Chapitre # (ALGO) 6 — Fichiers : lecture & écriture

Résumé & Plan

L'objectif de ce chapitre est de présenter brièvement comment importer et exporter des données extérieures dans son fichier Python principal ou inversement. Ces données peuvent être dans un autre fichier `.py`, ou bien un fichier texte `.txt` ou encore une base donnée contenue dans un fichier `.csv`.

Les structures de données Python que nous avons vues (chaînes, listes, tuples, dictionnaires) peuvent être utilisées au sein d'un programme, mais ne permettent pas de stocker durablement des données. Elles n'ont d'existence que dans la mémoire vive (ou RAM) de l'ordinateur, et à la fermeture de python, les données contenues dans ces variables sont perdues. Pour stocker ces données de manière durable (c'est à dire sur un disque dur, une clé USB ...), il faut utiliser une structure largement utilisée en informatique, dite de fichier.

Exercice 1 | Préparation des fichiers Solution

1. Récupérer tous les fichiers externes nécessaires à ce TP dans le répertoire partagé de la classe. Ils sont disponibles dans le dossier intitulé `TP_Fichiers`.
2. Les placer dans un dossier dans votre propre répertoire, et y enregistrer votre fichier Python `py` habituel.
3. Ouvrez les différents fichiers externes directement sur l'ordinateur, et observez leur contenu.

Attention

Il est impératif que tous vos fichiers soient au même endroit, afin qu'ils puissent ensuite être lus.

1. FICHIER .PY : IMPORTATION DE MODULE

Si l'on souhaite importer les données contenus dans un fichier python auxiliaire `fichier_aux.py` contenant :

```
import numpy as np
CHAINE = "bonjour les BCPST1"
```

dans un fichier principal, on place le fichier `fichier_aux.py` dans le même dossier que le fichier principal et on tape dans le fichier principal :

```
>>> from fichier_aux import *
```

Dans le fichier principal (éditeur ou console), on voit que tout a été correctement importé.

```
>>> np.sin # le module numpy a donc correctement été importé par \
↪ la même occasion
<ufunc 'sin'>
>>> CHAINE # la variable CHAINE existe bien dans le fichier \
↪ principal
'bonjour les BCPST1'
```

2. FICHIER .TXT

Un traitement sur un fichier se déroule toujours en trois temps :

- l'ouverture du fichier,
- le traitement proprement dit,
- la fermeture du fichier.

Il existe trois types d'ouverture de fichiers texte, chacun se faisant par l'intermédiaire de l'instruction suivante :

```
open('NomDeFichier.extension','option')
```

Le choix de l'option permet :

- l'ouverture en **lecture** (suppose que le fichier existe déjà), avec l'option `'r'` (pour read) ;
- l'ouverture en **écriture** (pour créer un nouveau fichier ou écraser un fichier existant), avec l'option `'w'` (pour write) ;
- l'ouverture en **ajout** (permet d'ajouter du texte à la fin d'un fichier existant), avec l'option `'a'` (pour append) ;

Dans tous les cas, `open('NomDeFichier.extension','option')` renvoie un objet de type fichier, qu'il convient d'affecter dans une variable du programme.

2.1. Lecture

On suppose qu'un fichier `texte.txt` existe (pour simplifier placé dans le même répertoire que le script python que nous allons utiliser, sans quoi il faudrait indiquer son chemin absolu) et nous souhaitons extraire son contenu. Nous allons utiliser l'option `'r'` pour un accès en lecture.

Exemple 1 On souhaite ici importer le contenu de `texte.txt`, contenant :

```
283 133 47 30 21 16 13 : températures année 1
1000 855 798 760 732 710 692 : températures année 2
```

On procède alors comme ci-après.

```
>>> fichier = open("texte.txt", "r")
>>> fichier.read() # convertit le tout en une grosse chaîne, \
↪ on affiche ici que les 55 premiers caractères
'283 133 47 30 21 16 13 : températures année 1\n1000 855 798 76
0 732 710 692 : températures année 2'
>>> fichier.close()
```

Les sauts de ligne sont codés par la chaîne `\n`, mais n'apparaissent pas comme des sauts dans la console. On peut aussi préférer lire le fichier ligne par ligne :

```
>>> fichier = open("texte.txt", "r")
>>> ligne_1 = fichier.readline()
>>> ligne_1
'283 133 47 30 21 16 13 : températures année 1\n'
>>> ligne_2 = fichier.readline() # on passe à la ligne suivante
>>> ligne_2
'1000 855 798 760 732 710 692 : températures année 2'
>>> fichier.close()
```

On peut également parcourir tout le fichier à l'aide d'une boucle `for`.

```
>>> fichier = open("texte.txt", "r")
>>> for ligne in fichier:
...     print(ligne)
...
283 133 47 30 21 16 13 : températures année 1

1000 855 798 760 732 710 692 : températures année 2
>>> fichier.close()
```

On peut également lire toutes les lignes d'un coup, et créer ainsi une liste des lignes.

```
>>> fichier = open("texte.txt", "r")
>>> lignes = fichier.readlines()
>>> lignes
['283 133 47 30 21 16 13 : températures année 1\n', '1000 855 7
98 760 732 710 692 : températures année 2']
>>> fichier.close()
```

Exercice 2 | Séquences d'ADN Solution Un brin d'ADN peut se représenter par un mot (en langage Python : une chaine de caractère) créé à partir d'un alphabet codant des molécules ordonnées dans un certain ordre (correspondant aux nucléotides). Dans les exemples qui suivent, les molécules considérées seront l'adenine (A), la cytosine (C), la guanine (G) et la thymine (T). Par exemple : *AGCTTTTCATTCTGACTGCAACGGGCAATATGTC* est un brin d'ADN.

1. Ouvrir (double clic) dans un éditeur de texte le fichier `hamming.txt`, observer son contenu (il contient deux lignes, chacune comportant une chaine de caractères correspondant à un brin d'ADN) puis le fermer.
2. Dans l'éditeur, écrire quelques lignes de code permettant d'ouvrir le fichier `hamming.txt` puis de stocker son contenu dans deux variables de type chaine de caractères qu'on appellera `brin_1` et `brin_2`.
3. Écrire à la suite du programme une fonction `repartition` qui prend un argument une chaine de caractères `brin` et qui compte et renvoie le nombre de A, le nombre de C, le nombre de G et le nombre de T composant `brin`. Tester la fonction sur `brin_1` et `brin_2`. Notez le résultat :

4. La « teneur en GC » d'un brin d'ADN est le pourcentage de symboles G et C dans la chaine de caractère correspondante. Écrire une fonction `GCcontent` qui prend en argument une chaine de caractères brin et qui renvoie la teneur en GC de `brin`. Tester la fonction sur `brin_1` et `brin_2`. Notez le résultat :

5. Une « mutation » est une erreur qui se produit lors de la création ou de la copie d'un brin d'ADN. La mutation la plus simple que l'on puisse considérer est celle où une base est remplacée par une autre. On cherche à déterminer le nombre de mutations différenciant deux brins d'ADN.
Étant donné deux chaines de caractères s et t de même longueur, la distance de HAMMNING entre s et t, notée $d_H(s,t)$, correspond au nombre de lettres qui diffèrent dans s et t. Par exemple, si `s = GAGCCTA` et `t = CATCGTA`, $d_H(s,t) = 3$.
Écrire dans le même script une fonction `Hamming` en langage Python qui prend en arguments deux chaines de caractères `s` et `t` et qui renvoie la distance de Hamming $d_H(s,t)$. Testez `Hamming(brin_1, brin_2)` et notez le résultat :

Exercice 3 | **Hauteur de vagues en fonction du temps** Solution Le fichier `hauteurs.txt` contient une seule ligne sur laquelle figurent les hauteurs des vagues, en mètres, mesurées par la balise maritime irlandaise M4 tous les jours à 11 h du 15 avril 2003 au 22 novembre 2018. Les données sont séparées par un espace.

1. Écrire une fonction `extraction1()` sans argument qui renvoie la liste des nombres écrits dans le fichier hauteurs `.txt`. *On pourra se servir de la méthode `split` sur les chaînes dont nous rappelons un exemple d'utilisation :*

```
>>> c = 'Les BCPST1 vous êtes toujours là ?'
>>> liste_mots = c.split()
>>> liste_mots
['Les', 'BCPST1', 'vous', 'êtes', 'toujours', 'là', '?']
```

2. Écrire une fonction `vagues()` qui affiche un graphique représentant l'évolution de la hauteur des vagues de 2003 à 2018. Quelle est la hauteur de la plus grande vague mesurée durant cette période? *On demande à ce que Python effectue ce calcul de maximum, pour le graphique on prévoira titre et légende*

2.2. Écriture

On peut aussi créer un nouveau fichier, et écrire du contenu dessus.

Commençons par créer un fichier texte en utilisant un script python. Pour que le fichier apparaissent dans le même dossier que celui dans lequel vous aurez sauvegardé le script, il faudra exécuter celui en choisissant dans le menu pyzo « Exécuter » puis « Démarrer le script » (raccourcis « Ctrl+Maj+E »).

Exemple 2 Pour créer un fichier texte de nom `texte2.txt`, on procède ainsi :

```
>>> Contenu = "Ceci est un exemple de fichier texte."
>>> fichier = open("texte2.txt","w")
>>> fichier.write(Contenu)
37
>>> fichier.close()
```

Le contenu du fichier apparaît ici sur une seule ligne. Pour séparer le texte sur plusieurs lignes, on insère le caractère spécial « `"\n"` » (bien que formé de deux signes, l'ensemble est considéré comme un seul caractère par Python) indiquant à l'ordinateur de revenir à la ligne à cet endroit. Par exemple :

```
>>> Contenu = "Ceci est un exemple\\n de fichier texte."
>>> fichier = open("texte3.txt","w")
>>> fichier.write(Contenu)
39
>>> fichier.close()
```

On peut aussi écrire plusieurs lignes d'un coup.

```
>>> Contenu = ["Ceci est un exemple \n", "de fichier texte."]
>>> fichier = open("texte4.txt","w")
>>> fichier.writelines(Contenu)
>>> fichier.close()
```

Exercice 4 | **Fichier de notes (écriture)** Solution On désire écrire une fonction d'entête `listeClasse(prenoms)` dont le but est de créer un fichier `notes.txt` contenant les prénoms de la liste `prenoms` ainsi que les notes obtenues à un devoir. On se limitera à un nombre faible d'élèves, on créera alors une liste de prénoms à utiliser. Par exemple,

```
prenoms = ["Ghislaine", "Georgette", "Alphonse", "Jean-Eude"]
```

Pour générer une note entre `0` et `20`, on procèdera ainsi :

```
>>> import random as rd
>>> round(20*rd.random(), 1)
```

```
16.5
```

Écrire la fonction `listeClasse` qui crée le fichier `notes.txt`, en assignant une note au hasard à chaque élève présent dans la liste `prenoms`. Le fichier devra être de la forme

```
prenom1 note1
prenom2 note2
...
```

Exercice 5 | Fichier de notes (exploitation) Solution On suppose construit le fichier de notes de l'exercice précédent.

1. Écrire une fonction d'en-tête `liste_notes()` qui lit le fichier `notes.txt` et renvoie une liste de listes de la forme `[[prenom1, note1], [prenom2, note2]]`.
   Indication : On pourra se servir à nouveau de la méthode `split` sur les chaînes
2. Dans cette question, on suppose que la liste renvoie par `liste_notes` est stockées dans une liste appelée `notes`.
   Écrire une fonction d'en-tête `stats(L)` qui à partir d'une liste `L` de même forme que `notes`, renvoie :
   - un couple de première coordonnée le prénom de l'élève ayant eu la plus petite note, et la note en question,
   - la moyenne,
   - un second couple de première coordonnée le prénom de l'élève ayant eu la plus grande note, et la note en question.

3. FICHIER .CSV

Pour échanger, partager et analyser les données collectées lors d'une expérience, elles doivent être enregistrées dans un fichier au format informatique ouvert. Le plus utilisé est le format de fichier CSV (**C**omma-**S**eparated **V**alues).

Le module `csv` contient des outils permettant de lire et manipuler des fichiers dont le format est `CSV`. On l'importe de la manière classique suivante.

```
>>> import csv
```

Lors du premier exercice, vous avez normalement récupéré un gros fichier de données au format `csv`.

On commence par ouvrir le fichier en mode lecture, puis on le décode en précisant le délimiteur (le symbole séparant chaque ligne de la base de données).

```
fichier = open('nomdufichier.csv', "r")
data = csv.reader(fichier, delimiter=";")
```

Dans notre cas :

```
>>> fichier = open('bor_arbres.csv', "r")
>>> data = csv.reader(fichier, delimiter=";")
```

On constate que `data` possède un type particulier.

```
>>> type(data)
<class '_csv.reader'>
```

Mais c'est un type itérable, au sens où l'on peut parcourir chacune des lignes à l'aide d'une simple boucle **`for`**. Dans l'exemple qui suit, on l'arrête au bout de 3 itérations, car la base est très grande.

```
>>> N = 0
>>> for ligne in data:
...     print(ligne)
...     N += 1
...     if N == 3:
...         break
...
['Geo Point', 'Geo Shape', 'gid', 'geom_o', 'geom_err', 'localisat
ion', 'tranche_age', 'nom', 'circonference', 'hauteur', 'typo_espa
ce', 'statut', 'diametre', 'famille', 'genre', 'variete', 'geograp
hie', 'date_plantation', 'cdate', 'mdate', 'age_tranche_basse']
['44.8451436,-0.5719386', '{"type": "Point", "coordinates": [-0.57
19386, 44.8451436]}', '3034', '0', '', 'des Quinconces (place)', '
49 - 74', 'Platanus x hispanica', '147', '20', 'ESPACE_PUBLIC_VEGE
TALISE', 'VIVANT', '47', 'Platanaceae', 'Platanus sp.', '', 'Hybri
de Platanus orientalis x P. occidentalis', '', '2019-04-02T00:00:0
0+02:00', '2019-04-02T00:00:00+02:00', '49']
['44.8451677,-0.5716563', '{"type": "Point", "coordinates": [-0.57
16563, 44.8451677]}', '3036', '0', '', 'des Quinconces (place)', '
83 - 124', 'Platanus x hispanica', '248', '21', 'ESPACE_PUBLIC_VEG
ETALISE', 'VIVANT', '79', 'Platanaceae', 'Platanus sp.', '', 'Hybr
ide Platanus orientalis x P. occidentalis', '', '2021-05-20T00:00:
00+02:00', '2021-05-20T00:00:00+02:00', '83']
```

La première ligne correspond donc aux champs de la base, le premier champ est donc le lieu géographique de plantation. Si l'on souhaite par exemple afficher l'ensemble des coordonnées uniquement, on fera :

```
>>> N = 0
```

```
>>> for ligne in data:
...     print(ligne[0])
...     N += 1
...     if N == 3:
...         break
```

Une fois nos données récupérées, on ferme le fichier.

```
>>> fichier.close()
```

4. POUR ALLER PLUS LOIN

Cette partie est à aborder en toute fin de TP uniquement, quand tout le reste aura été réussi. Elle est optionnelle.

4.1. Arbres de la ville de Bordeaux

Exercice 6 | Dictionnaire pour le traitement d'un jeu de données Solution On reconsidère la base de données de `https://opendata.bordeaux-metropole.fr` importée de la façon suivante (sous forme de dictionnaire) :

```
>>> import csv
>>> fichier = open('bor_arbres.csv', "r", encoding='utf-8')
>>> reader = csv.DictReader(fichier,delimiter=";")
>>> arbres = []
>>> for ligne in reader:
...     arbres.append(dict(ligne))
...
>>> fichier.close()
```

On peut alors afficher par exemple deux éléments de la variable `arbres`.

```
>>> arbres[0]
{'Geo Point': '44.8451436,-0.5719386', 'Geo Shape': '{"type": "Poi
nt", "coordinates": [-0.5719386, 44.8451436]}', 'gid': '3034', 'ge
om_o': '0', 'geom_err': '', 'localisation': 'des Quinconces (place
)', 'tranche_age': '49 - 74', 'nom': 'Platanus x hispanica', 'circ
onference': '147', 'hauteur': '20', 'typo_espace': 'ESPACE_PUBLIC_
VEGETALISE', 'statut': 'VIVANT', 'diametre': '47', 'famille': 'Pla
tanaceae', 'genre': 'Platanus sp.', 'variete': '', 'geographie': '
Hybride Platanus orientalis x P. occidentalis', 'date_plantation
': '', 'cdate': '2019-04-02T00:00:00+02:00', 'mdate': '2019-04-02T
00:00:00+02:00', 'age_tranche_basse': '49'}
>>> arbres[1440]
{'Geo Point': '44.8504672,-0.5610435', 'Geo Shape': '{"type": "Poi
nt", "coordinates": [-0.5610435, 44.8504672]}', 'gid': '36086', 'g
eom_o': '0', 'geom_err': '', 'localisation': 'Parc aux Angéliques_
E', 'tranche_age': '10', 'nom': 'Prunus avium', 'circonference': '
37', 'hauteur': '8', 'typo_espace': 'PARC_JARDIN_SQUARE', 'statut
': 'FOSSE', 'diametre': '12', 'famille': 'Rosaceae', 'genre': 'Pru
nus sp.', 'variete': '', 'geographie': 'Moyen-Orient, Asie du Sud
Ouest', 'date_plantation': '2011-12-01', 'cdate': '2019-01-02T00:0
0:00+01:00', 'mdate': '2019-01-02T00:00:00+01:00', 'age_tranche_ba
sse': '10'}
```

On répondra aux questions ci-après à l'aide de Python, sous forme d'une fonction ou non en fonction du besoin.

1. Quel est le type de la variable `arbres` ? Quelle est la nature des éléments de `arbres` ?
2. Combien y-a-t-il d'arbres répertoriés à Bordeaux ?
3. Écrire une fonction d'en-tête `liste_localisations()` sans argument qui renvoie la liste des localisations possibles des arbres. Combien y'en-a-t-il ?
4. Combien y-a-t-il d'arbres répertoriés au parc aux angéliques ? *On répondra à l'aide de la longueur d'une liste par compréhension, et de `localisation`.*
5. Combien y-a-t-il d'arbres répertoriés dans les parcs, jardins ou square ? *On répondra à l'aide d'une liste définie par compréhension, et de `typo_espace`.*
6. Combien y-a-t-il d'arbres répertoriés de hauteur supérieure ou égale à 10m ? *On répondra à l'aide d'une liste définie par compréhension, et de `hauteur`.*
7. Comme vous l'avez peut-être remarqué, les dates de plantation du champs `date_plantation` sont au format `aaaa-mm-jj` où `aaaa` représente l'année, `mm` représente le mois et `jj` représente le jour.

   7.1) Observez le code suivant :

   ```
   >>> a = "2017-04-01"
   >>> t = a.split('-')
   ```

```
>>> t
['2017', '04', '01']
>>> t[0]
'2017'
```

Que permet de faire la méthode `split()` sur les chaînes ?

7.2) Construire une liste des arbres plantés en 2017 ou après. Combien y'en a-t-il ? *On répondra à l'aide d'une liste définie par compréhension*

4.2. Classement à un concours

On dispose d'un fichier `Notes_candidats.csv`. Ce fichier contient les notes de candidats au concours Agro-Véto, les notes correspondent aux épreuves dans l'ordre suivant :

```
Bio Synthèse,Modélisation(Maths),RésProb
↪ (Physique),Calculs_raisonnements(Maths),SVT
↪ documents,Français,AnalyseDoc(Physique),LV
```

On consultera l'en-tête du fichier `csv`. Par ailleurs, une école donnée décide d'affecter chaque note d'un coefficient et calcule ainsi une moyenne par candidat. Pour l'exemple, on choisira les coefficients suivants (qui correspondent ici à la banque des écoles Agro, comme AGRO PARISTECH).

```
COEFFS = [4, 4, 4, 4, 4, 4 ,4, 3]
```

L'école en question ne peut accueillir que 5 candidats (concours sélectif) et on souhaite ici renvoier la liste des heureux(ses) élu(e)s.

Exercice 7 | Solution

1. Recopier et compléter le code suivant pour qu'il crée un dictionnaire de clefs les candidats, et valeurs une liste de notes obtenues au concours.

```
import csv

def dico_notes():
    """
    renvoie un dictionnaire avec comme clefs les candidats et \
    ↪ comme valeurs
    les notes par matière, dans le même ordre que dans la \
    ↪ liste COEFFS
    """
    fichier_notescand = open('Notes_candidats.csv' , "r")
    NOTES = csv.reader(fichier_notescand, delimiter = ",")
    dico_notescand = {}
    for L in NOTES[1:]:
        candidat = _________________
        notes = _________________
        if candidat != "Nom":
            dico_notescand[_________________] = _________________
    fichier_notescand.close()
    return dico_notescand
```

Dans la suite on stockera le résultat de cette fonction dans une variable `dico`, en faisant `dico = dico_notes()`.

2. Écrire une fonction d'en-tête `liste_moy()` et qui renvoie une liste de couples de la forme `(candidat, moyenne_concours)` où `candidat` est le nom du candidat, et `moyenne_concours` sa moyenne.
3. En déduire une fonction d'en-tête `classement()` qui renvoie la liste précédente, mais où les couples sont triés par ordre croissant de notes.
4. Déterminer alors les candidats admis dans l'école.
5. Le premier non-admis : combien de points lui manquait-il ?

5. SOLUTIONS DES EXERCICES

Solution (exercice 1) Énoncé

Solution (exercice 2) Énoncé

1. RAS

2.
```
>>> fichier = open("hamming.txt", "r")
>>> brin_1 = fichier.readline()
>>> brin_1
'GCCGGCATCCCAATCGACATGTGACAGGGTTACTGAGATAGTATGATCTTGAACATAAC
GGAGAATTCAATCGACCAATTGCCTGGTCCCAGTACGACACGGAAACGTGTTATCCCCGT
CTAAAGCGAAGCGCCCACCTTGGTACCATGCCAGCTGGTCTTCCTAACTAGGACGGTCAT
TTGCCCAAGGAACCCCTCTAGATGGATCATATGTATTTAAGTTTCATGGGCAGGTCGGTA
ACGCAGCCTAGTCACTCCAGGAAGTCTATGTACTTTAAGTTGAATCGCCCATCTCGATTA
CCGGATTTCGCCTGTCTTACGCTTTGGCTCATCTCGTGTCAAGCCATGCCGGCACACTCT
GTGCTGTGCGGATTGTATGATCGGGGAATCCCTTGTCAAAATGACCGCTATATGCGCCAT
GCCCTAGCCCAAGAGAGCCCCGGCTATGTTGGCGATACTCTTGCGTCACAAACGTAGTAA
GTGTTGCAGGGGGCTGGCATCGTGTGGGCACTGGAATACGTAGATGGCTGATCCCAGTTC
TGGCACGCAGCTCTATCACGTGTCTTAGCCTAGGACCCGGAACGCAAGGTAGGATAAATA
TCCGGCGATCAAGACCCAGCGTCGATGGTTCGAACCGCAGGTATACTGGATTGTCCGTAG
AATTTCATCCGCCCACGAAATAGCATCAGCTACGTTAGGGTAGAGCATCCAGGTGTGTAA
AATGTAGTGTTGGTGTATAATTCGTGCCTTCGGCATGGACAAGTGGAATTGCGGGGGTCC
GTGGCTGAGCGCAATGTAACAGTGATGTCGAGGGGCTACGAGCCTTTCCCCCCGGTTTAT
CTAGTCAATGCGGAGAAGCAACGGACTCTGTAATGGGTGGGGTAGCGTGAGGTCATAAAG
TAGCGTGCAATAATTACACCCAATTA\n'
>>> brin_1 = brin_1[:-1] # suppression du dernier caractère
>>> brin_1
'GCCGGCATCCCAATCGACATGTGACAGGGTTACTGAGATAGTATGATCTTGAACATAAC
GGAGAATTCAATCGACCAATTGCCTGGTCCCAGTACGACACGGAAACGTGTTATCCCCGT
CTAAAGCGAAGCGCCCACCTTGGTACCATGCCAGCTGGTCTTCCTAACTAGGACGGTCAT
TTGCCCAAGGAACCCCTCTAGATGGATCATATGTATTTAAGTTTCATGGGCAGGTCGGTA
ACGCAGCCTAGTCACTCCAGGAAGTCTATGTACTTTAAGTTGAATCGCCCATCTCGATTA
CCGGATTTCGCCTGTCTTACGCTTTGGCTCATCTCGTGTCAAGCCATGCCGGCACACTCT
GTGCTGTGCGGATTGTATGATCGGGGAATCCCTTGTCAAAATGACCGCTATATGCGCCAT
GCCCTAGCCCAAGAGAGCCCCGGCTATGTTGGCGATACTCTTGCGTCACAAACGTAGTAA
GTGTTGCAGGGGGCTGGCATCGTGTGGGCACTGGAATACGTAGATGGCTGATCCCAGTTC
TGGCACGCAGCTCTATCACGTGTCTTAGCCTAGGACCCGGAACGCAAGGTAGGATAAATA
TCCGGCGATCAAGACCCAGCGTCGATGGTTCGAACCGCAGGTATACTGGATTGTCCGTAG
AATTTCATCCGCCCACGAAATAGCATCAGCTACGTTAGGGTAGAGCATCCAGGTGTGTAA
AATGTAGTGTTGGTGTATAATTCGTGCCTTCGGCATGGACAAGTGGAATTGCGGGGGTCC
GTGGCTGAGCGCAATGTAACAGTGATGTCGAGGGGCTACGAGCCTTTCCCCCCGGTTTAT
CTAGTCAATGCGGAGAAGCAACGGACTCTGTAATGGGTGGGGTAGCGTGAGGTCATAAAG
TAGCGTGCAATAATTACACCCAATTA'
>>> brin_2 = fichier.readline()
>>> brin_2 = brin_2[:-1]
>>> fichier.close()
```

3.
```
def repartition(brin):
    nb_A, nb_C, nb_G, nb_T = 0, 0, 0, 0
    for x in brin:
        if x == "A":
            nb_A += 1
        elif x == "C":
            nb_C += 1
        elif x == "G":
            nb_G += 1
        else:
            nb_T += 1
    return nb_A, nb_C, nb_G, nb_T
```

```
>>> repartition(brin_1)
(223, 225, 249, 228)
>>> repartition(brin_2)
(236, 213, 264, 212)
```

4.
```
def GCcontent(brin):
    nb_A, nb_C, nb_G, nb_T = repartition(brin)
    return (nb_G + nb_C)/(nb_A+nb_C+nb_G+nb_T)
```

```
>>> GCcontent(brin_1)
0.5124324324324324
>>> GCcontent(brin_2)
0.5156756756756756
```

5.
```
def Hamming(s, t):
    nb = 0
    n = len(s)
    for i in range(n):
        if s[i] != t[i]:
            nb += 1
    return nb
```

```
>>> Hamming(brin_1, brin_2)
465
```

Solution (exercice 3) Énoncé

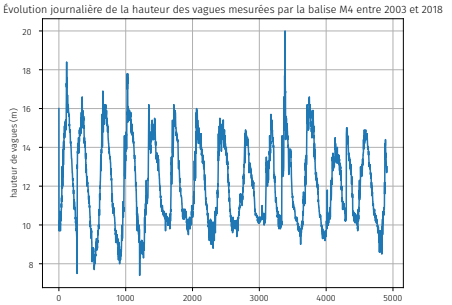
1. RAS
2.
```
def extraction1():
    fichier = open("hauteurs.txt", "r")
    c = fichier.readline()
    L_str = c.split()
    L = [] # on crée une nouvelle liste où on transforme \
↪ les chaînes en flottants
    for h in L_str:
        L.append(float(h))
    return L
```

```
>>> L = extraction1()
```

3. Pour le calcul du maximum, on peut exécuter la fonction `maximum` codée dans un précédent TP.

```
>>> maximum(L)
20.0
plt.plot(L) # affichage graphique
plt.ylabel("hauteur de vagues (m)")
plt.title ("Évolution journalière de la hauteur des vagues m
esurées par la balise M4 entre 2003 et 2018")
plt.grid()
```

```
plt.show()
```



Solution (exercice 4) Énoncé

```
prenoms = ["Ghislaine", "Georgette", "Alphonse", "Jean-Eude"]

def listeClasse(prenoms):
    fichier = open("notes.txt","w")
    for prenom in prenoms:
        note = round(20*rd.random(), 1)
        fichier.write(prenom + " " + str(note) + "\n")
    fichier.close()
```

```
>>> listeClasse(prenoms)
```

On peut observer en sortie le fichier crée :

```
Ghislaine 1.0
Georgette 13.4
Alphonse 19.7
Jean-Eude 6.0
```

Solution (exercice 5) Énoncé

1. Dans cet exercice, on utilisera le fichier ci-après :

```
Ghislaine 8.3
Georgette 11.2
Alphonse 2.5
Jean-Eude 16.3
```

```
def liste_notes():
    fichier = open("notes.txt","r")
    L = []
    for ligne in fichier:
        sep = ligne.split()
        L.append([sep[0], float(sep[1])])
```

```
    fichier.close()
    return L
```

2.
```
>>> L = liste_notes()
>>> L
[['Ghislaine', 1.0], ['Georgette', 13.4], ['Alphonse', \
↪ 19.7], ['Jean-Eude', 6.0]]
```

3. Il s'agit ensuite de faire une recherche de min, max selon la seconde coordonnée de chaque sous-liste.

```
def stats(L):
    mini = L[0][1]
    prenom_mini = L[0][0]
    maxi = L[0][1]
    prenom_maxi = L[0][0]
    S = 0 # somme des notes
    for couple in L:
        note = couple[1]
        prenom = couple[0]
        if note > maxi:
            maxi = note
            prenom_maxi = prenom
        if note < mini:
            mini = note
            prenom_mini = prenom
        S += note
    return (prenom_mini, mini), (prenom_maxi, maxi), \
    ↪ S/len(L)
```

```
>>> stats(L)
(('Ghislaine', 1.0), ('Alphonse', 19.7), 10.025)
```

Solution (exercice 6) Énoncé

La variable `arbres` est une liste de dictionnaires. Le nombre d'arbres correspond alors à la longueur de la liste `arbres`.

```
>>> len(arbres)
35499
def liste_localisations():
    localisations = []
    for arbre in arbres:
        loc = arbre["localisation"]
        if loc not in localisations:
            localisations.append(loc)
    return localisations
>>> localisations = liste_localisations()
>>> len(localisations)
967
>>> pjs = [arbre for arbre in arbres if arbre['typo_espace'] \
↪ == 'parc_jardin_square']
>>> len(pjs)
0
>>> parc_aux_angeliques = [arbre for arbre in arbres if "Parc a
ux Ang" in arbre['localisation']]
>>> len(parc_aux_angeliques) # le nombre d'arbres au parc des \
↪ angéliques
4426
>>> hauteur_sup_10 = [arbre for arbre in arbres if arbre['haute
ur'] != '' and int(arbre['hauteur']) >= 10]
>>> len(hauteur_sup_10) # le nombre d'arbres de hauteur >10
13980
```

La méthode `split` permet de séparer une chaîne suivant le symbole placé dans l'argument de `split` : elle crée alors une liste où chaque coordonnée est une chaîne issue de cette séparation. On s'en sert dans la suite afin de pouvoir récupérer l'année de plantation (qui nous intéresse).

```
>>> arbre2017 = [arbre for arbre in arbres if arbre['date_plant
ation'] != '' and int(arbre['date_plantation'].split("-")[0]) \
↪ >= 2017]
```

Solution (exercice 7) Énoncé

```
import csv
COEFFS = [4, 4, 4, 4, 4, 4 ,4, 3]

def dico_notes():
    """
    renvoie un dictionnaire avec comme clefs les candidats et \
    ↪ comme valeurs
    les notes par matière, dans le même ordre que COEFFS
    """
    fichier_notescand = open('Notes_candidats.csv' , "r")
    NOTES = csv.reader(fichier_notescand, delimiter = ",")
    dico_notescand = {}
```

```
    for L in NOTES:
        # on commence à 1 pour exclure l'en-tête
        candidat = L[0]
        notes = L[1:]
        if candidat != "Nom":
            dico_notescand[candidat] = notes
    fichier_notescand.close()
    return dico_notescand

dico = dico_notes()

def liste_moy():
    L = []
    for candidat in dico:
        notes = [float(x) for x in dico[candidat]]
        moyenne = 0
        for i in range(len(notes)):
            moyenne += notes[i]*COEFFS[i]
        L.append((candidat, moyenne/sum(COEFFS)))
    return L

moyennes = liste_moy()
```

```
>>> moyennes
```

```
[('C1', 7.638709677419355), ('C2', 7.509677419354838), ('C3', \
↪ 10.038709677419357), ('C4', 13.751612903225807), ('C5', \
↪ 9.154838709677419), ('C6', 9.683870967741935), ('C7', \
↪ 8.919354838709678), ('C8', 7.793548387096774), ('C9', \
↪ 10.67741935483871), ('C10', 12.806451612903226), ('C11', \
↪ 5.387096774193548), ('C12', 9.680645161290323), ('C13', \
↪ 8.364516129032257), ('C14', 9.838709677419354), ('C15', \
↪ 7.193548387096774), ('C16', 8.716129032258063), ('C17', \
↪ 11.080645161290322), ('C18', 7.667741935483871), ('C19', \
↪ 8.893548387096775), ('C20', 11.606451612903227), ('C21', \
↪ 7.6096774193548375), ('C22', 10.083870967741936), ('C23', \
↪ 11.983870967741936), ('C24', 12.193548387096774), ('C25', \
↪ 8.516129032258064), ('C26', 9.325806451612904), ('C27', \
↪ 9.864516129032259), ('C28', 5.025806451612904), ('C29', \
↪ 12.522580645161291), ('C30', 10.225806451612904), ('C31', \
↪ 8.983870967741936), ('C32', 7.441935483870967), ('C33', \
↪ 6.193548387096774), ('C34', 8.612903225806452), ('C35', \
↪ 9.361290322580645), ('C36', 7.580645161290323)]
```

Il nous reste ensuite à trier, la seule difficulté est qu'il s'agit ici de trier selon la deuxième coordonnée de chaque couple (la note). On peut par exemple employer le tri rapide, et l'adapter.

```
def classement(L):
    """
    on suppose ici que L est une liste de couples
    renvoie les couples triés dans l'ordre croissant selon la \
    ↪ deuxième coordonnée
    """
    if L == []:
        return []
    pivot = L[0]
    inf_pivot = []
    sup_pivot = []
    for x in L[1:]:
        if x[1] < pivot[1]:
            inf_pivot.append(x)
        else:
            sup_pivot.append(x)
    return tri_rapide_rec(inf_pivot) + [pivot] + \
    ↪ tri_rapide_rec(sup_pivot)
```

```
>>> moyennes_tri = classement(moyennes)
>>> moyennes_tri[-5:] # les admis
[('C5', 9.154838709677419), ('C6', 9.683870967741935), ('C7', \
↪ 8.919354838709678), ('C8', 7.793548387096774), ('C9', \
↪ 10.67741935483871)]
>>> moyennes_tri[-6][1]-moyennes_tri[-5][1] # points \
↪ manquants au 1er non admis
4.596774193548388
```