

TD 4b – Un programme installable

Guide détaillé, étape par étape

Ce guide détaille les étapes du TD 4b, pour le parcours avancé. Le TD suit le TD 4a, dans le même dossier `4a_recette/travail/recette/`. Il part du programme de l'étape B6 : `python recette.py crepes -p 6 -u US` écrit `sortie/crepes_6_US.csv`.

Le TD donne au programme la forme d'un projet Python : une fonction `main`, deux modules, un environnement décrit par un fichier, puis une commande `recette` installée par `pip`. Les diapositives de l'exposé qui précède le TD expliquent ces quatre notions. Chaque étape se termine par un commit.

Le guide donne les lignes à modifier sous la forme expliquée dans la partie B du guide du TD 4a : - pour une ligne à supprimer, + pour une ligne à ajouter.

Étape	Objectif	Commits de plus
C1	séparer les définitions du programme principal : <code>main</code>	1
C2	répartir le code en deux modules	2
C3	décrire l'environnement du projet dans un fichier	3
C4	installer le programme comme une commande	4
C5 (bonus)	écrire la page HTML par <code>pandoc</code>	5

C1 · Une fonction main

À faire : placer les lignes du programme dans une fonction main, appelée en bas du fichier sous `if __name__ == "__main__":` ; un commit.

À obtenir : `python recette.py crepes -p 6` affiche la même recette qu'avant ; `python -c "import recette"` n'affiche rien.

Python lit un fichier de haut en bas, qu'il soit lancé par python ou importé par un autre fichier. Importer `recette.py` exécute donc aussi son programme :

```
python -c "import recette"
```

s'arrête sur une erreur, car le programme attend ses arguments :

```
-c: error: the following arguments are required: nom
```

`python -c` exécute le code écrit entre guillemets.

1. Au-dessus du commentaire `# Les valeurs viennent de la ligne de commande`, écrire `def main():`.
2. Sélectionner toutes les lignes du programme, de ce commentaire jusqu'au dernier `print`, et appuyer sur Tab : elles se décalent de quatre espaces.
3. À la fin du fichier, ajouter l'appel :

```
# Vrai quand le fichier est lancé par `python`, faux quand il est importé.
if __name__ == "__main__":
    main()
```

Le bas du fichier devient :

```
def main():
    # 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.")
    ...
    ecrire_ingredients(fichier, ingredients)
    print("écrit :", fichier)

# Vrai quand le fichier est lancé par `python`, faux quand il est importé.
if __name__ == "__main__":
    main()
```

`__name__` vaut `"__main__"` quand le fichier est lancé par python, et `"recette"` quand il est importé.

Vérification : `python recette.py crepes -p 6` affiche la même recette qu'à l'étape B6 ; `python -c "import recette"` n'affiche rien.

```
git commit -am "Une fonction main"
```

C2 · Deux modules

À faire : créer `quantites.py` avec les tables et les fonctions ; les retirer de `recette.py`, qui les importe ; un commit.

À obtenir : la même recette ; `recette.py` ne contient plus que les chemins, `main` et son appel.

Un fichier Python est un module. `quantites.py` contiendra ce qui calcule (lire, écrire, adapter, convertir, afficher) et `recette.py` ce qui dépend du programme (les chemins, les arguments, l'ordre des appels).

C2.1 Le module `quantites.py`

Dans l'explorateur de VS Code, créer le fichier `quantites.py` dans `travail/recette/`. Y couper-coller depuis `recette.py` les deux tables et les cinq fonctions. Le fichier contient alors :

```
"""Les quantités d'une recette : lire et écrire un fichier CSV, adapter au nombre de
personnes, convertir, afficher."""

import csv

# 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)}

# ---- 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

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(["ingredient", "quantite", "unite"])
        for nom, quantite, unite in ingredients:
            ecrivain.writerow([nom, round(quantite, 1), unite])

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)

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

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

```

C2.2 recette.py importe le module

Dans `recette.py`, il reste les chemins, puis `main` et son appel. `import csv` n'y sert plus. Une ligne importe les tables et les fonctions de `quantites.py` :

```

2
3 import argparse
- 4 import csv
4 from pathlib import Path
+
+     from quantites import VERS_SI, VERS_US, adapter, afficher, convertir,
+     ecrire_ingredients, lire_ingredients
7
8 # ---- Les données -----

```

`from quantites import adapter` exécute `quantites.py` une fois, et le nom `adapter` est ensuite utilisable dans `recette.py`. Python cherche `quantites.py` dans le dossier du fichier lancé, puis dans les dossiers de l'environnement.

Vérification :

```

python recette.py crepes -p 6
python -c "import quantites; print(quantites.VERS_US)"

```

La première commande affiche la même recette. La seconde affiche la table :

```
{'g': ('oz', 0.03527399072294044), 'ml': ('cup', 0.0042267570629110525)}
```

```

git add quantites.py
git commit -am "Deux modules : quantites et recette"

```

C3 · Un environnement pour le projet

À faire : copier `environment.yml` dans le projet ; créer l'environnement `recette` et l'activer ; un `commit`.

À obtenir : l'invite commence par `(recette)` ; `which python` donne le Python de l'environnement ; le programme fonctionne.

Le fichier `environment.yml` décrit ce dont le projet a besoin. Une autre personne recrée le même environnement à partir de ce fichier.

```
cp ../../../../4b_paquet/depart/modeles/environment.yml .
```

```
# L'environnement du programme recette : conda env create -f environment.yml
name: recette
channels:
  - conda-forge
dependencies:
  - python=3.12
  - pandoc
  - pip
  - setuptools
```

`pandoc` sert au bonus C5. `pip` et `setuptools` servent à l'étape C4.

```
conda env create -f environment.yml
conda activate recette
which python
```

La création télécharge les paquets : plusieurs minutes.

Vérification : l'invite commence par `(recette)`, et `which python` affiche un chemin qui contient `envs/recette`.
`python recette.py crepes` affiche la recette : le programme n'emploie que la bibliothèque standard de Python.

```
git add environment.yml
git commit -m "L'environnement du projet"
```

C4 · Une commande installée

À faire : déplacer les deux modules dans `src/` avec `git mv` ; copier et compléter `pyproject.toml` ; `pip install -e .` ; un commit.

À obtenir : `recette crepes -p 6` affiche la recette, sans python ni nom de fichier ; `recette --help` affiche l'aide.

C4.1 Les modules dans `src/`

```
mkdir src
git mv recette.py quantites.py src/
git status
```

Vérification : `git status` affiche deux renommages (renamed:).

Le script est maintenant dans `src/`. `Path(__file__).parent` désigne donc `src/`, qui ne contient pas `recettes/`, et `python src/recette.py crepes` s'arrête sur une erreur dont la dernière ligne est :

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

La racine du projet est le dossier parent de `src/` : un `.` parent de plus, comme `..` dans un chemin.

```
11
12 # Les chemins partent du dossier du script : __file__ est le chemin de ce fichier
- 13 RACINE = Path(__file__).parent
+   RACINE = Path(__file__).parent.parent    # src/, puis le dossier du projet
14 DONNEES = RACINE / "recettes"
15 SORTIE = RACINE / "sortie"
```

Vérification : `python src/recette.py crepes` affiche la recette.

C4.2 Le fichier `pyproject.toml`

```
cp ../../../../4b_paquet/depart/modeles/pyproject.toml .
```

```
# La fiche d'identité du projet, lue par pip : `pip install -e .` installe la
# commande `recette`, qui appelle la fonction main du fichier src/recette.py.
```

```
[build-system]
requires = ["setuptools>=64"]
build-backend = "setuptools.build_meta"

[project]
name = "recette"
version = "0.1"
description = "(à compléter : une phrase)"
requires-python = ">=3.9"

[project.scripts]
recette = "recette:main"

[tool.setuptools]
package-dir = {"" = "src"}
py-modules = ["recette", "quantites"]
```

[project.scripts] déclare la commande : `recette = "recette:main"` fait de `recette` une commande qui appelle la fonction `main` du module `recette`. `package-dir` et `py-modules` disent à `pip` où sont les modules. Compléter la ligne `description`.

C4.3 Installer

Dans `travail/recette/`, l'environnement `recette` actif :

```
pip install -e . --no-build-isolation
```

`.` désigne le projet, dans le dossier courant. `-e` installe le projet en mode modifiable : la commande appelle le code de `src/`, et une modification de ce code vaut sans réinstaller. `--no-build-isolation` utilise le `setuptools` de l'environnement, sans le télécharger.

Vérification : la dernière ligne affichée est `Successfully installed recette-0.1`. Puis :

```
recette crepes -p 6
recette --help
which recette
```

La `recette` s'affiche, sans `python` ni nom de fichier. `which recette` donne un chemin dans l'environnement `recette`. Les chemins partent du script : lancée depuis un autre dossier, par exemple après `cd ..`, la commande trouve toujours `recettes/`, et écrit dans `sortie/` du projet.

Dans le `README`, s'il existe, remplacer `python recette.py` par `recette` et ajouter l'installation.

```
git add pyproject.toml
git commit -am "src/ et pyproject.toml : la commande recette"
```

`pip uninstall recette` retire la commande.

C5 (bonus) · La page HTML, par pandoc

À faire : ajouter l'option `--page`, qui écrit aussi la recette en page HTML par pandoc ; un commit.

À obtenir : `recette crepes -p 6 --page` écrit `sortie/crepes_6_SI.html`, qui s'ouvre dans le navigateur.

La fonction `ecrire_page` écrit un tableau Markdown, puis lance `pandoc` avec `subprocess.run`, comme au cours 3. Elle reste dans `recette.py`, car elle dépend d'un programme extérieur.

```
2
3 import argparse
+ import subprocess
5 from pathlib import Path
6
...
17
18
+ # ---- Les fonctions -----
+
+ def ecrire_page(chemin, titre, ingredients):
+     """Écrit la page HTML des ingrédients : un tableau Markdown, converti par
pandoc."""
+     lignes = ["# " + titre, "", "| Ingrédient | Quantité |", "|---|---|"]
+     for nom, quantite, unite in ingredients:
+         lignes.append("| " + nom + " | " + str(round(quantite, 1)) + " " + unite + "
|")
+     markdown = chemin.with_suffix(".md")
+     markdown.write_text("\n".join(lignes) + "\n", encoding="utf-8")
+     subprocess.run(["pandoc", str(markdown), "-o", str(chemin), "--standalone",
+                     "--metadata", "title=" + titre], check=True)
+
+
32 # ---- Le programme -----
33
...
38 analyseur.add_argument("-p", "--personnes", type=int, default=4, help="nombre de
personnes (défaut : 4)")
39 analyseur.add_argument("-u", "--unites", choices=("SI", "US"), default="SI",
help="système d'unités (défaut : SI)")
+ analyseur.add_argument("--page", action="store_true", help="écrit aussi la page
HTML, par pandoc")
41 options = analyseur.parse_args()
42 NOM = options.nom
...
61 print("écrit :", fichier)
62
+ if options.page:
+     page = SORTIE / (NOM + "_" + str(PERSONNES) + "_" + UNITES + ".html")
+     ecrire_page(page, NOM + " pour " + str(PERSONNES) + " personnes",
ingredients)
+     print("écrit :", page)
+
68
69 # Vrai quand le fichier est lancé par `python`, faux quand il est importé.
```

Vérification :

```
recette crepes -p 6 --page  
start sortie/crepes_6_SI.html
```

écrit : C:\Users\eleve\Desktop\cours4\4a_recette\travail\recette\sortie\crepes_6_SI.html

La page affiche le titre et le tableau des ingrédients.

```
git commit -am "L'option --page"
```