Devoir surveillé 1

Dans l'écriture de vos programmes en `Python`*, indiquez clairement l'indentation et veillez à respecter la syntaxe du langage. Expliquez succinctement ce que font vos programmes, notamment en commentant soigneusement votre code, et n'hésitez pas à donner des noms explicites aux variables.*

L'utilisation de tout matériel électronique est interdite.

Ce sujet contient sept questions de cours et un problème.

Durée : 2h.

Applications directes du cours

Partie I : Lecture de programmes

1. Les deux fonctions suivantes font-elles la même chose ?

```
def bonnet_blanc(x):
    if x < 0:
        return "bonnet"
    elif x < 1:
        return "blanc"
```

```
def blanc_bonnet(x):
    if x < 1:
        return "blanc"
    elif x < 0:
        return "bonnet"
```

2. Qu'affichent les programmes suivants ?

(a)
```
for k in range(5):
    print(2 ** k)
```

(b)
```
for k in range(3, 4):
    print(2 ** k)
```

(c)
```
for k in range(1, 7, 3):
    print(2 ** k)
```

(d)
```
ch = ""
for o in "Bonjour":
    ch = o + ch
    print(ch)
```

Partie II : Écriture de programmes

3. Écrire une fonction qui prend en argument un entier n et qui renvoie `True` si n est pair, `False` sinon.
4. Écrire une fonction qui prend en argument un entier n et qui renvoie le terme u_n de la suite définie par :
$$\begin{cases} u_0 = -2 \\ \forall k \in \mathbb{N}, u_{k+1} = 2u_k - 3. \end{cases}$$
5. Écrire une fonction qui prend en argument un entier n et qui renvoie la somme $\sum_{k=1}^{n} (k+1)^5$.
6. (a) Écrire une fonction qui prend en arguments une liste L et un objet x, et qui renvoie `True` si l'objet x est un élément de la liste L, `False` sinon.
   (b) Écrire une fonction qui prend en arguments une liste L et un objet x, et qui renvoie le nombre d'occurrences de l'objet x dans la liste L.
   (c) Écrire une fonction qui prend en arguments une liste L et un objet x, et qui renvoie la liste des indices des occurrences de l'objet x dans la liste L.
7. (a) Écrire une fonction qui prend en argument une liste de nombres (entiers ou flottants) non vide et qui renvoie le minimum de cette liste.
   On n'utilisera bien sûr pas la fonction `min`.
   (b) Écrire une fonction qui prend en argument une liste de nombres non vide et qui renvoie la liste des indices de toutes les occurrences du minimum dans la liste.
   (c) En utilisant la fonction précédente, écrire une fonction qui prend en argument une liste de nombres non vide et qui remplace dans la liste toutes les occurrences du minimum par la chaine de caractères `"J'ai cru voir un gros min."`.

Problème — Gattaca (d'après Agro-Véto 2022, filière TB)
Dans ce problème, on s'intéresse à la recherche d'une sous-chaine dans une chaine de caractères, qui est une question importante en informatique et qui possède de nombreuses applications intéressantes, notamment en bio-informatique, pour la recherche de séquences particulières de codons au sein de séquences d'ADN.

On commencera par écrire un algorithme naïf de recherche de sous-chaine, puis on implémentera dans la dernière partie — qui est une partie bonus — un algorithme plus efficace : celui de Knuth-Morris-Pratt.

On dira qu'une chaine de caractères `mot` de longueur p est une **sous-chaine** d'une chaine de caractères `texte` s'il existe un indice i tel que la chaine `texte[i : i + p]` soit égale à `mot`. Si i est un tel indice, alors on dira que la chaine `mot` **est contenue et commence** à l'indice i dans la chaine `texte`.

Exemples :

- ⋆ *la chaine* `"AAT"` *est une sous-chaine de la chaine* `"TTAATGCAATAACT"` *: les indices auxquels elle y est contenue et commence sont les indices* 2 *et* 7 *;*
- ⋆ *la chaine* `"AATC"` *n'est pas une sous-chaine de* `"TTAATGCAATAACT"`*.*

Partie I : Comparaison de chaines de caractères

1. Écrire une fonction qui prend en arguments deux chaines de caractères de même longueur et qui renvoie leur distance de Hamming, c'est-à-dire le nombre de positions auxquelles les deux chaines diffèrent. *La fonction renverra un message d'erreur si les deux chaines de caractères n'ont pas la même longueur.*
2. En appliquant la fonction précédente, écrire une fonction `egales` qui prend en arguments deux chaines de caractères et qui renvoie `True` si les deux chaines sont égales, `False` sinon.

Partie II : Force brute

3. Écrire une fonction `liste_occ` qui prend en arguments deux chaines de caractères `mot` et `texte`, et qui renvoie la liste des indices auxquels la chaine `mot` commence et est contenue dans la chaine `texte`. On écrira ici un algorithme naïf qui recherche la chaine `mot` à tous les indices valides dans la chaine `texte`.

   Exemples :

   - ⋆ *l'instruction* `liste_occ("AAT", "TTAATGCAATAACT")` *renverra* `[2, 7]` *;*
   - ⋆ *l'instruction* `liste_occ("AATC", "TTAATGCAATAACT")` *renverra la liste vide.*
4. En déduire une fonction qui prend en arguments deux chaines de caractères `mot` et `texte`, et qui renvoie le plus petit indice auquel la chaine `mot` est contenue et commence dans la chaine `texte` si la chaine `mot` est bien une sous-chaine de `texte`, un message d'erreur sinon.

Partie III : Bord d'une chaine de caractères

On dit qu'une chaine de caractères `souschaine` est :

- un **préfixe** d'une chaine de caractères `ch` s'il existe un indice j tel que les chaines `ch[ : j]` et `souschaine` soient égales ;
- un **suffixe** d'une chaine de caractères `ch` s'il existe un indice j tel que les chaines `ch[j : ]` et `souschaine` soient égales ;
- le **bord** d'une chaine de caractères `ch` si `souschaine` est la plus longue chaine de caractères distincte de `ch` qui soit à la fois un préfixe et un suffixe de `ch`.

Exemples :

- ⋆ *les chaines* `"ATC"` *et* `""` *sont des préfixes de la chaine* `"ATCGATG"` *;*

- ⋆ *les chaines* `"ATG"` *et* `""` *sont des suffixes de la chaine* `"ATCGATG"` *;*
- ⋆ *le bord de la chaine* `"AGTCGAGT"` *est la chaine* `"AGT"` *;*
- ⋆ *le bord de la chaine* `"ATCGATG"` *est la chaine vide* `""` *;*
- ⋆ *le bord de la chaine* `"A"` *est la chaine vide* `""`*, car le bord ne doit pas être égal à la chaine* `"A"` *tout entière ;*
- ⋆ *la chaine* `"TT"` *est à la fois un préfixe et suffixe de la chaine* `"TTTGACCTTT"`*, mais n'en est pas le bord car elle n'est pas de longueur maximale : le bord de la chaine* `"TTTGACCTTT"` *est* `"TTT"`*.*

5. Écrire une fonction qui prend en argument une chaine de caractères et qui renvoie la longueur du bord de la chaine de caractères.
6. Écrire une fonction `longueurs_bords` qui prend en argument une chaine de caractères `ch` et qui renvoie la liste `B` des longueurs du bord de chacun des préfixes non vides de `ch` : pour tout indice i de la chaine `ch`, l'élément `B[i]` devra être la longueur du bord du préfixe `ch[ : i + 1]` de `ch`.

   Exemple : l'instruction `longueurs_bords("AATGAATC")` *devra renvoyer la liste* `[0, 1, 0, 0, 1, 2, 3, 0]`*. En effet :*
   - ⋆ *le préfixe* `"A"` *a pour bord* `""`*, qui est de longueur* 0 *;*
   - ⋆ *le préfixe* `"AA"` *a pour bord* `"A"`*, qui est de longueur* 1 *;*
   - ⋆ *le préfixe* `"AAT"` *a pour bord* `""`*, qui est de longueur* 0 *;*
   - ⋆ *le préfixe* `"AATG"` *a pour bord* `""`*, qui est de longueur* 0 *;*
   - ⋆ *le préfixe* `"AATGA"` *a pour bord* `"A"`*, qui est de longueur* 1 *;*
   - ⋆ *le préfixe* `"AATGAA"` *a pour bord* `"AA"`*, qui est de longueur* 2 *;*
   - ⋆ *le préfixe* `"AATGAAT"` *a pour bord* `"AAT"`*, qui est de longueur* 3 *;*
   - ⋆ *le préfixe* `"AATGAATC"` *a pour bord* `""`*, qui est de longueur* 0*.*

La dernière partie est donnée en bonus : elle n'est à traiter que si le reste du sujet a été traité entièrement et soigneusement.

Partie IV : Algorithme de Knuth-Morris-Pratt

Dans cette partie, on présente l'algorithme de Knuth-Morris-Pratt, qui est une amélioration de l'algorithme naïf de recherche d'une sous-chaine dans une chaine de caractères écrit à la partie II.

On peut en effet remarquer que lorsque la recherche de la chaine `mot` dans la chaine `texte` échoue à l'indice i, il n'est pas toujours nécessaire de chercher `mot` à l'indice $i+1$: on peut parfois décaler la prochaine recherche de plusieurs caractères.

Le fonctionnement de l'algorithme de Knuth-Morris-Pratt, pour la recherche de la chaine `mot` dans la chaine `texte`, est le suivant.

- Tout d'abord, on détermine la liste `B` des longueurs des bords des préfixes de `mot`, comme à la question 6.
- On initialise alors à 0 une variable `i` qui représente l'indice dans la chaine `texte` auquel on commencera la recherche de `mot`.
- On initialise également à 0 une variable `j` qui représente l'indice d'un caractère dans la chaine `mot` : celui que l'on comparera avec les caractères de `texte`.
- Tant que `i` est un indice valide dans la chaine `texte` :
  - ⋆ on détermine le plus petit entier `j` tel que les caractères `texte[i + j]` et `mot[j]` soient différents (c'est-à-dire le plus grand entier `j` tel que les sous-chaines `texte[i : i + j]` et `mot[ : j]` soient égales) ;
  - ⋆ si `j` est égal à la longueur de `mot`, on a trouvé la chaine `mot` dans `texte` à l'indice `i` ;
  - ⋆ si `j` est égal à 0, on se décale d'un seul caractère (comme dans l'algorithme naïf) et on recommence la recherche de `mot` à l'indice `i + 1`, à partir de l'indice `j = 0` dans `mot` ;
  - ⋆ sinon, on se décale de `j - B[j - 1]` caractères et on cherche `mot` à l'indice `i + j - B[j - 1]`, en partant de l'indice `j = B[j - 1]` dans `mot`.

Illustrons l'exécution de cet algorithme sur les chaines suivantes :

`mot = "ATCGATG"` et `texte = "ATCGATCGATCGATGT"`.

Dans les tableaux qui suivent, les caractères dont la comparaison réussit sont entourés, ceux dont la comparaison échoue sont barrés.

La liste des longueurs des bords des préfixes de `mot` *est* `B = [0, 0, 0, 0, 1, 2, 0]`.

- *On commence la recherche à l'indice* `i = 0` *dans la chaine* `texte`*, à partir de l'indice* `j = 0` *dans* `mot`.

  Les six premières comparaisons réussissent mais la comparaison des caractères `texte[i + j]` *et* `mot[j]` *échoue pour* `j = 6` *: la chaine* `texte` *ne contient pas la chaine* `mot` *à l'indice* 0.

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`texte[i]`	(A)	(T)	(C)	(G)	(A)	(T)	Ø	G	A	T	C	G	A	T	G	T
`mot[j]`	(A)	(T)	(C)	(G)	(A)	(T)	Ø									
`j`	0	1	2	3	4	5	6									

  Le bord du préfixe `mot[ : j] = "ATCGAT"` *est* `"AT"`*, donc ce bord a pour longueur* `B[j - 1] = 2`.

  On reprend alors la recherche de `mot` *à l'indice* `i = i + j - B[j - 1]`*, c'est-à-dire à l'indice* `i = 4` *dans la chaine* `texte`.

  Puisqu'on sait déjà que les deux premiers caractères `"A"` *et* `"T"` *de* `mot` *coïncident avec ceux de* `texte` *aux indices* 4 *et* 5*, on peut reprendre la recherche à partir de l'indice* `j = B[j - 1] = 2` *dans* `mot`.

- *À partir de l'indice* `i = 4`*, les quatre premières comparaisons réussissent, mais la comparaison échoue à nouveau à l'indice* `j = 6` *: les caractères* `texte[i + 6]` *et* `mot[6]` *ne coïncident pas.*

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`texte[i]`	A	T	C	G	**A**	**T**	(C)	(G)	(A)	(T)	∅	G	A	T	G	T
`mot[j]`					**A**	**T**	(C)	(G)	(A)	(T)	∅					
`j`					0	1	2	3	4	5	6					

Comme précédemment, le bord du préfixe `mot[ : j] = "ATCGAT"` *est* `"AT"`*, sa longueur vaut* `B[j - 1] = 2`*.*

On reprend alors la recherche de `mot` *à l'indice* `i = i + j - B[j - 1]`*, c'est-à-dire à l'indice* `i = 8` *dans la chaine* `texte`*.*

Et puisqu'on sait que les deux premiers caractères coïncident, on reprend la recherche à partir de l'indice `j = B[j - 1]`*, c'est-à-dire à partir de l'indice* `j = 2` *dans* `mot`*.*

- *À partir de l'indice* `i = 8`*, toutes les comparaisons réussissent : pour tout indice* `j` *de* `mot`*, les caractères* `texte[i + j]` *et* `mot[j]` *coïncident.*

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`texte[i]`	A	T	C	G	A	T	C	G	**A**	**T**	(C)	(G)	(A)	(T)	(G)	T
`mot[j]`									**A**	**T**	(C)	(G)	(A)	(T)	(G)	
`j`									0	1	2	3	4	5	6	

Ainsi, la chaine de caractères `mot` *est contenue et commence à l'indice* 8 *dans la chaine* `texte`*.*

7. Écrire une fonction qui prend en arguments deux chaines de caractères `mot` et `texte`, et qui renvoie un message d'erreur si la chaine `mot` n'est pas une sous-chaine de `texte`, et qui renvoie le plus petit indice auquel la chaine `mot` est contenue et commence dans la chaine `texte` sinon, trouvé avec l'algorithme de Knuth-Morris-Pratt.