

Initiation à la programmation – TP9–TP10 3 et 4 novembre

Objectif : Manipuler des listes en python – compréhensions

Exercice 1 : manipulations élémentaires de listes

1. Écrire la fonction `tous_zero(l)` qui renvoie `True` si tous les éléments de `l` sont des 0 et `False` sinon.
2. Écrire la fonction `au moins_zero(l)` qui renvoie `True` si au moins un élément de `l` est un 0 et `False` sinon.
3. Écrire la fonction `compte(x,l)` qui compte et renvoie le nombre d'éléments égaux à `x` dans la liste `l`. Par exemple : Si `l_ex=[0,9, 10, 3, 7, 5, 11, 4, 5,0, 1, 7, 1, 6, 3, 3]`
`compte(3,l_ex)` doit renvoyer 3.
4. Écrire la fonction `pairs(l)` qui renvoie la liste des éléments pairs de la liste. Par exemple `pairs(l_ex)` renvoie `[0,10, 4,0,6]`
5. Écrire la fonction `positions_zero(l)` qui renvoie la liste des indices des zéros de la liste.
Par exemple `positions_zero(l_ex)` renvoie `[0, 9]`.
6. Écrire la fonction `supprime_double(l)` qui renvoie une nouvelle liste où les éléments n'apparaissent pas plusieurs fois.
Par exemple `supprime_double(l)` renvoie `[0,9,10, 3, 7, 5, 11, 4, 1, 6]`
7. Listes de chaînes de caractères : on considère une liste de chaînes de caractères. (1). Ecrire une fonction qui teste si les chaînes sont bien rangées dans l'ordre alphabétique (on rappelle que le test peut se faire avec `ch1<ch2`)

(2). Modifier le programme pour renvoyer le premier nom qui n'est pas dans l'ordre !

```
>>> test(["abricot","banane","orange","peche","pomme","poire","prune"])
False
>>> test(["abricot","banane","orange","peche","poire","pomme","prune"])
True
>>> testplus(["abricot","banane","orange","peche","pomme","poire","prune"])
'poire'
```

Exercice 2

Un exemple de fonctionnement des programmes est donné à la fin de l'exercice.

- a. Écrire la fonction `modifie(ch)` en langage Python qui prend en argument une string `ch` et calcule puis retourne la chaîne obtenue en remplaçant tous les `a` par `.`
- b. Écrire la fonction `modifie_liste` en langage Python qui prend en argument une liste de string `l` et renvoie la liste des chaînes modifiées comme à la question a.

```
>>> modifie("ananas")
'@n@n@s'
>>> l
['ananas', 'banane', 'pomme', 'mangue', 'fraise']
>>> modifie_listes(l)
['@n@n@s', 'b@n@ne', 'pomme', 'm@ngue', 'fr@ise']
```

Exercice 3 : modifications de listes

Écrire la fonction `absolue` (maliste) qui étant donnée maliste qui est une liste d'entiers, modifie la liste pour remplacer les nombres négatifs par leur opposé. Attention, il ne faut pas créer une nouvelle liste mais modifier la liste.

```
>>> liste1 = [-2, 5, 1, -1, 7]
>>> liste1
[-2, 5, 1, -1, 7]
>>> absolue(liste1)
>>> liste1
[2, 5, 1, 1, 7]
```

Exercice 4 :compréhensions sur les listes

Les fonctions qui suivent peuvent être écrites en une seule ligne avec le `for` en compréhension.

1. Écrire la fonction `pairs(l)` qui renvoie la liste des éléments pairs de la liste. Par exemple `pairs(l_ex)` renvoie `[0,10, 4,0,6]`.
2. Écrire la fonction `positions_zero(l)` qui renvoie la liste des indices des zéros de la liste. Par exemple `positons_zero(l_ex)` renvoie `[0, 9]`.
3. Ecrire la fonction `listeDiviseurs(n)` qui, étant donné un entier naturel `n`, renvoie la liste de ses diviseurs sauf lui-même. On n'oubliera pas le 1.
4. Écrire la fonction `somme(l1,l2)` qui étant données deux listes de nombres de même taille renvoie la liste obtenue en mettant dans la case `i` la somme des éléments des cases `i` de `L1` et de `l2`.
5. Écrire la fonction `extrait(c,l)` qui étant donnés un caractère `c` et une liste de string `l` renvoie la liste de strings qui contiennent la lettre `c`.
6. Écrire la fonction `couple(l1,L2)` qui étant donnés deux listes de tailles quelconques renvoie la liste de tous le couples qu'on peut former avec un élément de `L1` et un élément de `l2`

```
>>> couple([1,2,3],["a","b","c","d","e"])
[(1, 'a'), (1, 'b'), (1, 'c'), (1, 'd'), (1, 'e'), (2, 'a'), (2, 'b'), (2, 'c'), (2, 'd'), (2, 'e'),
(3, 'a'), (3, 'b'), (3, 'c'), (3, 'd'), (3, 'e')]
>>> listeDiviseur(30)
[1, 2, 3, 5, 6, 10, 15]
>>> extrait("e", ["ananas", "pomme", "poire", "kiwi", "noix"])
['pomme', 'poire']
```

```
>>> somme([3,2,-1,-3],[8,0,2,5])
[11, 2, 1, 2]
```

Exercice 5 : test de modélisation d'un dé

On veut tester le générateur aléatoire de python sur la simulation d'un dé.

1. Écrire la fonction tirage() qui renvoie une face de dé c'est à dire un entier entre 0 et 5 (compris)
2. Écrire la fonction serie(n) qui effectue n tirage de dés et renvoie une liste de taille 6 dans laquelle on trouve les nombres de fois que chaque face du dé a été obtenue. On mettra dans la case i de la liste le nombre de fois où le i est sorti.
3. Écrire la fonction affiche(l) qui affiche les éléments de la liste l séparés par des espaces (sans, et []).
4. Écrire la fonction somme(l1,l2) qui étant données deux listes de nombres de même taille renvoie la liste obtenue en mettant dans la case i la somme des éléments des cases i de l1 et de l2.

```
>>> de=tirage()
>>> de
4

>>> serie(10)
[0, 2, 3, 1, 1, 3]

>>> affiche([2,2,0,3,2,1])
2 2 0 3 2 1

>>> l1=[0, 1, 5, 0, 3, 1]
>>> l2=[0, 1, 5, 0, 3, 1]
>>> somme(l1,l2)
[0, 2, 10, 0, 6, 2]
```

5. Écrire la fonction plusieursSeries(n) qui enchaîne les séries de n tirages en affichant au fur et à mesure les résultats de la série qui vient d'être faite et un bilan global de tous les tirages et demande à l'utilisateur s'il doit continuer ou pas. Cela doit se présenter comme dans l'exemple ci-dessous :

```
>>> plusieursSeries(10)
resultats pour cette serie
1 7 0 1 1 0
bilan des resultats
1 7 0 1 1 0

encore ? (o/n)o

resultats pour cette serie
2 2 1 0 3 2
bilan des resultats
3 9 1 1 4 2
```

```

encore ? (o/n)o

resultats pour cette serie
0 4 1 3 0 2
bilan des resultats
3 13 2 4 4 4

encore ? (o/n)o

resultats pour cette serie
0 3 1 1 1 4
bilan des resultats
3 16 3 5 5 8

encore ? (o/n)o

resultats pour cette serie
3 0 2 2 1 2
bilan des resultats
6 16 5 7 6 10

encore ? (o/n)o

resultats pour cette serie
0 1 0 3 1 5
bilan des resultats
6 17 5 10 7 15

encore ? (o/n)n

```

Exercice 6 : listes de string

On travaille avec des listes de strings . Dans cet exemple les listes contiennent des « ingrédients » de pizzas ...
Exemples :

```

l1= ['jambon', 'mozzarella', 'sauce_tomate', 'champignons']
l2= ['chevre', 'lardons', 'creme_fraiche', 'mozzarella']
l3= ['jambon', 'tomates', 'mozzarella', 'sauce_tomate', 'champignons', 'poivrons', 'oignons']

```

1. Écrire la fonction **inclus**(l1,l2) qui teste si tous les éléments de la liste l1 appartiennent à la liste l2
2. Écrire la fonction **intersection**(l1,l2) qui renvoie la liste des éléments qui sont dans l1 et dans l2 (éléments communs). Par exemple **intersection**(l1,l2) renvoie ['mozzarella']
3. Écrire la fonction **affichage**(l) qui affiche successivement les éléments de la liste l sans aller à la ligne ni pendant l’affichage ni à la fin. Par exemple **affichage**(l3) produira l’affichage suivant : jambon tomates mozarella sauce_tomate champignons poivrons oignons
4. (a) On veut faire une saisie d’une liste de taille quelconque d’ingrédients. Pour cela le programme doit demander à l’utilisateur de saisir un ingrédient ou 0 pour arrêter la saisie. Ensuite la liste des ingrédients saisie doit être renvoyée. Écrire la fonction **saisie**() .

```
>>> saisie()
```

```

ingredient suivant    poivrons
ingredient suivant    oignons
ingredient suivant    tomates
ingredient suivant    fromage
ingredient suivant    0
['poivrons', 'oignons', 'tomates', 'fromage']

```

- (b) Pour cette saisie, on voudrait tester que les mots saisis ne contiennent pas d'espaces mais uniquement des lettres sans caractères spéciaux sauf l'underscore. On définit la variable globale `permis="abcdefghijklmnopqrstuvwxyz_"`
Écrire la fonction `correct(mot)` qui teste si la string `mot` est écrite uniquement avec des caractères de la chaîne `permis` et `False` sinon
- (c) En déduire la fonction `saisieTest()` qui fait la saisie comme à la question a mais en refusant les mots qui ne sont pas corrects . Si un mot ne vérifie pas le test correct alors il n'est pas mis dans la liste et on affiche « saisie incorrecte » pour prévenir l'utilisateur

```

>>> saisieTest()
ingredient suivant    poivrons
ingredient suivant    oignons
ingredient suivant    sauce tomate
saisie incorrecte
ingredient suivant    sauce_tomate
ingredient suivant    tomates
ingredient suivant    fromage
ingredient suivant    merguez
ingredient suivant    0
['poivrons', 'oignons', 'sauce_tomate', 'tomates', 'fromage', 'merguez']

```

Exercice 7 : les entiers parfaits

On appelle nombre parfait un entier naturel qui est égal à la somme de ses diviseurs sauf lui-même. Par exemple, 6 est un nombre parfait car $6 = 1 + 2 + 3$. On rappelle que `p%n` renvoie le reste de la division euclidienne de `p` par `n`.

- (a) Ecrire la fonction `sommeEntiers(l)` qui, étant donnée une liste `l` d'entiers, renvoie la somme des éléments de cette liste.
- (b) Ecrire la fonction `listeDiviseurs(n)` qui, étant donné un entier naturel `n`, renvoie la liste de ses diviseurs sauf lui-même. On n'oubliera pas le 1.
- (c) En déduire la fonction `entierParfait(n)` qui teste si l'entier naturel `n` est un entier parfait.
- (d) Ecrire la fonction `listeEntiersParfaits(b)` qui étant donnée un entier naturel `b` renvoie la liste des entiers parfaits strictement plus petits que `b`.

```

>>> SommeEntiers([1, 2, 3, 5, 6, 10, 15])
42

>>> listeDiviseurs(30)
[1, 2, 3, 5, 6, 10, 15]

>>> entierParfait(6)
True

```

```
>>>entierParfait(16)
False

>>>listeEntierParfaits(1000)
(6, 28, 496)
```

Exercice 8 :jeu type UNO (mais pas UNO)

Le jeu se compose de cartes qui ont une couleur ('n', 'b', 'v', 'r') et une hauteur comprise entre 1 et 13 . Il y a 2 cartes de chaque (hauteur, couleur) plus 2 jokers. Il y a donc $(4*13)*2+2$ cartes soient 106 cartes (chaque carte étant présente 2 fois). Les cartes seront représentées par un tuple (hauteur, couleur) sauf les jokers qui seront représentés par ('j', 'j'). Un jeu (la main d'un joueur) sera une liste de cartes. Lors des tours de jeu, les joueurs pourront poser des cartes en les empilant sur un plateau. Un plateau sera représenté par une liste de cartes. On mettra la nouvelle carte au début de la liste à chaque fois. Donc la carte la plus importante du plateau (la seule qu'on voit) est la première carte de la liste plateau.

0.1 Création du jeu

Écrire la fonction creation (sans paramètre) qui renvoie la liste des cartes du jeu.

```
>>> creation()
[(1, 'b'), (1, 'v'), (1, 'n'), (1, 'r'), (2, 'b'), (2, 'v'), (2, 'n'), (2, 'r'), (3, 'b'), (3, 'v'), (3, 'n'), (3, 'r'), (4, 'b'), (4, 'v'), (4, 'n'), (4, 'r'), (5, 'b'), (5, 'v'), (5, 'n'), (5, 'r'), (6, 'b'), (6, 'v'), (6, 'n'), (6, 'r'), (7, 'b'), (7, 'v'), (7, 'n'), (7, 'r'), (8, 'b'), (8, 'v'), (8, 'n'), (8, 'r'), (9, 'b'), (9, 'v'), (9, 'n'), (9, 'r'), (10, 'b'), (10, 'v'), (10, 'n'), (10, 'r'), (11, 'b'), (11, 'v'), (11, 'n'), (11, 'r'), (12, 'b'), (12, 'v'), (12, 'n'), (12, 'r'), (13, 'b'), (13, 'v'), (13, 'n'), (13, 'r'), ('j', 'j'), ('j', 'j')]
```

0.2 Compter les points

Un ensemble de cartes sera représenté par une liste de cartes. Chaque carte vaut sa hauteur comme nombre de points sauf le joker qui vaut 25 points. Ecrire la fonction ComptePoints(jeu) qui compte le nombre de points d'un jeu.

```
>>> points([(1, 'b'), (1, 'b'), (3, 'n'), (7, 'b'), (12, 'r'), ('j', 'j'), (8, 'b'), (7, 'v')])
64
```

0.3 Cartes Jouables

a. Ecrire la fonction compatible(cartel, carte2) qui teste si les cartes cartes1 et cartes2 peuvent être mises l'une sur l'autre : c'est possible si l'une des cartes est un joker ou si les 2 cartes sont de même hauteur ou de même couleur.

b. En déduire la fonction Jouable(plateauEx, cartel) qui teste si on peut poser la carte cartel sur le plateau plateauEx. La carte est jouable si elle est compatible avec la carte du dessus du plateau c'est à dire la première carte du plateau (liste de cartes).

c. En déduire la fonction cartesJouables(plateauEx , Exjeu) qui renvoie la liste des cartes du jeu Exjeu (liste de cartes) qui sont jouables sur le plateau plateauEx.

Remarque : comme les cartes sont en double on peut avoir 2 fois le même carte dans un jeu et donc dans les cartes jouables.

```

>>> compatible((1, 'v'), (1, 'n'))
True
>>> compatible((1, 'v'), (3, 'n'))
False
>>> compatible(('j', 'j'), (3, 'n'))
True

>>> plateauEx=[(1, 'v'), (1, 'n'), (4, 'n'), (6, 'n')]
>>> jouable(plateauEx, (3, 'b'))
False
>>> jouable(plateauEx, (1, 'b'))
True

>>> monjeu=[(2, 'n'), (6, 'r'), (3, 'r'), (8, 'b'), (2, 'v'), (2, 'v'), (12, 'n'), (8, 'n')]
>>> CartesJouables(plateauEx, monjeu)
[(2, 'v'), (2, 'v')]

```

0.4 Pioche et distribution des cartes

a. Ecrire la fonction `pioche(tas,n)` qui prend `n` cartes au hasard dans la liste de cartes `tas` et qui renvoie un tuple formé de la liste de ces `n` cartes et de la liste des cartes restantes dans le tas

b. Ecrire la fonction `distribue(tas, nbjoueurs, nb)` qui distribue aléatoirement `n` cartes à chacun des `nbjoueurs`. On renverra un tuple composé de la liste des `nbjoueurs` jeux et de la liste des cartes restantes. Si il n'y a pas assez de cartes on ne fera rien et on renverra `False`.

0.5 Jeu

Il n'y a plus qu'à programmer le jeu à l'aide de toutes ces fonctions !

Exercice défi bonus : jeu 2048 sur une ligne

Considérons une liste d'entiers `a`. Pousser `a` vers la gauche revient à déplacer tous les entiers dans `a` différents de 0 vers la gauche, tout en fusionnant deux entiers identiques consécutifs en les additionnant. Par exemple, pousser `[0,4,0,0,2,0,4,0,0,4,0,16,2,0,0,2]` vers la gauche la transforme en `[4,2,8,16,4,0,0,0,0,0,0,0,0,0,0]`.

De la même manière, pousser `a` vers la droite revient à déplacer tous les entiers dans `a` différents de 0 vers la droite, toujours en fusionnant deux entiers identiques consécutifs. Ainsi, pousser `[0,0,2,0,2,0,2,32,0,4,4,0,4,4,0,8]` vers la droite la transforme en `[0,0,0,0,0,0,0,0,0,0,2,4,32,8,8,8]`.

Écrire une fonction `pousse` prenant en paramètre une liste d'entiers `a` et un caractère `s` qui pousse `a` vers la gauche si `s` vaut `'g'`, et qui pousse `a` vers la droite si `s` vaut `'d'`.

Sur la base de ces fonctions, réalisez un jeu 2048 sur une ligne. À chaque mouvement, un chiffre 2 ou un 4 apparaît dans une case vide (case contenant un zéro) de manière aléatoire. Le jeu comporte 16 tuiles.