TD 2 (Révision)

Thibault PENNING

Exercice 0.1 (Somme cumulé)**.** Écrire une fonction `somme_cum` qui prend en entrée un tableau et retourne un autre tableau contenant la somme cumulée de chaque élément du tableau (i.e. le $i^{\text{ème}}$ élément du tableau retourné est la somme des éléments $[0, \dots, i]$)

Exemple :

```
>>> somme_cum([1, 2, 3, 4, 5, 6])
[ 1,  3,  6, 10, 15, 21]
```

> **Note**
>
> Ce genre de fonction est particulièrement utile dans le domaine des simulations physiques, puisqu'il permet simplement de donner une approximation d'une intégration.

Exercice 0.2 (Test de parité)**.** Écrire une fonction `parite` qui prend en entrée un nombre entier binaire, et qui renvoie `True` ssi le dernier bit est de même parité que l'ensemble des autres bits (i.e. le dernier bit est 0 ssi il y a un nombre pair de 1).

Exemple:

```
>>> parite(1011010)
True
>>> parite(1011011)
False
```

> **Note**
>
> Ce genre de test simple est implémenté pour la détermination d'erreur lors de transmission de message.

Bonus

Il est possible de réaliser ce test seulement avec des opérations booléennes. Seriez-vous le trouver ?

Exercice 0.3 (Taux d'intéret)**.** Écrire une fonction `benefice` qui prend en entrée une somme initiale d'un compte, un taux d'intérêt et un nombre d'années, et qui retourne le tableau donnant le bénéfice donner chaque année, ainsi que la valeur total final du compte après la fin de toutes ces années.

Note

Un taux d'intérêt est calculé sur la valeur totale du compte à cette année. Pour un taux de 5%, avec une valeur sur un compte de 1000, le bénéfice seras de 5% de 1000 soit un bénéfice de 50, et la nouvelle valeur du compte seras de 1050.

Exercice 0.4 (Vérification tableau croissant)**.** Écrire une fonction `croissant` qui prend en entrée un tableau, et revoie si un tableau est croissant ou non.

Exemple:

```
>>> croissant([1,5,9,30,100])
True
>>> croissant([1,5,30,9,100])
False
```

Exercice 0.5 (Insertion par dichotomie)**.** Écrire la fonction `insertion_dicho` qui prend en entrée une liste d'entier trié dans l'ordre croissant et un entier, et retourne la même liste avec l'entier placé au bon endroit. La méthode utilisée est la dichotomie, on l'on cherche la place à insérer en réduisant au fur et a mesure l'intervalle ou le nombre peut se situer par comparaison.

Exemple:

```
>>> insertion_dicho([1,3,5,9,20,30], 15)
[1,3,5,9,15,20,30]
```

Exercice 0.6 (Crible d'Ératosthène)**.** Le crible est une méthode simple pour connaitre tous les nombres premiers de 2 à n.

(1) Écrire une fonction `crible` qui prend une entrée une borne n, et qui retourne un tableau de taille $n-2$ où chaque élément d'index i est vrai ssi le nombre $i+2$ est vrai (par exemple le premier élément est à vrai ssi 2 est premier (il l'est)).

Le crible est simple. Pour chaque nombre, on regarde si ce dernier est marqué comme premier. S'il ne l'est pas nous continuons. Si il l'est nous marquons tous ces multiple comme non premier. Seuls les nombres premiers sont nécessaires grâce à la version faible du théorème fondamental de l'arithmétique.

(2) Écrire une fonction `premier` qui prend en entrée un nombre n, et utilise la fonction `crible` afin de retourner l'ensemble des nombres premier entre 2 et n.

> **Astuce**
>
> Si cela est plus simple pour vous vous pouvez faire commencer les liste à 1 ou 0. Considérez les 2 comme non premier.

Exemple:

```
>>> crible(20)
[True, True, False, True, False, True, False, False, False, True, False, True, False, False,
>>> premier(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

> **Note**
>
> Une version plus optimiser du crible existe en pesant que les multiples de $[2, ...i-1]$ ont déjà été vu lorsque l'on commence à i. Seriez-vous la trouver ?

Exercice 0.7 (Jeu des bâtons)**.** Le jeu des bâtons est un jeu inspiré d'un célèbre jeu télévisé. Le jeu commence avec 21 bâtons, et chaque personne prennent à tour de rôle 1, 2 ou 3 bâtons aux choix. Le jeu se déroule jusqu'à ce qu'il n'y ait plus de bâton. La dernière personne qui prend le ou les derniers battons a perdu.

Coder ce jeu. Vous demanderez à l'utilisateur son nombre de bâtons (en faisans attention à ce qu'il rentre) et modifiera l'état du jeu, choisira le nombre de bâtons au hasard pour l'adversaire (qui est don l'ordinateur).

> **Bonus**
>
> Il existe une stratégie simple lorsque le nombre de bâtons de départ est un nombre tel que : $\exists k \in \mathbb{N},\ \text{tq} n = 4 * k + 1$ ce qui est le cas quand $n = 21$ et que le joueur commence en dernier. Trouver cette stratégie et modifier le programme pour l'implémenter.

Exercice 0.8 (Sous mots dans un mot)**.** Écrire une fonction `sous_mot` qui prend en argument 2 chaine de caractère, un mot, et un sous mot, et renvoie `True` ssi le sous mot est présent dans le mot.

Exemple :

```
>>> sous_mot("dysfonctionement", "fonction")
True
>>> sous_mot("pyramidale", "amide")
False
```

Exercice 0.9 (Statistique d'un texte)**.** Écrire une fonction `stats_lettre` qui prend en entrée un texte et qui affiche le nombre d'occurrences de chaque lettre présent.

Si possible n'utilisez pas des tableaux mais une autre structure de donnée vue en classe…

Exemple:

```
>>> stats_lettre("Hello World !")
H = 1   E = 1   L = 3   O = 2   W = 1   R = 1   D = 1   ! = 1
```

> **Note**
>
> Ce genre de petite fonction est très utile pour casser des codes de chiffrement à permutation fixe. En effet, sur des longs textes les fréquences des lettres est fortement similaire.

Exercice 0.10 (Tri à bulle)**.** Le tri bulle est l'un des tris le plus simple qui existe. Il consiste en la procédure suivante : comparer 2 éléments par 2 éléments (en commençant par les 2 premiers de la liste) et changer leur position si l'élément le plus grand est placé avant l'élément le plus petit. Continuer au 2 éléments d'un cran au-dessus (dont l'un est un élément déjà vu) et répéter l'opération. Lorsque l'on répète cette boucle autant de fois qu'il y a d'élément, le tableau est trié.