TP - Exercices d'algorithmique

Aide

Pour entrer un texte en Python, on utilise la fonction `input` :

```
text = input("Entrez votre texte: ")
```

Pour convertir un texte en nombre:

```
texte = "35"
nombre = int(texte)
```

Remarque: Cela lèvera une erreur si le texte ne correspond pas à un nombre

1 - Fibonacci

1. Créer une fonction prenant un paramètre `n`, qui affiche les `n` premiers nombres de la suite de Fibonacci

```
0
1
1   (0 + 1)
2   (1 + 1)
3   (2 + 1)
5   (3 + 2)
8   (5 + 3)
```

2. Tester et valider cette fonction.

Le 50e terme doit être égal à `7778742049`

2 - Calculatrice

1. Créer un script qui demande à l'utilisateur de rentrer une opération mathématique
2. Séparer le texte par opérandes et opérateurs

Pour `32 + 5`, l'opérande de gauche est `32`, celle de droite est `5`, l'opérateur est `+`

Vous n'avez le droit d'utiliser uniquement *une boucle* `for` et des conditions

3. À partir de ces éléments, calculer le résultat de l'opération, et afficher son résultat

Note: On ne supportera pas les nombres à virgule ou négatifs, uniquement entiers positifs

4. Implémenter ce système pour les opérations de bases `+`, `-`, `*` et `/`

3 - Jeu du pendu

1. Créer une fonction `pendu` prenant en paramètre le mot à deviner

Cette fonction créé une boucle qui ne se termine que lorsque le joueur a trouvé toutes les lettres du mot à deviner

Pour chaque lettre que le joueur rentre, il faut remplacer *tous les endroits* où cette lettre apparaît dans le mot

La fonction, une fois le mot deviné, retourne combien d'essai il aura fallu avant que le mot ait été trouvé

2. Tester cette fonction pour plusieurs mots de taille différente, en validant le

fonctionnement

4 - Pattern matching

1. Créer une fonction `match_pattern`, prenant en paramètre un texte, et un pattern.

L'objectif de cette fonction est de tester si le pattern est visible dans le texte, et de retourner tous les *indexes* auxquel on peut retrouver ce pattern

La fonction retournera une liste des indexes auxquel le pattern a été trouvé

Example:

```
(index) 0        9  12
Texte = AABAACAADAABAABA
Pattern = AABA

Résultat: 0, 9 et 12
```

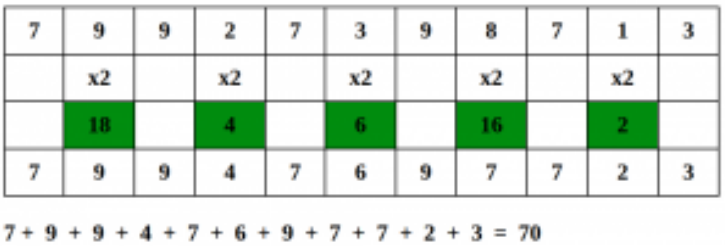
2. Tester cette fonction sur plusieurs textes, avec plusieurs patterns

5 - Algorithme de Luhn

L'algorithme de Luhn est utilisé pour vérifier qu'une faute de frappe ne s'est pas glissée lors de la saisie des numéros d'une carte bancaire.

Pour cela:

- On itère *1 chiffre sur 2* à partir de l'*avant-dernier* chiffre
- Ce nombre est multiplié par 2
- Si le nombre est supérieur ou égal à `10`, on soustrait `9`
- On additionne *tous les chiffres* de la carte bancaire, dont ceux modifiés
- Le resultat doit être divisible par `10` sans reste.



1. Créer une boucle qui itère sur tous les chiffres de la carte bancaire par la fin

Créer une condition pour détecter si le chiffre doit être multiplié par 2, et si on doit effectuer une soustraction par `9`

Quel que soit le résultat de cette condition, ajouter le nombre à la somme finale

2. Créer une fonction `luhn` qui entoure cette boucle, et retourne un `bool`.

Si la somme finale est divisible par 10 sans reste (regardez l'opérateur `%`), retourner `True`, car le numéro est valide

Si la somme finale n'est pas divisible par 10 sans reste, retourner `False`

3. Valider cet algorithme sur ces numéros:

```
4137 8947 1175 5904   Valide
9724 8708 6           Valide
4539 7043 5470 6091   Non valide
7992 7398 713         Valide
```