TD 12

FICHIERS ET CHAÎNES – PREMIÈRE FOURNÉE

1 Manipulation de chaînes de caractères

On rappelle quelques fonctions utiles agissant sur les chaînes de caractères. Dans tout ce qui suit, `s` est une chaîne de caractère.

- `s.rstrip()` renvoie la chaîne `s` dans laquelle on a supprimé tous les caractères de *white space* situés à la fin de la chaîne : `"toto  \t \r \n\n".rstrip()` renvoie `"toto"`.
- `s.split(t)` renvoie une liste constituée des morceaux de `s` délimités par les occurrences de `t`. Sur des exemples :

```
>>> "12,27,34".split(",")
['12', '27', '34']
>>> "132 23 baba".split(" ")
['132', '23', 'baba']
>>> "132 ,23 baba".split(" ")
['132', ',23', 'baba']
```

- Si `liste` est une liste de chaînes de caractères, `s.join(liste)` renvoie la chaîne formée des éléments de `liste` mis bout à bout, avec une copie de `s` intercalé entre deux éléments de `liste` successifs :

```
>>> m = ["a", "bb", "ccc"]
>>> " ".join(m)
'a bb ccc'
>>> ",".join(m)
'a,bb,ccc'
>>> "XYZ".join(m)
'aXYZbbXYZccc'
>>> "".join(m)
'abbccc'
```

  On remarquera que `"".join(liste)` renvoie la concaténation des éléments de `liste`.
- `int(s)` convertit une chaîne en entier (pour peu que ça ait un sens et `float(s)` la convertit en «flottant» (nombre à virgule).

EXERCICE 1

1. *Écrire une fonction `somme_liste(l)` qui calcule la somme des éléments d'une liste.*
2. *Écrire une fonction `somme_chaine(s)` qui prend une chaîne comme `"12 3.2 17.5"` (des nombres contenant éventuellement un point comme séparateur décimal et séparés par un espace) et renvoie (dans ce cas)* 32,7.
3. *Modifier cette fonction pour que l'on puisse spécifier le séparateur et pour qu'elle ignore les caractères «blancs» à la fin de la chaîne.*

```
>>> somme_chaine("32.3,3,1.4\n", ",")
36.699999999999996
>>> somme_chaine("32.3 3 1.4", " ")
36.699999999999996
```

4. *Écrire une fonction `affiche_matrice(M)` qui prend une matrice M sous forme de liste de listes et l'affiche de manière (à peu près) lisible. On ne cherchera pas à faire trop raffiné :*

```
>>> M = [[1, 23, 10], [22, 1, -8.1]]
>>> affiche_matrice(M)
1 23 10
22 1 -8.1
```

suffit largement. On peut bien sûr aligner les colonnes, mais c'est plus compliqué et inintéressant.

```
>>> M = [[1, 23, 10], [22, 1, -8.1], [123, -1, 9], [1, 2, 19090801]]
>>> pprint_matrice(M)
  1 23        10
 22  1      -8.1
123 -1         9
  1  2 19090801
```

2 Lecture d'un fichier

EXERCICE 2

1. *Repérer le répertoire dans lequel se trouve votre fichier* `TD12.py`.
2. *Récupérer sur mon site (*i.e. `http://bianquis.org`*) le fichier* `premiers.txt` *et le sauvegarder dans le même répertoire.*
3. *Ouvrir le fichier* `premiers.txt` *à l'aide d'un éditeur de texte (le bloc-notes de Windows fera l'affaire, n'utilisez surtout pas un traitement de texte comme Word ou LibreOffice).*
4. *Observez la structure des informations : comment les données sont-elles séparées, à quelle ligne commencent-elles ?*

EXERCICE 3

1. *Calculer avec Python le nombre de lignes et le nombre de caractères du fichier* `premiers.txt`.
2. *Calculer la somme des nombres premiers inférieurs à* 10^5.
3. *Calculer le nombre de nombres premiers inférieurs à* 10 000.

EXERCICE 4

1. *Créer une liste* `liste_premiers` *qui contient les nombres premiers inférieurs à* 10^5.
   On doit donc avoir
   `liste_premiers[:10] = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]`
   et
   `liste_premiers[-10:] = [99877, 99881, 99901, 99907, 99923, 99929, 99961, 99971, 99989, 99991]`.
2. *Créer une liste* `est_premier` *indexée de* 0 *à* 10^5 *exclu telle que* `est_premier[n]` *contienne* `True` *si n est premier,* `False` *sinon.*
   On doit donc avoir
   `est_premier[:10] = [False, False, True, True, False, True, False, True, False, False]`
   et
   `est_premier[-10:] = [False, True, False, False, False, False, False, False, False, False]`.
3. *En utilisant ces deux listes, écrire une fonction* `test_primalite(n)` *qui prend un entier n et renvoie* `True` *s'il est premier,* `False` *sinon. Cette fonction devra avoir le comportement suivant :*
   - *si* $n < 10^5$*, elle renvoie un résultat (correct) «immédiatement» ;*
   - *si* $10^5 \leqslant n < 10^{10}$ *elle renvoie un résultat (correct) en un temps raisonnable (ça devrait avoir l'air d'être instantané à l'œil nu) ;*
   - *si* $n \geqslant 10^{10}$*, elle renvoie* `False` *ou un message dans lequel elle avoue son ignorance.*

EXERCICE 5 **Théorème des nombres premiers**

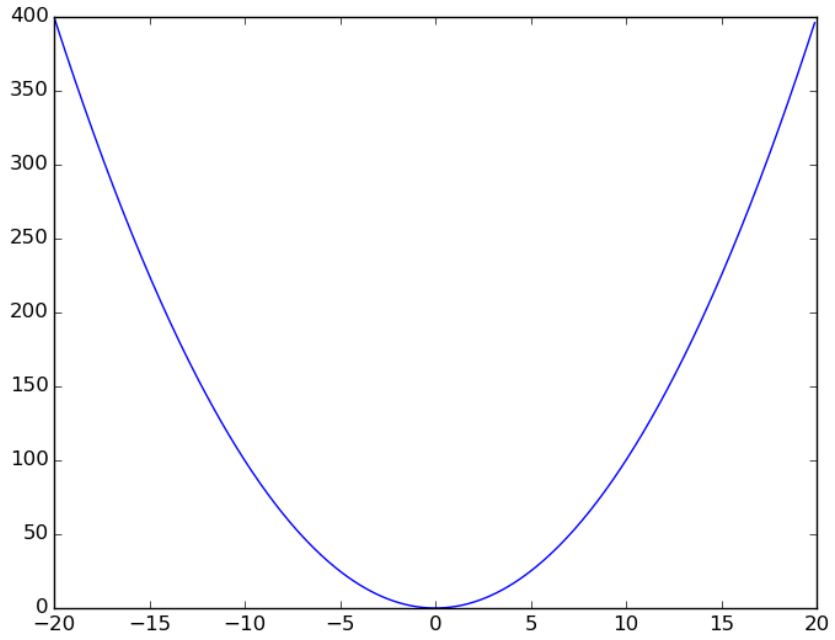
Pour $n \in \mathbb{N}$*, on définit* $\pi(n)$ *comme le nombre de nombres premiers inférieurs ou égaux à n. On remarquera que, si* $n > 0$*, la quantité* $\frac{\pi(n)}{n}$ *représente la proportion des nombres de* $[\![1, n]\!]$ *qui sont premiers.*

1. *Calculer la liste* `pi` *de longueur* 10^5 *telle que* `pi[k]` *soit égal à* $\pi(k)$*. Attention, si vous recommencez le calcul depuis le début pour chaque k, cela va prendre longtemps…*
2. *Le* théorème des nombres premiers *affirme que* $\frac{\pi(n)}{n} \underset{n\to+\infty}{\sim} \frac{1}{\ln n}$*. Pour visualiser ce phénomène, on souhaite tracer la courbe des* $\frac{\pi(n)}{n \ln n}$ *pour n variant de* 1 *à* 10^5.
   Pour cela, nous allons utiliser le module `pyplot` *de la bibliothèque* `matplotlib`*. Pour y avoir accès (ainsi qu'à la fonction* ln*), il faut rajouter les lignes suivantes en entête de fichier :*

```
import matplotlib.pyplot as plt
from math import log
```

On dispose ainsi d'une fonction `plt.plot(X, Y)` *qui prend en entrée une liste X d'abscisses et une liste Y d'ordonnées (qui doivent être de même longueur) et trace la courbe reliant tous les points* (x_i, y_i)*. Par exemple :*

```
X = [-20 + .1 * x for x in range(400)]
Y = [x**2 for x in X]
plt.plot(X, Y)
plt.show()
# Les trois premières lignes créent le graphique, la quatrième l'affiche.
```



Faire afficher le graphique de la fonction $n \mapsto \frac{\pi(n)\ln n}{n}$ *jusqu'à* $n = 10^5$ *(en prenant quelques centaines de points) :*