

Un script Python, pas à pas

TD d'application des cours 1 à 3 : créer et versionner le dossier d'un projet, puis y construire un script Python qui calcule les quantités d'une recette, un commit par étape

`cours4/4a_recette/` $\approx 30' + 75'$

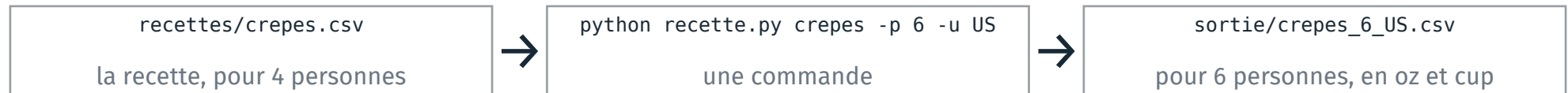
Le guide et le matériel de départ

Le guide du TD détaille chaque étape : dossier, code, commandes et vérifications. Les diapositives suivantes en sont le résumé.

Fichier	Emplacement	Usage
guide_4a_recette.pdf, .html, guide.ipynb	cours4/4a_recette/	le détail des étapes, le code à copier
recette.py	depart/	le programme de départ, avec deux erreurs de syntaxe à corriger
recettes/	depart/	quatre recettes en CSV, pour 4 personnes ; <code>cookies.csv</code> en unités américaines
README.md	depart/modeles/	un modèle de README à adapter et compléter (étape B7)

Les étapes du TD

Le script calcule une recette pour un nombre de personnes, en SI ou US.



Étape	Objectif	Résultat
A1-A3	préparer le projet : dossier, éditeur, dépôt	un dépôt ; un commit
B1	lire un message d'erreur et corriger	la recette pour 4 personnes s'affiche
B2	parcourir une liste dans une fonction	la recette pour 6 personnes
B3	convertir avec un dictionnaire	la recette en oz et cup
B4, B5	lire, puis écrire un fichier CSV	le résultat écrit dans <code>sortie/</code>
B6	lire les arguments de la ligne de commande	<code>recette.py cookies -p 8</code>

Aide-mémoire : le terminal

Rappel du cours 1, pour l'étape A1. Les commandes se tapent dans Git Bash ; les chemins partent du dossier courant.

<code>pwd</code>	affiche le dossier courant
<code>ls depart</code>	liste le contenu du dossier <code>depart</code>
<code>cd travail/recette, cd ..</code>	descend dans un dossier ; remonte d'un niveau
<code>mkdir travail/recette</code>	crée un dossier
<code>cp depart/recette.py travail/recette/</code>	copie un fichier dans un dossier
<code>cp depart/recettes/*.csv ...</code>	copie tous les fichiers dont le nom se termine par <code>.csv</code>
<code>Tab, flèche haut</code>	complète un nom ; rappelle la commande précédente

Une commande qui réussit n'affiche souvent rien : `ls` vérifie le résultat.

Aide-mémoire : une étape, un commit

Rappel du cours 2, pour toutes les étapes. Les commandes se tapent dans le terminal Git Bash de VS Code, dans `travail/recette/`.

<code>git init</code>	crée le dépôt dans le dossier courant
<code>git config user.name "Prénom Nom"</code>	le nom qui signe les commits, pour ce dépôt ; de même
<code>user.email</code>	
<code>.gitignore</code>	une ligne par fichier ou dossier que git ne suit pas : ici <code>sortie/</code>
<code>git status</code>	les fichiers modifiés depuis le dernier commit
<code>git diff</code>	les lignes modifiées ; <code>q</code> pour quitter
<code>git commit -am "..."</code>	enregistre les fichiers suivis et modifiés
<code>git restore recette.py</code>	remet le fichier dans l'état du dernier commit

Aide-mémoire : les données du programme

Rappel des cours 1 et 3, pour la partie B. Un ingrédient est un tuple de trois valeurs ; la recette est une liste d'ingrédients.

Écrit	Signification
<code>("Farine", 250, "g")</code>	un tuple : le nom, la quantité, l'unité
<code>[("Farine", 250, "g"), ("Lait", 500, "ml")]</code>	une liste de deux tuples
<code>for nom, quantite, unite in ingredients:</code>	une boucle : les trois valeurs de chaque tuple, l'un après l'autre
<code>resultat.append((nom, quantite, unite))</code>	ajoute un tuple à la fin de la liste <code>resultat</code>
<code>VERS_US = {"g": ("oz", 1 / 28.3495)}</code>	un dictionnaire : la clé "g" et sa valeur
<code>unite in VERS_US, VERS_US[unite]</code>	vrai si la clé existe ; la valeur associée à la clé

Les supports en ligne

Le site du module donne les pages de cours, les guides des TD et les archives des séances. Il est en cours d'écriture : des pages manquent ou changent d'une semaine à l'autre.

Page	Contenu
geodata-ing1-info-01.github.io/cours-info-01	l'accueil : une page par séance, avec ses TD
.../cours4/index.html	la séance 4 : le guide du TD 4a, celui du TD 4b
.../telechargements/	une archive par séance, les fichiers des TD

Étape A1 · Le dossier du projet, dans le terminal

Objectif : créer le dossier du projet par des commandes, en copiant les fichiers livrés.

Attendu à la fin de l'étape

- ✓ travail/recette/ contient `recette.py` et le dossier `recettes/`
- ✓ travail/recette/recettes/ contient les quatre fichiers CSV

Pistes

1. extraire l'archive sur le Bureau, comme au cours 3
2. ouvrir Git Bash dans `4a_recette/` : clic droit, « Open Git Bash here »
3. créer les dossiers avec `mkdir`, copier les fichiers avec `cp` (`*.csv` pour les quatre CSV)
4. vérifier chaque dossier avec `ls`

Guide, étape A1.

```
4a_recette/  
├─ depart/  
│   ├─ recette.py  
│   └─ recettes/           quatre CSV  
└─ travail/  
    └─ recette/           le projet  
        ├─ recette.py  
        └─ recettes/
```

Étape A2 · Le projet dans VS Code

Objectif : un éditeur ouvert sur le dossier du projet, avec un terminal Git Bash dans ce dossier.

Attendu à la fin de l'étape

- ✓ `travail/recette/` ouvert dans VS Code
- ✓ un terminal Git Bash dont l'invite commence par `(base)` et se termine par `travail/recette`
- ✓ les espaces du code affichés par des points

Pistes

1. ouvrir le dossier : File, Open Folder, `travail/recette`
2. choisir Git Bash : `Ctrl + Maj + P`, « Terminal: Select Default Profile »
3. ouvrir un terminal : Terminal, New Terminal
4. afficher les espaces : `Ctrl + ,`, « Render Whitespace », `all`

Guide, étape A2. Sur un thème sombre, les points se voient mal : un thème clair (`Ctrl + K`, puis `Ctrl + T`, « Light Modern ») les rend plus visibles.

Étape A3 · Le dépôt git

Objectif : enregistrer l'état de départ du projet, pour que chaque étape suivante ait son commit.

Attendu à la fin de l'étape

- ✓ un dépôt git dans `travail/recette/`
- ✓ un fichier `.gitignore` qui contient `sortie/`
- ✓ un premier commit, avec `recette.py`, `recettes/` et `.gitignore`

Pistes

1. créer le dépôt : `git init`
2. donner son nom et son adresse, pour ce dépôt seulement : `git config user.name, user.email`
3. créer `.gitignore` dans VS Code (New File), y écrire la ligne `sortie/`, enregistrer
4. vérifier avec `git status`, puis `git add .` et `git commit -m`

Guide, étape A3.

Étape B1 · Corriger le programme de départ

Objectif : lire un message d'erreur de Python et corriger la ligne qu'il désigne.

Attendu à la fin de l'étape

- ✓ `python recette.py` affiche la recette des crêpes pour 4 personnes
- ✓ un commit par erreur corrigée

Pistes

1. lancer `python recette.py`, lire le numéro de ligne, puis la dernière ligne du message
2. `SyntaxError: expected ':'` : ajouter le deux-points qui manque, enregistrer, puis faire un commit
3. relancer ; `TabError` : remplacer la flèche (une tabulation) par des espaces, enregistrer, puis faire un commit

Guide, étape B1 : le programme de départ en entier.

Étape B2 · Multiplier les quantités

Objectif : afficher la recette pour un autre nombre de personnes que celui de la recette, en multipliant chaque quantité par le même nombre.

Attendu à la fin de l'étape

- ✓ `python recette.py` affiche la recette des crêpes pour 6 personnes : Farine : 375.0 g, Lait : 750.0 ml
- ✓ une fonction `adapter(ingredients, personnes_recette, personnes)`, qui renvoie une nouvelle liste d'ingrédients
- ✓ un commit

Pistes

1. dans `adapter`, calculer le facteur : `personnes / personnes_recette`, soit $6 / 4 = 1,5$
2. reprendre la boucle de `afficher`
3. partir d'une liste vide, y ajouter chaque ingrédient multiplié par le facteur (`append`), la renvoyer (`return`)
4. dans le programme, appeler `adapter` avec `PERSONNES_RECETTE` et `PERSONNES`, puis `afficher`

Guide, étape B2 : le code de la fonction. À partir de cette étape, le guide ne donne que les lignes à modifier.

Étape B3 · Convertir les unités

Objectif : afficher la recette en unités américaines ou SI, avec une table de conversion.

Attendu à la fin de l'étape

- ✓ avec `UNITES = "US"`, la recette s'affiche en oz et cup : Farine : 13.2 oz
- ✓ les œufs, sans unité, gardent leur quantité
- ✓ une fonction `convertir(ingredients, table)` ; un commit

Pistes

1. écrire deux dictionnaires, `VERS_US` et `VERS_SI` : pour chaque unité, l'autre unité et le facteur
2. dans `convertir`, reprendre la boucle de `adapter`
3. si l'unité est une clé de la table (`if unite in table:`), multiplier la quantité et changer l'unité ; sinon, garder l'ingrédient
4. dans le programme, après `adapter`, appeler `convertir` avec `VERS_US` ou `VERS_SI` selon `UNITES`

Guide, étape B3. 1 oz = 28,3495 g ; 1 cup = 236,588 ml.

Un dictionnaire

```
VERS_US = {
    "g": ("oz", 1 / 28.3495),
    "ml": ("cup", 1 / 236.588),
}

"g" in VERS_US    # True
"" in VERS_US     # False : les œufs

unite, facteur = VERS_US["ml"]
# unite : "cup"
# facteur : 1 / 236.588
```

Étape B4 · Lire un fichier CSV

Objectif : lire les ingrédients dans un fichier, pour toutes les recettes.

Attendu à la fin de l'étape

- ✓ `python recette.py` affiche la même recette qu'à l'étape B3, lue dans `recettes/crepes.csv`
- ✓ avec `NOM = "cookies"` et `UNITES = "SI"`, les cookies s'affichent en g et ml
- ✓ la liste initiale des ingrédients, `INGREDIENTS`, a disparu ; un commit

Pistes

1. écrire `lire_ingredients(chemin)` : ouvrir le fichier, le lire avec `csv.reader` (`fichiers.ipynb`, § 10)
2. laisser de côté la ligne des noms de colonnes : `next(lecteur)`
3. convertir la quantité, lue comme une chaîne, en nombre : `float`
4. construire les chemins depuis le dossier du script : `RACINE = Path(__file__).parent, DONNEES = RACINE / "recettes"`
5. lire `DONNEES / (NOM + ".csv")` à la place de la liste initiale

Guide, étape B4. Documentation du module `csv` : docs.python.org/fr/3/library/csv.html.

recettes/crepes.csv

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

Étape B5 · Écrire un fichier CSV

Objectif : écrire le résultat dans un fichier, avec les mêmes colonnes que les fichiers lus.

Attendu à la fin de l'étape

- ✓ `python recette.py` écrit aussi `sortie/crepes_6_US.csv`
- ✓ le fichier écrit a la même première ligne que ceux de `recettes/`
- ✓ `git status` ne liste pas `sortie/` ; un commit

Pistes

1. écrire `ecrire_ingredients(chemin, ingredients)` : ouvrir le fichier en écriture, `open(chemin, "w", ...)` (fichiers.ipynb, § 9)
2. avec `csv.writer`, écrire la ligne des colonnes, puis une ligne par ingrédient : `writerow`
3. dans le programme, créer `sortie/` (`SORTIE = RACINE / "sortie", SORTIE.mkdir(exist_ok=True)`), puis construire le nom du fichier avec `NOM`, `PERSONNES` et `UNITES`

Guide, étape B5. Documentation du module `csv` : docs.python.org/fr/3/library/csv.html.

Étape B6 · Les arguments, avec argparse

Objectif : donner la recette, le nombre de personnes et les unités au lancement de la commande, au lieu de les changer dans le code. Les fonctions et le calcul ne changent pas.

Attendu à la fin de l'étape

- ✓ `python recette.py cookies -p 8 -u SI` affiche les cookies pour 8 personnes en SI, et écrit leur CSV
- ✓ `python recette.py --help` décrit les trois arguments
- ✓ un commit

Pistes

1. créer l'analyseur : `argparse.ArgumentParser` (cours 3)
2. déclarer l'argument `nom`, puis les options `-p` (`type=int, default=4`) et `-u` (`choices, default`)
3. remplacer les trois lignes `NOM = ..., PERSONNES = ..., UNITES = ...` par les valeurs lues : `options.nom, options.personnes, options.unites`

Guide, étape B6. Fin du TD en séance ; l'étape B7 est un bonus.

Étape B7 (bonus) · Un README, une étiquette

Objectif : qu'une autre personne sache lancer le programme, et nommer la version finale.

Attendu à la fin de l'étape

- ✓ un README.md complété, dont les commandes fonctionnent
- ✓ l'étiquette v1.0 sur le dernier commit
- ✓ un commit : le README

Pistes

1. copier le modèle : `cp ../../depart/modeles/README.md .`
2. remplacer chaque « (À compléter ...) » ; l'aperçu : `Ctrl + Maj + V`
3. faire le commit, puis poser l'étiquette : `git tag -a v1.0 -m "Première version"`

Guide, étape B7.