Examen machine Python 1 (L3BI / M1BI-IPFB / M1ISDD)

lundi 16 décembre 2024

Le barème et le temps passé sur chaque exercice sont indicatifs.

Les programmes demandés doivent être écrits en Python en utilisant les modules standards de Python.

Respectez impérativement les consignes qui vous sont données, notamment sur le nommage des programmes.

Un répertoire `$HOME/exam-python` a été créé pour vous. **Vous devez impérativement composer dedans**, pensez à y mettre tous les fichiers que vous créerez. Tout autre fichier mis ailleurs ne sera pas considéré pour la correction.

Vous exécuterez la commande suivante afin que nous puissions vous identifier :

```
$ touch $HOME/exam-unix/NOM_prenom
```

où `NOM` est à remplacer par votre propre nom de famille en majuscule, `prenom` par votre prénom en minuscule, avec les espaces remplacés par des caractères *underscore* (tiret bas).

Pour cet examen, vous aurez besoin des fichiers `coor_calpha.dat`, `phrase_codee.txt` et `vador2human.py` qui sont déjà présents dans le répertoire `$HOME/exam-python` (bien vérifier avant de commencer).

Enfin, configurez votre éditeur de texte pour que l'indentation soit fixée à 4 espaces.

Pour *gedit*, il faut aller dans le menu "Éditeur de texte", puis "Préférences", puis dans l'onglet "Éditeur", fixer la largeur des tabulations à 4 et cocher "Insérer des espaces au lieu des tabulations".

---

1 Exercice 1 (3 points, 18')

Créez un script `boucles.py` qui génère la sortie suivante à l'aide d'une double boucle imbriquée et d'un `if` :

```
 9 -8  7 -6
-8  7 -6  5
 7 -6  5 -4
-6  5 -4  3
 5 -4  3 -2
```

2 Exercice 2 (2.5 points, 15')

Le fichier `/usr/share/dict/french` contient le dictionnaire des mots français. Ecrire un script `extrait_mot20lettres.py` qui lit le fichier `/usr/share/dict/french` extrait chaque mot de 20 lettres et écrit chacun de ces mots dans un fichier `mots_20lettres.txt`. Lorsqu'on lancera le script, celui-ci affichera à l'écran :

```
$ python ./extrait_mot20lettres.py
Le script a extrait XX mots de 20 lettres.
```

où `XX` représente le nombre de mots de 20 lettres extrait par votre script.

3 Exercice 3 (4 points, 24')

Dark Vador a émis un message codé qu'il convient de décrypter. Heureusement, Han Solo a trouvé le moyen de décoder le message. Il s'agit d'un dictionnaire `vador2human.py` qui pour chaque lettre du message de Vador donne l'équivalent de la lettre en français, sachant que l'ordre des lettres n'est pas modifié (le caractère 1 de Vador correspond au caractère 1 de la phrase décodée, pareil pour le caractère 2 et ainsi de suite). La phrase codée quand à elle se trouve dans le fichier `phrase_codee.txt`. Par simplicité, tous les accents et caractères de ponctuation ont été enlevés (sauf les espaces), et on ne considère que les minuscules.

Ecrivez un script `decode.py` qui lit la phrase codée dans le fichier `phrase_codee.txt` et l'affiche à l'écran. Ensuite, le script doit décoder cette phrase en langage humain (français) à l'aide du dictionnaire se trouvant dans le fichier `vador2human.py`. Le script importera ce fichier en tant que module pour récupérer le dictionnaire à l'intérieur. Enfin, le script affichera la phrase décodée à l'écran. Sortie attendue :

```
$ python ./decode.py
Phrase codée:
'pbpjcwnsjowc njykjhzri djkarjykjxkpyykfnjywduwukjckjhn unwxxwrp djckjyjfdpokna'
Phrase décodée:
'[...]'
```

Notes : vous remplacerez `[...]` par la phrase décodée ; les guillemets simples ne font pas partie de la phrase.

4 Exercice 4 (4 points, 24')

Les diviseurs stricts d'un nombre sont tous ses diviseurs excepté lui-même. Par exemple, `6` admet comme diviseurs stricts `1`, `2` et `3`.

D'après Wikipedia, un nombre abondant est un nombre entier naturel non nul qui est strictement inférieur à la somme de ses diviseurs stricts. Par exemple, `10` n'est pas un nombre abondant car la somme de ses diviseurs `1 + 2 + 5 = 8` est inférieure à lui-même (`8 < 10`). Autre exemple, `12` est un nombre abondant car `1 + 2 + 3 + 4 + 6 = 16` et `16 > 12`. Ecrivez un script `abondant.py` qui détermine tous les nombres abondants de `1` à `100`. La sortie souhaitée est la suivante :

```
$ python ./abondant.py
12 est un nombre abondant car sum([1, 2, 3, 4, 6]) = 16 > 12.
18 est un nombre abondant car sum([1, 2, 3, 6, 9]) = 21 > 18.
[...]
Il y a XXX nombres abondants entre 1 et 100
```

où `XXX` correspond au nombre de nombres abondants que votre script a trouvé entre 1 et 100. Le script contiendra une fonction `extrait_diviseurs(nombre)` qui reçoit en argument un entier `nombre` et qui renvoit une liste des diviseurs stricts de ce nombre. Par exemple, si la fonction reçoit `6` comme argument, elle renverra la liste `[1, 2, 3]`.

5 Exercice 5 (6.5 points, 39')

Dans cet exercice, on souhaite écrire un script `get_contacts.py` qui extrait les contacts entre les différents carbones alpha (CA) d'une protéine. Le script prendra comme premier argument un nom de fichier contenant les coordonnées des CA (`coor_calpha.dat`) et en deuxième argument un *float* déterminant une distance seuil en Å. Le programme parcourra ensuite chaque paire de CA distincte et affichera s'il existe un contact si la distance entre eux est inférieure au seuil. Attention, dans cet affichage on veut les contacts entre CA séparés de plus de 5 résidus dans la séquence.

Le fichier `coor_calpha.dat` contient 4 colonnes, contenant respectivement le numéro de résidu et les coordonnées x, y et z du CA :

```
1 -7.542 5.187 9.163
2 -3.813 5.485 8.490
[...]
89 7.604 11.308 1.435
```

Le script contiendra une fonction `extrait_CA(filename)` où `filename` est une *string* contenant un nom de fichier (contenant les coordonnées des CA comme dans `coor_calpha.dat`). La fonction renverra un dictionnaire de ce type : `{1: (-7.542, 5.187, 9.163), 2: (-3.813, 5.485, 8.49), ...}`. Chaque clé est un entier représentant le numéro de résidu et chaque valeur est un tuple de *floats* contenant les coordonnées (x, y, z) du CA.

Le script contiendra une fonction `calc_dist(atom1, atom2)` où `atom1` et `atom2` sont des tuples contenant les coordonnées d'un atome. Cette fonction renverra un *float* correspondant à la distance euclidienne entre les 2 atomes.

Le programme principal récupèrera le dictionnaire contenant les coordonnées des CA en appelant la fonction `extrait_CA(filename)`. Ensuite, il fera une double boucle sur chaque paire de résidus et affichera chaque paire de CA dont la distance est inférieure au seuil. Afin de ne considérer que les paires séparées de plus de 5 résidus, vous pourrez utiliser un `if`.

Le programme affichera l'usage suivant si le bon nombre d'arguments n'est pas donné :

```
$ python ./get_contacts.py
Usage: python get_contacts.py coor_calpha.dat seuil
```

On testera le programme avec un seuil de 7 Å. La sortie souhaitée montrera chaque contact et sa distance exacte en Å avec une décimale :

```
$ python ./get_contacts.py coor_calpha.dat 7
contact  1 - 46 ->   6.7 A
contact  1 - 47 ->   6.2 A
[...]
contact 78 - 84 ->   5.3 A
```