

***Bilingi
Nathan***

Rapport du TP 8

Introduction :

Le but de ce projet est d'automatiser, en utilisant le langage python, la correction de 500 programmes écrits en C par des élèves. L'un des principaux enjeux de ce projet est d'apprendre à utiliser la commande subprocess.

I-La démarche

Dans un premier temps voyons la méthode que nous avons employé pour répondre au problème donné.

Tout d'abord, nous avons dézippé le fichier « Rendus_eleves.zip ». Ensuite nous avons récupéré le nom des fichiers présents dans le dossier `eleves_bis` à l'aide de la commande `ls` et nous avons créé une liste avec ces fichiers. Par la suite, pour chaque fichier, nous avons utilisé la fonction « remplir », qui fonctionne de la façon suivante : elle vérifie si un fichier contient des erreurs en regardant si ce qui est renvoyé par la commande de compilation renvoie des lignes contenant la chaîne « `error:` ». Pour cela, on utilise la commande `gcc`, on récupère son résultat et on découpe la chaîne à chaque fois que le mot « `error :` » est présent (cela donnera une liste).

Si c'est le cas, c'est-à-dire si la liste obtenue est de taille 2 ou plus, alors la fonction qui teste des calculs ne se lancera pas sur ce fichier. Par la suite, on ajoute à une liste le nom de l'élève et son prénom à l'aide de deux fonctions qui jouent ces rôles. Pour cela on transforme la chaîne de caractère « `nom_prenom.c` » en une liste de deux éléments, en considérant le caractère « `_` » comme un séparateur. On récupère le premier élément de la liste pour avoir le nom, et on retire le « `.c` » au deuxième élément de la liste pour avoir le prénom. On ajoute ces deux éléments à une liste et on sépare les deux éléments par des « `;` ». Par la suite, avec la méthode décrite précédemment, on écrit « `0` » ou « `1` » dans la liste, selon si la fonction qui teste s'il y a des erreurs nous renvoie 0 ou 1. 0 permet de dire que la compilation a échoué, 1 que la compilation a fonctionné. Encore une fois on ajoute un « `;` » après le nombre pour le séparer de l'élément suivant (on écrira un « `;` » après chaque nombre).

Ensuite, on utilise une fonction qui teste le nombre de Warning en regardant le nombre de fois que `gcc` renvoie la chaîne « `warning :` ». Pour cela on utilise le même procédé que celui employé avec « `error :` » sauf que cette fois-ci on comptera la taille de la liste. La taille de la liste obtenue, -1, correspondra au nombre de warning. On ajoute le nombre obtenu à la liste précédente. Ensuite si la compilation du fichier a fonctionné, on lance une fonction qui va tester différents calculs sur le fichier exécutable et renvoyer le nombre de fois que cela a fonctionné, on ajoutera ce nombre à la liste. Au contraire si il n'y a pas eu d'exécutable, on ajoute directement « `0` » à la liste pour dire que le fichier ne passe aucun test étant donné qu'il ne produit pas d'exécutable.

Finalement on compte le nombre de ligne de documentation en vérifiant avec les commandes `subprocess` et `grep` le nombre de fois que l'on retrouve la chaîne de caractère « `/*` » ou « `*/` ». On ajoute le nombre obtenu dans la liste.

Après cela on écrit dans un fichier csv les différents nombres présents dans la liste. Le programme saura où placer chaque élément car il y aura un « `;` » pour distinguer une colonne d'une autre.

Finalement on répète l'opération pour chaque fichier .c, jusqu'à ce que le fichier csv soit complet.

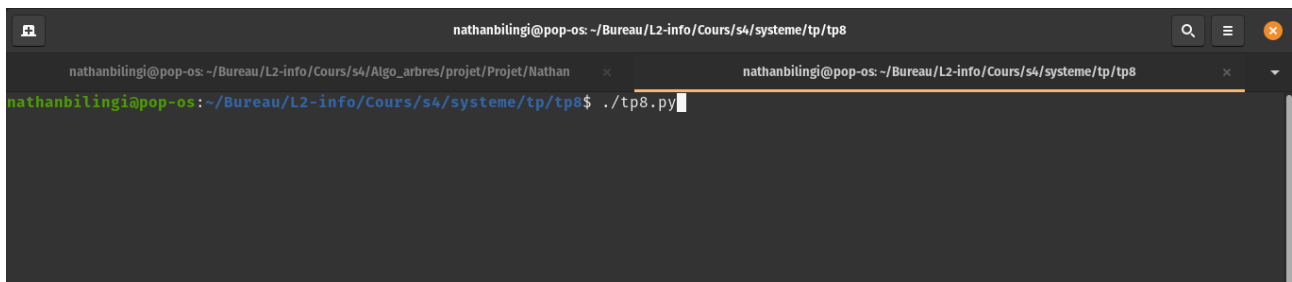
Ensuite, une fois les 6 premières colonnes du tableau remplies, on applique directement des calculs dans le fichiers csv pour compléter les 2 dernières colonnes (voir **II-mode d'emploi**).

II-Mode d'emploi

Dans un deuxième temps, voyons comment faire fonctionner le projet.

Tout d'abord, il est indispensable de mettre dans un dossier le fichier « tp8.py » et le fichier zip « Rendus_eleves.zip ». Ensuite, depuis votre terminal, il faut se déplacer jusqu'au dossier où vous avez placé votre fichier tp8.py, et écrire `chmod +x tp8.py`.

Après cela exécutez le fichier tp8.py en écrivant : « `./tp8.py` » (voir l'image ci-dessous).

A screenshot of a terminal window with a dark background. The title bar shows the user 'nathanbilingi' on a 'pop-os' machine, with the current directory being '~/Bureau/L2-info/Cours/s4/systeme/tp/tp8'. There are two tabs open: one for the current directory and another for a subdirectory. The terminal prompt is 'nathanbilingi@pop-os: ~/Bureau/L2-info/Cours/s4/systeme/tp/tp8\$' and the command './tp8.py' has been entered, with the cursor at the end of the line.

Attendez jusqu'à ce que le terminal vous redonne la main.

Une fois que vous aurez à nouveau la main, un fichier « file.csv » ce sera normalement formé si tout s'est bien passé. Ouvrez ce fichier csv, et dans la colonne G, ligne 1, écrivez la formule suivante : `=SI(SI(C1=1;3;0)-0,5*D1<0;0;SI(C1=1;3;0)-0,5*D1)`

Dans la colonne H, ligne 1, écrivez la formule suivante :

`=G1+(5/7)*E1+SI(F1*(2/3)>2;2;F1*(2/3))`

Étendez les formules sur toutes les colonnes H et G et vous obtenez finalement le produit final (voir l'image ci-dessous).

	A	B	C	D	E	F	G	H	I	
1	Bostan	Abdel	1	0	7	1	3	8,666666667		
2	Kiripoulos	Abdel	1	0	7	0	3	8		
3	Pouchkine	Abdel	1	0	1	0	3	3,714285714		
4	Satoures	Abdel	1	4	0	0	1	1		
5	Waldman	Abdel	1	0	7	1	3	8,666666667		
6	Allard	Abdoul	1	2	0	0	2	2		
7	Bicha	Abdoul	1	0	7	0	3	8		
8	Bond	Abdoul	1	0	7	1	3	8,666666667		
9	Carline	Abdoul	1	1	7	0	2,5	7,5		
10	Elfasi	Abdoul	1	0	7	0	3	8		
11	Rigot	Abdoul	1	0	7	1	3	8,666666667		
12	Ronchard	Abdoul	1	0	1	0	3	3,714285714		
13	Allard	Alban	1	0	7	0	3	8		
14	Barret	Alban	0	0	0	0	0	0		
15	Bloube	Alban	1	2	7	0	2	7		
16	Boulh	Alban	0	3	0	0	0	0		
17	Kerten	Alban	0	2	0	0	0	0		
18	Kiripoulos	Alban	1	0	7	1	3	8,666666667		
19	Peng	Alban	1	0	1	0	3	3,714285714		
20	Rabit	Alban	1	0	7	0	3	8		
21	Rochard	Alban	1	0	7	1	3	8,666666667		
22	Bent	Alexandra	1	2	0	0	2	2		
23	Boulh	Alexandra	1	0	7	1	3	8,666666667		
24	Collard	Alexandra	1	1	1	0	2,5	3,214285714		
25	Djoko	Alexandra	1	0	7	0	3	8		
26	Ladass	Alexandra	0	0	0	0	0	0		

Conclusion

En résumé, l'ensemble du programme est fonctionnel. Toutefois j'ai rencontré des difficultés lors de la réalisation du projet, notamment lorsqu'il a fallu réaliser un « pipe » car la commande `subprocess.run` ne pouvait pas réaliser cette tâche. Il a fallu chercher une commande qui permettait de réaliser des pipes et c'est à travers la commande `subprocess.Popen` que j'ai pu réaliser la commande que je souhaitais.

Hormis cela, je n'ai pas fait face à un problème qui m'a demandé beaucoup de temps pour être résolu.