Exercices d'entraînement

1. Calculer la somme des éléments d'une liste ★

Pour une liste de nombres entiers, donnez la somme de ses éléments. Pour effectuer ce type de calcul, il faut utiliser une variable supplémentaire qui accumule les valeurs dans la liste lors du parcours. Pensez à initialiser cette variable correctement. La liste est de taille quelconque, tenez-en compte dans l'écriture du programme.

2. Répartir les nombres pairs et impairs dans deux listes ★

Étant donné une liste de nombres entiers, répartissez ces éléments dans une liste `IMPAIR` contenant les nombres impairs et une liste `PAIR` contenant les nombres pairs. Affichez le résultat. Pour tester si un nombre k est pair, l'expression k%2 vous sera utile.

3. Compter les occurrences de chaque nombre dans une liste ★

Pour une liste de nombres entiers compris entre 0 et 9, comptez le nombre d'occurrences de chaque nombre dans la liste. Ainsi, pour la liste L = [2, 5, 2, 5, 5, 8], nous avons deux occurrences du chiffre 2, trois occurrences du chiffre 5 et une occurrence du chiffre 8. Nous vous proposons deux approches, vous pouvez programmer l'une ou l'autre :

- Effectuez d'abord une boucle principale sur les nombres recherchés. Ensuite, faites une boucle imbriquée qui compte le nombre d'occurrences de ce nombre dans la liste. S'il apparaît au moins une fois, affichez son nombre d'occurrences.
- Initialisez une liste avec dix valeurs 0, avec la syntaxe `L=[0]*10`. Chaque case de cette liste sert à compter le nombre d'occurrences de chaque valeur : 0, 1,…, 9. Ensuite, faites une boucle pour parcourir la liste. Chaque fois que vous lisez une valeur v, augmentez le compteur correspondant. Une fois la liste parcourue, parcourez la liste des comptages et affichez les chiffres qui sont apparus au moins une fois.

4. Faire apparaître une texture de grillage ★

Nous vous proposons de faire apparaître la texture suivante composée de caractères X et d'espace :

```
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X X X
```

Voici quelques conseils :

- Utilisez une boucle principale pour créer 10 lignes.

- Utilisez une boucle secondaire pour vous déplacer sur 40 colonnes.
- Pour savoir si vous devez afficher un caractère X ou un espace, testez l'expression `(x+y)%2`.
- Pour éviter un retour à la ligne intempestif, utilisez le paramètre `end` de la fonction `print()`.

5. Calculer les racines d'un polynôme du second degré ★

Demandez à l'utilisateur d'entrer les trois coefficients a, b et c du polynôme $ax^2 + bx + c = 0$. Ils seront entrés sur la même ligne, séparés par un espace. Calculez ensuite le déterminant $\Delta=b^2-4ac$. Si Δ est strictement positif, alors affichez les valeurs des deux racines : $(-b\pm\sqrt{\Delta}) / 2a$. Si Δ est nul, alors affichez l'unique solution : -b / 2a. Et si Δ est négatif, informez l'utilisateur qu'il n'y a pas de solution. Voici quelques conseils :

- Une fois les trois paramètres entrés en une fois sur la même ligne, utiliser la fonction `split()` des chaînes de caractères pour récupérer les trois coefficients.
- Pour convertir les coefficients en flottant, utilisez la fonction `float()`.
- Pour le calcul de la racine carré, faites une importation du package math en début de programme en tapant : `import math`. L'appel à la fonction racine carrée se fait ensuite en écrivant : `math.sqrt()`.
- Le but de cet exercice, hormis de faire une application mathématique, est de gérer correctement trois scénarios distincts. Il faut donc ajuster correctement les instructions `if/else` pour ne pas avoir la réponse : "le polynôme a deux racines", doublé du message : "le polynôme n'a pas de solution".
- Pour tester les trois configurations, nous vous proposons les tests suivants :
  - $x^2 - 1 = 0 \rightarrow$ deux racines : 1 et -1
  - $x^2 = 0 \rightarrow$ une racine : 0
  - $x^2 + 1 = 0 \rightarrow$ aucune solution

6. Trouver tous les nombres premiers jusqu'à 100 ★

Un nombre est premier s'il est divisible uniquement par 1 et lui-même. Ainsi, on peut dire que le nombre k est premier s'il n'est pas divisible par aucun des nombres compris entre 2 et k-1. Si le nombre k est divisible par la valeur b, cela veut dire que le reste de la division de k par b est nul, soit en langage informatique que l'expression k%b est égale à zéro. Voici quelques conseils pour mettre en place ce programme :

- Créez une boucle principale pour parcourir tous les nombres de 1 à 100.
- Créez une boucle imbriquée pour tester si k, le nombre courant, est divisible par un nombre compris entre 2 et k-1.
- Si aucun nombre ne divise k, alors affichez le nombre premier trouvé.
- Pour définir une plage d'indices allant de la valeur a jusqu'à la valeur b incluse, utilisez la syntaxe `range(a,b+1)`.

```
>> Nombres premiers trouvés :
>> 1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97
```

7. Vérifier si une liste est triée ★

Soit une liste de nombres entiers L = [1, 5, 11, 19, 8, 50, 77]. Vérifiez qu'ils sont placés par ordre croissant de

gauche à droite. Pour cela, il faut parcourir la liste et comparer chaque élément à l'élément suivant. Il y a un piège lorsque l'on arrive vers la fin de la liste, à vous de trouver lequel. Lorsque vous trouvez deux éléments qui ne sont pas dans le bon ordre, affichez un message d'erreur.

8. Décaler une liste ★

Pour une liste L donnée, décalez ses éléments d'une case vers la droite et affichez le résultat. Recommencez autant de fois qu'il y a d'éléments dans la liste. Voici quelques conseils :

- Par exemple, le décalage à droite de la liste [0,1,2,3,4,5] donne [5,0,1,2,3,4].
- Il n'est pas nécessaire d'utiliser une boucle `for` pour construire le décalage. Il suffit de remarquer qu'un seul élément est finalement déplacé à chaque décalage. Il suffit donc de le retirer et de l'insérer au bon endroit.

Voici un exemple :

```
>> [0, 1, 2, 3, 4, 5, 6, 7]
>> [7, 0, 1, 2, 3, 4, 5, 6]
>> [6, 7, 0, 1, 2, 3, 4, 5]
>> [5, 6, 7, 0, 1, 2, 3, 4]
>> ...
```

9. Construire un tableau de comparaison ★ ★

À partir de deux listes de nombres entiers, disposées l'une verticalement et l'autre horizontalement, remplissez un tableau indiquant si le nombre présent à gauche sur cette ligne est supérieur, inférieur ou égal au nombre en haut de la colonne. Voici quelques conseils :

- Utilisez deux boucles `for` imbriquées permettant de parcourir chacune des deux listes.
- Les colonnes doivent faire trois caractères de large. Ainsi, pour que les colonnes soient bien agencées, utilisez le formatage des chaînes de caractères en utilisant la syntaxe : `print("{0:3}".format(var), end = "")`.

```
   1 15 18 55 60 65 77 80 85 93 99
5  >  <  <  <  <  <  <  <  <  <  <
15 >  =  <  <  <  <  <  <  <  <  <
20 >  >  >  <  <  <  <  <  <  <  <
77 >  >  >  >  >  >  =  <  <  <  <
98 >  >  >  >  >  >  >  >  >  >  <
```

10. Fusionner deux listes triées ★ ★ ★

À partir de deux listes de nombres triés, créez une troisième liste triée contenant les valeurs des deux listes données. Voici un exercice difficile. Seulement quelques lignes sont nécessaires pour créer ce programme, mais il n'est pas si simple à mettre en place. Voici quelques conseils :

- Ne tentez pas d'utiliser une double boucle `for`, cela ne vous aidera pas.

- Créez deux variables i et j donnant la position de parcours dans chaque liste.
- Pour la boucle principale, utilisez une boucle while dont la condition d'entrée consiste à vérifier qu'il reste des éléments à parcourir dans les deux listes.
- Comparez les deux valeurs L1[i] et L2[j] : la plus petite valeur est insérée dans la liste résultat. Par exemple, si la plus petite valeur est associée à la liste Liste1, la position i est incrémentée pour passer à la position suivante : i+=1.
- Lorsque tous les éléments d'une des deux listes ont été parcourus, vous ne pouvez pas continuer à comparer les éléments entre les deux listes. Vous sortez donc de la boucle principale.
- Une fois tous les éléments d'une liste traitée, il reste à basculer les éléments restants dans l'autre liste vers la liste résultat. Une boucle est nécessaire pour traiter ce cas. Attention, à ce niveau, deux scénarios sont possibles.

Une fois le programme réalisé, nous vous conseillons de le tester sur Python Tutor avec des listes de quelques éléments. Il est vraiment intéressant de comprendre son comportement, car dans cet algorithme la boucle while permet d'avancer dans deux listes à la fois alors que traditionnellement une boucle avance seulement dans une liste.