

TD 1d — Trois programmes fautifs

Guide détaillé, étape par étape

Le TD fait afficher par l'éditeur de code les caractères qu'on ne voit pas, les espaces et les tabulations, puis corriger trois programmes Python courts qui s'arrêtent sur un message d'erreur. Chacun contient une faute d'un genre différent. On lance le programme, on lit le message, on corrige la copie, et on relance jusqu'à ce que le message disparaisse. Le TD dure une dizaine de minutes.

Le TD se fait dans VS Code, configuré au TD 1c : l'extension Python est installée, et le terminal intégré s'ouvre dans l'environnement base d'Anaconda. Les commandes se tapent dans ce terminal.

Étape	Objectif	Ce qu'on fait
1	travailler sur des copies des programmes	préparer le dossier du TD dans l'éditeur
2	voir les espaces, les tabulations et les fins de ligne	afficher les caractères invisibles
3	lire un message d'erreur et trouver la faute	corriger <code>surface.py</code>
4	corriger la ligne que le message désigne	corriger <code>moyenne.py</code>
5	remplacer un chemin absolu par un chemin relatif	corriger <code>chemin.py</code>

Chaque étape commence par un encadré qui la résume. Ce que chaque étape fait constater est expliqué à la fin du guide, dans « Ce que le TD fait constater » : faire l'étape d'abord, et noter ce qu'on observe, avant de lire l'explication.

1 · Préparer le dossier du TD dans l'éditeur

À faire : copier les trois fichiers de `depart\` dans `travail\` ; ouvrir le dossier `cours1\1d_erreurs\` dans VS Code ; ouvrir un terminal.

À obtenir : l'arborescence de VS Code montre `depart\` et `travail\`, et l'invite du terminal commence par `(base)` et se termine par `1d_erreurs>`.

Les deux dossiers du TD

Le dossier du TD est dans l'archive de la séance, extraite au TD 1a :

```
C:\Users\eleve\Desktop\info01\cours1\1d_erreurs\  
├─ depart\  
│   ├── chemin.py      compte les caractères du poème du TD 1a  
│   ├── moyenne.py     calcule la moyenne de quatre altitudes  
│   └── surface.py     calcule la surface d'une parcelle rectangulaire  
├─ travail\           vide  
├─ td_1d_erreurs.pdf  la feuille du TD  
└─ README.md
```

Comme au TD 1a, `depart\` contient les fichiers fournis, et ne se modifie pas ; les corrections se font sur des copies, dans `travail\`. Si une copie est abîmée, on en refait une à partir de `depart\`.

Dans l'explorateur de fichiers, ouvrir `depart\`, sélectionner les trois fichiers (`Ctrl + A`), les copier (`Ctrl + C`), ouvrir `travail\` et coller (`Ctrl + V`).

Vérification : `travail\` contient `chemin.py`, `moyenne.py` et `surface.py`.

Ouvrir le dossier dans l'éditeur

1. Dans VS Code, menu File, Open Folder (Fichier, Ouvrir le dossier, si l'interface est en français).
2. Choisir `Bureau\info01\cours1\1d_erreurs`, et cliquer sur « Sélectionner un dossier ». Ouvrir le dossier du TD lui-même, et non un fichier seul ni le dossier `cours1\` entier : le terminal s'ouvre dans le dossier ouvert, et le chemin écrit à l'étape 5 suppose que ce soit `1d_erreurs\`.
3. Si VS Code demande si l'on fait confiance aux auteurs du dossier, répondre « Yes, I trust the authors ».
4. Menu Terminal, New Terminal (Terminal, Nouveau terminal). Le terminal s'ouvre en bas de la fenêtre, déjà placé dans le dossier ouvert.

Vérification : l'invite du terminal a la forme suivante.

```
(base) C:\Users\eleve\Desktop\info01\cours1\1d_erreurs>
```

Si elle ne commence pas par `(base)`, taper `conda activate base`, puis Entrée. Si elle ne se termine pas par `1d_erreurs>`, le dossier ouvert à l'étape 2 n'est pas le bon : le rouvrir, puis ouvrir un nouveau terminal.

2 · Afficher les caractères invisibles

À faire : régler l'éditeur pour qu'il affiche les espaces et les tabulations ; lire deux indications de la barre d'état.

À obtenir : dans un fichier ouvert, un point entre les mots, et des points au début des lignes indentées.

Le rendu des espaces

Un espace et une tabulation ne se distinguent pas à l'œil : les deux laissent un blanc. VS Code peut les afficher, par un point médian (·) pour chaque espace et une flèche (→) pour chaque tabulation.

Le réglage se fait une fois, et reste valable toute l'année :

1. ouvrir les réglages, `Ctrl + ,` (menu File, Preferences, Settings) ;
2. dans la zone de recherche, en haut, taper `render whitespace` ;
3. dans la liste « Editor: Render Whitespace », choisir `all`.

Le menu View, Appearance, Render Whitespace (Affichage, Apparence, Rendu des espaces) active ou désactive le même affichage, sans proposer de valeur. La palette de commandes y donne aussi accès : `Ctrl + Maj + P`, puis taper `render whitespace`.

Ouvrir `travail\moyenne.py` dans l'arborescence de gauche, par un clic.

Vérification : un point apparaît entre chaque mot, et quatre points au début de la ligne 7, `total = total + altitude`.

La barre d'état

La barre d'état est la bande colorée, en bas de la fenêtre. Sur la droite, avec un fichier Python ouvert, elle affiche entre autres :

L'indication	Ce qu'elle dit
Spaces: 4	ce que la touche de tabulation insère : ici quatre espaces ; un clic dessus permet de le changer
LF ou CRLF	comment les lignes du fichier se terminent
UTF-8	l'encodage du fichier
Python	le langage que l'éditeur a associé au fichier, d'après son extension

À noter : laquelle des deux fins de ligne, LF ou CRLF, la barre d'état affiche pour `moyenne.py`.

3 · Corriger surface.py

À faire : lancer `travail\surface.py` ; lire le message ; trouver la faute dans l'éditeur ; la corriger ; relancer.

À obtenir : `python travail/surface.py` affiche une surface, et plus aucun message d'erreur.

Lancer et lire le message

Dans le terminal, taper la commande suivante, puis Entrée :

```
python travail/surface.py
```

Le message se lit de bas en haut. La dernière ligne donne le genre de l'erreur et sa description ; au-dessus, `File "... \travail\surface.py", line 6` désigne le fichier et le numéro de la ligne fautive, puis vient la ligne elle-même :

```
return aire
TabError: inconsistent use of tabs and spaces in indentation
```

Sous Windows, la commande s'écrit aussi `python travail\surface.py` : les deux séparateurs sont acceptés.

Trouver la faute

Ouvrir `travail\surface.py` dans l'éditeur, et regarder les lignes 5 et 6, qui forment le corps de la fonction `surface`. Elles paraissent alignées. L'éditeur souligne déjà l'une d'elles ; passer la souris sur le soulignement affiche le problème qu'il a relevé.

À noter : ce que le rendu des espaces montre au début de la ligne 5, et au début de la ligne 6.

Corriger et relancer

1. Au début de la ligne 6, supprimer le blanc qui précède `return`.
2. Taper quatre espaces, pour que la ligne soit indentée comme la ligne 5. Avec `Spaces : 4` dans la barre d'état, la touche de tabulation insère ces quatre espaces.
3. Enregistrer, `Ctrl + S`. Un point blanc sur l'onglet du fichier signale une modification non enregistrée : le terminal exécute le fichier tel qu'il est sur le disque.
4. Relancer `python travail/surface.py`. La flèche vers le haut du clavier rappelle la commande précédente dans le terminal.

La palette de commandes propose aussi « Convert Indentation to Spaces », qui remplace toutes les tabulations d'indentation du fichier par des espaces.

Vérification : le terminal affiche `294.0`, et rien d'autre.

4 · Corriger moyenne.py

À faire : lancer `travail\moyenne.py` ; lire le message ; corriger la ligne qu'il désigne ; relancer.

À obtenir : `python travail/moyenne.py` affiche une moyenne, et plus aucun message d'erreur.

Lancer et lire le message

```
python travail/moyenne.py
```

Le message désigne la ligne 6. Il la recopie, et place un accent circonflexe, ^, sous l'endroit où Python s'est arrêté :

```
for altitude in altitudes
                        ^
SyntaxError: expected ':'
```

À noter : ce que le message dit attendre, et l'endroit précis que désigne le ^.

Corriger et relancer

1. Ouvrir `travail\moyenne.py`, et aller à la ligne 6. Son numéro est dans la marge de gauche ; `Ctrl + G`, puis 6, y conduit directement.
2. Ajouter à cette ligne le caractère que le message demande, à l'endroit qu'il désigne.
3. Enregistrer, puis relancer `python travail/moyenne.py`.

Vérification : le terminal affiche `130.05`, et rien d'autre.

5 · Corriger chemin.py

À faire : lancer `travail\chemin.py` ; lire le message ; remplacer le chemin du fichier lu par un chemin relatif ; relancer.

À obtenir : `python travail/chemin.py` affiche un nombre de caractères, et plus aucun message d'erreur.

Lancer et lire le message

```
python travail/chemin.py
```

Le message se termine par :

```
FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/alice/cours1/1a_formats/depart/raven_une_ligne.txt'
```

Ouvrir `travail\chemin.py`. Le programme ouvre un fichier texte, le lit en entier, et affiche son nombre de caractères. Cette fois, l'éditeur ne souligne rien.

À noter : à quel poste, et à quel utilisateur, appartient le chemin écrit à la ligne 3 ; si ce chemin existe sur le vôtre.

Retrouver le fichier sur son poste

Le fichier que le programme veut lire est `raven_une_ligne.txt`, le poème du TD 1a. Sur les postes de la salle, il est ici :

```
C:\Users\eleve\Desktop\info01\cours1\1a_formats\depart\raven_une_ligne.txt
```

Le dossier courant du terminal est `C:\Users\eleve\Desktop\info01\cours1\1d_erreurs`. Un chemin relatif se lit à partir de ce dossier courant : `..` désigne le dossier parent, ici `cours1\`, et la suite du chemin descend de là jusqu'au fichier.

Corriger et relancer

1. À la ligne 3 de `travail\chemin.py`, remplacer le texte entre guillemets par :

```
../1a_formats/depart/raven_une_ligne.txt
```

Garder les guillemets, et écrire les séparateurs avec `/`, que Python accepte sous Windows comme ailleurs.

2. Enregistrer, puis relancer `python travail/chemin.py`, depuis le même terminal.

Vérification : le terminal affiche 1341 caractères.

Si le message `FileNotFoundError` revient, avec le nouveau chemin, vérifier dans l'ordre : que l'invite se termine par `1d_erreurs>` ; que le dossier `cours1\1a_formats\depart\` existe et contient `raven_une_ligne.txt` ; que le chemin ne contient pas de faute de frappe.

À noter : à partir de quel dossier le chemin relatif est compté, celui du fichier `chemin.py` ou celui du terminal. Pour le vérifier, taper `cd travail`, puis `python chemin.py`, et comparer ; revenir ensuite avec `cd ...`

Ce que le TD fait constater

Cette section se lit après avoir fait les étapes.

Étape 2 : ce que l'éditeur affiche des caractères invisibles

Le rendu des espaces montre la différence entre un espace et une tabulation, qui est dans le fichier et ne se voit pas autrement. Cet affichage est le seul moyen de repérer une indentation qui mélange les deux, et il ressortira au cours 2 : git y signalera des lignes modifiées qui semblent identiques, et qui ne diffèrent que par des espaces et des tabulations.

Spaces : 4 indique que la touche de tabulation insère quatre espaces. L'extension Python règle d'elle-même cette valeur à quatre, qui est la convention du langage.

LF et CRLF nomment la façon dont les lignes se terminent. Linux et macOS terminent une ligne par un caractère, LF (saut de ligne, 0A dans la table ASCII du TD 1a) ; Windows par deux, CR puis LF. Les fichiers du TD sont écrits avec LF. Un même texte n'a donc pas la même taille selon le système qui l'a enregistré ; le cours 2 y revient avec git.

Les trois fautes

Fichier	Ce que dit le message	La faute
surface.py	TabError: inconsistent use of tabs and spaces in indentation, ligne 6	la ligne 5 est indentée par quatre espaces, la ligne 6 par une tabulation
moyenne.py	SyntaxError: expected ':', ligne 6	il manque les deux-points à la fin de la ligne du for
chemin.py	FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/alice/...'	le chemin est celui du poste d'Alice ; il faut écrire ../1a_formats/depart/raven_une_ligne.txt

Messages relevés avec Python 3.12. Une fois corrigés, les trois programmes affichent 294.0, 130.05 et 1341 caractères.

Les trois fautes sont rangées par difficulté de lecture. Dans `surface.py`, les lignes 5 et 6 sont alignées à l'écran, et diffèrent dans le fichier : la ligne 5 commence par quatre espaces, la ligne 6 par une tabulation. Python refuse une indentation qui mélange les deux, et seul le rendu des espaces montre ce mélange à l'écran. Dans `moyenne.py`, le message suffit à trouver la faute : il nomme le caractère attendu, :, et le ^ désigne la fin de la ligne 6, où il manque. En Python, une ligne qui ouvre un bloc (for, if, def) se termine par deux-points.

L'éditeur souligne les deux premières fautes avant tout lancement : elles enfreignent les règles d'écriture de Python, que l'extension vérifie pendant la frappe.

Un programme correct sur un autre poste

La troisième faute est d'une autre nature. `chemin.py` respecte les règles d'écriture du langage, l'éditeur ne souligne rien, et le programme fonctionne sur le poste d'Alice, qui l'a écrit. Il échoue ailleurs parce qu'il contient un chemin **absolu**, `C:/Users/alice/...`, qui part de la racine du disque et n'existe que sur ce poste-là.

Un chemin **relatif** part du dossier courant, celui où se trouve le terminal ; le dossier du fichier `.py` n'intervient pas. Depuis `1d_erreurs\`, `../1a_formats/depart/raven_une_ligne.txt` remonte d'un dossier, puis descend dans celui du TD 1a. Il désigne le bon fichier sur tout poste où l'archive du cours est extraite de la même façon, quel que soit le nom de l'utilisateur, et quel que soit le système. Lancé depuis `travail\`, le même programme échoue : le dossier parent y est `1d_erreurs\`, qui ne contient pas `1a_formats\`.

Le choix du dossier ouvert dans VS Code a donc une conséquence sur les chemins relatifs : le terminal intégré, et le bouton d'exécution de l'éditeur, prennent le dossier ouvert pour dossier courant. Un chemin absolu dans un programme est l'erreur la plus fréquente des rendus de code des autres cours : le programme ne fonctionne que sur le poste de son auteur.

Ce que « le programme fonctionne » veut dire ici

La vérification demandée à chaque étape est que le programme se lance et n'affiche plus de message d'erreur. Elle ne dit pas que le résultat est juste : un programme qui calcule autre chose que ce qu'on voulait s'exécute sans message. La vérification de l'éditeur porte de même sur les règles d'écriture du langage, et non sur ce que le programme calcule.