Python 1 – session 1

mardi 28/10/2025

Consignes

* **La durée de l'examen est de 2 heures**. Un tiers-temps supplémentaire est accordé aux étudiants dont le besoin est validé par le relais handicap.
* Le travail demandé est un travail strictement individuel et personnel. Vos scripts seront systématiquement analysés avec des outils de détection de fraude.
* Toute communication, avec des entités humaines ou non-humaines (comme des intelligences artificielles), de façon automatisée ou non, est interdite, considérée comme une fraude et donc passible de poursuites.
* Déposez, en une seule fois, vos scripts sur Moodle avant l'heure indiquée.
* Pour cet examen, vous utiliserez les machines des salles informatiques et une session utilisateur spéciale. Pensez à configurer votre éditeur de texte pour que l'indentation soit fixée à 4 espaces.
* Indiquez votre nom, prénom et cursus dans un commentaire dans la première ligne de tous vos scripts.
* Utilisez impérativement les noms de scripts, de fonctions et de variables précisés dans l'énoncé.
* Ajoutez des commentaires pertinents pour expliquer les parties les plus importantes de vos codes.
* N'utilisez que les bibliothèques Python standards.

Le non-respect de ces consignes sera sanctionné.

Un barème pour chaque exercice est fourni à titre indicatif.

Fichiers fournis sur Moodle

Vous trouverez sur Moodle des fichiers suivants :

* `exam.pdf`
* `9ic4.pdb`
* `pdb_format_33.pdf`
* `seq.txt`
* `malediction_du_corbeau.txt`
* `notes_python1.dat`

Fichiers à rendre sur Moodle

Vous déposerez sur Moodle les scripts suivants que vous aurez créés :

* `mystere.py`
* `count.py`
* `gc.py`
* `words.py`
* `notes.py`

1 Exercice 1 (2 points) ❓

Créez un nouveau script `mystere.py` qui affiche la solution à l'énigme suivante :

> *Je suis un nombre mystère compris entre 1000 et 6000 inclus. Le chiffre des unités est le double de celui des centaines et le triple de celui des dizaines. La somme du chiffre des milliers et de celui des centaines vaut 8. Qui suis-je ?*

2 Exercice 2 (4 points) ⚛️

Pour cet exercice, vous utiliserez le fichier `9ic4.pdb`. Il s'agit d'un fichier PDB, dont le format est détaillé dans le document `pdb_format_33.pdf`.

Créez un nouveau script `count.py`.

Créez la fonction `count_atoms()`. Cette fonction prend en argument le nom d'un fichier `.pdb` sous la forme d'une chaîne de caractères et réalise les actions suivantes :

* Compter le nombre d'atomes de chaque type dans le fichier PDB dont le nom est donné en argument. Ces comptages sont stockés dans un dictionnaire dont les clés sont les noms des atomes et les valeurs le nombre d'atomes.
* Afficher le nombre d'atomes pour chaque type d'atome, selon le format :
  `Atom XX is found YY times.`
  (où `XX` est le nom de l'atome et `YY` le nombre d'atomes correspondant).
* Afficher enfin l'atome le plus fréquent, selon le format :
  `The most common atom is: ZZ`
  (où `ZZ` est le nom de l'atome le plus fréquent).

Dans le script `count.py`, créez la constante `PDBNAME` qui contient la chaîne de caractères `"9ic4.pdb"` et appelez la fonction `count_atoms()` avec cette constante.

Voici une partie de la sortie renvoyée par le script `count.py` (`[...]` indique une coupure) :

```
$ python3 count.py
Atom N is found 374 times.
Atom C is found 1304 times.
[...]
Atom S is found 4 times.
The most common atom is: H.
```

Remarque : pour extraire le nom de chaque atome, utilisez la colonne *element symbol* du fichier PDB (voir le document de référence du format PDB : `pdb_format_33.pdf`)

3 Exercice 3 (4 points) 🧬

Le fichier `seq.txt` contient cinq séquences d'ADN, à raison d'une séquence par ligne.

Créez un nouveau script `gc.py`.

Créez la fonction `read_sequences()` qui prend en argument le nom d'un fichier sous la forme d'une chaîne de caractères et renvoie une liste dont les éléments sont les séquences lues dans le fichier.

Créez la fonction `compute_gc()` qui prend en argument une séquence d'ADN sous la forme d'une chaîne de caractères et renvoie la proportion de GC de la séquence. La proportion de GC est le rapport entre le nombre de bases G et C et le nombre total de bases :

$$GC = \frac{nb_G + nb_C}{nb_G + nb_C + nb_A + nb_T} = \frac{nb_G + nb_C}{sequence\ length}$$

Dans le programme principal de votre script, utilisez la fonction `read_sequences()` pour extraire les séquences du fichier `seq.txt`. Pour chaque séquence extraite, utilisez la fonction `compute_gc()` pour calculer la proportion de GC. Affichez le résultat sous la forme :
`Prop. GC sequence XX: YY`
(où `XX` est le numéro de la séquence, commençant à 1, et `YY` est la proportion de GC, affichée avec deux chiffres après la virgule).

Voici une partie de la sortie renvoyée par le script `gc.py` (`[...]` indique une coupure) :

```
$ python3 gc.py
Prop. GC sequence 1: 0.48
Prop. GC sequence 2: 0.65
[...]
```

4 Exercice 4 (5 points) 🎃

Le fichier `malediction_du_corbeau.txt` contient un extrait du roman « La malédiction du corbeau » de Jean-Paul Nozière. Ce texte ne contient ni majuscule, ni accent, mais les signes de ponctuations ont été conservés.

Créez un nouveau script `words.py` qui prend en argument le nom d'un fichier passé depuis la ligne de commande, lit ce fichier, extrait chaque mot du texte, compte la fréquence de chaque mot dans un dictionnaire puis affiche le nombre de mots différents, le mot le plus long et le mot le plus court dans le format de sortie donné ci-dessous.

Conseils :

* Conservez les apostrophes à l'intérieur des mots (ex : `"l'atmosphere"` est considéré comme un seul mot).
* Les signes de ponctuation restent attachés aux mots auxquels ils sont collés (ex : `"nature,"` est considéré comme un seul mot).

Voici la sortie renvoyée par le script `words.py` appliqué au fichier `malediction_du_corbeau.txt` :

```
$ python3 words.py malediction_du_corbeau.txt
Number of different words: 126
Longest word: inhospitalieres
Shortest word: a
```

5 Exercice 5 (5 points) 👻

Pour Halloween, M. Rouaud adore faire des farces à M. Poulain. Et sa dernière n'est pas des moindres ! Il a modifié le fichier `notes_python1.dat` contenant les notes des étudiants de Python 1, alors que M. Poulain avait laissé son ordinateur portable sans surveillance… Toutefois, M. Rouaud n'a pas modifié ce fichier au hasard. Voici ce qu'il a fait :

* Il a inversé les lettres des prénoms des étudiants. *pierre* est donc devenu *erreip*.
* Si le prénom contient la lettre « a », il a rajouté 6 points à la note.
* Pour les 5 premières notes du fichier, il a ajouté 3 points supplémentaires.
* Pour les 5 dernières notes du fichier, il a retiré 4 points.

Aidez ce pauvre M. Poulain à retrouver les notes de ses étudiants. Pour cela, créez un nouveau script `notes.py`. Créez la fonction `fix_grade()` qui prend en argument le nom d'un fichier de notes, lit ce fichier, corrige les noms des étudiants et leurs notes et écrit le tout dans un nouveau fichier `notes_python1_fixed.dat`. Les notes seront arrondies à deux chiffres après la virgule. Le format du fichier `notes_python1_fixed.dat` doit être identique à celui de `notes_python1.dat`.

Pour vous aider, voici les premières lignes du fichier `notes_python1_fixed.dat` (`[...]` indique une coupure) :

```
adrien,15.80
anne,5.44
benjamin,6.26
sacha,16.67
lucie,2.44
maxence,19.20
[...]
```