Partiel d'Informatique-Algorithmique
Mardi 23 octobre 2018 – Durée : 1h50

Documents papier et informatique autorisés – Accès internet et téléphones portables interdits.

Important

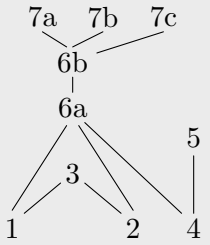
A. Créez votre répertoire de partiel `E2_Nom_Prenom` sur le « bureau » de l'ordinateur. À la fin, il devra contenir seulement deux fichiers : `numchiffre.py` et `kaprekar.py`. Lorsque vous avez terminé ou que le temps imparti est écoulé, vous devrez remettre votre répertoire de partiel à l'enseignant·e ou imprimer les fichiers et les rendre sous format papier.

B. Il est nécessaire de remplir les en-têtes des fonctions et des modules, de **tester vos fonctions**, et de commenter lorsque cela vous semble nécessaire.

C. N'oubliez pas que si vous avez un doute sur ce que produit une expression, vous pouvez la tester dans l'interpréteur (le shell) Python ou en afficher la documentation en tapant `help`.

D. **Sauvegardez régulièrement votre travail.**

E. Les questions dépendent les unes des autres :



Par exemple il est indispensable de traiter les questions 1 et 2 et 4 avant d'aborder la question 6a. Si vous n'arrivez pas à écrire une fonction, simulez son fonctionnement en lui faisant retourner une valeur du même type que celui qui est demandé dans l'énoncé, et passez à la suite.

Introduction

Un nombre de Kaprekar est un entier naturel qui, lorsqu'il est élevé au carré, peut être séparé en une partie gauche et une partie droite dont la somme donne le nombre initial. Par exemple 55 est un nombre de Kaprekar car $55^2 = 3025$ et $55 = 30 + 25$, et 12 n'en est pas un car $12^2 = 144$ mais ni $1 + 44$, ni $14 + 4$, ni 144 ne sont égaux à 12. Il n'y a pas besoin que les deux parties aient le même nombre de chiffres, ni qu'elles commencent par un chiffre non nul, ainsi par exemple 4879 est un nombre de Kaprekar puisque $4879^2 = 23804641$ et $238 + 04641 = 4879$.

Travail à rendre

Module `numchiffre.py`

Créer un module `numchiffre.py` qui contiendra les en-têtes habituels, les définitions des fonctions ci-dessous (questions 1 et 2) et les tests (question 3).

1. (3 points) Créer une fonction `liste_chiffres` avec un paramètre entier, qui renvoie la liste de ses chiffres en base 10. Par exemple l'appel `liste_chiffres(729)` devrait renvoyer la liste `[7,2,9]`. *Les fonctions de transtypage, en particulier `str`, ainsi que la fonction `reverse` sont autorisées, mais pas obligatoires, pour cette question ; on rappelle que si `chaine_chiffres` est une chaîne de caractères ne comprenant que des chiffres, alors `[int(c) for c in chaine_chiffres]` produit une liste d'entiers.*

Solution: On donne deux solutions.

```
1. def liste_chiffres_1(p_nombre):                    # 'p_nombre' est de type int
2. """ renvoie la liste des chiffres de 'p_nombre'"""
3.     return [int(x) for x in str(p_nombre)]         # 'str(p_nombre)' est de type str

1. def liste_chiffres_2(p_nombre):      # 'p_nombre' est de type int
2. """ renvoie la liste des chiffres de 'p_nombre'"""
```

```
3.      liste = []                      # initialise le resultat a la liste vide
4.      while p_nombre > 0:             # tant que 'p_nombre' est non nul
5.          liste.append(n % 10)        # ajoute a liste le dernier chiffre de 'p_nombre'
6.          n = n // 10                 # on ne garde que le quotient de la division par 10
7.      liste.reverse()                 # renverse la liste
8.      return liste                    # renvoie le resultat de type list
```

2. (3 points) Créer une fonction `forme_nombre` qui prend en argument une liste de chiffres (entiers entre 0 et 9) et renvoie le nombre formé sur ces chiffres. Par exemple un appel sur la liste `[0,2,1,7]` devrait renvoyer l'entier `217`. *Vous pouvez vous aider de l'une des deux écritures suivantes (de votre choix)*

$$217 = \mathbf{7} \times 10^0 + \mathbf{1} \times 10^1 + \mathbf{2} \times 10^2 + \mathbf{0} \times 10^3 \quad (1)$$
$$= \mathbf{7} + 10\,(\mathbf{1} + 10\,(\mathbf{2} + 10 \cdot \mathbf{0}))\,. \quad (2)$$

On pourra utiliser une boucle `for`.

Solution: On donne trois solutions. La première utilise le passage par une chaîne de caractères, la seconde et la troisième utilisent les écritures données par l'énoncé.

```
1. def forme_nombre_0(p_liste):             # 'p_liste' est de type list
2. """ renvoie le nombre dont l'ecriture decimale est donnee par 'p_liste' """
3.     chaine = ""                          # 'chaine' est intitialisee a la chaine vide
4.     for k in range(len(p_liste)):        # parcours de 'p_liste', 'k' est la position.
5.         chaine = chaine+str(p_liste[k])  # ajoute le chiffre en position k a 'chaine'
6.     return int(chaine)                   # renvoie le transtypage de 'chaine' en entier

1. def forme_nombre_1(p_liste):             # 'p_liste' est de type list
2. """ renvoie le nombre dont l'ecriture decimale est donnee par 'p_liste' """
3.     valeur = 0                           # 'valeur' initialisee a 0
4.     longueur = len(p_liste)              # 'longueur' contient le nombre de chiffres
5.     for k in range(longueur):            # iterateur 'k', parcours de 'p_liste'
6.         valeur=valeur
                +p_liste[longueur-k]*10**k  # mise à jour de 'valeur' suivant (1)
7.     return valeur                        # renvoie 'valeur', de type int

1. def forme_nombre_2(p_liste):             # 'p_liste' est de type list
2. """ renvoie le nombre dont l'ecriture decimale est donnee par 'p_liste' """
3.     valeur = 0                           # 'valeur' initialisee a 0
4.     for d in p_liste:                    # parcours de 'p_liste': d est un chiffre
5.         valeur = valeur * 10 + d         # mise a jour de 'valeur' suivant (2)
6.     return valeur                        # renvoie 'valeur', de type int
```

3. (2 points) Importer la fonction `randrange` du module `random` à l'endroit adapté dans `numchiffre.py`. Dans le corps du module `numchiffre.py` ajouter les instructions :

```
if __name__ == "__main__" :
    for j in range(98,102):
        print (j,forme_nombre(liste_chiffres(j)))
    liste_test = []
    for k in range(5):
        liste_test.append(randrange(10))
    print (liste _test, forme_nombre(liste_test))
```

Executer `numchiffre.py`, recopier le résultat d'execution à la fin et commenter : le résultat est-il conforme à ce qui est attendu ?

Solution:

```
(executing file "numchiffre.py")
98 98
99 99
100 100
101 101
[2,3,6,4,4] 23644
```

Pour des nombres à deux ou trois chiffres (entre 98 et 101 inclus) transformés en listes de chiffres par `liste_test`, `forme_nombre` permet de retrouver le nombre initial. Inversement, `liste_chiffre` est capable de retrouver les chiffres d'un nombre à 5 chiffres aléatoires.

Programme principal

Créer un fichier `kaprekar.py`, importer le module `numchiffre.py`.

4. (3 points) Créer une fonction `decoupe` qui prend en argument deux paramètres : une liste `p_liste` et un entier positif `p_dec` et qui renvoie deux listes, l'une contenant les éléments de `p_liste` de 0 à `p_dec` (inclus), l'autre contenant les éléments de `p_liste` en positions de `p_dec+1` à la fin. *On pourra renvoyer le résultat sous forme de tuple ou bien de liste, selon votre préférence.* Voici quelques exemples :
   — `decoupe([3,4,3,2,1,2,1,0],4)` devrait renvoyer le tuple `([3,4,3,2,1],[2,1,0])` ou bien la liste `[[3,4,3,2,1],[2,1,0]]`.
   — `decoupe([5,0,9,2,8,4,1,5,7],0)` devrait renvoyer le tuple `([5],[0,9,2,8,4,1,5,7])` ou bien la liste `[[5],[0,9,2,8,4,1,5,7]]`.

Solution: Solution avec un tuple en sortie :

```
1. def decoupe(p_liste, p_dec):    # 'p_liste' est une liste, 'p_dec' est un entier
2. """ renvoie deux sous-listes de p_liste : les elements de position 0 a p_dec,
       puis les elements de position p_dec+1 a la fin """
3.     return (p_liste[:p_dec], p_liste[p_dec:])  # renvoie un tuple
```

5. (1 ½ points) Ecrire des instructions permettant de tester la fonction `decoupe`. Recopier vos résultats d'execution à la fin du programme.

6. (3 points) (a) Ecrire une fonction `somme_parties` qui prend en argument un entier naturel et renvoie la liste des sommes des parties de son écriture décimale. Par exemple `somme_parties(3025)` devrait renvoyer `[28,55,307,3025]` car $3 + 025 = 28$ et $30 + 25 = 55$ et $302 + 5 = 307$ et $3025 = 3025$. *On appelera les fonctions précédemment définies.*

   (b) Ecrire une fonction `est_kaprekar` qui prend en argument un entier naturel `p_entier` et renvoie un booléen dont la valeur est `True` si `p_entier` est un nombre de Kaprekar et `False` sinon. *Il est recommandé de relire la définition et les exemples avant d'écrire cette fonction.* L'opération `in` est autorisée pour rechercher dans la liste renvoyée par `somme_partie`.

Solution: Question (a)

```
1. def somme_parties(p_entier):              # 'p_entier' est de type int
2. """ renvoie la liste des sommes des parties correspondant aux decoupes de
       l'ecriture decimale de 'p_entier' """
3.     l = liste_chiffre(p_entier)           # 'l' est de type list
4.     sommes = []                           # 'sommes' est initialisee a la liste vide
5.     for dec in range(len(l)):             # 'dec' est la position de decoupe
6.         p1, p2 = decoupe(l, dec)          # affectation multiple
7.         sommes.append(forme_nombre(p1)    # ajoute la somme des nombres
                         +forme_nombre(p2))    formes sur p1, p2
8.     return sommes                         # renvoie le resultat
```

NB : Si on a choisi une liste plutôt qu'un tuple à la question 4, la ligne 6 doit être remplacée par

```
...
6. p1 = decoupe(l,dec)[0]
7. p2 = decoupe(l,dec)[1]
...
```

Question (b)

```
1. def est_kaprekar(p_entier):                          # 'p_entier' est de type int
2. """ indique si p_entier est un nombre de Kaprekar, ou non """
3.     return p_entier in somme_parties(p_entier**2)  # renvoie un booleen
```

7. (4 ½ points) (a) Dans le corps du programme principal, écrire les instructions nécessaires pour demander à l'utilisateur·trice de saisir un entier naturel (refuser tant qu'il n'est pas positif) et répondre s'il s'agit d'un nombre de Kaprekar ou non. Recopier les résultats d'execution à la fin (on pourra essayer : `22,-4,9,297,7777`).

   (b) Donner tous les nombres de Kaprekar entre 1 et 1000 (on ne demande pas les décompositions correspondantes). *Vous devez en trouver* 11.

   (c) Il existe un unique nombre de Kaprekar strictement compris entre 82600 et 83000. Quel est-il ? Vous pourrez utiliser une boucle `for` ou `while` en commentant votre choix.

Solution:

```
# ------ Programme principal -------

""" Question (a) """
1. n=-1
2. while n<0:                                        # tant que n ne convient pas
3.     n = int(input("Entrez un entier naturel"))   # (re)demande d'entrer n
4. if est_kaprekar(n):
5.     print("L'entier naturel ", n, " est un nombre de Kaprekar")
6. else:
7.     print("L'entier naturel ", n, " n'est pas un nombre de Kaprekar")

""" Question (b) """
1. liste_kaprekar = []               # initialise la liste des nombres de Kaprekar trouves
2. compteur_kaprekar = 0             # initialise le compteur de nombres de Kaprekar
3. for k in range (1,1001):
4.     if est_Kaprekar(k):
5.         liste_kaprekar.append(k)  # ajoute 'k' dans la liste
6.         compteur_kaprekar += 1    # incremente le compteur
7. print("Il y a ",
   compteur_kaprekar , " nombres de
   Kaprekar entre 1 et 1000 :")
8. print(liste_kaprekar)             # affiche la liste

""" Question (c) """
1. k = 82600                                  # initialise 'k'
2. kaprekar_trouve = False                    # la variable booleenne 'kaprekar_trouve'
                                                indique si le nombre a ete trouve
3. while k<83001 and not(kaprekar_trouve):  # tant que k < 83001 et qu'on n'a pas trouve
4.     kaprekar_trouve = est_kaprekar(k)    # mise a jour de 'kaprekar_trouve'
5.     if kaprekar_trouve:                  # si c'est trouve
6.         print("C'est", k)                # on l'affiche
```

```
7.     k = k+1                              # incremente 'k'

# ------ Resultats d'execution -------

(executing file "kaprekar.py")
...
Il y a 11 nombres de Kaprekar entre 1 et 1000 :
[1, 9, 10, 45, 55, 99, 100, 297, 703, 999, 1000]
C'est 82656
```

Pour la question (c) on a préféré la boucle `while` à la boucle `for`, a posteriori c'était mieux car le nombre recherché est très proche de 82600. Il était donc inutile de tester les nombres de 82657 à 83000.

Bonus (seulement si tout le reste est parfait)

Il n'est pas nécessaire de traiter ces questions pour avoir une excellente note.

8. (½ point) Réécrire les fonctions du module `numchiffre.py` de manière récursive.

Solution:

```
1. def liste_chiffres_rec(p_nombre):
2. """ fonction recursive, renvoie la liste des chiffres de 'p_nombre'"""
3.     if p_nombre < 10:
4.         return [p_nombre]                        # si p_nombre < 10 son ecriture n'est
                                                      composee que d'un seul chiffre
5.     else:
6.         l = liste_chiffres_rec(x // 10)
7.         l.append(x % 10)
8.         return l

1. def forme_nombre_rec(p_chiffres):     # p_chiffres est une liste
2. """ fonction recursive, renvoie le nombre dont l'ecriture decimale est 'p_chiffres' """
3.     try:
4.         return p_chiffres.pop() + 10*forme_nombre_rec(p_chiffres)    # utilise (2)
5.     except IndexError:                # cas terminal : chiffres.pop() souleve IndexError
                                           car 'p_chiffres' est vide
6.         return 0

""" effet de bord : transforme le parametre en la liste vide """
""" pour l'eviter """

1. def forme_nombre_rec_mieux(p_chiffres):
2. """ renvoie le nombre dont l'ecriture decimale est 'p_chiffres' sans effet de bord """
3.     if len(p_chiffres) > 0:  # evite le cas ou 'p_chiffres' est vide
4.         return p_chiffres[-1] + 10*forme_nombre_rec_mieux(p_chiffres[:-1])
5.     else:                    # renvoie 0 si 'p_chiffres' est vide
6.         return 0
```

9. (½ point) Donner une liste des petits nombres de Kaprekar en base 2.

Solution: On remplace `10` par `2` dans `liste_chiffres_rec` et `forme_nombre_rec`. Voici une liste :

```
[1, 11, 110, 111, 1010, 1111, 11100, 11111, 100100, 110011,
111111, 1010101, 1011011, 1111000, 1111111, 10001000, 10010011,
10101011, 10111011, 11001101, 11111111, 101010110, 101011111, 101101101, 111110000]
```

Parmi ceux-ci, 6 (110), 28 (11100) et 496 (111110000) sont des nombres parfaits.