

TD 2b — Trois programmes fautifs

Guide détaillé, étape par étape (version 2, proposition)

Version 2 du cours 2 : proposition de travail pour 2027-2028. Ce guide reprend celui du TD 1d de 2026. Joué au cours 1 v2 dans Notepad++, il est passé au cours 2 v2 le 27/09/2026, après la partie sur l'éditeur de code : les programmes s'ouvrent dans VS Code et se lancent dans son terminal, Git Bash.

Le TD fait lire dans VS 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 suppose le TD 2a fait : VS Code a Git Bash pour terminal, affiche les espaces, et enregistre les fichiers après un court délai.

Étape	Objectif	Ce qu'on fait
1	travailler sur des copies des programmes	préparer le dossier du TD
2	voir les caractères invisibles, l'encodage et les fins de ligne	lire les caractères invisibles et la barre d'état
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

À faire : ouvrir le dossier cours2/2b_erreurs/ dans VS Code, avec un terminal Git Bash ; copier les trois fichiers de depart/ dans travail/.

À obtenir : l'invite du terminal se termine par 2b_erreurs, et travail/ contient chemin.py, moyenne.py et surface.py.

Les deux dossiers du TD

Le dossier du TD est dans l'archive de la séance :

```
~/Desktop/info01/cours2/2b_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_2b_erreurs.pdf   la feuille du TD  
└─ README.md
```

Comme au cours 1, 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/.

Ouvrir le dossier dans VS Code

1. Dans VS Code : Fichier, Ouvrir le dossier..., puis choisir Bureau\info01\cours2\2b_erreurs.
2. Menu Terminal, Nouveau terminal. Le terminal s'ouvre en bas de la fenêtre, dans le dossier ouvert.

Vérification : l'invite a la forme suivante (le nom du poste change d'une machine à l'autre).

```
(base)  
eleve@POSTE MINGW64 ~/Desktop/info01/cours2/2b_erreurs  
$
```

Si le terminal n'est pas Git Bash, le choisir par la flèche à côté du +, comme au TD 2a. Si la ligne (base) manque, et que python --version ne répond pas, le réglage de conda dans Git Bash (TD 1d) n'a pas été fait sur ce poste.

Copier les fichiers

```
cp depart/*.py travail/  
ls travail
```

cp depart/*.py travail/ copie dans travail/ les fichiers de depart/ dont le nom se termine par .py. Le motif * a été vu au TD 1c.

Vérification : ls travail affiche chemin.py, moyenne.py et surface.py, et l'explorateur de VS Code les montre sous travail.

2 · Lire les caractères invisibles et la barre d'état

À faire : ouvrir `travail/moyenne.py` ; repérer les espaces dessinés ; lire deux indications de la barre d'état.

À obtenir : des points au début des lignes indentées, et LF dans la barre d'état.

Les espaces dessinés

Un espace et une tabulation ne se distinguent pas à l'œil : les deux laissent un blanc. Le réglage du TD 2a, « render whitespace » à all, fait dessiner par VS Code un point pour chaque espace et une flèche pour chaque tabulation. S'il n'a pas été fait : Ctrl + , , chercher « render whitespace », choisir all.

Ouvrir `travail/moyenne.py`, par un clic dans l'explorateur de VS Code.

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 en bas de la fenêtre. Avec un fichier Python ouvert, elle affiche entre autres, à droite :

L'indication	Ce qu'elle dit
Ln 7, Col 5	la ligne et la colonne du curseur
Espaces : 4	l'indentation que VS Code emploie pour ce fichier
UTF-8	l'encodage du fichier
LF ou CRLF	comment les lignes du fichier se terminent
Python	le langage que l'éditeur a associé au fichier, d'après son extension

Intitulés à vérifier sur la version de VS Code des postes.

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 de VS Code, 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
```

Dans Git Bash, le séparateur des dossiers s'écrit `/`. Écrit avec `\`, `travail\surface.py` y devient `travailsurface.py`, un fichier qui n'existe pas.

Trouver la faute

Ouvrir `travail/surface.py`, et regarder les lignes 5 et 6, qui forment le corps de la fonction `surface`. Elles paraissent alignées. VS Code souligne la ligne 6.

À noter : ce que VS Code dessine 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 à la barre d'espace, pour que la ligne soit indentée comme la ligne 5, et vérifier que quatre points s'affichent.
3. Enregistrer, `Ctrl + S`, ou attendre l'enregistrement automatique réglé au TD 2a. Un point blanc sur l'onglet du fichier signale une modification non enregistrée : `python` 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.

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. Passer à l'onglet de `moyenne.py`, et aller à la ligne 6. Son numéro est dans la marge de gauche ; `Ctrl + G`, puis 6 et Entrée, y conduit directement. VS Code souligne déjà cette ligne.
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. VS Code 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 `~/Desktop/info01/cours2/2b_erreurs`, celui que l'invite affiche. Un chemin relatif se lit à partir de ce dossier courant : `..` désigne le dossier parent, ici `cours2/`, `../..` le parent de celui-ci, `info01/`, 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 :

```
../../cours1/1a_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 `2b_erreurs` ; que `ls ../../cours1/1a_formats/depart` affiche `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

L'affichage des caractères invisibles 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 avec git, dans la suite de la séance : git signale des lignes modifiées qui semblent identiques, et qui ne diffèrent que par des espaces et des tabulations.

Unix (LF) et Windows (CR LF) nomment la façon dont les lignes se terminent. Linux et macOS terminent une ligne par un caractère, LF (saut de ligne, de valeur 0A) ; Windows par deux, CR puis LF. Les fichiers du TD sont écrits avec LF : la barre d'état de VS Code affiche LF. Un même texte n'a donc pas la même taille selon le système qui l'a enregistré ; la partie sur git y revient.

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 ../../cours1/1a_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 l'affichage des caractères invisibles 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.

VS Code souligne les deux premières fautes avant tout lancement : il vérifie les règles d'écriture de Python, comme le montre la diapositive « Texte brut et règles du langage ». Au cours 1, Notepad++ colorait le même code sans rien souligner.

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 `2b_erreurs/`, `../../cours1/1a_formats/depart/raven_une_ligne.txt` remonte de deux dossiers, jusqu'à `info01/`, puis descend dans le TD 1a. Il désigne le bon fichier sur tout poste où les archives des cours sont extraites 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 : deux dossiers plus haut, on est dans `cours2/`, qui ne contient pas `cours1/`.

Le dossier dans lequel le terminal a été ouvert, ou celui où `cd` a conduit, décide donc de ce que désigne un chemin relatif. 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.