Le **mystère** de la question 7. Il ne faut pas faire une boucle sur `d` et incrémenter le compteur quand `d` est bien le chiffre significatif courant mais utiliser le chiffre significatif courant pour savoir où incrémenter le tableau.

Un peu de traitement de données

I. Bibliothèques et modules

Pour éviter de réinventer la roue et de réécrire sans cesse les mêmes fonctions, on organise en général des fonctionnalités sous forme de *bibliothèque* que l'on peut charger au besoin pour utiliser ces fonctionnalités. Un autre avantage est que la programmation des fonctions peut être ainsi faite une bonne fois pour toutes, de façon efficace et sans erreur.

Ainsi, il existe par exemple une bibliothèque avec des fonctions mathématiques, nommée `math`. À la base, la plupart des fonctions mathématiques ne sont pas définies en Python :

```
>>> cos(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
```

Mais elles sont présentes dans la bibliothèque `math` que l'on peut charger au besoin.

```
>>> import math
>>> math.cos(1)
0.5403023058681398
```

Syntaxe de base pour les modules

Importation de base

```
import math
```

Importation avec renommage

```
import math as mon_super_module
```

C'est plutôt utile pour importer un module avec un long nom (on en rencontrera un plus tard) et l'utiliser avec un identifiant plus court.

Importation ciblée de fonctions On peut aussi faire une importation ciblée de certains fonctions d'un module pour ne pas avoir à faire précéder l'appel à la fonction du nom du module.

```
from math import cos, sin, pi
```

On peut maintenant écrire `cos(2 * pi / 3)` à la place de `math.cos(2 * math.pi / 3)`.

Pour connaître les fonctions définies par un module, un bon moyen est d'utiliser sa documentation en ligne. Celle du module `math` se trouve ici[1][2] :

https://docs.python.org/3/library/math.html.

Il est aussi possible d'obtenir des informations en restant dans la console Python. Ainsi, la commande `dir(math)` renvoie une liste des différentes commandes définies par le module. Pour avoir la description d'une commande spécifique, on utilise la fonction `help`. Ainsi, pour décrire la fonction `math.cos`, on exécutera :

```
help(math.cos)
```

Exercice 1 Trouver une autre fonction définie dans le module `math` (essayez de faire un peu original), afficher sa documentation et essayer de la comprendre.

Dans la suite de ce TP, nous allons utiliser, en plus du module `math`, des modules pour lire des données et pour les représenter graphiquement à l'écran.

II. La loi de Benford

La loi de Benford est une loi de distribution statistique d'apparition des premiers chiffres de données numériques que l'on observe dans de nombreux contextes, de la vie réelle ou en mathématiques.

Elle dit que les fréquences d'apparition des premiers chiffres (ou chiffres significatifs) des données ne suite pas la loi uniforme, mais que la probabilité d'avoir comme chiffre significatif $d \in [\![1,9]\!]$ est :

$$b_d = \log_{10}\left(1+\frac{1}{d}\right)$$

Ainsi, la probabilité qu'une donnée commence par un 1 est de $\log_{10}(2) \simeq 0.30$ mais elle est seulement d'environ 0.046 pour un 9.

Exercice 2 Écrire une fonction `benford` telle que `benford(`d`)` renvoie b_d (on suppose sans le vérifier que $d \in [\![1,9]\!]$).

[1] Le lien du PDF est cliquable.

[2] Si vous cherchez bien, vous pouvez même trouver la documentation en français.

Exercice 3 Écrire une fonction `loi_benford` qui ne prends pas d'arguments et renvoie un tableau t de taille 10 tel que $t[0] = 0$ et $t[d] = b_d$ sinon.

Exercice 4 Montrer à la main – c'est une question de mathématiques – que l'on a bien une probabilité, i.e. que $\sum_{d=1}^{9} b_d = 1$ et même, plus généralement, que

$$\forall\, n \geq 2, \quad \sum_{d=1}^{n-1} \log_n\left(1+\frac{1}{d}\right) = 1$$

En effet, la loi de Benford n'est pas spécifique à la base 10.

Une tentative de justification On a encore du mal à expliquer correctement l'apparition de la loi de Benford dans de nombreux cas. Il en existe un cependant où l'on a une explication satisfaisante : les données financières. Ainsi, si l'on regarde les cotations en bourse, on obtient une certaine répartition des chiffres significatifs. Cependant, cette répartition ne dépend pas *a priori* de la devise utilisée. On devrait donc avoir une répartition similaire si l'on exprime les cotations en euros, en dollars, en yen, etc. Cela revient à avoir une *invariance* de la loi par multiplication (ici, le taux de conversion d'une devise à une autre), et on peut montrer que la loi de Benford est précisément l'unique loi vérifiant cette invariance.

III. Application à des jeux de données

Nous allons maintenant essayer de faire apparaître cette loi. Pour cela, nous allons importer des données à partir d'un fichier pour les traîter. Cela va être l'occasion d'utiliser un nouveau module.

1. Le format csv

Le format `csv` (pour *Comma-Separated Values*) est un format textuel très simple pour représenter des données organisées en tableaux.

Pour chaque ligne du fichier (qui correspond à une ligne du tableau), les données sont représentées à la suite, séparées par des virgules comme son nom l'indique. Notons qu'il existe plusieurs variantes concernant le caractère utilisé pour la séparation, puisque l'on trouve aussi fréquemment des tabulations ou des points-virgules.

De plus, on a parfois la première ligne du fichier utilisée pour indiquer les entêtes des différentes colonnes.

Exercice 5 Ouvrir le fichier `bourse.csv` (par exemple dans *LibreOffice*) et analyser sa structure :

⋆ Combien de colonnes comporte-t'il ?

- ⋆ Quel est le caractère séparateur ?
- ⋆ Y a-t'il une ligne d'entêtes ?
- ⋆ Si oui, quels sont les intitulés des colonnes ?

Pour lire ce fichier en Python, nous allons utiliser la bibliothèque justement nommée `csv`[3].

```
def données_boursières():
    with open("bourse.csv", "r") as fichier:
        données = csv.DictReader(fichier, delimiter=";")
        for ligne in données:
            print(ligne)
            break
```

Si vous exécutez cette fonction[4], vous allez avoir l'affichage suivant :

```
{'Compagnie': '3D SYSTEMS CORP.', 'Cotation':'29.34'}
```

Il s'agit d'un *dictionnaire,* une nouvelle structure de données que nous allons découvrir maintenant.

Question 6 À votre avis, à quoi sert le **break** dans cette fonction ? Que se passe-t'il si vous le supprimez et exécutez à nouveau la fonction ?

2. Dictionnaires

Un *dictionnaire* est une structure de données permettant de stocker des données à l'aide d'association *clé–valeur*. Sa syntaxe Python repose sur l'usage d'accolades, entre lesquels figurent un ensemble d'association notées à l'aide de « : » et séparées par des virgules.

Voici un description rapide des manipulations de base[5].

Utilisation des dictionnaires

Création

```
>>> d1 = {} # création d'un dictionnaire vide.
>>> d2 = dict() # variante pour un dictionnaire vide.
>>> d = {"nom": "Bond", "prénom": "James"}
      # création d'un dictionnaire non vide
      # avec deux paires clé-valeur.
```

Nombre d'associations

```
>>> len(d)
2
```

Valeur associée à une clé

```
>>> d["nom"]
'Bond'
>>> d["matricule"] # clé non présente
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'matricule'
```

Test de la présence d'une clé

```
>>> "nom" in d
True
>>> "matricule" in d
False
```

Modification ou création d'une association

```
>>> d["nom"] = "Flantier" # modification
>>> d["matricule"] = "007" # nouvelle association
>>> d[12] = True
# on n'est pas contraint aux chaînes de caractères
```

Parcours par clés

[3] Je vous laisse trouver la documentation.
[4] Faites-le, après vous être assuré que le module `csv` est bien importé !
[5] N'hésitez pas à exécuter vous-même ces commandes, et en essayer d'autres.

```
>>> d # on en est là
{'nom': 'Flantier', 'prénom': 'James',
 'matricule': '007', 12: True}
>>> for c in d: # parcours par clé
...     print(c)
...
nom
prénom
matricule
12
```

On peut aussi écrire `for c in d.keys(): ...` mais c'est plus long (et pas très utile).

On peut aussi faire un parcours par valeurs en utilisant `d.values()` mais on perd la clé correspondant.

3. Retour sur la lecture des données

La fonction `données_boursières` vue plus tôt a, une fois le **break** supprimé, la structure suivante :

```
def fonction():
    ... # initialisation
    with open("fichier.csv", "r") as fichier:
        données = csv.DictReader(fichier, delimiter=";")
        for ligne in données:
            ... # traitement de la ligne courante
    ... # fin de la fonction
```

Par exemple, une fonction qui compte le nombre de lignes (hors entête) pourrait s'écrire ainsi :

```
def nombre_de_lignes(nom_de_fichier):
    compte = 0
    with open(nom_de_fichier, "r") as fichier:
        données = csv.DictReader(fichier)
        for ligne in données:
            compte = compte + 1
    return compte
```

On a défini dans le fichier `tp3.py` une fonction `chiffre_significatif`, qui est utilisable et dont la description est accessible en exécutant la commande :

```
help(chiffre_significatif)
```

comme expliqué précedemment.

Exercice 7 Écrire une fonction `analyse_boursière` qui envoie un tableau t de taille 10 telle que $t[d]$ est égal aux nombre de lignes du fichier pour lesquelles le chiffre significatif de la cotation est d.

Exercice 8 Transformer la fonction précédente pour qu'elle renvoie non pas le tableau contenant le nombre d'occurences de chaque chiffre significatif, mais leur pourcentage.

4. Représentation des données

Nous allons utiliser la bibliothèque `matplotlib` et plus particulièrement le module `pyplot` de cette bibliothèque que l'on renommera `plt`. L'import à utiliser est donc :

```
import matplotlib.pyplot as plt
```

Il s'agit d'une bibliothèque très riche, et nous allons seulement en égratigner la surface. Voici un survol de quelques commandes de base[6].

Quelques commandes de `matplotlib`

Tracé simple On ne spécifie que les ordonnées.

```
>>> plt.plot([1, 3, 2, 3, 1])
```

Tracé simple, deux On spécifie les abscisses et les ordonnées.

```
>>> plt.plot([0, 2, 4, 2, 0], [0, 1, 0, -1, 0])
```

Affichage du tracé

```
>>> plt.show()
```

Tracé de fonction

[6] Bien sûr, je vous conseille de les essayer.

```
>>> xs = [k / 10 for k in range(41)]
>>> ys = [cos(x) for x in xs]
>>> plt.plot(xs, ys)
>>> plt.show()
```

Variantes

```
>>> plt.bar(['a', 'b', 'c'], [3, 1, 2])
<BarContainer object of 3 artists>
>>> plt.plot([1, 2, 3],"go") # g pour green, o pour rond
[<matplotlib.lines.Line2D object at 0x11df02fa0>]
>>> plt.show()
```

Lorsque vous trouverez un peu de temps, vous pouvez le tutoriel du site de la bibliothèque.

Exercice 9 Représenter sur la même figure, représentez la loi de Benford et la loi obtenue à partir des données boursières.

5. Un autre jeu de données

Le fichier `communes.csv` compile le recensement de populations des communes de France, obtenue sur le site de l'INSEE.

Exercice 10 Reprenez les étapes précédentes, et calculer la loi des chiffres significatifs des populations totales des communes, et représenter la loi sur le même graphique que la loi de Benford.

IV. Exercices complémentaires

1. Utilisation des dictionnaires

Exercice 11 À partir des données du recensement, déterminer...
- ⋆ pour chaque département, le nom de la commune la plus peuplée ;
- ⋆ pour chaque région, la liste des codes des départements la constituant ;
- ⋆ la population totale de chaque région.

À chaque fois, pour remplir le dictionnaire, il faudra s'assurer que la clé existe bien (cela correspond, pour un tableau, à vérifier que l'indice est correct).

2. Une mesure de similitude

Pour mesurer l'adéquation d'une loi à la loi de Benford, la représentation graphique est une aide mais ne permet pas de quantifier cette adéquation. Pour faire cela, on peut utiliser l'entropie relative ou divergence de Kullback-Leibler.

Étant donné deux lois $(p_i)_{i\in[\![1,n]\!]}$ et $(q_i)_{i\in[\![1,n]\!]}$, cette divergence est définie comme

$$D(p|q) = \sum_{i=1}^{n} p_i \ln\left(\frac{p_i}{q_i}\right)$$

Il est clair que si $p = q$, alors $D(p|q) = 0$. On peut aussi montrer que l'on a toujours $D(p|q) \geq 0$ (mais vous avez besoin d'un outil nommé *convexité* pour faire cela). Ainsi, plus la divergence est basse et proche de 0, meilleur est la similitude.

Question 12 Écrire une fonction qui calcule cette divergence étant donné deux tableaux de même taille représentant les lois à comparer.

Question 13 Calculer pour les deux lois obtenus à partir des jeux de données précédents, leur divergence par rapport à la loi de Benford (en utilisant la loi de Benford comme loi q à laquelle on compare la loi p obtenue à partir des données statistiques).

Question 14 Sachant que les données boursières suivent la loi de Benford, comme on l'a expliqué plus tôt, peut-on dire que la loi obtenue à partir des données de rencensement suit la loi de Benford ?