds (6 kyu et 7 kyu)

- **SOLUTION - LENGTH**

```

var length=(h)=>h?1+length(h.next):0

```

- **SOLUTION - INDEXOF**

```

var indexOf=(h,v)=>{
  var idx=0
  while(h) {
    if (h.data===v) return idx
    else {idx++; h=h.next}
  }
  return -1
}

```

```

var indexOf=(h,v)=>{
  var idx=0
  while(h) {
    if (h.data===v) return idx
    }else {idx++; h=h.next}
  }
  return -1
}

```

▪ SOLUTION 1 – LASTINDEXOF

```
function lastIndexOf(head, value) {
  var ans = - 1;
  for (var i = 0; head; head = head.next, i++){
    if (head.data === value){
      ans = i;
    }
  }
  return ans;
}
```

▪ SOLUTION 2 – LASTINDEXOF

```
function lastIndexOf(head, value) {
  const list = (node) => node ? [node.data, ...list(node.next)] : []
  return list(head).lastIndexOf(value);
}
```

▪ SOLUTION 1 – ANYMATCH ET ALLMATCH

```
function anyMatch(head, p) {
  return head ? p(head.data) || anyMatch( head.next, p ) : false
}
```

```
function allMatch(head, p) {
  return head ? p(head.data) && allMatch( head.next, p ) : true
}
```

▪ SOLUTION 2 – ANYMATCH ET ALLMATCH

```
function anyMatch(head, p) {
  for (; head; head = head.next) {
    if (p(head.data)) return true;
  }
  return false;
}
```

```
function allMatch(head, p) {
  return !anyMatch(head, x => !p(x));
}
```

▪ SOLUTION 3 – ANYMATCH ET ALLMATCH

```
function anyMatch(h, p) {
  while(h) {
    if (p(h.data)) return true
    h=h.next
  }
  return false
}
```

```
var allMatch=(h,p)=> !anyMatch(h, x => !p(x))
```

▪ SOLUTION – COUNTIF

```
countIf = (h, p) => h ? +p(h.data) + countIf(h.next, p) : 0
```

- **SOLUTION – FILTER**

```
var filter=(h,p)=>h?  
  p(h.data)?new Node(h.data,filter(h.next,p)):filter(h.next,p)  
  :null
```

- **SOLUTION - MAP**

```
var map=(h,f)=>h?new Node(f(h.data),map(h.next, f)):null
```

- **SOLUTION - REDUCE**

```
reduce=(h, f, init) => h? reduce(h.next,f,f(init,h.data)) : init
```

- **SOLUTION – FUSION DE 2 LISTES ORDONNÉES**

```
mergeTwoLists =(l1, l2) => {  
  if (!l1 || !l2) return l1 || l2  
  if (l1.data < l2.data) {l1.next = mergeTwoLists(l1.next, l2); return l1}  
  else {l2.next = mergeTwoLists(l1, l2.next); return l2}  
}
```

#116 - Somme max dans un arbre (6kyu)

- **SOLUTION**

```
maxSum = (r) => r?  
  r.value+Math.max(maxSum(r.left),maxSum(r.right))  
  : 0
```

#117 - 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(':')  
}
```

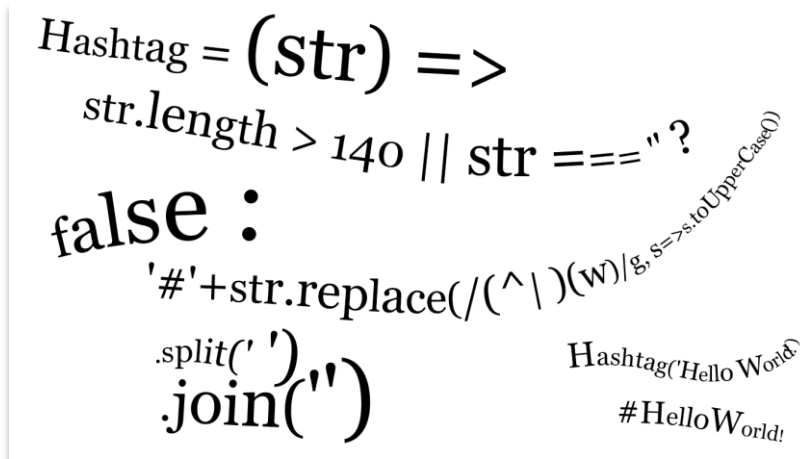
- **SOLUTION 2**

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

#118 - 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);
}

```

#119 - 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");
}

```

#120 - 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());
}
```

#121 - 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('');
  });
}
```

#122 - 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];
}
```

#123 - 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)
  }, [])
}
```

#124 - Un problème de poids (5 kyu)

▪ SOLUTION 1

```
sum=n=>[...n.toString()].reduce((a,c)=>a+ +c,0)
orderWeight = (strng) => strng
  .split(' ')
  .sort((a,b)=>sum(a)-sum(b) || (a<b?-1:1))
  .join(' ')
```

▪ SOLUTION 2

```
function orderWeight(strng) {
  const sum = (str)=>str.split('').reduce((sum,el)=>(sum+(+el)),0);
  function comp(a,b){
    let sumA = sum(a);
    let sumB = sum(b);
    return sumA === sumB ? a.localeCompare(b) : sumA - sumB;
  };
  return strng.split(' ').sort(comp).join(' ');
}
```

#125 - 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;
}
```

#126 - 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();
  });
}
```

#127 - Troues 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)
  }
}

```

#128 - 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;
}

```

#129 - 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;
}

```

#130 - 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;
}

```

#131 - Combinaisons téléphoniques (5 kyu)

▪ SOLUTION 1

```

var letterCombinations = function(digits) {
    var map = {"2": ["a", "b", "c"], "3": ["d", "e", "f"], "4": ["g", "h", "i"], "5": ["j",
"k", "l"], "6": ["m", "n", "o"], "7": ["p", "q", "r", "s"], "8": ["t", "u", "v"], "9": ["w",
"x", "y", "z"]};
    var rtn = map[digits[0]];
    digits = digits.substr(1);
    digits.split("").forEach((digit) => {
        var t = [];
        map[digit].forEach((letter) => t = t.concat(rtn.map((item) => item + letter)));
        rtn = t;
    });
    return rtn || [];
};

```

#132 - 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 Schroeppe12 = 0xAAAAAAAA;
const intToNegabinary = i => ( ( i + Schroeppe12 ) ^ Schroeppe12 ).toString(2);
```

```
parseInt("0xA",16)
→ 10
parseInt("0xAA",16)
→ 170
(10).toString(2)           // On alterne 1 et 0
→ "1010"
(170).toString(2)
→ "10101010"
parseInt("0xAAAAAAAA",16)
→ 2863311530
(2863311530).toString(2)
→ "10101010101010101010101010101010"
```

▪ SOLUTION 3

```
function intToNegabinary( number ) {
  var Schroeppe14 = 0xAAAAAAAA;
  // Convert to NegaQuaternary String
  return ( ( number + Schroeppe14 ) ^ Schroeppe14 ).toString(2);
}
function negabinaryToInt(s) {
  return [...s].reduce((i,d)=>+d-i*2, 0);
}
```

#133 - 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;
}
```

#134 - 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)]=""
  }
  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])
}

```

#135 - Hunger Games au zoo (5 kyu)

▪ SOLUTION 1

```

var whoEatsWho = function(zoo) {
  var input = zoo.split(',')
  if (zoo=="") return ['', '']
  var out=[zoo]
  var animals=['antelope','grass','big-fish','little-fish','bug',
  'leaves','bear','chicken','cow','sheep','fox','giraffe','lion','panda']
  var eat={0:[1], 1:[], 2:[3], 3:[], 4:[5], 5:[], 6:[2,4,5,7,8,9], 7:[4], 8:[1], 9:[1],
10:[7,9], 11:[5], 12:[0,8], 13:[5], 14:[]}
  var anim=(k)=>{var a=animals.indexOf(input[k]); return (a>-1) ? a : 14 }
  var canEat=(k,dir)=>eat[anim(k)].indexOf(anim(k+dir))>-1
  var doIt=(k,dir)=>{out.push(input[k]+" eats "+input[k+dir]); input.splice(k+dir,1);
n=0}
  var n=0
  while (n<input.length) {
    if (canEat(n,-1)) doIt(n,-1)
    else if (canEat(n,1)) doIt(n,1)
    else n++
  }
  return out.concat(input.join(','))
}

```

```
}
```

▪ SOLUTION 2

```
const eats={
  antelope:{grass: true},
  'big-fish':{'little-fish': true},
  bug:{leaves: true},
  bear: {'big-fish': true, bug: true, chicken: true, cow: true, leaves: true, sheep:
true},
  chicken:{bug: true},
  cow:{grass: true},
  fox:{chicken: true, sheep: true},
  giraffe:{leaves: true},
  lion:{antelope: true, cow: true},
  panda:{leaves: true},
  sheep:{grass: true}
}

function whoEatsWho(zoo) {
  var res=zoo.split(','), proc=[];
  for(let i=0; i<res.length; i++) {
    if(eats[res[i]]&&eats[res[i]][res[i-1]]) {proc.push(`${res[i]} eats ${res.splice(i-
1,1)[0]}`); i=-1;}
    else if(eats[res[i]]&&eats[res[i]][res[i+1]]) {proc.push(`${res[i]} eats
${res.splice(i+1,1)[0]}`); i=-1;}
  }
  return [zoo, ...proc, res.join(',')];
}
```

#136 - 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);
}

```

#137 - banana (5 kyu)

▪ SOLUTION 1

```

var bananas = function(s, text = 'banana') {
  if (s.length < text.length) return []
  if (!text.length) return [ s.split('').map(() => '-').join('') ]
  return [].concat(
    s[0] === text[0] ? bananas(s.slice(1), text.slice(1)).map(x => s[0] + x) : [],
    bananas(s.slice(1), text).map(x => '-' + x)
  )
}

```

▪ SOLUTION 2

```

const banana = 'banana';

var bananas = function(s,idx=0,last=0,r) {
  var lst = [];
  for(let i=last; i<s.length; i++) if(s[i]===banana[idx]) {
    if(idx+1===banana.length) lst.push([i]);
    else lst = lst.concat(bananas(s,idx+1,i+1,true).map(l=>[i,...l]));
  }
  return r?lst:lst.map(vs=>vs.reduce((a,i)=>(a[i]=s[i],a),Array(s.length).fill('-'))).join(''));
}

```

▪ SOLUTION 3

```

var bananas = function(s) {
  const target = "banana",
        result = [],
        stack = [[0, // position in input string
                  0, // position in target string
                  ""]]; // acc string

  while (stack.length) {
    const [i,j,acc] = stack.pop();

    if (s.length - (i+1) >= target.length - j) {

```

```

    // if it's possible to use the remaining letters to match the target
    stack.push([i+1, j, acc + "-"]);
  }

  if (s[i] === target[j]) {
    if (j === target.length - 1) {
      // if we have a complete match, save it
      result.push(acc + s[i] + "-".repeat(s.length - (i+1)));
    } else if (s.length - (i+1) >= target.length - (j+1)) {
      // else if it's possible to use the remaining letters to make a match
      stack.push([i+1, j+1, acc + s[i]]);
    }
  }
}
}

return result;
}

```

• SOLUTION 4

```

var bananas = function(s) {
  for (var v,b,r=[],l=s.length,i=0;i<l<<l;i++)
    if ((b=[...v=i.toString(2)].reduce((s,a)=>s+ +a,0))==6) {
      for (var c='',v=('0'.repeat(l)+v).slice(-l),j=0;j<l;j++) c+=v[j]=='1'?s[j]:'-';
      if (c.replace(/-/g,'')=='banana') r.push(c);
    }
  return r
}

```

#138 - 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));
};

```

#139 - Hauteur de pluie (5 kyu)

▪ SOLUTION 1

```

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 2

```

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;
}

```

#140 - Vecteurs (5 kyu)

▪ SOLUTION 1

```

class Vector {
    constructor(x,y,z) { this.i=x, this.j=y, this.k=z; }
    getMagnitude() { return Math.sqrt(this.dot(this)); }
}

```

```

getMagnitude() { return Math.hypot(this.i,this.j,this.k); }
static getI() { return new Vector(1,0,0); }
static getJ() { return new Vector(0,1,0); }
static getK() { return new Vector(0,0,1); }
add(that) { return new Vector(this.i+that.i,this.j+that.j,this.k+that.k); }
multiplyByScalar(n) { return new Vector(this.i*n,this.j*n,this.k*n); }
dot(that) { return this.i*that.i+this.j*that.j+this.k*that.k; }
cross(that) { return new Vector( this.j*that.k-this.k*that.j, this.k*that.i-
this.i*that.k, this.i*that.j-this.j*that.i ); }
isParallelTo(that) { return Boolean(this.getMagnitude()) &&
Boolean(that.getMagnitude()) && Math.abs(this.cross(that).getMagnitude())<Vector.EPSILON
; }
isPerpendicularTo(that) { return Boolean(this.getMagnitude()) &&
Boolean(that.getMagnitude()) && Math.abs(this.dot(that))<Vector.EPSILON ; }
normalize() { return this.multiplyByScalar(1/this.getMagnitude()); }
isNormalized() { return Math.abs(this.getMagnitude()-1)<Vector.EPSILON; }
static get EPSILON() { return 1e-6; }
}

```

#141 - NSA et espionnage (5 kyu)

▪ SOLUTION 1

```

var NSA = {
  log: function(person) {
    try {
      return person._log || 'No Entries';
    }
    finally {
      delete person._log;
    }
  }
};

function communicator(type) {
  return function(phone) {
    for (var i = 1; i < arguments.length; ++i)
      this._log = (this._log ? this._log + '\n' : '') + this.name + ' ' + type + 'ed ' +
arguments[i].name + ' from ' + phone.owner.name + '\s phone(' + phone.number + ')';
  };
}

function Person(name) {
  this.name = name;
}

Person.prototype = {
  call: communicator('call'),
  text: communicator('text')
};

```

▪ SOLUTION 2

```

var NSA = {
  log: function (person) {
    var log = person.log.length ? person.log.join('\n') : 'No Entries';
    person.log = [];
    return log;
  }
};

```



```

var Person = function(name) {
  this.name = name;
  this.call = function(cellphone, callee) {
    this.log.push(this.name + " called " + callee.name + " from " + cellphone.owner.name
+ "'s phone(" + cellphone.number + ")");
  };
  this.text = function(cellphone) {
    for (var i = 1; i < arguments.length; i++) {
      var callee = arguments[i];
      this.log.push(this.name + " texted " + callee.name + " from " +
cellphone.owner.name + "'s phone(" + cellphone.number + ")");
    }
  };
  this.log = [];
}

```

▪ SOLUTION 3

```

const NSA = (function() {
  const m = new Map();
  return {
    log(person) {
      if (m.has(person)) {
        const s = m.get(person).join('\n');
        m.delete(person);
        return s;
      } else {
        return 'No Entries';
      }
    },

    tap(person, action, to, phone) {
      const s = `${person.name} ${action}ed ${to.name} from ${phone.owner.name}'s
phone(${phone.number})`;
      if (m.has(person)) {
        m.get(person).push(s);
      } else {
        m.set(person, [s]);
      }
    }
  };
})();

class Person {
  constructor(name) {
    this.name = name;
  }
  call(phone, to) {
    NSA.tap(this, 'call', to, phone);
  }
  text(phone, ...rs) {
    for (const to of rs) NSA.tap(this, 'text', to, phone);
  }
}

```

#142 - Matrices infinies (5 kyu)

▪ SOLUTION 1

```
function findTrue(mat) {
  for(let d=0;; d++) for(let i=0; i<=d; i++) if(mat[i][d-i]===true) return [i,d-i];
}
```

On parcourt les éléments de la matrice en diagonale ((0,0), (0,1), (1,0), (0,2), (1,1), (2,0) etc.) jusqu'à trouver **true**.

#143 - Triangle Pascal (4 kyu)

▪ SOLUTION 1

```
function pascalsTriangle(n) {
  n--
  s=[1]
  p=[1]
  while(n-->0) {
    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)});
}
```

#144 - Écriture d'intervalles (4 kyu)

▪ SOLUTION 1

```
function solution(list){
  var l=list.map((v,k)=>(v==list[k-1]+1 && v==list[k+1]-1) ? "-" : v)
  return l.reduce((a,v,k)=>(v!='-') ? a.concat(v+((l[k+1]=='-') ? '-' : '')) :
a, []).join(',').replace(/-,/g, '-')
}
```

▪ SOLUTION 2

```
function solution(list){
  for(var i = 0; i < list.length; i++){
    var j = i;
    while(list[j] - list[j+1] == -1) j++;
    if(j != i && j-i>1) list.splice(i, j-i+1, list[i] + '-' + list[j]);
  }
  return list.join();
}
```

▪ SOLUTION 3

```
function solution(list) {
  for (var a = [], i = 0; i < list.length;) {
    for (var j = 1; i + j < list.length && list[i+j] == list[i] + j; j++);
    if (j >= 3) { a.push( `${list[i]}-${list[i+j-1]}` ); i += j; }
    else a.push(list[i++]);
  }
  return a.join(',');
}
```

#145 - Somme d'intervalles (4 kyu)

▪ SOLUTION 1

```
function sumIntervals(intervals){
  var red=intervals.reduce((a,u,k)=>{
    idx=a.findIndex(i=>i[0]<=u[1] && u[0]<=i[1])
    if (idx>-1) a[idx]=[Math.min(a[idx][0],u[0]),Math.max(a[idx][1],u[1])]
    else a.push(u)
    return a
  },[])
  return red.reduce((a,u)=>a+u[1]-u[0],0)
}
```

▪ SOLUTION 2

```
function sumIntervals(intervals){
  var numbers = {};
  intervals.forEach(function(x) {
    for (var i = x[0]; i < x[1]; i++) {
      numbers[i] = i;
    }
  });
  return Object.keys(numbers).length;
}
```

▪ SOLUTION 3

```
function sumIntervals(intervals){
  return intervals
  .sort((x,y)=>x[0]-y[0])
  .reduce((res, next) => {
    var newEnd = Math.max(next[1], res.end);
    res.sum += newEnd - Math.max(next[0], res.end);
    res.end = newEnd;
    return res;
  }, {sum: 0, end: -Infinity})
  .sum;
}
```

#146 - 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;
  }
  return s.length==0
}
```

▪ 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);
}
```

```
}
```

#147 - 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(/[+-]?[^\d+]/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(/^\+/, '');
}

```

#148 - 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('')
}
```

#149 - 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");
}
```

#150 - 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;
}
```

#151 - 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))
```

#152 - 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(''));
}

```

#153 - 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('/');
}
```

#154 - Serpent (4 kyu)

▪ SOLUTION 1

```
snail = function(array) {
  var directions = [[1, 0], [0, 1], [-1, 0], [0, -1]];
  for (var result = [], i = 2 * array[0].length, dir = 0, pos = [-1, 0]; i > 1; i--, dir
= (dir + 1) % 4) {
    for (var l = 0, add = directions[dir]; l < Math.floor(i / 2); l++) {
      pos = [pos[0] + add[0], pos[1] + add[1]];
      result.push(array[pos[1]][pos[0]]);
    }
  }
  return result;
}
```

▪ SOLUTION 2

```
snail = function(array) {
  var result;
  while (array.length) {
    result = (result ? result.concat(array.shift()) : array.shift());
    for (var i = 0; i < array.length; i++)
      result.push(array[i].pop());
    result = result.concat((array.pop() || []).reverse());
    for (var i = array.length - 1; i >= 0; i--)
      result.push(array[i].shift());
  }
  return result;
}
```

▪ SOLUTION 3

```
function snail(array) {
  var results = [];
  while(array.length > 0) {
    results = results.concat(array.shift());
    array.forEach(function (current) {
      results.push(current.pop());
    });
    array.forEach(function (current) {
      current.reverse();
    });
    array.reverse();
  }
  return results;
}
```

#155 - Nombre binaire multiple de 3

▪ SOLUTION 1

```
multipleOf3Regex = /^(1(01*0)*1|0)*$/
```

#156 - 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);
}
```

#157 - 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];
}
```

#158 - 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;
}

```

```
}
```

#159 - 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];
}
```

#160 - 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);
}

```