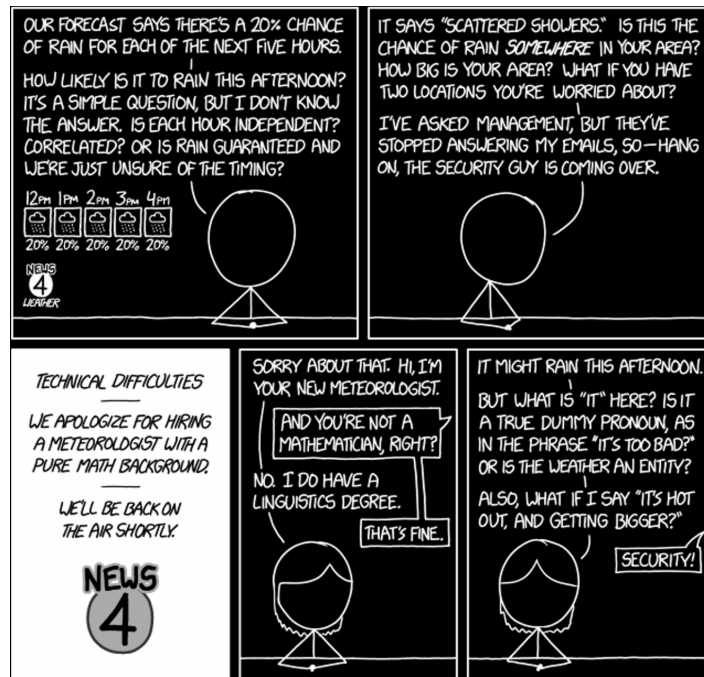




On aborde dans ce chapitre l'étude des structures de liste et de chaîne de caractères qui permettent le stockage des n-uplets et des textes.



2	Listes et chaînes de caractères	1
1	Les listes	2
1.1	Modes de définition d'une liste et lecture	2
1.2	Opérations	4
1.3	Copie	5
2	Les chaînes de caractères	6
3	Itération sur une liste ou une chaîne	8
4	Exercices	9
5	Indications	14

1. Les listes

Les listes de Python sont des tableaux dynamiques. Essentiellement, cela signifie que la taille d'une liste n'a pas à être fixée à l'avance : l'utilisateur peut y ajouter à loisir des éléments sans se soucier de l'implémentation en mémoire. On peut se représenter une liste comme une succession de cases de la mémoire contenant des valeurs et numérotées à partir de l'indice 0 :

23	26	27	29	32	28	25	29	33
0	1	2	3	4	5	6	7	8

1.1. Modes de définition d'une liste et lecture

Le type correspondant est `list`. Le lecteur trouvera ci-dessous la syntaxe de définition d'une liste :

Action	Code
Définition d'une liste	<code>t=[1,-2,6]</code>
Définition d'une liste constante de longueur n	<code>t=[const]*n</code>
Définition de la liste vide	<code>t=[]</code>
Longueur de la liste <code>t</code> (nombre de termes)	<code>len(t)</code>
Valeur de la cellule d'indice <code>i</code> de la liste <code>t</code>	<code>t[i]</code>

Une liste peut être de longueur quelconque (dans la limite des capacités de l'ordinateur). Ses éléments peuvent être de types différents. Tous les objets manipulés par Python peuvent être éléments d'une liste, y compris les listes. L'accès aux cellules se fait en lecture et en enregistrement.

```
>>> t1=[True,4,4.2,0,2,1]
>>> t2,t3=[6,7,3,-1],[]
>>> len(t1),len(t2),len(t3)
6,4,0
>>> T=[1,6,False,print,[True,2]]
>>> type(T),type(T[2]),type(T[4][1])
class 'list', class 'bool', class 'int'
```

```
>>> 6*T[0]-T[1]
0
>>> T[0]+T[4][1]
3
>>> T[2]=-1
>>> T
[1,6,-1,print,[True,2]]
```

On voit sur cet exemple qu'une liste peut contenir des listes. Ceci est particulièrement intéressant pour implémenter des tableaux multidimensionnels.

Par exemple, les tableaux ci-contre (une matrice et une pyramide) peuvent être codés par les listes suivantes :

	1 2 1 0	1
	3 4 5 0	2 3
	5 2 0 1	4 5 6

`tm=[[1,2,1,0],[3,4,5,0],[5,2,0,1]]` et `tp=[[1],[2,3],[4,5,6]]`

Dans le cas d'un tableau bidimensionnel, il est conseillé de coder chaque ligne du tableau par une liste et l'ensemble du tableau par la liste formée de ces listes. La syntaxe est alors proche de celle

des matrices : la ligne i du tableau t est codée par la liste $t[i]$ et le terme d'indice j de celle-ci par $t[i][j]$ (sous réserve que les deux indices soient corrects). Par exemple, la valeur de $tm[1][2]$ est 5, $tp[1]$ vaut $[2, 3]$ et $tp[2][2]$ vaut 6.

Le langage Python admet une syntaxe très proche des Mathématiques pour définir une liste à partir de l'expression de ses éléments.

Action	Code
Définition d'une liste en compréhension	<code>t=[expr for k in range(a,b,r) if cond]</code> où <code>expr</code> est une expression dépendant de <code>k</code> et <code>cond</code> est une expression booléenne dépendant de <code>k</code> .

Voici quelques exemples d'utilisation de cette puissante syntaxe :

```
>>> t1=[k**2 for k in range(7)]
>>> t1
[0,1,4,9,16,25,36]
>>> t2=[i for i in range(10) if i%3!=1]
>>> t2
[0,2,3,5,6,8,9]
```

On notera la proximité de la définition de $t1$ avec l'écriture ensembliste $\{k^2; k \in \llbracket 0,6 \rrbracket\}$.

La seconde liste rassemble les entiers i compris entre 0 et 9 tels que $i \neq 1$ [3].

Expression renvoyée	Code
Tranche de i à $j-1$	<code>t[i:j]</code>
Tranche de i à la fin	<code>t[i:]</code>
Tranche de 0 à $i-1$	<code>t[:i]</code>

Python permet le « slicing » (découpage en sous-listes) :

```
>>> t=[0,5,5,1]
>>> t[1:3]
[5,5]
```

```
>>> t[1:3]=[0,0]
>>> t
[0,0,0,1]
```

Nous terminerons ce paragraphe par une mise en garde. Le code qui suit se solde par un message d'erreur à l'exécution :

```
>>> t=[3,0,1,2]
>>> t[4]
```

```
Traceback (most recent call last): File "<console>",
line 1, in <module> IndexError: list index out of range
```

Le problème est facilement identifiable : le terme d'indice 4 n'existe pas.



L'erreur « list index out of range »

On vérifiera scrupuleusement la validité des indices de listes au sein d'un script. Notamment en cas de boucle, il faudra s'assurer qu'aucun dépassement n'a lieu.

Le langage Python offre une seconde numérotation des listes par des indices négatifs : le terme d'indice -1 désigne le dernier terme, -2 l'avant-dernier, etc.

```
>>> t=[3,0,1,2]
>>> t[-1],t[-4],t[-2],t[-3]
2,3,1,0
```

1.2. Opérations

Le type *list* de Python supporte plusieurs opérations élémentaires.

Action	Code
Ajout d'un élément x en fin de liste	<code>t.append(x)</code>
Renvoi et suppression de la fin de liste	<code>t.pop()</code>
Concaténation de deux listes t1 et t2	<code>t3=t1+t2</code>

Illustration des opérations d'ajout en fin de liste, de suppression de la tête et de concaténation :

```
>>> t1=[6,1]
>>> t1.append(2)
>>> t1
[6,1,2]
>>> t1=t1+[4]
>>> t1
[6,1,2,4]
```

```
>>> t2=[-6,9]
>>> t1=t2+t1
[-6,9,6,1,2,4]
>>> t1.pop()
4
>>> t1
[-6,9,6,1,2]
```

Par exemple, pour définir la liste

$[t[0], t[0]+t[1], \dots, t[0]+\dots+t[n-1]]$ où t est une liste d'entiers

on peut opter pour deux stratégies : créer la future liste en l'initialisant avec des zéros ou la construire dynamiquement par ajouts successifs.

```
def sommes_partielles_v1(t):
    n=len(t)
    s=[0]*n
    s[0]=t[0]
    for i in range(1,n):
        s[i]=s[i-1]+t[i]
    return s
```

```
def sommes_partielles_v2(t):
    n=len(t)
    s=[]
    s.append(t[0])
    for i in range(1,n):
        s.append(s[i-1]+t[i])
    return s
```

Il ne faut pas confondre les deux syntaxes. Par exemple, en initialisant `s=[]`, l'instruction `s[0]=t[0]` se solde par un message d'erreur puisque la cellule d'indice 0 de `s` n'existe pas.



Les deux grands modes de création d'une liste

Outre les modes de définition déjà étudiés, on retiendra qu'on peut construire une liste :

- ⇒ *Dynamiquement* : en partant de la liste vide `[]` et en utilisant la méthode `append`.
- ⇒ *Statiquement* : en initialisant la liste à une constante `a` (par exemple) `t=[a]*n` et en modifiant ses cellules via la syntaxe `t[i]=...`

La méthode dynamique s'impose bien-sûr dans le cas où la taille de la liste n'est pas connue à l'avance.

Remarquons qu'il est possible d'ajouter un terme à une liste par concaténation :

```
>>> t=[0,1,2,4,5,6]
>>> t=t+[7]
[0,1,2,3,4,5,6,7]
```

Cette méthode est cependant déconseillée pour des listes de grande taille.



Il faut privilégier la méthode `append`

Afin d'optimiser l'exécution du code, mieux vaut éviter l'utilisation de l'opérateur `+` pour l'ajout d'un terme (`t=t+[a]`) car son utilisation a pour effet une recopie de l'intégralité de la liste en mémoire contrairement à `append`¹.

On peut tester l'égalité de deux listes au moyen de l'opérateur `==`, cf. ci-dessous :

```
>>> t1,t2=[1,2,3],[4,5]
>>> t1+t2==[1,2,3,4,5]
True
```



Test d'égalité pour les listes

Au concours, on ne pourra utiliser cet opérateur que si l'énoncé l'indique clairement en préambule ou en annexe.

1.3. Copie

La copie d'une liste nécessite quelques précautions :

```
>>> t=[2,5,7]
>>> s=t
>>> t[0]=-12
```

```
>>> s,t
([-12,5,7],[-12,5,7])
```

1. En fait, cela n'est vrai qu'en moyenne.

Dans certains algorithmes, il est important de sauvegarder la valeur d'une liste *avant* de la modifier.



Attention aux copies de liste

Si `t` est une liste, alors l'affectation `s=t` aura pour effet de créer un objet identique à `t` : toute modification de l'une des deux listes `t` ou `s` impactera l'autre variable. Il existe bien entendu des moyens de créer une copie *indépendante* d'une liste `t`. Ceci est une conséquence du caractère mutable des variables de type `list`, contrairement aux variables de type numérique (`int`, `float` et `bool`).

La réalisation d'une sauvegarde d'une liste (on dit aussi une copie) avant sa modification est possible au moyen de très nombreuses syntaxes. On retiendra les deux suivantes :

Action	Code
Copie d'une liste <code>t</code>	<code>tbis=t.copy()</code> ou <code>tbis=t[:]</code>

Attention toutefois aux listes contenant des listes pour lesquelles cette méthode ne fonctionne pas².

```
>>> t=[2,5,7]
>>> s=t.copy()
>>> t[0]=-12
>>> s,t
([2,5,7],[-12,5,7])
```

```
>>> t=[[2,5],[7,-1]]
>>> s=t.copy()
>>> t[0][0]=-12
>>> s,t
([[[-12,5],[7,-1]], [[-12,5],[7,-1]]])
```

Contrairement à ses arguments de type numérique, une fonction peut modifier toute liste qui lui est donnée en argument.



Listes et fonctions

Une fonction peut modifier ses paramètres de type `list`.

```
def perm(t):
    if len(t)>1:
        t[0],t[1]=t[1],t[0]
```

On remarquera que cette fonction *ne renvoie rien* mais *modifie* son paramètre `t` en permu-

tant ses deux premiers termes (on peut la qualifier de procédure).

```
>>> t=[1,2,3,4,5]
>>> perm(t)
>>> t
[2,1,3,4,5]
```

2. Les chaînes de caractères

Les chaînes de caractères permettent de manipuler du texte. Dans certains langages, il existe un type `char` (pour caractères en anglais) à partir duquel est construit le type composé des chaînes de caractères.

2. La fonction `deepcopy` du module `copy` résout ce problème mais dépasse le cadre du programme officiel.

tères. Sous Python, les deux types ont fusionné. On peut se *représenter* une chaîne comme un tableau de caractères³ :

C	E	C	I		E	S	T		U	N	E		C	H	A	Î	N	E
---	---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	---	---	---

Le type des chaînes de caractères est `str`, de l'anglais *string*. Les chaînes peuvent être délimitées par des simples quotes (apostrophes) ou doubles quotes (guillemets) :

Action	Code
Définition d'une chaîne de caractères <code>ch</code>	<code>ch='texte' ou ch="texte"</code>
Définition de la chaîne vide	<code>ch='' ou ch=""</code>
Longueur d'une chaîne de caractères <code>ch</code>	<code>len(ch)</code>

On peut utiliser l'un des caractères `'` et `"` dans une chaîne mais il faut alors la délimiter au moyen de l'autre caractère :

```
>>> d="L'art "
>>> len(d)
4
```

L'espace étant un caractère, on prendra garde à n'en pas mettre lorsqu'on définit la chaîne vide.

Expression renvoyée	Code
Accès en lecture au caractère d'indice <code>i</code>	<code>ch[i]</code>
Tranche de <code>i</code> à <code>j - 1</code>	<code>ch[i:j]</code>
Tranche de <code>i</code> à la fin	<code>ch[i:]</code>
Tranche de 0 à <code>i - 1</code>	<code>ch[:i]</code>

L'accès aux caractères d'une chaîne s'effectue comme pour les tableaux et les listes.

Le premier caractère d'une chaîne est en position 0, le dernier en position $n - 1$ où n est la longueur de la chaîne.

Le slicing s'applique encore ici.

```
>>> mot='Find me '
>>> print(mot[4])

>>> print(mot[5])
m
```

```
>>> mot[0:2]
'Fi'
>>> mot[2:5]
'nd '
```



Le type `str` n'est pas mutable

Les chaînes de caractères ne sont pas modifiables.

3. Attention, ne pas confondre la chaîne "AB" avec un tableau de caractères ["A", "B"], il ne s'agit que d'une *représentation*.

```
>>> s="PCSI 2 LLG"
>>> s[0]
'P'
>>> s[0]='M'
```

```
Traceback (most recent call last):
File "<console>", line 1, in <module>
TypeError: 'str' object does not support
item assignment
```

Voici le minimum à connaître :

Résultat	Code
Concaténation des chaînes ch1 et ch2	ch1+ch2
Répète n fois la chaîne ch	ch*n

```
>>> mot='Help '
>>> mot=mot+'me '
>>> mot
'Help me '
>>> mot*3
'Help meHelp meHelp me '
```

L'opérateur == permet de tester l'égalité de deux chaînes mais, comme dans le cas des listes, il faudra au concours que l'énoncé autorise explicitement son utilisation.

```
>>> "HX6"=="HX7"
False
```

3. Itération sur une liste ou une chaîne

Les listes de Python sont itérables : dans une boucle for, on peut remplacer l'objet range(i,j,p) par une liste. Cela est intéressant lorsque les indices ne servent qu'à décrire les éléments de la liste : on y gagne en clarté. Les fonctions ci-dessous renvoient le minimum d'une liste numérique non vide :

```
def minimum1(t):
    min=t[0]
    for element in t:
        if element<min:
            min=element
    return min
```

```
def minimum2(t):
    n,min=len(t),t[0]
    for indice in range(n):
        if t[indice]<min:
            min=t[indice]
    return min
```

Tout comme les listes, les chaînes sont des objets itérables du langage Python.

```
nom='FIN'
for a in nom:
    print(a)
```

F
I
N

4. Exercices

1



Génération de listes

Écrire un code créant les listes suivantes :

a. $t1 = [1, 1, 1, 1, \dots, 1, 1, 1]$ (14 fois)

c. $t3 = [0, 0, 0, 1, 1, 1, 2, 2, 2, \dots, 11, 11, 11]$

b. $t2 = [30, 29, 28, 27, \dots, 13, 12, 11, 10]$

d. $t4 = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, \dots, 11, 11]$

2



Algorithme de recherche linéaire dans une liste

Écrire une fonction `search(t, x)` d'argument une liste `t` et un objet `x` et renvoyant `True` si `x` figure dans `t`, `False` sinon.

3



Maximum et moyenne d'une liste numérique

Écrire des fonctions `maximum` et `moyenne` prenant en argument une liste numérique `t` non vide et renvoyant respectivement :

1. le plus grand élément de `t`;

2. la moyenne des termes de `t`.

4

Testeur de monotonie *f*

Écrire une fonction `testMonotonie(t)` prenant en argument une liste `t` d'entiers naturels et renvoyant `True` si elle est monotone (au sens large) et `False` sinon.

5

Recherche linéaire d'un facteur dans une chaîne *f*

On appelle facteur d'une chaîne de caractères `c`, toute chaîne `f` formée de caractères consécutifs de `c`. Par exemple, "BC" est un facteur de "ABCD", au contraire de "BD". Écrire une fonction `rechercheFacteur` prenant en argument deux chaînes `c` et `f` renvoyant `True` si `f` est un facteur de `s` et `False` sinon. On pourra tester l'égalité de deux chaînes au moyen de l'opérateur `==`.

6

Shuffle *f*

On considère un jeu de 52 cartes numérotées de 0 à 51 : 0 1 2 3 ... 51

On le coupe par le milieu en deux paquets :

0 1 2 ... 25 et 26 27 28 ... 51

puis on *mélange* les deux paquets de la manière suivante :

0 26 1 27 2 28 ... 25 51

Écrire un programme calculant le nombre minimal (non nul) de mélanges successifs auxquels il faut procéder pour retrouver l'ordre initial.

7

Repérer le minimum d'une liste *ff*

Écrire une fonction `indMin` d'argument une liste `t` d'entiers renvoyant la liste du ou des indices où le minimum de la liste est atteint. Par exemple, l'appel `indMin([2,0,3,0])` doit renvoyer `[1,3]`.

8

Champernowne's constant *ff*

La constante de *Champernowne* est le nombre obtenu en concaténant derrière la virgule les entiers naturels consécutifs :

$$c := 0.1234567891011121314151617181920212223\dots$$

Écrire une fonction `d` prenant en argument un entier naturel non nul n et renvoyant la n -ème décimale de c . Par exemple, les appels `d(1)` et `d(15)` renvoient respectivement 1 et 2. On pourra utiliser les fonctions `int` et `str` :

```
>>> int('1234')+2
1236
```

```
>>> str(1234)
"1234"
```

9

Longueurs des séries d'une liste binaire *ff*

Pour une liste `t` non vide à valeurs binaires (i.e. dans $\{0,1\}$), on s'intéresse à la liste des longueurs des séries successives de 1 qu'elle contient. Par exemple, pour les listes suivantes

`[1,0,1,1,1,0,1,1]` , `[0,1,0,0,1,0,1,0,1,1,0,0,0]` et `[1,0,1,1,1,1,1]`

on obtient respectivement `[1,3,2]`, `[1,1,1,2]` et `[1,5]`. Pour une liste ne comportant que des zéros, on conviendra que le résultat est la liste vide `[]`.

Écrire une fonction `lg_series` prenant en argument une liste `t` à valeurs dans $\{0,1\}$ et renvoyant la liste des longueurs des séries successives de 1.

10

Recherche dichotomique dans une liste triée *ff*

Pour déterminer si un entier x est présent dans une liste d'entiers `t` triée dans l'ordre croissant, on peut envisager un algorithme plus efficace qu'un parcours *linéaire* de la liste :

- ⇒ On calcule l'indice m du « milieu » de la liste (avec des guillemets car la liste peut contenir un nombre pair de termes) ;
- ⇒ Si `t[m] < x`, alors x n'est pas dans la portion de `t` située entre l'indice initial et m (car `t` est triée dans l'ordre croissant) : on recherche x entre l'indice $m+1$ et l'indice de fin ;
- ⇒ Si `t[m] > x`, alors x n'est pas dans la portion de `t` située entre l'indice m et l'indice final : on recherche x entre l'indice initial et $m-1$.

En itérant cette séquence, on recherche donc x dans des segments successifs de la liste dont la longueur est à peu près divisée par deux à chaque fois.

Écrire une fonction `biSearch(t, x)` prenant en argument une liste d'entiers naturels t triée dans l'ordre croissant et un entier x , renvoyant `True` si l'entier figure dans la liste, et `False` sinon.

11 Γ -règles *ff*

On appelle Γ -règle toute règle graduée commençant à 0 formée d'une suite strictement croissante d'entiers naturels et telle que tous les écarts entre deux éléments distincts de la graduation soient différents.

On représente une Γ -règle par une liste d'entiers t de longueur n , telle que :

$$\Rightarrow t[0] = 0$$

$$\Rightarrow \text{Pour } i \in [0, n-2], t[i] < t[i+1]$$

$$\Rightarrow \text{Les entiers } t[j] - t[i] \text{ (où } 0 \leq i < j \leq n-1) \text{ sont deux à deux distincts.}$$

1. Les listes suivantes sont-elles des Γ -règles ?

$$t_1 = [0, 1, 5, 8] \text{ et } t_2 = [0, 3, 8, 10, 11]$$

2. À toute Γ -règle est associée une règle symétrique, qui est aussi une Γ -règle. Par exemple, la symétrique de t_1 est $t_3 = [0, 3, 7, 8]$:



Écrire une fonction `symetrie` prenant en argument une Γ -règle t et renvoyant sa symétrique.

3. Écrire une fonction `croiss` prenant en argument une liste t d'entiers naturels et qui renvoie `True` si $t[0] = 0$ et t est strictement croissante, et `False` sinon.

4. Écrire une fonction `test` prenant en argument une liste t d'entiers naturels et qui renvoie `True` si t est une Γ -règle et `False` sinon.

On pourra utiliser l'instruction `x in t` qui renvoie `True` si l'objet x appartient à la liste t et `False` sinon.

12Tableaux de Young *ff*

On appelle triangle tout tableau d'entiers naturels vérifiant la condition suivante : chaque ligne a une longueur inférieure ou égale à la précédente. Un tableau de Young est un triangle dont toutes les lignes et les colonnes sont triées dans l'ordre croissant. En voici un exemple de tableau de Young ci-contre.

1	2	5	7	10
2	6	8	10	
2	7	9		
3				

1	2	2	3
2	6	7	
5	8	9	
7	10		
10			

On définit le transposé d'un triangle en transformant en colonnes chacune des lignes du tableau (il est clair que ce transposé est aussi un triangle et que le transposé d'un tableau de Young est un tableau de Young).

Ainsi, le transposé du tableau de Young ci-dessus est le tableau de Young ci-contre.

Un triangle sera codé par une liste de listes. Par exemple, le premier triangle ci-dessus est codé par :

$[[1, 2, 5, 7, 10], [2, 6, 8, 10], [2, 7, 9], [3]]$

1.
 - a. Écrire une fonction `isCrescent` prenant en argument une liste d'entiers `t` et renvoyant `True` si cette liste est croissante au sens large et `False` sinon.
 - b. Écrire une fonction `trans` prenant en argument un triangle `t` et renvoyant son transposé.
 - c. Écrire une fonction `isYoung` prenant en argument une liste de listes d'entiers `t` et renvoyant `True` si cette liste représente un tableau de Young et `False` sinon.
2.
 - a. Écrire une fonction `search` prenant en arguments un tableau de Young `t` et un entier naturel `x` et renvoyant `True` si l'entier est dans le tableau et `False` sinon.
 - b. On peut améliorer l'algorithme de recherche de `x` dans `t`. On se place sur la dernière cellule `y` de la première ligne de `t` et on itère l'action suivante : si `y` vaut `x`, alors la recherche est finie; si `y < x`, alors `x` ne peut se trouver sur cette ligne, on peut descendre d'une ligne; si `x < y`, alors `x` ne peut se trouver sur cette colonne, on se déplace à gauche sur la même ligne. En déduire une fonction `searchBis` analogue à `search`.

13



Gestion de vols Paris–Rome *fff*

Soit n et m des entiers naturels non nuls. On considère n personnes, numérotées de 0 à $n - 1$, voulant prendre l'avion à Paris pour se rendre à Rome. Les vols sont numérotés de 0 à $m - 1$ et partent dans l'ordre de leur indice. Les vœux des voyageurs ainsi que les capacités des différents vols sont enregistrés dans deux listes `voeux` et `charges` telles que :

⇒ Pour tout i dans $\llbracket 0, n - 1 \rrbracket$, `voeux[i]` est le vol que souhaite prendre la personne i .

⇒ Pour tout $j \in \llbracket 0, m - 1 \rrbracket$, `charge[j]` est la charge du vol j , c'est-à-dire le nombre maximal de voyageurs que l'avion peut contenir.

1. Écrire une fonction `planning` qui prend en entrée les listes `voeux` et `charges` renvoyant une liste `vols` de longueur m telle que `vols[j]` est la liste dans l'ordre croissant des voyageurs qui désirent prendre le vol numéro j . Par exemple si `voeux = [3, 1, 2, 3, 2, 1, 2]` et $m = 4$ alors

`vols = [[], [1, 5], [2, 4, 6], [0, 3]]`

2. Écrire une fonction `test1`, qui prend en entrée les listes `voeux` et `charges` renvoyant le booléen `True` s'il est possible de satisfaire tous les voyageurs et `False` sinon.
3. Dans cette question, on suppose que tout voyageur accepte de prendre un vol ultérieur au sien, quel qu'il soit.
 - a. Écrire une fonction `test2`, qui prend en entrée les listes `voeux` et `charges` renvoyant le booléen `True` s'il est possible de satisfaire tous les voyageurs avec ce nouveau protocole, et `False` sinon.
 - b. On suppose qu'il est possible de faire voyager tout le monde dans ces nouvelles conditions. Écrire alors une fonction `repartition2`, qui prend en entrée les listes `voeux` et `charges` renvoyant une liste `rep` telle que `rep[j]` est la liste des voyageurs qui voyageront sur le vol numéro j .
4. Dans cette dernière question, on suppose que, pour tout voyageur i , si le vol `voeux[i]` est trop plein pour que le voyageur i puisse le prendre, celui-ci est prêt à prendre le vol `voeux[i] + 1` s'il existe. Considérons par exemple `voeux = [1, 2, 0, 0, 1, 0]` et `charges = [2, 2, 2]`. On peut calculer

la liste des nombres de demandes par vol : $[3, 2, 1]$ (trois personnes demandent le vol 0, deux le vol 1 et une le vol 2). On peut satisfaire les voyageurs en déplaçant un des voyageurs demandant le vol 1 au vol 2 et un des voyageurs demandant le vol 0 au vol 1. En revanche, dans le cas où $\text{voeux} = [1, 2, 0, 0, 1, 0, 0]$ et $\text{charges} = [1, 2, 4]$, on ne peut satisfaire la demande car la liste des nombres de demande par vol est $[4, 2, 1]$ (il faudrait déplacer 3 personnes du vol 0 vers le vol 1, ce qui est impossible).

- a. Écrire une fonction `test3`, qui prend en entrée les listes `voeux` et `charges` renvoyant le booléen `True` s'il est possible de satisfaire tous les voyageurs avec ce nouveau protocole, et `False` sinon.
- b. On suppose qu'il est possible de faire voyager tout le monde dans ces nouvelles conditions. Écrire alors une fonction `repartition3`, qui prend en entrée les listes `voeux` et `charges` renvoyant une liste `rep` telle que `rep[j]` est la liste des voyageurs qui voyageront sur le vol numéro `j`.

14

Plus longue sous-liste croissante *fff*

On appelle sous-liste d'une liste t toute liste de la forme

$[t[n_0], t[n_1], \dots, t[n_k]]$ où $0 \leq n_0 < n_1 < \dots < n_k < n$ où n est la longueur de t

Elle est dite croissante si de plus $t[n_0] \leq t[n_1] \leq \dots \leq t[n_k]$. Par convention, une liste de longueur 1 est croissante. On s'intéresse dans le sujet à la longueur maximale d'une sous-liste croissante de t . Pour $[11, 6, 2, 24, 25, 12, 21, 41, 34, 30]$, cette longueur maximale vaut 4 : par exemple, $[2, 24, 25, 41]$ réalise ce maximum. On propose un algorithme efficace pour calculer la longueur d'une plus grande sous-liste croissante.

Notons t la liste donnée en argument. On calcule une nouvelle liste M de la manière suivante : s'il n'existe pas d'élément inférieur ou égal à $t[i]$ avant $t[i]$, alors $M[i] = 1$, sinon $M[i]$ vaut 1 plus le maximum des $M[j]$, pour $j < i$ tel que $t[j] \leq t[i]$.

1. Comment trouver la longueur d'une plus grande sous-liste croissante de t en utilisant M ?
2. Écrire une fonction `longMax` d'argument une liste t d'entiers et renvoyant la longueur d'une plus grande sous-liste croissante de t .
3. En vous inspirant de la fonction précédente, écrire une fonction `seqMax` d'argument une liste d'entiers t renvoyant une sous-liste de t , croissante et de longueur maximale.

Commentaire

Cette approche relève de la programmation dynamique, que vous étudierez en seconde année. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en stockant les résultats intermédiaires.

5. Indications

1 ↪ _____

On peut utiliser une définition en compréhension ou tout simplement créer la liste au moyen d'une boucle `for`.

2 ↪ _____

Attention, il s'agit de reprogrammer l'expression booléenne `x in t` au moyen d'une boucle.

3 ↪ _____

Balayer la liste au moyen d'une boucle `for` en utilisant une variable auxiliaire.

4 ↪ _____

Utiliser, par exemple, deux variables booléennes : une pour la croissance et une autre pour la décroissance.

5 ↪ _____

Tester l'égalité de `f` avec une tranche de la forme `c[i:i+q]` où `q` est la longueur de `f`.

6 ↪ _____

Utiliser des listes. On trouve 8.

7 ↪ _____

Adapter l'algorithme de recherche du minimum vu en cours.

8 ↪ _____

Générer une chaîne contenant le début de l'écriture décimale de `x` tant qu'elle ne contient pas la décimale recherchée.

9 ↪ _____

Il faut bien réfléchir à la stratégie avant de coder quoi que ce soit : afin de calculer la longueur d'une série, on peut par exemple utiliser un compteur qui est réinitialisé à un dès le début d'une nouvelle série.

10 ↪ _____

On pourra utiliser deux variables `d` (indice de début) et `f` (indice de fin) qui vérifieront la propriété suivante tout au long de l'algorithme : à la fin de chaque itération, si `x` est dans `t`, alors il est nécessairement dans la portion de la liste située entre les indices `d` et `f` (au sens large).

11 ↪ _____

Il faut tester tous les écarts, pas seulement les écarts successifs.

12

Pour construire le transposé de t , on pourra initialiser un triangle de taille adéquate avec des zéros puis itérer sur t pour remplacer les zéros.

13

Dans le cas du troisième protocole, il faut bien comprendre que pour chaque vol, on doit prioritairement « déplacer » les passagers du vol précédent si c'est possible puis compléter par les voyageurs initialement prévu pour le vol.

14

La case $M[i]$ contient la longueur du plus grand sous-tableau croissant de t terminant en position i .