

TD 1i – Le client, le noyau, et où ils sont installés (facultatif)

Guide détaillé, étape par étape

L'exécution d'un notebook fait intervenir deux programmes : un client, qui affiche la page, et un noyau, qui exécute les cellules. Le TD vérifie ce que contiennent deux environnements conda, constate ce qui manque à l'environnement recette du TD 1h pour ouvrir un notebook, puis installe les programmes qui manquent, de deux façons : le client et le noyau dans le même environnement, ou le client dans un environnement et le noyau dans un autre. Le TD se termine par le notebook `recette.ipynb`, qui reprend le programme du TD 1h. Il dure une vingtaine de minutes, et il est facultatif : il se fait en séance si le temps le permet, ou seul ensuite.

Le TD nécessite l'invite de commandes d'Anaconda, où se tapent toutes les commandes, un navigateur pour JupyterLab, et VSCode avec l'extension Jupyter du TD 1g. Il suppose que le TD 1h a été fait : l'environnement recette doit exister. Les commandes `conda install` et `conda create` téléchargent des paquets, et nécessitent une session réseau ouverte.

Étape	Objectif	Ce qu'on fait
1	relever ce qu'un environnement contient	constater ce que les environnements contiennent, et ce qui manque
2	ouvrir un notebook avec le client et le noyau au même endroit	installer le client et le noyau dans le même environnement
3	séparer le client du noyau	installer un noyau dans un autre environnement que le client
4	exécuter un notebook dans l'environnement de son projet	exécuter le notebook du projet recette

Chaque étape commence par un encadré qui la résume. Les numéros entre parenthèses, (1) à (15), sont ceux de la feuille du TD. 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 · Ce que l'environnement contient, et ce qui manque

À faire : lister JupyterLab et ipykernel dans base ; lancer `jupyter lab` dans recette ; choisir recette comme noyau dans VSCode.

À obtenir : la réponse de `conda list` pour base, le message de l'invite de commandes d'Anaconda pour `jupyter lab`, et la proposition de VSCode.

Les fichiers du TD

Ouvrir `info01\cours1\li_noyaux\`. Le TD ne fournit qu'un fichier, et emploie un notebook du TD 1g :

```
cours1\
├─ 1g_notebooks\
│   └─ altitudes.ipynb      le notebook du TD 1g, employé aux étapes 1 à 3
├─ 1h_recette\
│   └─ depart\recette\data\ les données du projet recette, lues à l'étape 4
└─ 1i_noyaux\
    ├─ recette.ipynb        le programme du TD 1h, une fonction par cellule
    ├─ td_li_noyaux.pdf     la feuille du TD
    └─ README.md
```

Le TD n'a pas de dossier `depart\` ni `travail\` : il installe des paquets dans des environnements conda, et ne modifie que `recette.ipynb`.

L'invite de commandes d'Anaconda et l'environnement recette

Menu Démarrer, taper `anaconda`, choisir « Anaconda Prompt ». L'invite commence par `(base)`. Taper :

```
conda env list
```

La liste doit contenir `recette`, avec un dossier du type `C:\Users\<nom>\.conda\envs\recette`. Si elle ne le contient pas, refaire les étapes 1 et 4 du TD 1h : `conda create -n recette -c conda-forge python=3.12`, puis `conda install -n recette -c conda-forge markdown tabulate`.

(1) Ce que base contient

```
conda list -n base "jupyterlab|ipykernel"
```

`-n base` désigne l'environnement à lister. Ce qui suit est un motif : `conda list` n'affiche que les paquets dont le nom contient `jupyterlab` ou `ipykernel`, la barre verticale signifiant « ou ». Les guillemets sont nécessaires : sans eux, l'invite de commandes d'Anaconda lirait la barre verticale comme une commande à part. La feuille du TD écrit `conda list -n base jupyterlab ipykernel`, que `conda` refuse (`unrecognized arguments: ipykernel`) : il n'accepte qu'un motif.

La commande affiche un tableau, une ligne par paquet trouvé : nom, version, et canal d'origine. Des paquets voisins, comme `jupyterlab_server`, y apparaissent aussi, puisque leur nom contient `jupyterlab`.

À noter : les paquets que le tableau contient. La réponse dépend de la façon dont `conda` a été installé sur la machine : si le TD se fait aussi sur un portable personnel, y taper la même commande et comparer.

(2) Lancer JupyterLab dans recette

```
conda activate recette
jupyter lab
```

L'invite passe de `(base)` à `(recette)` après la première commande.

À noter : la réponse à `jupyter lab`.

(3) Choisir recette comme noyau dans VSCode

1. Ouvrir VSCode, puis le fichier `cours1\lg_notebooks\altitudes.ipynb` (menu File, Open File...).
2. En haut à droite du notebook, cliquer sur le nom du noyau, ou sur « Select Kernel » si aucun n'est choisi.
3. Choisir « Python Environments... », puis recette dans la liste. Si recette n'y est pas, taper dans la palette (Ctrl + Maj + P) « Python: Clear Cache and Reload Window », puis recommencer.
4. Exécuter la première cellule (Maj + Entrée).

VSCode affiche alors un message qui propose d'installer un paquet, avec un bouton « Install ». Ne pas l'accepter : fermer le message. L'étape 2 fait la même installation à la main.

À noter : le nom du paquet que VSCode propose d'installer.

2 · Tout dans le même environnement

À faire : installer JupyterLab dans recette ; le lancer ; y ouvrir `altitudes.ipynb` ; l'arrêter.

À obtenir : JupyterLab ouvert dans le navigateur, puis arrêté par `Ctrl + C` dans l'invite de commandes d'Anaconda.

(4) Installer JupyterLab dans recette

Dans l'invite de commandes d'Anaconda :

```
conda install -n recette -c conda-forge jupyterlab
```

conda calcule ce qu'il faut installer, puis affiche la liste des paquets et demande `Proceed ([y]/n)?`. Répondre y puis Entrée. Le téléchargement et l'installation prennent une à plusieurs minutes.

À noter : le nombre de paquets de la liste, et si `ipykernel` en fait partie, alors que la commande ne le nomme pas.

(5) Lancer JupyterLab

JupyterLab montre les fichiers du dossier d'où il est lancé. Se placer d'abord dans `cours1`, pour qu'il montre les dossiers des TD 1g et 1i :

```
conda activate recette
cd C:\Users\eleve\Desktop\info01\cours1
jupyter lab
```

L'invite de commandes d'Anaconda affiche plusieurs lignes, dont une adresse, puis le navigateur s'ouvre sur JupyterLab. Si le navigateur ne s'ouvre pas, copier l'adresse qui commence par `http://localhost` et la coller dans sa barre d'adresse. L'invite de commandes d'Anaconda reste occupée tant que JupyterLab s'exécute : ne pas le fermer.

À noter : l'adresse affichée, et ce que désigne `localhost`.

(6) Ouvrir `altitudes.ipynb`

Dans le panneau de gauche de JupyterLab, double-cliquer sur `1g_notebooks`, puis sur `altitudes.ipynb`. Exécuter les cellules (`Maj + Entrée`).

À noter : le nom du noyau, affiché en haut à droite du notebook.

(7) Arrêter JupyterLab

1. Fermer l'onglet de JupyterLab dans le navigateur.
2. Revenir à l'invite de commandes d'Anaconda, et taper `Ctrl + C`. JupyterLab demande s'il doit s'arrêter ; taper `Ctrl + C` une seconde fois, ou y puis Entrée.
3. Revenir au navigateur, rouvrir l'adresse de l'étape 5.

Vérification : (`recette`) est de nouveau affiché en début de ligne, dans l'invite de commandes d'Anaconda.

À noter : ce qu'affiche le navigateur quand on rouvre l'adresse, et ce qui continuerait de s'exécuter si on s'était contenté de fermer l'onglet.

3 · Le client d'un côté, le noyau de l'autre

À faire : créer un environnement `altitudes` qui a un noyau mais pas de client ; le déclarer ; l'employer depuis le JupyterLab de recette.

À obtenir : `import numpy` s'exécute dans `altitudes.ipynb`, ouvert dans le JupyterLab de recette, avec le noyau `altitudes`.

(8) Créer l'environnement `altitudes`

Dans l'invite de commandes d'Anaconda :

```
conda create -n altitudes -c conda-forge python=3.12 ipykernel numpy
```

Répondre `y` à `Proceed ([y]/n)?`. L'environnement contient Python, le noyau (`ipykernel`) et `numpy`, et pas JupyterLab.

(9) Déclarer le noyau

```
conda run -n altitudes python -m ipykernel install --user --name altitudes
```

`conda run -n altitudes` lance la commande qui suit avec le Python d'`altitudes`, sans activer l'environnement. `ipykernel install` écrit une déclaration de noyau, sous le nom donné par `--name`, dans un dossier propre au compte (`--user`), que tous les clients Jupyter lisent.

À noter : la dernière ligne affichée, et le dossier qu'elle nomme.

(10) Lister les noyaux déclarés

```
conda activate recette
jupyter kernelspec list
```

La commande `jupyter` est ici celle de l'environnement `recette`, installée à l'étape 4. `jupyter kernelspec list` affiche un nom de noyau par ligne, suivi du dossier de sa déclaration.

Ouvrir ce dossier pour le noyau `altitudes`, dans l'explorateur de fichiers (coller le chemin dans la barre d'adresse), puis ouvrir `kernel.json` dans Notepad++ ou le Bloc-notes. Ne rien modifier.

À noter : le nombre de noyaux listés ; ce que contient `kernel.json`, et en particulier le premier chemin de la liste `argv`.

(11) Employer le noyau `altitudes`

1. Relancer JupyterLab depuis `cours1`, comme à l'étape 5 (`cd` puis `jupyter lab`, dans `recette`).
2. Ouvrir `lg_notebooks\altitudes.ipynb`.
3. Menu Kernel, Change Kernel..., choisir `altitudes`, puis Select. Le nom affiché en haut à droite devient `altitudes`.
4. Ajouter une cellule à la fin (bouton + de la barre du notebook), y taper `import numpy`, et l'exécuter (Maj + Entrée).

Pour voir quel interpréteur exécute les cellules, ajouter une cellule avec :

```
import sys
sys.executable
```

À noter : si l'import de `numpy` réussit, sachant que `recette` n'a pas `numpy` ; l'interpréteur affiché par `sys.executable`.

VSCode lit la même liste de noyaux : dans VSCode, « Select Kernel », puis « Jupyter Kernel... », propose `altitudes`.

Supprimer ensuite les cellules ajoutées (clic dans la cellule, puis menu Edit, Delete Cells) : JupyterLab enregistre le notebook de lui-même à intervalles réguliers. Laisser JupyterLab ouvert pour l'étape 4.

4 · Le notebook du projet recette

À faire : ouvrir `li_noyaux\recette.ipynb` avec le noyau de recette ; commenter la première cellule ; tout exécuter ; changer le nombre de personnes.

À obtenir : la recette des crêpes affichée, mise en forme, avec son tableau d'ingrédients, pour le nombre de personnes choisi.

(12) Ouvrir le notebook et lire la première cellule

Dans le panneau de gauche de JupyterLab, revenir à `cours1` (clic sur le dossier dans le chemin affiché au-dessus de la liste), puis ouvrir `li_noyaux\recette.ipynb`. Menu Kernel, Change Kernel..., choisir le noyau de recette, « Python 3 (ipykernel) ».

La première cellule de code est :

```
from pathlib import Path

DONNEES = Path("../lh_recette/depart/recette/data")

PERSONNES = 4
UNITES = "SI"      # "SI" ou "US"

assert DONNEES.is_dir(), f"dossier introuvable : {DONNEES.resolve()}"
sorted(f.name for f in DONNEES.iterdir())
```

À noter : d'où part le chemin de `DONNEES`, et le dossier qu'il désigne sur le disque.

(13) Commenter la première cellule

En Python, un commentaire commence par `#` et va jusqu'à la fin de la ligne. Ajouter un commentaire à chaque ligne de la cellule, au-dessus d'elle ou à sa suite, qui explique à quoi la ligne sert dans ce notebook. « On importe Path » répète la ligne ; le commentaire attendu explique pourquoi le notebook a besoin de Path.

Le texte du notebook demande aussi d'adapter le chemin à l'emplacement des données. Le chemin écrit désigne le projet fourni au TD 1h, dans `depart\`. Pour lire la copie de travail, le chemin devient `Path("../lh_recette/travail/recette/data")`. Le chemin s'écrit avec des `/`, qui fonctionnent aussi sous Windows.

Exécuter la cellule (Maj + Entrée).

À noter : ce que la cellule affiche. Si elle affiche une `AssertionError` avec `dossier introuvable`, lire le chemin complet que le message donne, et corriger `DONNEES`.

(14) Redémarrer et tout exécuter

Menu Kernel, « Restart Kernel and Run All Cells... » (la feuille l'appelle « Noyau, Redémarrer et tout exécuter »), puis confirmer par Restart.

À noter : ce qu'affichent les deux dernières cellules, et la différence entre elles.

(15) Changer le nombre de personnes

Dans la première cellule, remplacer `PERSONNES = 4` par une autre valeur, puis refaire l'étape 14. Essayer aussi `UNITES = "US"`.

À noter : ce qui change dans la recette affichée, et ce qui ne change pas.

Rendre la main

Arrêter JupyterLab (Ctrl + C deux fois dans l'invite de commandes d'Anaconda), puis retirer la déclaration du noyau et l'environnement `altitudes` :

```
jupyter kernelspec remove altitudes
conda activate base
conda env remove -n altitudes
```

La première commande demande confirmation : répondre y. conda refuse de supprimer l'environnement actif, d'où le conda activate base avant la suppression.

En cas d'erreur

- 'conda' n'est pas reconnu en tant que commande interne ou externe : la commande a été tapée dans un cmd ou un PowerShell ordinaire. La taper dans l'invite de commandes d'Anaconda.
- EnvironmentNotWritableError à conda install : la commande s'appliquait à base, que le compte élève ne peut pas modifier. Vérifier que la commande contient -n recette.
- CondaToSNonInteractiveError: Terms of Service have not been accepted, ou une question Do you accept the Terms of Service (ToS) ... [(a)ccept/(r) eject/(v)iew] : répondre a, ou relancer la commande en ajoutant --override-channels avant -c conda-forge.
- CondaHTTPError: HTTP 000 CONNECTION FAILED : la session réseau n'est pas ouverte. L'ouvrir, et relancer la commande.
- « Solving environment » qui dure plusieurs minutes : la première commande qui emploie conda-forge télécharge la liste de ses paquets. Attendre.
- VSCode ne propose pas recette ou altitudes : palette, « Python: Clear Cache and Reload Window ».

Ce que le TD fait constater

Cette section se lit après avoir fait les étapes.

Étape 1 : ce qui manque à recette

	Ce qu'on fait	Ce qu'on constate
1	<code>conda list -n base</code> , limité à jupyterlab et ipykernel	les deux avec Anaconda, aucun des deux avec Miniforge
2	<code>conda activate recette</code> , puis <code>jupyter lab</code>	commande introuvable : pas de client dans cet environnement
3	dans VSCode, choisir recette comme noyau	l'éditeur propose d'installer ipykernel : pas de noyau non plus

La distribution Anaconda installe plus de six cents paquets dans base, JupyterLab et ipykernel compris. Mini-conda et Miniforge n'y installent que conda, Python et leurs dépendances. Les postes de la salle ont Anaconda, et c'est pour cette raison que le TD 1g ouvrait un notebook sans rien installer. Sur un portable installé avec Miniforge, base ne contient ni JupyterLab ni ipykernel, et le TD 1g ne s'y fait pas sans installer d'abord ces paquets. Le contenu d'un environnement se vérifie par `conda list`, plutôt que d'être supposé. `conda list` accepte un motif, ce qui dispense de filtrer sa sortie avec un autre outil.

`jupyter lab` n'est pas une commande du système : c'est un programme installé dans un environnement, que l'invite de commandes d'Anaconda ne trouve que si cet environnement est actif. `recette` ne contient que ce que le TD 1h y a mis, Python, markdown et `tabulate`, et le projet lui-même.

VSCode est un client, installé une fois pour toutes sur le poste. Pour exécuter un notebook avec `recette`, il manque à cet environnement le noyau, ipykernel.

Étape 2 : le client et le noyau au même endroit

	Ce qu'on fait	Ce qu'on constate
4	<code>conda install -n recette -c conda-forge jupyterlab</code>	ipykernel s'installe avec JupyterLab, sans avoir été demandé
5	<code>conda activate recette</code> , puis <code>jupyter lab</code>	une adresse localhost s'affiche, et le client démarre
6	ouvrir <code>altitudes.ipynb</code> , lire le noyau en haut à droite	Python 3 (ipykernel), celui de recette
7	fermer l'onglet, puis Ctrl + C deux fois	le serveur s'arrête, et la page ne répond plus

Le paquet `jupyterlab` déclare `ipykernel` parmi ses dépendances : en installer un installe l'autre, avec plusieurs dizaines de paquets. C'est une dépendance transitive, comme au TD 1h, et l'environnement obtenu reproduit à la main la configuration du base d'Anaconda.

`jupyter lab` lance un serveur sur la machine elle-même, que `localhost` désigne, et le navigateur s'y connecte. Anaconda Navigator effectuait ces deux étapes, activer l'environnement et lancer JupyterLab, pour l'environnement choisi en haut de sa page d'accueil.

Le serveur s'exécute tant que l'invite de commandes d'Anaconda qui l'a lancé est ouverte. Fermer l'onglet du navigateur ne l'arrête pas : un serveur resté actif conserve son port, et le JupyterLab lancé ensuite en prend un autre, ou échoue.

Cette façon de faire est la plus simple, et elle suffit sur un poste de TP. Elle installe un client par projet : cinq projets nécessitent cinq installations de JupyterLab, et autant de mises à jour.

Étape 3 : le client dans un environnement, le noyau dans un autre

	Ce qu'on fait	Ce qu'on constate
8	<code>conda create -n altitudes -c conda-forge python=3.12 ipykernel numpy</code>	
9	<code>conda run -n altitudes python -m ipykernel install --user -- name altitudes</code>	Installed kernelspec altitudes
10	<code>conda activate recette, puis jupyter kernelspec list</code>	deux noyaux, dont altitudes, qui n'est pas dans recette
11	choisir le noyau altitudes, taper <code>import numpy</code>	l'import passe, alors que recette n'a pas numpy

Le client ne cherche pas les environnements conda : il lit un dossier de déclarations de noyaux. `ipykernel install` y a écrit un `kernel.json`, dont l'information utile est un chemin, celui de l'interpréteur d'altitudes, qui se termine par `envs\altitudes\python.exe`. altitudes n'a pas JupyterLab, et n'en a pas besoin.

À l'étape 11, le client s'exécute dans `recette` et le code s'exécute dans `altitudes` : `sys.executable` affiche l'interpréteur d'altitudes. Les deux programmes sont dans deux environnements différents. VSCode, qui est aussi un client, lit la même liste de déclarations : c'est pour cette raison qu'il demande de choisir un noyau à l'ouverture d'un notebook. Cette façon de faire, un client installé une fois et un noyau par projet, est courante en entreprise.

Une bibliothèque qui manque à un notebook s'installe dans l'environnement du noyau, jamais dans celui du client. Le `ModuleNotFoundError` du TD 1h se retrouve dans un notebook quand cette règle n'est pas suivie.

Étape 4 : le notebook du projet recette

	Ce qu'on fait	Ce qu'on constate
12	lire la première cellule	un chemin relatif, qui remonte d'un dossier puis descend dans celui du TD 1h
13	commenter chaque ligne, puis exécuter	les fichiers du dossier de données sont listés
14	redémarrer le noyau et tout exécuter	la recette s'affiche mise en forme, tableau compris
15	changer PERSONNES, puis tout ré-exécuter	les quantités changent, la recette non

Le chemin `../1h_recette/depart/recette/data` part du dossier du notebook, `1i_noyaux`, remonte d'un dossier par `..`, jusqu'à `cours1`, puis descend dans les données du TD 1h. C'est un chemin relatif, au sens de la partie 1 et du TD 1d. Il ne fonctionne que si l'arborescence de l'archive n'a pas été modifiée. La cellule affiche `['ingredients.csv', 'recette.md', 'style.css']` : le notebook lit les deux premiers, et `style.css` sert au programme du TD 1h pour sa page HTML.

L'assert arrête le notebook dès la première cellule si le dossier n'existe pas, avec un message qui donne le chemin complet cherché. Sans lui, l'erreur serait un `FileNotFoundError` deux cellules plus bas, à la lecture du premier fichier, loin de sa cause.

Redémarrer le noyau efface tout ce qu'il gardait en mémoire : les variables et les fonctions définies. Tout réexécuter depuis la première cellule vérifie que le notebook fonctionne dans l'ordre où il est écrit, et pas seulement dans l'ordre où on a exécuté ses cellules. La dernière cellule emploie `ingredients` et `source`, calculés plus haut : la relancer seule, après avoir changé `PERSONNES`, afficherait l'ancienne recette.

L'avant-dernière cellule affiche le texte Markdown, barres verticales comprises ; la dernière le fait afficher mis en forme par le notebook, avec Markdown, importé d'`IPython.display`. Pour quatre personnes, le tableau donne 240 g de farine, 500 ml de lait, 4 œufs, 4 g de sel et 48 g de beurre fondu. Le fichier `ingredients.csv` donne les quantités pour une personne, et le notebook les multiplie ; le texte de la préparation, lu dans `recette.md`, ne change pas.

Le noyau à choisir est celui de `recette`, l'environnement du projet. Ce notebook réécrit ses fonctions au lieu de les importer, y compris une version réduite de `tabulate`, et n'emploie que la bibliothèque standard et `IPython`. Il s'exécute donc aussi dans le navigateur, avec `JupyterLite`, en y déposant `ingredients.csv` et `recette.md` et en remplaçant le chemin par `Path("." )`.