Travaux pratiques

informatique commune

Introduction aux listes

Les listes Python

Le type *list* est une structure de données énumérable qui permet de représenter une séquence finie d'éléments. Ces derniers sont séparés par une virgule est enclos dans des crochets, comme par exemple :

```
In [1]: l = [10, 23, 45, -2, 98]
```

A l'instar des chaînes de caractères, chaque élément peut être obtenu par le biais de son indice (avec les mêmes conventions : l'indice du premier élément est 0), et le slicing est possible : si `l` est une liste, `l[i: j: p]` désigne la liste des éléments dont les indices sont compris entre i (au sens large) et j (au sens strict) espacés d'un pas p.

Mutation d'une liste

Cependant, à la différence d'une chaîne de caractères, une liste est un objet *mutable* : il est possible de modifier physiquement une portion de la liste sans recréer celle-ci dans son entier. Les opérations de mutation consistent en :

- la modification d'une case définie par son indice : `l[i] = v` remplace le contenu de `l[i]` par la nouvelle valeur `v` ;
- la modification d'une portion de liste définie par un *slicing* : `l[i: j: p] = lv` remplace la sous-liste décrite par le slicing par la liste `lv` (qui doit être de même taille) ;
- l'ajout d'un élément en fin de liste : `l.append(v)` ajoute à la liste `l` un élément supplémentaire `v` en fin de liste ;
- la suppression d'une case définie par son indice : `del l[i]` supprime la case d'indice `i` ;
- la suppression d'une valeur présente dans une liste : `l.remove(x)` supprime la première occurrence de `x` dans `l` ;
- la suppression de la dernière case d'une liste : `l.pop()` supprimé *et retourne la valeur* de la dernière case de la liste ;
- la duplication d'une liste : `nl = l.copy()` crée une nouvelle liste `nl`, copie de la liste `l`.

Notez que cette liste n'est pas exhaustive ; d'autres opérations de mutation existent.

Création d'une liste

Il existe de nombreuses façons de créer une liste. Lorsque le nombre d'éléments est réduit, on peut tout simplement les énumérer comme dans le premier exemple ci-dessus.
Une deuxième manière de procéder consiste à définir une liste par *compréhension* : `l = [x for x in enum if expr]` retourne la liste des éléments de l'objet énumérable `enum` qui vérifient l'expression booléenne `expr`. Par exemple, pour définir la liste des entiers inférieurs à 100 et non divisibles par 3 on écrira :

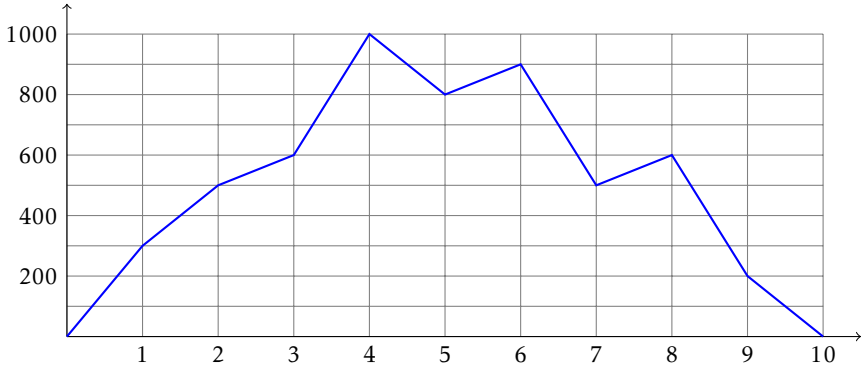
```
In [2]: [x for x in range(100) if x % 3 != 0]
Out[2]: [1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, ...
```

Les opérations de concaténation et de duplication permettent de créer une liste d'une taille donnée : `lst = [None] * 100` crée une liste de 100 cases (indexées entre 0 et 99) remplie par la valeur `None`.
Enfin, lorsqu'on ne connait pas au moment de sa création le nombre de cases que devra posséder une liste, on débute par une liste vide : `l = []` que l'on remplit progressivement avec la méthode `append`. L'exemple précédent peut aussi être obtenu à l'aide du script :

```
l = []
for x in range(100):
    if x % 3 != 0:
        l.append(x)
```

Exercice 1. Parcours d'un tableau

Un *altimètre* est un instrument de mesure permettant de déterminer la distance verticale entre un point et une hauteur de référence. Lors d'une randonnée en montagne, Alice étalonne son altimètre à son point de départ, puis mesure à chaque heure l'altitude relative atteinte.
À la fin de sa randonnée, elle obtient une liste d'entiers naturels qu'elle range dans une liste Python nommée `alt` (illustration figure 1).



`alt = [300, 500, 600, 1000, 800, 900, 500, 600, 200, 0].`

Figure 1 – La randonnée d'Alice a duré 10 heures, elle a atteint une altitude relative de 300m au bout d'une heure, de 500m au bout de deux heures, etc.

Alice souhaite maintenant calculer certaines valeurs relatives à son parcours.

a) Quelle fonction déjà existante en Python lui permet de calculer la durée de sa randonnée ?

b) Définir en Python une fonction baptisée `altmax`, qui prend en argument une liste `alt` et qui retourne l'altitude relative maximale atteinte lors de sa randonnée.
Par exemple, dans le cas de l'illustration numérique donnée ci-dessus la fonction devra renvoyer la valeur 1000.

c) Alice cherche maintenant à savoir à quel moment son ascension a été la plus rapide en calculant le dénivelé maximal réalisé en une heure. Elle cherche donc la plus grande différence entre deux emplacements consécutifs de la liste. Par exemple, pour la liste donnée en illustration le dénivelé maximal est égal à 400 et a été réalisé entre la troisième et la quatrième heure.
Définir en Python une fonction baptisée `denivmax` prenant en argument une liste `alt` et retournant le dénivelé maximal réalisé en une heure durant sa randonnée.

d) Modifier la fonction précédente pour qu'elle retourne non pas le dénivelé maximal mais l'heure à laquelle débute la réalisation de ce dénivelé. Pour la liste donnée en exemple, cette fonction devra donc retourner la valeur 3.

e) Définir une fonction baptisée `denivtotal` retournant la somme des dénivelés *positifs* réalisés durant cette randonnée. Pour l'exemple donné en illustration cette fonction devra donc retourner la valeur 1200.

f) Enfin, on appelle *sommet* toute altitude relative strictement supérieure à l'altitude qu'elle précède et à l'altitude qui lui succède dans la liste `alt`. Dans l'exemple qui nous sert à illustrer cet exercice, la randonnée d'Alice présente trois sommets de valeurs 1000, 900 et 600 atteints à la quatrième, sixième et huitième heure de marche.
Rédiger fonction `sommets` retournant la liste des sommets de la randonnée.
Indication : créer une liste vide et y ajouter par la méthode `append` les sommets au fur et à mesure de leur découverte.

Exercice 2. Plus grand plateau

On considère un tableau de bits dont les éléments sont égaux aux entiers 0 ou 1. Rédiger en Python une fonction calculant le nombre maximal de 0 consécutifs présents dans ce tableau. Par exemple, pour le tableau suivant la fonction devra retourner la valeur 7 :

1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0

7

Exercice 3. Nombre moyen d'éléments absents

a) Dans le module `numpy.random` se trouve une fonction nommée `randint`. Utilisez-la pour créer une tableau de 100 entiers tirés au hasard entre 0 et 99.

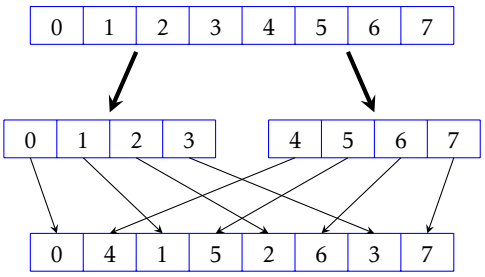
b) Calculer le nombre d'éléments de $[\![0, 99]\!]$ qui n'appartiennent pas à ce tableau

c) Recommencer cette expérience un grand nombre de fois pour évaluer le nombre moyen d'éléments d'absents (la valeur théorique est d'environ 36,6).

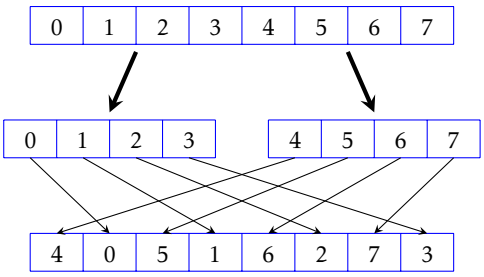
Exercice 4. In and Out Shuffle

Une méthode traditionnelle pour mélanger un paquet de cartes consiste à couper le paquet en deux puis à entrelacer ces deux parties. Lorsque les deux parties sont égales et que l'entrelacement se fait carte par carte, le mélange est dit parfait. Il y a deux types de mélanges parfaits :

- le *out shuffle* lorsqu'on reconstitue le jeu en commençant par la première carte de la première moitié ;
- le *in shuffle* lorsqu'on reconstitue le jeu en commençant par la première carte de la seconde moitié.



Out shuffle d'une liste à 8 éléments.



In shuffle d'une liste à 8 éléments.

a) Utiliser le *slicing* pour décrire :
- la première puis la seconde moitié de la liste ;
- les éléments de rangs pairs puis ceux de rangs impairs.

b) En déduire une fonction `out_shuffle` effectuant le mélange *out shuffle* d'une liste contenant un nombre pair d'éléments.

c) Pour un jeu de 52 cartes, combien de mélanges doit-on effectuer pour retrouver la liste dans son état initial ?

d) Mêmes questions avec le *in shuffle*.

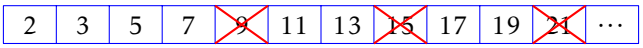
Exercice 5. Nombres premiers

On propose dans cet exercice plusieurs méthodes pour générer la liste des nombres premiers inférieurs ou égaux à 1000.

Première méthode. Définir une fonction qui prend en paramètre un entier n et retourne le plus petit nombre premier p strictement supérieur à n, puis utiliser cette fonction pour générer la liste demandée.

Deuxième méthode. On se propose maintenant de générer cette liste en utilisant le principe du crible d'Ératosthène. L'algorithme procède par élimination : partant de la liste les entiers de 2 à 1000, on supprime de celle-ci les multiples de 2 :

L'entier suivant, 3, est forcément premier. On élimine à son tour ses multiples de la liste :



et on procède ainsi jusqu'à parcourir toute la liste.

Troisième méthode. Bien que cela ne soit pas très efficace, on peut enfin générer cette liste par compréhension en générant successivement les trois listes :
- $\ell_1 = \left\{ n \;\middle|\; n \in [\![2, 1000]\!] \right\}$;
- $\ell_2 = \left\{ (n, \mathscr{D}_n) \;\middle|\; n \in \ell_1 \right\}$ où $\mathscr{D}_n = \left\{ k \in [\![1, n]\!] \;\middle|\; k \text{ divise } n \right\}$ est la liste des diviseurs de n ;
- $\ell_3 = \left\{ n \;\middle|\; (n, \mathscr{D}_n) \in \ell_2 \text{ et } \operatorname{card} \mathscr{D}_n = 2 \right\}$.

Exercice 6. Amnistie à la prison de Sikinia

Dans la prison centrale de Sikinia, il y a 100 cellules numérotées de 1 à 100. Les portes des cellules peuvent être dans deux états : ouvertes ou fermées. On peut passer d'un état à un autre en faisant faire un demi-tour au bouton de la porte. Au moment où commence l'histoire, toutes les cellules sont fermées et occupées.
Pour fêter le vingtième anniversaire de la république de Sikinia, le président décide d'une amnistie. Il donne au directeur de la prison les ordres suivants :

Tournez successivement d'un demi-tour les boutons :
- *de toutes les portes ;*
- *puis d'une porte sur deux à partir de la deuxième ;*
- *puis d'une porte sur trois à partir de la troisième ;*

– *puis d'une porte sur quatre à partir de la quatrième ;*

– *et ainsi jusqu'à la dernière cellule.*

Libérez alors les prisonniers dont la porte de la cellule est ouverte.

Quelles sont les numéros des cellules dont les occupants seront libérés ?

Conjecturer (et éventuellement prouver) le résultat dans le cas général de n cellules.

Exercice 7. Permutation de Josephus

On considère n personnes rangées en cercle et un entier positif $m \leqslant n$. En commençant par une personne désignée, on fait le tour du cercle en éliminant une personne toutes les m. Après chaque élimination on continue à compter avec le cercle qui reste. Le processus se poursuit jusqu'à ce que les n personnes aient été éliminées.
L'ordre dans lequel les personnes sont éliminées du cercle définit la *permutation de Josephus*. Par exemple, la permutation de Josephus pour $n = 7$ et $m = 3$ est $[3, 6, 2, 7, 5, 1, 4]$.
Définir une fonction Python qui prend en paramètres les entiers n et m et qui retourne la permutation de Josephus correspondante.

Et pour ceux qui s'ennuient

Exercice 8. Génération de permutations

Une *permutation* est un arrangement ordonné d'objets. Par exemple, 3124 est une permutation des chiffres 1, 2, 3 et 4. Si on classe par ordre croissant toutes les permutations des chiffres 0, 1 et 2 on obtient :

012 021 102 120 201 210

Quelle est la millionième permutation des chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 lorsqu'on les range par ordre croissant ?

Exercice 9. Nombres de Hamming

Un nombre de Hamming est un entier naturel non nul dont les diviseurs premiers sont inclus dans $\{2, 3, 5\}$:

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, …

Lorsqu'on les ordonne par ordre croissant, quel est le 2000^{e} nombre de Hamming ?