TP 5 - Listes et fonctions

Récupérez sur la page du cours le fichier `iutk.py` (celui de ce TP) qui fournit la bibliothèque graphique que nous allons utiliser et placez-le dans votre répertoire de travail.

Attention : chaque fonction que vous écrivez doit être précédée d'un commentaire qui indique ce qu'elle fait en fonction des paramètres donnés !

Exercice 1. Récupérez le programme `liste100000.py` sur la page du cours. Ce programme construit la liste des nombres entre 0 et 99 999 de deux manières différentes et affiche le temps d'exécution pour chacune des méthodes. Vous pouvez consulter la documentation du module `time` ici :`http://docs.python.org/3.2/library/time.html`

Expliquez l'énorme différence de temps entre les deux méthodes.

Exercice 2. Salutations

Vous devez tester les 3 **fonctions** suivantes dans le programme que vous rendez (`Exo2.py`).

1. Écrire une fonction `bonjour` sans argument qui **affiche** le texte *bonjour*.
2. Écrire une fonction `disBonjour` qui prend un entier `n` et **affiche** `n` fois le texte *bonjour*. Par exemple, `disBonjour(3)` affichera :

   ```
   bonjour
   bonjour
   bonjour
   ```

3. Écrire une fonction `renvoieBonjour` qui prend un entier `n` et **renvoie** une chaîne composée de `n` fois le texte *bonjour* séparés par des espaces. Attention, cette fonction n'affiche rien !

Exercice 3. Carrés

1. Écrire une fonction `carre` qui prend un entier en argument et renvoie son carré.
   Par exemple, `carre(5)` renverra `25`.
2. Écrire un programme utilisant la fonction `carre` qui demande un nombre `n` à l'utilisateur et affiche les `n` premiers carrés.
   Par exemple, si l'utilisateur entre 5, le programme affichera :

   ```
   1
   4
   9
   16
   25
   ```

Exercice 4. Nombre à la demande

Pour chaque question, vous devez utiliser les fonctions des questions précédentes.

1. Écrire (**et tester**) une fonction `demandeEntier` qui ne prend pas d'argument et qui demande à l'utilisateur de rentrer un nombre et le renvoie. Attention, cette fonction doit renvoyer un `int` et pas une chaîne !
2. Écrire (et tester) une fonction `estEntre` qui prend trois arguments entiers `val`, `a` et `b` et renvoie `True` si `val` est compris entre `a` et `b` et `False` sinon.

3. Écrire (et tester) une fonction `demandeEntre` qui prend deux arguments entiers `a` et `b` et redemande à l'utilisateur de rentrer un nombre jusqu'à ce que le nombre entré soit compris entre `a` et `b`. Pour finir, la fonction renvoie le nombre choisi.

   Par exemple, `demandeEntre(1,5)` redemande à l'utilisateur de rentrer un nombre jusqu'à ce que ce nombre soit compris entre 1 et 5.
4. Écrire un programme qui tire au hasard un nombre entre 1 et 10 puis demande à l'utilisateur de rentrer un nombre entre 1 et 10 jusqu'à ce qu'il trouve le nombre secret.

Exercice 5. Nombres mystères
Que fait la fonction suivante ?

```
1 def mystere(n):
2     for i in range(2,n-1):
3         if n%i == 0:
4             return False
5     return n>1
```

Écrire un programme qui demande un nombre n à l'utilisateur et indique *tous* les nombres premiers qui sont inférieurs ou égaux à n.

Exercice 6. Liste

1. Écrire une fonction `listeAleatoire` qui prend un entier `n` en argument et renvoie une liste de taille `n` contenant des nombres entre 1 et 10 tirés au hasard. Vous utiliserez pour ce faire la fonction `randint`.
2. Écrire un programme qui crée une liste de taille 20 avec la fonction `listeAleatoire` et l'affiche.
3. Écrire une fonction `maxListe` qui prend une liste `l` en argument et renvoie le plus grand entier apparaissant dans la liste.

   Par exemple, `maxListe([1,2,4,1])` renverra `4`.
4. Compléter le programme de la question 2 pour qu'il affiche en plus le maximum de la liste.
5. Écrire une fonction `singletons` qui prend une liste d'entiers et renvoie la liste contenant les mêmes nombres mais au plus une fois et dans le même ordre.

   Par exemple, `singletons([2,1,2,1,3,2,1,4]` renverra la liste `[2,1,3,4]`.
6. Compléter le programme de la question 4 pour qu'il affiche en plus la liste sans doublons.
7. Écrire une fonction `nbOccurences` qui prend un nombre `n` et une liste `l` et renvoie le nombre de fois que l'entier `n` apparaît dans la liste `l`. Par exemple,

   ```
   nbOccurences(1,[1,2,4,1])
   2
   nbOccurences(5,[1,2,4,1])
   0
   nbOccurences(2,[1,2,4,1])
   1
   ```
8. Compléter le programme de la question 6 pour qu'il affiche pour chaque élément de la liste le nombre de fois où il apparaît dans la liste.

   Par exemple, si la liste est `[1,1,9,1,2,9,4,2,2,1,1,1,9,1,2,9,4,2,2,1]`, le programme affichera :

```
1 apparait 8 fois
2 apparait 6 fois
4 apparait 2 fois
9 apparait 4 fois
```

9. Écrire une fonction `plusFrequent` qui prend une liste `l` d'entiers en argument et qui renvoie le nombre qui apparaît le plus de fois dans la liste. Si plusieurs nombres sont possibles, on renverra le plus grand.

   Par exemple, dans la liste `[1,2,1,1,2,2,3,3,4]`, les nombres 1 et 2 apparaissent tous les deux 3 fois. On renverra donc 2 qui est le plus grand des candidats.

Exercice 7. Morpion ★

Le but de l'exercice est de réaliser le jeu du *morpion*. Le résultat est donné dans le programme `morpion.pyc`. Vous pouvez lancer le programme, en tapant :

```
python3 morpion.pyc
```

Afin que votre programme soit lisible et bien structuré, nous vous conseillons de :

1. fabriquer une liste de taille 9 représentant les cases, et d'utiliser 3 valeurs différentes pour décrire l'état d'une case.
2. découper votre code en fonctions :
   (a) une fonction qui affiche la grille avec les pions joués,
   (b) une fonction qui teste si un joueur a gagné,
   (c) une fonction qui teste si la grille est pleine,
   (d) une fonction qui fait jouer un des 2 joueurs,
   (e) le reste dans la boucle principale.

Si vous avez fini, faites une version où l'on peut jouer contre l'ordinateur.