Devoir surveillé 01

Informatique

Durée : 3h - Calculatrice NON autorisée

Ce devoir comporte une série de questions et 2 problèmes indépendants. Vous pouvez les traiter dans l'ordre de votre choix, mais vous ordonnerez vos copies dans l'ordre croissant. Une grande importance sera accordée à la présentation et à la précision des arguments avancés pour répondre aux questions, ainsi qu'au respect de la syntaxe et de l'indentation (on tracera des traits verticaux pour repérer l'alignement). Il est conseillé de commenter sommairement les programmes si ceux-ci ne sont pas très simples.

Questions diverses

1. Bonus : Donner les représentations binaires et hexadécimales du nombre décimal 365.
2. On appelle suite de Catalan l'unique suite $(c_n)_{n\in\mathbb{N}}$ telle que $c_0 = 1$ et $\forall n \in \mathbb{N}$, $c_n = \sum_{k=0}^{n-1} c_k c_{n-1-k}$. Par exemple $c_1 = c_0^2$.
   (a) Calculer c_2.
   (b) Écrire une fonction **`Catalan(n)`** qui prend comme argument un entier n et rend comme résultat c_n.
   (c) Faire une étude de la complexité de la fonction **`Catalan`**, en fonction de n (on attend la complexité asymptotique).
3. On donne ci-dessous un algorithme possible de recherche par dichotomie prenant comme argument une liste de nombres triés par ordre croissant.

```
def rech_dicho(x,L):
    g,d = 0, len(L)-1
    if x < L[g] or L[d] < x:
        return None
    if L[d] == x:
        return d
    while d-g > 1 :
        m = (g+d)//2
        if L[m] > x:
            d = m
        else:
            g = m
    if L[g] == x:
        return g
    else:
        return None
```

   (a) L'appliquer à la liste **`[2,5,7,7,7,7,9,13,15,17]`** pour rechercher la valeur 7. On précisera les valeurs successives de **`g`** et **`d`** au début de l'appel, et celles de **`g`**, **`d`** et **`m`** à la fin de chaque boucle while.

   (b) Montrer soigneusement que la propriété P : « x est dans l'intervalle $[\mathtt{L[g]}, \mathtt{L[d]}[$ » est un invariant de boucle.

   (c) Expliquer, en utilisant cet invariant, le fonctionnement de l'ensemble du programme et justifier en particulier qu'on sort bien de la boucle **`while`**.

Exercice 1 : Loi de Benford

On prend un journal au hasard, et on recense les nombres de toute nature rencontrés dans ses articles (prix, nombre de personnes, statistiques, numéros divers). En ne considérant que le premier chiffre significatif de ces nombres (par exemple 2 dans 23 euros, 8 dans 85 000 personnes, etc...) on pourrait s'attendre à ce que la probabilité de rencontrer chaque chiffre (de 1 à 9) soit de 1/9. De façon apparemment paradoxale, il n'en est rien. Des études statistiques, appuyées par des considérations théoriques, ont montré que la probabilité de rencontrer le chiffre c est donnée sous certaines conditions par la loi de Benford (log désigne le logarithme décimal).

$$p(c) = \log\left(1 + \frac{1}{c}\right)$$

Ainsi, d'après cette loi étonnante, il serait beaucoup plus probable de rencontrer comme premier chiffre un 1 ($p(1) = 0{,}3$) qu'un 9 ($p(9) = 0{,}05$).
On se propose de tester la validité de cette loi sur un cas particulier d'ensemble de nombres : la liste de la population de chacune des 36 687 communes de France, mesurée grâce au recensement de 2011. Cette information - disponible sur le site de l'INSEE - est supposée avoir été récupérée dans un fichier **`communes.txt`**, contenant le nom de la commune, une virgule, et la population de la commune, comme par exemple :

1. Écrire une suite d'instructions qui ouvre le fichier communes.txt et crée deux listes nommées **`lcom`** et **`lpop`** contenant respectivement la liste des noms de communes et la liste des populations (qui doivent être des entiers). On pourra se reporter à l'annexe.

2. Écrire une fonction **`pop_sup(lpop,n)`** qui renvoie le nombre de villes dont la population est supérieure ou égale à **`n`**.

3. Écrire une fonction **`villes_mortes(lcom,lpop)`** qui renvoie sous forme de liste le nom des communes contenant 0 habitants [1]

4. Écrire une fonction **`commune_plus_peuplee(lcom,lpop)`** qui renvoie le nom de la commune la plus peuplée de France. On demande de le faire sans utiliser la fonction **`max()`**.

5. Écrire une fonction **`lprem(L)`** qui prend en argument une liste **`L`** de nombres entiers naturels et qui renvoie la liste des premiers chiffres de chaque élément de **`L`**, en oubliant les entiers nuls.

   Par exemple, **`lprems([25111,12,589,0,92556])`** doit renvoyer **`[2,1,5,9]`**.

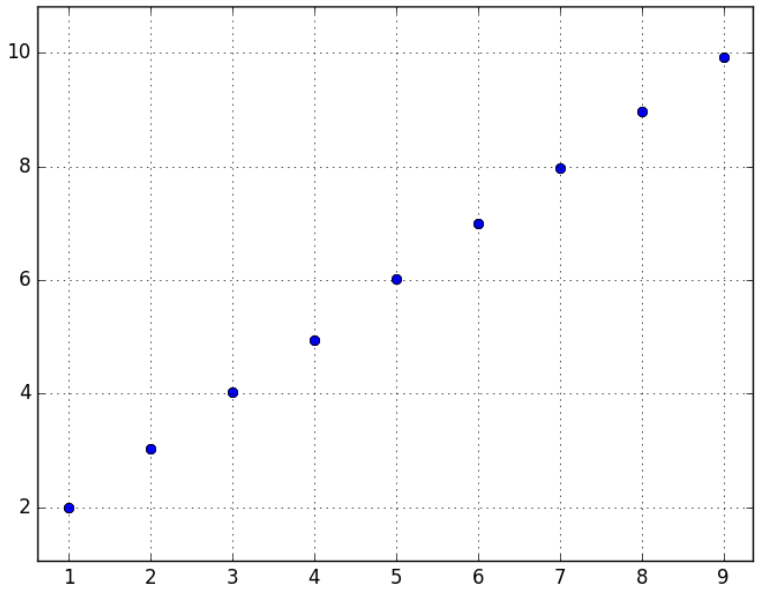
6. Écrire une fonction **`proportion(L)`** qui à une liste d'entiers quelconque renvoie la liste des pourcentages d'apparition de chaque premier chiffre, de 1 à 9.
   Par exemple, **`proportion([11,125,18544,25,98])`** doit renvoyer **`[0.6,0.2,0,0,0,0,0,0,0.2]`**.

7. On écrit alors les instructions suivantes :

```
lprop=proportion(lpop)
f=[(i+1)*10**(lprop[i]) for i in range(9)]

plt.plot([i+1 for i in range(9)],f,'bo')
plt.grid()
plt.show()
```

   Et on obtient le graphique suivant :



   Montrer que ce résultat confirme la loi de Benford sur cette série de données.

1. Cela peut paraître bizarre, mais cela existe : les communes « mortes pour la France » furent totalement dévastées lors la bataille de Verdun en 1916 et ne furent jamais reconstruites, en raison de la présence trop importante de munitions non explosées et de sols bouleversés et pollués.
Pour conserver leur mémoire, l'État, en 1919, lors des premières élections municipales organisées après la Grande guerre, a décidé de les doter d'un conseil municipal restreint.

Exercice 2 : Hauteurs de marées

À partir d'une base de données du service hydrographique et océanographique de la marine (shom) établie à partir du 1er janvier 2013, il est possible d'avoir accès à un relevé précis de la hauteur d'eau effectué toutes les 10 minutes au port du Pouliguen, accompagné de la date et de l'heure. On utilise pour cela une variable **data** qui a la structure d'une liste de listes, chaque sous-liste correspondant à un triplet **[date,heure,hauteur]**. Pour illustrer les choses, voici le début de **data** :

```
[['01/01/2013','00:00',2.0],
['01/01/2013','00:10',1.995],
['01/01/2013','00:20',1.999],
['01/01/2013','00:30',2.029],...]
```

On va d'abord exploiter **data** dans le but d'obtenir les représentations graphiques ci-dessous :

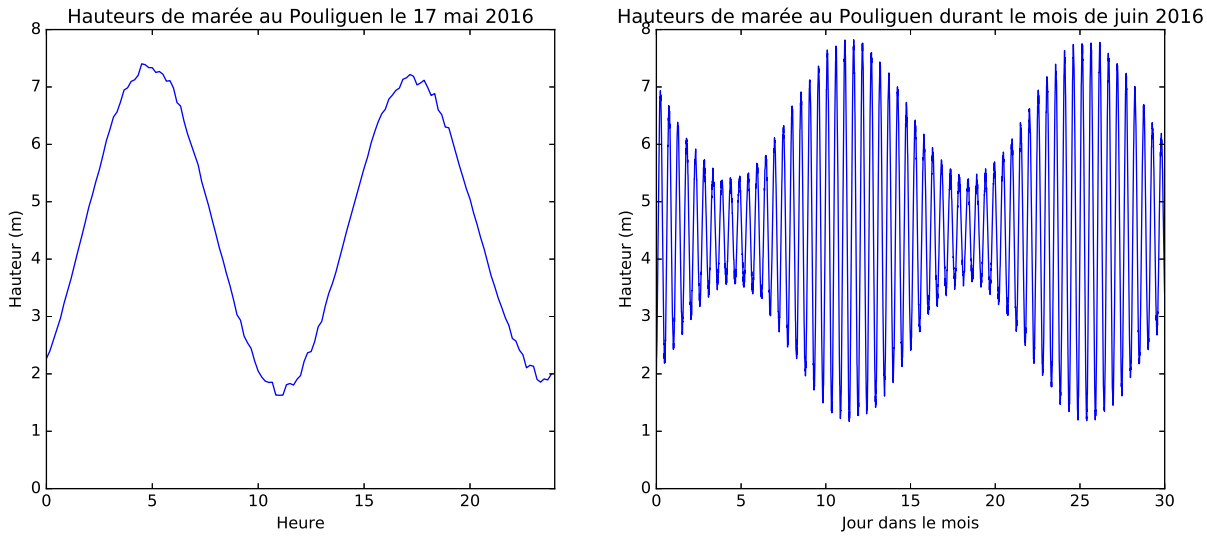


FIGURE 1 – Représentations graphiques

1. Que renvoie **data[2][1]** ?
2. On cherche à extraire des informations de **data** nécessaires au tracé des deux graphiques précédents. Écrire un programme qui :
   - récupère les deux extraits de **data** correspondant à la liste des hauteurs (et seulement des hauteurs) pour le 17 mai 2016 et le mois de juin 2016, sous forme de deux listes **h_17mai16** et **h_juin16**.
   - construit deux listes de flottants **dates1** correspondant aux différents instants du 17 mai 2016 exprimés en heures, et **dates2** les instants de juin 2016 exprimés en jours.

   Il est demandé de commenter sommairement votre programme.
3. Préciser enfin comment obtenir, à partir des différentes listes créées, les représentations graphiques de la figure 1, en utilisant la fonction **plot** du module **matplotlib** (on se préoccupera pas de l'importation de ce module, ni des légendes, ni du subplot : on demande seulement deux graphiques bruts).

On cherche maintenant à déterminer la liste des amplitudes (encore appelées marnages) correspondant à la différence de hauteur entre une pleine mer (ou marée haute) et une basse mer (ou marée basse) à partir de la liste des amplitudes.

4. Définir une fonction **`moyenne(L)`** qui calcule la valeur moyenne d'une liste de valeurs.

5. Écrire le code permettant d'extraire dans une liste **`height`** toutes les hauteurs (et seulement les hauteurs) contenues dans **`data`**.

6. On souhaite récupérer les données associées aux amplitudes de chaque marée. Pour cela, on va procéder de la manière suivante : on cherche à construire d'abord la liste de tous les indices pour lesquels le niveau de l'eau passe par sa valeur moyenne en montée. On nomme ces indices les passages par la moyenne en montée (abréviation PMM).

   On propose ci-après une fonction **`construction_successeurs(height)`** qui prend en argument une liste de hauteurs d'eau et qui renvoie la liste des indices qui suivent immédiatement les points pour lesquels la hauteur d'eau passe par sa valeur moyenne de manière croissante. Recopier la focntion sur votre copie en complétant les lignes nécessaires.

```
def construction_successeurs(height):
    m=                      #ligne à compléter
    liste_PMM=[]
    for i in range(    ):    #ligne à compléter
        if                : #ligne à compléter
        liste_PMM.append(i)
    return liste_PMM
```

7. Proposer alors une fonction **`decompose_maree(liste_niveaux)`** qui permet de décomposer une liste de niveaux en liste de marées. On omettra les données précédant le premier PMM et celles succédant au dernier PMM. Par exemple **`decompose_maree([-1, 1, 2, -2, 2, 1, -6, -4, 2, 5])`** (noter que cette liste est de moyenne nulle) retournera **`[[1, 2, -2], [2, 1, -6, -4]]`**.

8. Écrire une fonction **`marnage(height)`** qui renvoie une liste des amplitudes de toutes les marées à partir de la liste **`height`**. On utilisera judicieusement les fonctions précédentes.

9. Quelle est la complexité de la fonction **`marnage`** en fonction de n=**`len(height)`** ?

Opérations et fonctions Python disponibles

Fonctions

- **range(n)** renvoie la séquence des **n** premiers entiers $(0 \to n-1)$
- **list(range(n))** renvoie une liste contenant les **n** premiers entiers dans l'ordre croissant : **list(range(5))** $\to$ **[0, 1, 2, 3, 4]**
- **random.random()** renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme
- **random.randrange(a, b)** renvoie un entier aléatoire compris entre **a** et **b**-1 inclus (**a** et **b** entiers)
- **random.sample(u, n)** renvoie une liste de **n** éléments distincts de la liste **u** choisis aléatoirement, si **n**>**len(u)**, déclenche l'exception **ValueError**
- **math.sqrt(x)** calcule la racine carrée du nombre **x**
- **math.ceil(x)** renvoie le plus petit entier supérieur ou égal à **x**
- **math.floor(x)** renvoie le plus grand entier inférieur ou égal à **x**

Opérations sur les listes

- **len(u)** donne le nombre d'éléments de la liste **u** : **len([1, 2, 3])** $\to$ **3** ; **len([[1,2], [3,4]])** $\to$ **2**
- **u + v** construit une liste constituée de la concaténation des listes **u** et **v** : **[1, 2] + [3, 4, 5]** $\to$ **[1, 2, 3, 4, 5]**
- **n * u** construit une liste constitué de la liste **u** concaténée **n** fois avec elle-même : **3 * [1, 2]** $\to$ **[1, 2, 1, 2, 1, 2]**
- **e in u** et **e not in u** déterminent si l'objet **e** figure dans la liste **u**
  2 in [1, 2, 3]$\to$**True** ; **2 not in [1, 2, 3]** $\to$ **False**
- **u.append(e)** ajoute l'élément **e** à la fin de la liste **u** (similaire à **u = u + [e]**)
- **u[i], u[j] = u[j], u[i]** permute les éléments d'indice **i** et **j** dans la liste **u**

Import et manipulation de fichiers

- **open('nomfichier.txt','r')** renvoie un objet python qui contient les données du fichier ;
- **pfichier.read()** renvoie une chaine de caractère contenant l'intégralité du fichier texte original. **pfichier** est le fichier python crée avec **open** ;
- **pfichier.readlines()** renvoie une liste dont les éléments sont les différentes lignes du fichier texte original. Les éléments de la liste sont des chaînes de caractères ;
- **chaine.split(',')** renvoie une liste a partir des caractères de **chaine** (de type chaîne de caractère uniquement), en utilisant le séparateur **,**. Par exemple, si **chaine='moi,ici,je suis là'**, alors **chaine.split(,)** renvoie **['moi','ici','je suis là']**.

Tracé de courbes avec matplotlib.pyplot

On suppose que le module est importé avec **import matplolib.pyplot as plt**.

- **plt.plot(x, y, fmt)** prend en argument deux séquences **x**, **y** et une chaîne de caractère de formatage **fmt** optionnelle, et trace des lignes ou des marques selon les axes de coordonnées en utilisant des couples (x, y).

  Par exemple :

```
plot (x, y)         # trace les points de x et y avec le style de ligne et la couleur par défaut
plot (x, y, 'bo')   # trace les points de x et y avec avec des ronds bleus
plot (y)            # trace y en utilisant les indices 0 à N-1 comme abscisses
plot (y, 'r+')      # trace y avec des croix rouges
```

- les options de formatage sont, entre autres, **'-'** ligne continue, **'-'** tirets, **':'** ligne pointillée, **'.'** marque point, **'o'** marque disque, **'*'** marque étoile, **'s'** marque carré, **'d'** marque diamant, ...
- **plt.plot(x1, y1, fmt1, x2, y2, fmt2)** effectue deux (ou plus) tracés avec les groupes de données fournies.
- **plt.plot(x,y, label="courbe 1")** ajoute une étiquette à la courbe (ne sera visible que si la légende du graphique est affichée),

- `plt.legend()` ajoute une légende au graphique
- `plt.xlabel(" x en m")` et `plt.ylabel(" y en m/s")` ajoute les titres correspondants aux axes,
- `plt.title("titre du ...")` ajoute un titre au graphique,
- `plt.grid()` ajoute une grille,
- `plt.axis("equal")` force l'affichage d'un repère orthonormé.