

TD 4a — Un script Python, pas à pas

Guide détaillé, étape par étape

Ce guide détaille les étapes de la feuille du TD 4a. Pour chaque étape, il indique le dossier dans lequel se placer, ce qu'il faut écrire ou taper, et comment vérifier le résultat.

Le TD revoit les opérations des cours 1 à 3 sur un petit programme. La partie A crée le dossier du projet dans le terminal, l'ouvre dans VS Code et crée son dépôt git. La partie B écrit le programme `recette.py` étape par étape : il calcule les quantités d'une recette pour un nombre de personnes, en unités SI ou en unités américaines. Chaque étape se termine par un commit.

Le code de ce guide se copie depuis `guide_4a_recette.html`, ouvert dans un navigateur, ou depuis `guide.ipynb`, ouvert dans JupyterLab. Copié depuis le PDF, il perd ses indentations.

Étape	Objectif	Commits à la fin
A1	créer le dossier du projet dans le terminal	0
A2	ouvrir le projet dans VS Code, avec Git Bash pour terminal	0
A3	versionner le programme de départ	1
B1	lire un message d'erreur, corriger le programme	3
B2	adapter les quantités à un nombre de personnes	4
B3	convertir les unités, dans les deux sens	5
B4	lire les ingrédients dans un fichier CSV	6
B5	écrire le résultat dans un fichier CSV	7
B6	lire les valeurs sur la ligne de commande	8
B7 (bonus)	un README, une étiquette git	9

Le parcours avancé fait ce TD plus vite, puis le TD 4b dans le même dossier.

A1 · Le dossier du projet, dans le terminal

À faire : récupérer l'archive de la séance ; ouvrir Git Bash dans `cours4/4a_recette/` ; créer le dossier `travail/recette/` et y copier le programme et les recettes, par des commandes.

À obtenir : `ls travail/recette` affiche `recette.py` et `recettes` ; `ls travail/recette/recettes` affiche les quatre fichiers CSV.

A1.1 Récupérer l'archive

Comme au cours 3 : ouvrir le dossier partagé `formationTemp`, copier `info01-cours4.zip` sur le Bureau, puis clic droit sur l'archive, Extraire tout. Dans le dossier proposé, effacer la fin, `\info01-cours4`, puis Extraire. Le dossier extrait est `Desktop\cours4`.

```
cours4/
├── 4a_recette/
│   ├── depart/
│   │   ├── recette.py          ← le programme de départ
│   │   └── recettes/          ← quatre recettes, pour 4 personnes
│   │       ├── cookies.csv
│   │       ├── crepes.csv
│   │       ├── mousse_chocolat.csv
│   │       └── pate_pizza.csv
│   └── modeles/
│       └── README.md          ← le modèle de l'étape B7
├── travail/                  (vide)
├── guide_4a_recette.pdf      ← ce guide
├── td_4a_recette.pdf         ← la feuille du TD
└── 4b_paquet/               ← le TD 4b, parcours avancé
```

A1.2 Ouvrir Git Bash dans le dossier du TD

Dans l'explorateur de fichiers Windows, ouvrir `cours4`, puis clic droit sur le dossier `4a_recette`, « Afficher d'autres options », « Open Git Bash here ». Une fenêtre Git Bash s'ouvre dans ce dossier.

À défaut, ouvrir Git Bash depuis le menu Démarrer, puis :

```
cd ~/Desktop/cours4/4a_recette
```

Vérification :

```
pwd
```

affiche `/c/Users/eleve/Desktop/cours4/4a_recette`. Dans Git Bash, les chemins s'écrivent avec des `/`, et `C:\Users` devient `/c/Users`.

A1.3 Regarder le contenu

```
ls
ls depart
ls depart/recettes
```

Vérification : la première commande liste `depart`, `travail` et les deux PDF ; la troisième liste les quatre fichiers CSV.

A1.4 Créer le dossier du projet et y copier les fichiers

Le projet est un dossier à part, dans `travail/` : les fichiers de `depart/` restent tels qu'ils ont été livrés.

```
mkdir travail/recette
cp depart/recette.py travail/recette/
mkdir travail/recette/recettes
cp depart/recettes/*.csv travail/recette/recettes/
```

mkdir crée un dossier. cp copie le fichier nommé en premier dans le dossier nommé en second. *.csv désigne tous les fichiers dont le nom se termine par .csv. Une commande qui réussit n'affiche rien.

Vérification :

```
ls travail/recette
ls travail/recette/recettes
```

La première commande affiche recette.py et recettes, la seconde les quatre fichiers CSV. Si cp affiche No such file or directory, relire le chemin : la touche Tab complète les noms de dossiers et de fichiers.

A2 · Le projet dans VS Code

À faire : ouvrir travail/recette/ dans VS Code ; choisir Git Bash comme terminal ; afficher les espaces et les tabulations.

À obtenir : le terminal de VS Code a une invite qui commence par (base) et se termine par travail/recette ; les espaces du code apparaissent sous forme de points.

A2.1 Ouvrir le dossier

Lancer VS Code depuis le menu Démarrer. Menu File, Open Folder... : aller dans le Bureau, puis cours4, 4a_recette, travail, cliquer une fois sur recette, et « Sélectionner un dossier ». Si VS Code demande si l'on fait confiance aux auteurs des fichiers, répondre « Yes, I trust the authors ».

Le dossier ouvert est le projet : l'explorateur, à gauche, montre ses fichiers, et le terminal s'ouvre dans ce dossier.

A2.2 Git Bash comme terminal

1. Palette de commandes : Ctrl + Maj + P.
2. Taper default profile, puis choisir « Terminal: Select Default Profile ».
3. Choisir « Git Bash ».
4. Menu Terminal, New Terminal.

Vérification : l'invite du terminal est

```
(base)
eleve@POSTE MINGW64 ~/Desktop/cours4/4a_recette/travail/recette
$
```

Si l'invite commence par PS, le terminal est PowerShell : le fermer (icône de corbeille), puis en ouvrir un nouveau. S'il n'y a pas de (base), conda n'est pas encore réglé dans Git Bash sur ce poste. Le régler une fois, puis fermer le terminal et en ouvrir un nouveau :

```
source /c/ProgramData/anaconda3/etc/profile.d/conda.sh
conda init bash
```

A2.3 Afficher les espaces et les tabulations

Réglages : Ctrl + ,. Dans la zone de recherche, taper render whitespace, et choisir all pour « Editor: Render Whitespace ».

Vérification : ouvrir recette.py dans l'explorateur. Chaque espace du code apparaît comme un point gris, et chaque tabulation comme une flèche. L'étape B1 s'en sert.

Sur un thème sombre, les points et les flèches se voient mal. Un thème clair les rend plus visibles : Ctrl + K, puis Ctrl + T, et choisir « Light Modern » (ou « Light+ »). Le même choix se trouve dans le menu File, Preferences, Theme, Color Theme.

A3 · Le dépôt git

À faire : dans le terminal de VS Code : `git init`, votre nom et votre adresse pour ce dépôt, un fichier `.gitignore` qui contient `sortie/`, puis le premier commit.

À obtenir : `git log --oneline` affiche une ligne.

```
git init
git config user.name "Prénom Nom"
git config user.email "prenom.nom@exemple.fr"
```

Le poste est partagé : le nom et l'adresse sont réglés pour ce dépôt seulement, sans `--global`.

Le programme écrira ses résultats dans un dossier `sortie/`. Ces fichiers ne sont pas versionnés, puisque le programme les refait. Le fichier `.gitignore` donne la liste des fichiers et des dossiers que git ne suit pas, un par ligne. Dans l'explorateur de VS Code, créer un fichier (icône New File) nommé `.gitignore`, y écrire la ligne `sortie/`, puis enregistrer (Ctrl + S).

```
git status
```

Vérification : `git status` liste `.gitignore`, `recette.py` et `recettes/` parmi les fichiers non suivis.

```
git add .
git commit -m "Le programme de départ et les recettes"
git log --oneline
```

Vérification : `git log --oneline` affiche une ligne.

Partie B · Le programme

Le programme se lance dans le terminal de VS Code, dans `travail/recette/ : python recette.py`. Chaque étape modifie `recette.py`, le relance, puis enregistre l'état du fichier par un `commit`.

Les lignes à modifier. Le guide donne le programme en entier une seule fois, à l'étape B1. Pour les étapes suivantes, il ne donne que les lignes qui changent, comme dans cet extrait de l'étape B2 :

```
33 # ---- Le programme -----
34
- 35 print("Crêpes pour", PERSONNES_RECETTE, "personnes")
- 36 afficher(INGREDIENTS)
+     # Les valeurs à changer
+     PERSONNES = 6           # le nombre de personnes voulu
```

Une ligne marquée - est à supprimer, une ligne marquée + est à ajouter. Les lignes sans signe ne changent pas et indiquent l'endroit de la modification. Le signe ... remplace des lignes qui ne changent pas.

Le nombre qui suit le signe est le numéro de la ligne dans VS Code, quand les modifications sont faites de haut en bas. Une ligne ajoutée n'a pas de numéro. Si le fichier n'a pas les mêmes lignes vides que le guide, les numéros diffèrent de quelques lignes, et les lignes sans signe permettent de retrouver l'endroit.

Une ligne ajoutée se copie sans le + ni les cinq espaces qui le suivent. Pour plusieurs lignes, les coller dans VS Code, sélectionner leurs six premiers caractères en rectangle (Maj+Alt en faisant glisser la souris), puis appuyer sur Suppr.

Si une étape échoue. `git status` montre les fichiers modifiés depuis le dernier `commit`. `git diff` montre les lignes modifiées (q pour quitter). `git restore recette.py` remet le fichier dans l'état du dernier `commit`.

B1 · Corriger le programme de départ

À faire : lancer `python recette.py`, lire le message d'erreur, corriger la ligne qu'il désigne, puis un `commit` ; recommencer jusqu'à ce que le programme affiche la recette.

À obtenir : la recette des crêpes pour 4 personnes s'affiche ; un `commit` par erreur corrigée.

Le programme de départ a trois parties : les données (la recette des crêpes écrite dans une liste), les fonctions, puis le programme, qui appelle les fonctions. Chaque ingrédient est un tuple de trois valeurs : le nom, la quantité et l'unité.

```
"""Une recette à l'échelle : les ingrédients pour un nombre de personnes, en unités SI ou
US."""

# ---- Les données -----

# La recette des crêpes : (ingrédient, quantité, unité)
INGREDIENTS = [
    ("Farine", 250, "g"),
    ("Lait", 500, "ml"),
    ("Œufs", 4, ""),
    ("Sel", 2, "g"),
    ("Beurre fondu", 50, "g"),
]
PERSONNES_RECETTE = 4      # les recettes sont écrites pour 4 personnes

# ---- Les fonctions -----

def afficher(ingredients):
```

```

"""Affiche une ligne par ingrédient : le nom, la quantité et l'unité."""
for nom, quantite, unite in ingredients
print(nom, ":", round(quantite, 1), unite)

# ---- Le programme -----

print("Crêpes pour", PERSONNES_RECETTE, "personnes")
afficher(INGREDIENTS)

```

Lancer le programme :

```
python recette.py
```

Python s'arrête sur une erreur :

```

File "C:\Users\eleve\Desktop\cours4\4a_recette\travail\recette\recette.py", line 20
    for nom, quantite, unite in ingredients
                                ^
SyntaxError: expected ':'

```

Le message donne le fichier et le numéro de la ligne, recopie la ligne, puis nomme l'erreur. `SyntaxError: expected ':'` signale un deux-points manquant : une ligne `for` se termine par `:`. Corriger la ligne 20 et enregistrer (Ctrl + S), puis faire le commit de cette correction, avant de relancer le programme :

```
git commit -am "Correction : le deux-points de la boucle"
```

-a ajoute au commit les fichiers déjà suivis et modifiés. Avant le commit, `git diff` montre la ligne modifiée, en rouge avant et en vert après (q pour quitter).

Relancer le programme. Python s'arrête sur une seconde erreur :

```

File "C:\Users\eleve\Desktop\cours4\4a_recette\travail\recette\recette.py", line 21
    print(nom, ":", round(quantite, 1), unite)
TabError: inconsistent use of tabs and spaces in indentation

```

`TabError` signale un mélange d'espaces et de tabulations dans l'indentation d'un même bloc. Avec le réglage de l'étape A2, la ligne 21 commence par une flèche (une tabulation), et les autres lignes par des points (des espaces). Effacer la flèche, la remplacer par huit espaces, puis enregistrer.

Vérification : `python recette.py` affiche

```

Crêpes pour 4 personnes
Farine : 250 g
Lait : 500 ml
Œufs : 4
Sel : 2 g
Beurre fondu : 50 g

```

La barre d'état de VS Code, en bas à droite, indique Spaces : 4 : la touche Tab insère quatre espaces dans ce fichier. Faire le commit de cette seconde correction :

```

git commit -am "Correction : l'indentation de la ligne 21"
git log --oneline

```

Vérification : `git log --oneline` affiche trois lignes, une par commit.

B2 · Multiplier les quantités

À faire : écrire la fonction `adapter`, qui reçoit le nombre de personnes de la recette et le nombre voulu, et renvoie les ingrédients pour ce nombre ; l'appeler pour 6 personnes ; un commit.

À obtenir : `python recette.py` affiche la recette pour 6 personnes, avec Farine : 375.0 g ; un commit.

La recette est écrite pour 4 personnes (`PERSONNES_RECETTE`). Pour 6 personnes, chaque quantité est multipliée par $6 / 4 = 1,5$. La fonction `adapter` reçoit les deux nombres de personnes, celui de la recette et celui voulu, et calcule ce facteur.

Elle parcourt ensuite la liste des ingrédients avec une boucle `for`, comme `afficher`. Elle construit une nouvelle liste, `resultat`, un ingrédient à la fois, avec `append`, et la renvoie.

```
22
23
+     def adapter(ingredients, personnes_recette, personnes):
+         """Les ingrédients pour `personnes` personnes, d'une recette écrite pour
+         `personnes_recette` personnes."""
+         facteur = personnes / personnes_recette
+         resultat = []
+         for nom, quantite, unite in ingredients:
+             resultat.append((nom, quantite * facteur, unite))
+         return resultat
+
+
33 # ---- Le programme -----
34
- 35 print("Crêpes pour", PERSONNES_RECETTE, "personnes")
- 36 afficher(INGREDIENTS)
+     # Les valeurs à changer
+     PERSONNES = 6             # le nombre de personnes voulu
+
+     ingredients = adapter(INGREDIENTS, PERSONNES_RECETTE, PERSONNES)
+
+     print("Crêpes pour", PERSONNES, "personnes")
+     afficher(ingredients)
```

Vérification : `python recette.py` affiche

```
Crêpes pour 6 personnes
Farine : 375.0 g
Lait : 750.0 ml
Œufs : 6.0
Sel : 3.0 g
Beurre fondu : 75.0 g
```

Avec `PERSONNES = 4`, les quantités sont celles de départ ; avec `PERSONNES = 2`, elles sont divisées par deux. Remettre `PERSONNES = 6` avant le commit.

```
git commit -am "Les quantités pour un nombre de personnes"
```

B3 · Convertir les unités

À faire : écrire les deux tables de conversion et la fonction convertir ; convertir la recette en unités américaines ; un commit.

À obtenir : Farine : 13.2 oz et Lait : 3.2 cup ; les œufs gardent leur quantité ; un commit.

Une recette américaine donne les masses en onces (oz) et les volumes en tasses (cup) : 1 oz = 28,3495 g et 1 cup = 236,588 ml. Deux dictionnaires donnent, pour chaque unité, l'unité de l'autre système et le nombre par lequel multiplier la quantité. `VERS_US["g"]` vaut ("oz", 1 / 28.3495).

La fonction convertir reçoit l'une des deux tables. Une unité absente de la table, comme l'unité vide des œufs, ne change pas.

```
12 ]
13 PERSONNES_RECETTE = 4      # les recettes sont écrites pour 4 personnes
+
+   # Pour chaque unité : l'unité de l'autre système, et le nombre par lequel
+   # multiplier la quantité
+   VERS_US = {"g": ("oz", 1 / 28.3495), "ml": ("cup", 1 / 236.588)}
+   VERS_SI = {"oz": ("g", 28.3495), "cup": ("ml", 236.588)}
19
20
...
36
37
+   def convertir(ingredients, table):
+       """Les ingrédients, chaque unité présente dans `table` remplacée par celle de
+       l'autre système."""
+       resultat = []
+       for nom, quantite, unite in ingredients:
+           if unite in table:
+               nouvelle_unite, facteur = table[unite]
+               resultat.append((nom, quantite * facteur, nouvelle_unite))
+           else:
+               resultat.append((nom, quantite, unite))
+       return resultat
+
+
50 # ---- Le programme -----
51
52 # Les valeurs à changer
53 PERSONNES = 6              # le nombre de personnes voulu
+   UNITES = "US"            # le système d'unités voulu : "SI" ou "US"
55
56 ingredients = adapter(INGREDIENTS, PERSONNES_RECETTE, PERSONNES)
57
- 58 print("Crêpes pour", PERSONNES, "personnes")
+   if UNITES == "US":
+       ingredients = convertir(ingredients, VERS_US)
+   else:
+       ingredients = convertir(ingredients, VERS_SI)
+
+   print("Crêpes pour", PERSONNES, "personnes, en unités", UNITES)
64 afficher(ingredients)
```

Vérification : python recette.py affiche

```
Crêpes pour 6 personnes, en unités US
Farine : 13.2 oz
Lait : 3.2 cup
Œufs : 6.0
Sel : 0.1 oz
Beurre fondu : 2.6 oz
```

Avec UNITES = "SI", les quantités sont celles de l'étape B2 : la recette est déjà en grammes et en millilitres, et VERS_SI ne contient ni g ni ml. La conversion des unités américaines vers le SI sert à l'étape B4, avec la recette des cookies. Remettre UNITES = "US" avant le commit.

```
git commit -am "La conversion des unités"
```

B4 · Lire les ingrédients dans un fichier CSV

À faire : écrire la fonction `lire_ingredients`, qui lit un fichier de recettes/ ; supprimer la liste initiale des ingrédients, `INGREDIENTS` ; un commit.

À obtenir : la même sortie qu'à l'étape B3, lue dans `recettes/crepes.csv` ; `NOM = "cookies"` et `UNITES = "SI"` donnent des grammes et des millilitres ; un commit.

Ouvrir `recettes/crepes.csv` dans VS Code :

```
ingredient,quantite,unite
Farine,250,g
Lait,500,ml
Eufs,4,
Sel,2,g
Beurre fondu,50,g
```

La première ligne nomme les colonnes, les suivantes donnent un ingrédient chacune. La bibliothèque `csv` découpe chaque ligne en une liste de chaînes (notebook `fichiers.ipynb` du cours 3, section 10 ; documentation officielle : <https://docs.python.org/fr/3/library/csv.html>). `next(lecteur)` lit la première ligne et la laisse de côté. `float` convertit la quantité, lue comme une chaîne, en nombre.

Les chemins se construisent avec `pathlib`, à partir du dossier du script. `__file__` est le chemin du fichier `recette.py` en cours d'exécution, et `.parent` le dossier qui le contient, `travail/recette/`. / ajoute un nom au chemin. Construits ainsi, les chemins ne dépendent pas du dossier du terminal, et le programme trouve `recettes/` d'où qu'on le lance.

```
1 """Une recette à l'échelle : les ingrédients pour un nombre de personnes, en unités
SI ou US."""
+
+ import csv
+ from pathlib import Path
5
6 # ---- Les données -----
7
8 # La recette des crêpes : (ingrédient, quantité, unité)
9 INGREDIENTS = [
10     ("Farine", 250, "g"),
11     ("Lait", 500, "ml"),
12     ("Eufs", 4, ""),
13     ("Sel", 2, "g"),
14     ("Beurre fondu", 50, "g"),
15 ]
8 PERSONNES_RECETTE = 4      # les recettes sont écrites pour 4 personnes
9
...
13 VERS_SI = {"oz": ("g", 28.3495), "cup": ("ml", 236.588)}
14
+ # Les chemins partent du dossier du script : __file__ est le chemin de ce fichier
+ RACINE = Path(__file__).parent
+ DONNEES = RACINE / "recettes"
+
19
20 # ---- Les fonctions -----
+
+ def lire_ingredients(chemin):
+     """Les ingrédients du fichier CSV : une liste de (ingrédient, quantité,
```

```

unité)."""
+     ingredients = []
+     with open(chemin, encoding="utf-8", newline="") as fichier:
+         lecteur = csv.reader(fichier)
+         next(lecteur)          # la première ligne nomme les colonnes
+         for nom, quantite, unite in lecteur:
+             ingredients.append((nom, float(quantite), unite))
+     return ingredients
+
32
33 def afficher(ingredients):
    ...
61
62 # Les valeurs à changer
+     NOM = "crepes"           # la recette : un fichier de recettes/, sans .csv
64 PERSONNES = 6               # le nombre de personnes voulu
65 UNITES = "US"               # le système d'unités voulu : "SI" ou "US"
66
- 67 ingredients = adapter(INGREDIENTS, PERSONNES_RECETTE, PERSONNES)
+     ingredients = lire_ingredients(DONNEES / (NOM + ".csv"))
+     ingredients = adapter(ingredients, PERSONNES_RECETTE, PERSONNES)
69
70 if UNITES == "US":
    ...
73     ingredients = convertir(ingredients, VERS_SI)
74
- 75 print("Crêpes pour", PERSONNES, "personnes, en unités", UNITES)
+     print(NOM, "pour", PERSONNES, "personnes, en unités", UNITES)
76 afficher(ingredients)

```

Vérification : python recette.py affiche la même recette qu'à l'étape B3, lue cette fois dans le fichier :

```

crepes pour 6 personnes, en unités US
Farine : 13.2 oz
Lait : 3.2 cup
Œufs : 6.0
Sel : 0.1 oz
Beurre fondu : 2.6 oz

```

Avec NOM = "cookies" et UNITES = "SI", la recette américaine est convertie en grammes et en millilitres :

```

cookies pour 6 personnes, en unités SI
Farine : 532.3 ml
Beurre mou : 170.1 g
Sucre roux : 148.8 g
Œufs : 1.5
Pépites de chocolat : 212.6 g
Pincée de sel : 1.5

```

Remettre NOM = "crepes" et UNITES = "US" avant le commit.

```
git commit -am "Les ingrédients lus dans un fichier CSV"
```

B5 · Écrire le résultat dans un fichier CSV

À faire : écrire la fonction `ecrire_ingredients` ; écrire la recette calculée dans `sortie/` ; un commit.

À obtenir : `sortie/crepes_6_US.csv`, avec les mêmes colonnes que les fichiers de recettes/ ; `git status` ne liste pas `sortie/` ; un commit.

`open(chemin, "w", ...)` ouvre le fichier en écriture : il est créé, ou vidé s'il existe (notebook `fichiers.ipynb`, section 9). `csv.writer` écrit chaque liste passée à `writerow` sur une ligne (documentation officielle du module `csv` : <https://docs.python.org/fr/3/library/csv.html>), les valeurs séparées par des virgules. `SORTIE.mkdir(exist_ok=True)` crée le dossier `sortie/` s'il n'existe pas encore.

```
16 RACINE = Path(__file__).parent
17 DONNEES = RACINE / "recettes"
+   SORTIE = RACINE / "sortie"
19
20
...
30         ingredients.append((nom, float(quantite), unite))
31     return ingredients
+
+
+   def ecrire_ingredients(chemin, ingredients):
+       """Écrit les ingrédients dans un fichier CSV, avec les mêmes colonnes que les
fichiers lus."""
+       with open(chemin, "w", encoding="utf-8", newline="") as fichier:
+           ecrivain = csv.writer(fichier)
+           ecrivain.writerow(["ingrédient", "quantite", "unite"])
+           for nom, quantite, unite in ingredients:
+               ecrivain.writerow([nom, round(quantite, 1), unite])
41
42
...
85 print(NOM, "pour", PERSONNES, "personnes, en unités", UNITES)
86 afficher(ingredients)
+
+   # Le résultat, dans sortie/
+   SORTIE.mkdir(exist_ok=True)
+   fichier = SORTIE / (NOM + "_" + str(PERSONNES) + "_" + UNITES + ".csv")
+   ecrire_ingredients(fichier, ingredients)
+   print("écrit :", fichier)
```

Vérification : `python recette.py` affiche la recette, puis

```
écrit : C:\Users\eleve\Desktop\cours4\4a_recette\travail\recette\sortie\crepes_6_US.csv
```

Ouvrir `sortie/crepes_6_US.csv` dans VS Code :

```
ingredient,quantite,unite
Farine,13.2,oz
Lait,3.2,cup
Œufs,6.0,
Sel,0.1,oz
Beurre fondu,2.6,oz
```

`git status` ne liste pas `sortie/`, écarté par `.gitignore`.

```
git commit -am "Le résultat écrit dans un fichier CSV"
```

B6 · Les valeurs sur la ligne de commande

À faire : remplacer les trois valeurs écrites dans le programme par trois arguments lus par argparse ; un commit.

À obtenir : `python recette.py cookies -p 8 -u SI` écrit `sortie/cookies_8_SI.csv` ; `python recette.py --help` décrit les trois arguments ; un commit.

Pour changer de recette, de nombre de personnes ou d'unités, il faut jusqu'ici modifier les trois valeurs dans le code. argparse (cours 3) les lit sur la ligne de commande, au lancement. Les fonctions et le calcul ne changent pas. nom est un argument obligatoire. -p et -u sont des options, avec une valeur par défaut. choices limite les valeurs possibles, et type=int convertit la valeur lue en nombre entier.

```
1 """Une recette à l'échelle : les ingrédients pour un nombre de personnes, en unités
SI ou US."""
2
+ import argparse
4 import csv
5 from pathlib import Path
...
71 # ---- Le programme -----
72
- 73 # Les valeurs à changer
- 74 NOM = "crepes"           # la recette : un fichier de recettes/, sans .csv
- 75 PERSONNES = 6           # le nombre de personnes voulu
- 76 UNITES = "US"           # le système d'unités voulu : "SI" ou "US"
+ # Les valeurs viennent de la ligne de commande
+ analyseur = argparse.ArgumentParser(description="Les ingrédients d'une recette pour
un nombre de personnes, en unités SI ou US.")
+ analyseur.add_argument("nom", help="la recette : un fichier de recettes/, sans .csv")
+ analyseur.add_argument("-p", "--personnes", type=int, default=4, help="nombre de
personnes (défaut : 4)")
+ analyseur.add_argument("-u", "--unites", choices=("SI", "US"), default="SI",
help="système d'unités (défaut : SI)")
+ options = analyseur.parse_args()
+ NOM = options.nom
+ PERSONNES = options.personnes
+ UNITES = options.unites
82
83 ingredients = lire_ingredients(DONNEES / (NOM + ".csv"))
```

Vérifications :

Commande	Ce qui doit s'afficher
<code>python recette.py cookies -p 8 -u SI</code>	la recette des cookies pour 8 personnes, en grammes et en millilitres, puis écrit : ... \\sortie\\cookies_8_SI.csv
<code>python recette.py crepes</code>	la recette pour 4 personnes en SI, les valeurs par défaut
<code>python recette.py</code>	error: the following arguments are required: nom
<code>python recette.py crepes -u FR</code>	error: argument -u/--unites: invalid choice: 'FR' (choose from 'SI', 'US')
<code>python recette.py --help</code>	l'aide : les trois arguments et leur texte

Un nom de recette inconnu arrête le programme sur une erreur, dont la dernière ligne donne la cause :

```
python recette.py gaufres
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'C:\\Users\\eleve\\Desktop\\cours4\\4a_recette\\travail\\recette\\recettes\\gaufres.csv'
```

```
git commit -am "Les valeurs lues sur la ligne de commande"
```

B7 (bonus) · Un README, une étiquette

À faire : compléter un README à partir du modèle ; poser l'étiquette v1.0 sur le dernier commit.

À obtenir : `git log --oneline` affiche neuf lignes, la première marquée tag: v1.0.

Copier le modèle dans le projet, depuis travail/recette/ :

```
cp ../../depart/modeles/README.md .
```

`..` désigne le dossier parent : `../../` remonte de travail/recette/ à 4a_recette/. Le `.` final désigne le dossier courant.

Ouvrir README.md dans VS Code et remplacer chaque passage « (À compléter ...) ». `Ctrl + Maj + V` affiche l'aperçu. Chaque commande écrite dans le README doit fonctionner quand on la colle dans le terminal.

```
git add README.md
git commit -m "Un README"
git tag -a v1.0 -m "Première version"
git log --oneline
```

Une étiquette git, un *tag*, nomme un commit : v1.0 désigne la version finale sans chercher son identifiant.