`ENSIBS CDA S5 Programmation: Python`

Consignes et questions pour le TP

⚠️Vous travaillez obligatoirement en binôme⚠️

Ne divisez pas le travail : vous traitez à deux la même question, puis vous alternez les rôles (développement/vérification) à la question suivante. Vous vous exercez ici à la communication, élément clé de ce module focalisé sur la perfection et non la vitesse.

> *"Pair programming increases the person-hours required to deliver code compared to programmers working individually. However, the resulting code has fewer defects." — Pair programming | Wikipedia*

Vous devez déclarer votre binôme au premier TD, il n'est pas permis d'en changer.

Vous ne pouvez utiliser que les librairies standard de Python (pas de numpy ou quoi que ce soit d'autre nécessitant un pip install). Développez dans un espace vierge (python virtual environment) afin d'éviter l'import de librairies tierces que vous auriez déjà installées sur votre système. La correction de votre travail s'effectuera sous Linux (Debian based) avec la version la plus récente de Python3.

Exercice 00 "HelloWorld"

Soit un code de langue ("en" ou "fr") suivi d'un espace puis un mot défini sur `[a-zA-Z]+`, concaténez `Hello` (respectivement `Bonjour`) avec ce mot sur la sortie standard. Si ce mot dépasse 16 caractères, retourner `NAME TOO LONG` sur la sortie standard. Chaque exercice fonctionne de la même manière : votre script nommé `main.py` doit pouvoir être appelé depuis le répertoire racine `CDA_S5_Programmation_Python` en prenant comme seul paramètre un chemin UNIX vers un fichier existant, ce fichier contenant pour chaque ligne un ensemble de valeurs consommables par le script (une ligne = un appel à la fonction principale de votre script). Observez bien les fichiers d'exemples `INPUT.txt` et `EXPECTED.txt` dans chaque dossier d'exercice : le fichier d'input donne un ou plusieurs exemples de lignes de valeurs à consommer (les exemples ne sont pas exhaustifs !), le fichier expected donne la correspondance exacte de l'appel à votre script avec ces inputs qui doit être affichée sur la sortie standard lorsque votre script est exécuté ainsi :

```
python3 ex0/main.py path/to/INPUT.txt
```

Notez que le chemin vers le fichier d'input peut être complètement arbitraire, ce fichier doit exister et être disponible en lecture, ne traitez pas l'erreur dans le cas contraire.

Vous devez faire extrêmement attention aux formats indiqués, toute erreur d'affichage même minime sur la sortie standard (un espace ou un retour à la ligne en trop) entraîne l'échec de votre réponse et ne vous rapporte aucun point (les exercices de 1 à 9 valent 0.45 point, l'exercice 10 bonus rapporte 1.5 point, la note maximale est de 4 points). Tout appel à vos scripts termine par un et un seul retour à la ligne sur la sortie standard. Si le fichier d'input comporte plusieurs lignes impliquant plusieurs appels à la fonction principale de votre script, il n'y a pas de séparation (ie. pas d'ajout de retour à la ligne supplémentaire ni autre décoration) entre les différents appels.

Si le format d'input n'est pas respecté d'après la spécification de l'exercice, alors il s'agit d'une faute de l'utilisateur que vous devez traiter en retournant `BAD INPUT` suivi d'un retour à la ligne sur la sortie standard, puis passez à la ligne d'input suivante s'il en reste. En aucun cas votre script ne doit crasher, afficher une exception, ou autre élément non prévu par la spécification (cf. paragraphe précédent concernant les exigences d'affichage). Le format d'input doit être la première vérification effectuée, et une éventuelle erreur est traitée prioritairement : aucun autre message d'erreur ne doit être affiché pour la ligne d'input concernée en cas de non respect de la spécification.

Votre fichier `main.py` doit comporter une stricte séparation entre : - le code qui traite le fichier d'entrée (parsing,type casting, gestion des erreurs BAD INPUT) et qui gère l'affichage (ie. les print) - et le code métier qui suppose des inputs typés et conformes à la spécification et qui dispose d'au moins une fonction frontale (cf. fonction helloworld pour cet exercice) appelée par la fonction main ; aucune fonction métier ne doit afficher quoi que ce soit sur la sortie standard (ni stdout ni stderr d'ailleurs).

En utilisant `pytest`, vous devez implémenter dans `test.py` les tests de la (ou des) fonction(s) que vous avez définie(s) dans `main.py` (excepté la fonction "main" elle-même, qui gère les erreurs d'input). Il s'agit de montrer des exemples de cas nominaux intéressants, les cas limites, et les cas marginaux. Ne testez pas de cas impliquant des inputs illégaux (non-conformes à la spécification, chemins de fichiers inexistants, fichiers ciblés mal formés, etc.) ; pour rappel ces cas doivent être tous traités en produisant `BAD INPUT` sur la sortie standard (et rien d'autre). Les cas limites et marginaux doivent produire des exceptions en interne côté main.py, ces exceptions doivent être illustrées dans test.py.

Enfin générez avec `pydoc` un fichier html par exercice dans le dossier doc, en suivant la convention de nommage `doc_ex(0)<num>.html` (exemple : `doc_ex00.html`). Vous devez documenter toutes vos fonctions métier (ie. toutes sauf la fonction main) en utilisant le format docstring.

Vous trouverez la solution de cet exercice ainsi que ses tests dans le dossier ex00. Inspirez-

vous de sa structure pour les réponses aux exercices suivants.

Exercice 01

Transformez une valeur de type Integer ou sinon Float donnée en entrée vers sa représentation (en Float ou Integer) la plus compacte en terme de nombre de caractères (notation classique ou scientifique, exprimée en valeur décimale), vous laisserez les arrondis par défaut (ie. vous ne paramétrez pas le fait d'arrondir les nombres à l'affichage). Ne corrigez pas les éventuelles erreurs de précision.

Exercice 02

Soient au moins deux valeurs de type String en ASCII exprimées sur `[a-zA-Z]+` et séparées par le caractère espace, retourner la première selon l'ordre alphabétique ou l'ordre d'apparition pour deux mots de même rang.

Exercice 03

Soient deux valeurs de type Float A et B séparées par le caractère espace, retourner la valeur arrondie à 5 chiffres après la virgule de `sqrt((sin(A)/A)*(sin(B)/B))`. S'il est impossible de produire un résultat dans l'ensemble des réels, retournez `ARITHMETIC ERROR` suivi d'un retour à la ligne sur la sortie standard.

Exercice 04

Soit une valeur de type String quelconque non vide, s'il s'agit d'une séquence ADN valide (contient seulement les caractères `a`, `c`, `g`, ou `t` en majuscules ou minuscules) alors retourner sa conversion en ARN en majuscules (chaque `t` ou `T` devient un `U`), retourner "NOT A DNA SEQUENCE" sinon.

Exercice 05

Soient deux séquences A et B d'ADN valides (retourner "BAD INPUT" sinon) selon les critères de l'exercice précédent et séparées par le caractère espace, retourner une liste d'entiers dont le premier est le nombre de fois qu'apparaît la séquence ADN A dans la séquence ADN B suivi des index des positions (on compte à partir de 0) de A dans B. Cette liste est formée exclusivement d'entiers séparés par le caractère `,`. Si A n'existe pas dans B, retourner False.

Exercice 06

Soit une suite de valeurs séparées par le caractère espace formée dans l'ordre d'abord par deux entiers M et N (avec M<=N), puis une liste L non vide de mots formées sur `[a-zA-Z]+`, et C une valeur de type String correspondant à un chemin UNIX désignant un fichier existant F au format csv correctement formé (séparateur accepté : `,`) et comprenant des valeurs exclusivement numériques dans les cases concernées par les calculs : retourner la somme des lignes M à N (inclus) de valeurs – ne comptez pas la ligne d'identifiants et comptez la première ligne de valeurs à partir de 0 – (N devant être inférieur ou égal au nombre de lignes de valeurs disponibles) elles-mêmes étant le produit des valeurs présentes dans les colonnes identifiées par les mots de la liste L.

NB : un csv mal formé ou employant un autre séparateur que la virgule ou non composé exclusivement de valeurs numériques dans les cases concernées par les calculs constitue une erreur de la part du client, donc un non respect de la spécification qui doit être traité comme d'habitude : `BAD INPUT` sur la sortie standard.

NB2 : un chemin UNIX vers un fichier est valide si un système UNIX peut vérifier que le chemin existe et est accessible en lecture ; un chemin donné peut être soit absolu, soit relatif au répertoire courant (pwd)

Exercice 07

Soit une liste de mots formés sur `[a-zA-Z]+` et séparés par le caractère ASCII "espace" en input, retourner en minuscule dans l'ordre alphabétique pour chaque mot (une seule fois par mot et en ignorant la casse) le nombre de fois que ce mot apparaît dans la liste. Utilisez le format suivant (pas d'espace ni de virgule finale) :

```
mot1:fréquence1,mot2:fréquence2,mot3:fréquence3
```

Exercice 08

Soit une suite de valeurs séparées par le caractère ASCII "espace" formée dans l'ordre d'abord par une suite S de couples {entier,mot} (chaque entier étant unique, chaque mot étant formé sur `[a-zA-Z]+`) et terminant par un entier K>0. Retourner la liste des entiers (dans l'ordre) de 1 à K inclus, un entier par ligne, si un entier est divisible par un ou plusieurs entiers de la suite S afficher à la place la concaténation – dans l'ordre de spécification – des mots associés à chaque entier diviseur. La suite S à le droit d'être vide (on affiche une liste d'entiers de 1 à la borne indiquée par le dernier — et seul — argument K en input).

Exercice 09

Soit une ligne d'input composée d'une chaîne S définie sur `[a-zA-Z\*]+`, puis un caractére espace, puis un chemin UNIX vers un fichier existant F contenant des lignes de mots formés sur `[a-zA-Z]*`. Retourner dans leur ordre d'apparition dans F et en les séparant par un caractère espace tous les mots correspondant à la chaîne S recherchée ; cette correspondance est valide si S est une sous-chaîne du mot, le caractère * désignant une suite possiblement vide de caractères joker, toute suite de joker étant systématiquement correspondante. Par exemple la chaîne de recherche `or*h*a` est une sous-chaîne valide du mot `Morbihan`. Respectez bien la casse.

Exercice 10 (Bonus)

Soit R une valeur de type String correspondant à une expression régulière légale en Python, retourner `True` si — appliquée tronçon par tronçon — R permet de filtrer dans leur ordre de définition les tronçons continus de lettres ASCII (seulement les lettres majuscules et minuscules) de toute chaîne de caractère UTF-8 non vide et discrimine le reste ; retourner `False` sinon.

Exemple, soit `Hello Stéphanie!` une chaîne de caractères arbitraire S. L'isolation de ses lettres ASCII dans l'ordre est : `HelloStphanie`. Soit `[a-zA-Z]+` une expression régulière R, si l'on applique R par tronçon sur S via : `"".join(re.findall("[a-zA-Z]+", "Hello Stéphanie!"))` on obtient bien `HelloStphanie`, soit le résultat de la concaténation dans l'ordre de trois tronçons de caractères ASCII `Hello`, `St` et `phanie`, les autres caractères ayant été ignorés.

La difficulté de cet exercice repose sur le fait que la correspondance entre expressions régulières s'applique sur toute chaîne UTF-8 arbitraire et ne doit pas dépendre d'un ou plusieurs exemples de tests. Vous ne pouvez pas vous limiter à tester `Hello Stéphanie!`, sinon votre programme affirmerait par exemple que `[a-zèA-Z]+` est équivalente à `[a-zA-Z]+`, ce qui est faux car le filtrage de cette autre chaîne `Hello Stèphanie!` est différent pour ces deux expressions régulières.

Dépôt sur Moodle

Vous avez jusqu'à la fin de cette période école, weekend inclu, pour déposer une archive de votre travail sur l'espace Moodle dédié à ce travail.

Un pénalité d'1 point sur 4 s'applique en cas de retard jusqu'à 24 heures après la date limite indiquée sur la page. Tout travail non déposé sur Moodle avant ce dernier délai sera considéré comme non rendu (ne me l'envoyez pas par un autre canal) et se verra attribuer un grade de 0. Soumettez au moins 10 minutes avant la clôture pour ne pas risquer de voir votre dépoôt bloqué ; l'horloge du système Moodle fait foi.

Pour soumettre votre travail, remplissez le fichier META.txt avec vos noms+prénoms+n°étudiant en respectant le format indiqué par les exemples contenus dans ce fichier, zippez (format zip exclusivement, pas de rar ou autre) le dossier `CDA_S5_Programmation_Python`. Votre archive ne doit contenir que ce dossier à la racine (dézippez-là pour vérifier).

Nommez votre archive de la manière suivante : <id-email-binome-1>_<id-email-binome-2>.zip

`id-email-binome-N` signifiant la partie gauche (avant le @) de votre adresse email étudiante, qui est sous la forme `e<2xxx>@etud.univ-ubs.fr`. Exemple de nommage: `louis.e2303695_vincent.e2300080.zip`

Vérifiez deux fois votre numéro d'étudiant et celui de votre partenaire. Celui-ci commence par un "e" (cf. votre email).

Une pénalité d'1 point sur 4 s'applique en cas de non respect des conventions de nommage.

Ne soumettez pas votre travail en doublon. Désignez qui de vous deux doit déposer votre archive sur Moodle.