

MODULE

Introduction à l'informatique

Objectifs, contenu et organisation du module

1re année géomatique

15 septembre

Objectif du cours

Consolider ou acquérir les bases informatiques nécessaires aux autres enseignements, en particulier ceux de programmation et les TD utilisant Python.

Demandé dans les autres modules et cours	(re)vu dans ce module
« ouvrez le projet fourni »	travailler dans un éditeur de code, lire une arborescence
« installez Python et numpy »	créer un environnement et le réinstaller ailleurs
« le script lit <code>donnees.csv</code> »	manipuler des fichiers depuis Python
« rendez votre code »	versionner avec git, partager un dépôt, documentation README.md

Objectifs du module liés à la programmation

Maîtriser la forme d'un projet de code : documentation, organisation des fichiers, bibliothèques et environnements.

	ce cours	cours de programmation
Quel algorithme choisir ?		oui
Comment écrire cette boucle ?		oui
Où mettre ce fichier ?	oui	
Comment lancer le script ailleurs ?	oui	

Les trois compétences du module

Éditer

éditeur de code, arborescence de projet

Versionner

git : enregistrer, revenir, partager

Structurer

README, environnement, ligne de commande

Des notions de culture informatique au fil de l'eau, rattachées aux enseignements et projets en cours.

Culture informatique

ordres de grandeur, sécurité informatique, du programme au logiciel

Organisation : sept séances de deux heures

Chaque séance alterne explications courtes et TD sur machine.

	Sujet	Type
1	Logiciel, programmation et formats de fichier	cours
2	Ligne de commande et git local	cours
3	Binaire, données et construction d'une CLI	cours
4	Premier projet : les séances 1 à 3 appliquées	projet
5	Matériel, réseau, SSH et secrets	cours
6	Forge et git en équipe	cours
7	Second projet : tout le module appliqué	projet

Attention. Une partie de la séance 1 reprend des notions surement vues au lycée et/ou prépa.

Cours 1

Introduction à l'informatique

1re année géomatique
15 septembre

Les fichiers du cours

Une archive par séance, un dossier par TD. Aujourd'hui, tout est dans `cours1/`.

```
cours1/          un dossier par TD, dans l'ordre de la séance
├── 1a_formats/
├── 1b_archive_odt/ facultatif
├── 1c_vscode_python/
├── 1d_erreurs/
├── 1e_hello_cpp/  facultatif
├── 1f_markdown/
├── 1g_notebooks/
├── 1h_recette/
├── 1i_noyaux/     facultatif
└── 1j_trajet/     facultatif
```

Dans chaque dossier, la feuille du TD en PDF.

Contenu de la séance

Quatre parties, des notions les plus générales jusqu'à la structure d'un projet Python.

Partie	Nature	Durée
Logiciels et formats de fichier	cours et TD 1a	25'
Programmation et éditeur de code	cours et TD 1c, 1d	35'
Markdown et notebook	cours et TD 1f, 1g	30'
Bibliothèques et environnements Python	cours et TD 1h	30'

Attention. La première partie de révision avance vite : posez vos questions tout de suite, tout ce qui suit s'appuie dessus.

Durées indicatives. Les TD 1b, 1e, 1i et 1j sont facultatifs.

Logiciels et formats de fichier

Ce qu'un logiciel fait, où il s'exécute, notions de base sur la manipulation et rôles des fichiers.

Logiciel : définition et termes courants

Définition : logiciel

Ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de données.

Source : *Journal officiel* du 22/09/2000 (vocabulaire de l'informatique). Le seul de ces mots à avoir une définition officielle.

Autres termes courants, qui désignent chacun un type de logiciel.

application

app

webapp

OS

driver

Les définitions d'usage : Grand dictionnaire terminologique de l'OQLF.

Le système d'exploitation

Un programme ne s'adresse pas directement au matériel : il passe par le système.



Vos programmes

LibreOffice, un navigateur, votre script

« ouvre releve.csv » ↓ ↑ le contenu



Système d'exploitation

Windows, macOS, Linux

« écris ces octets » ↓ ↑ les octets lus sur le disque



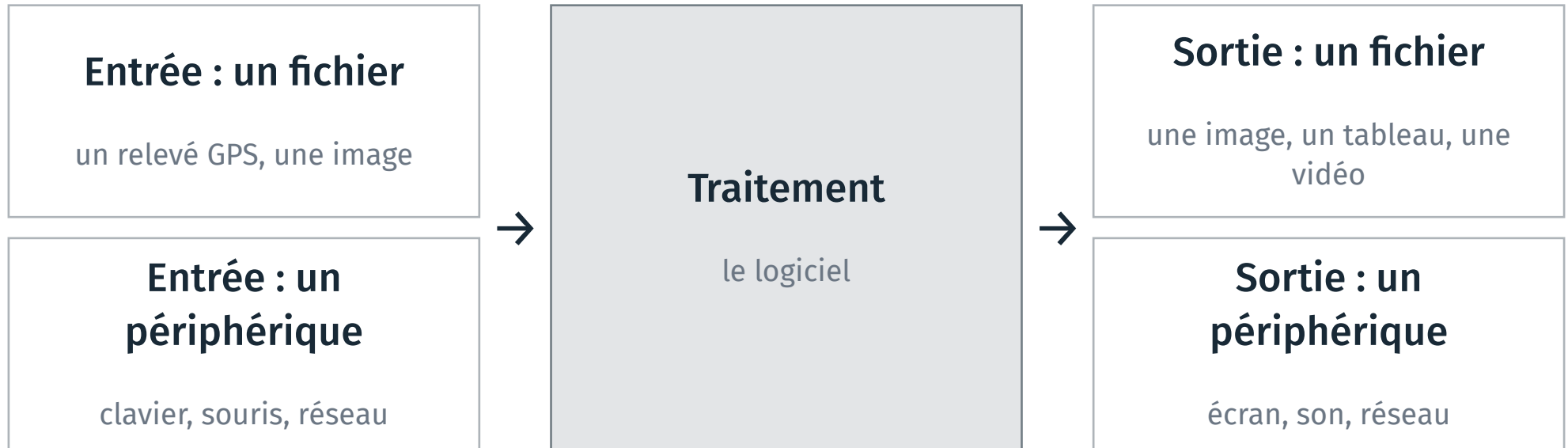
Matériel

processeur, mémoire, disque, réseau

Quel système d'exploitation tourne sur votre téléphone ?

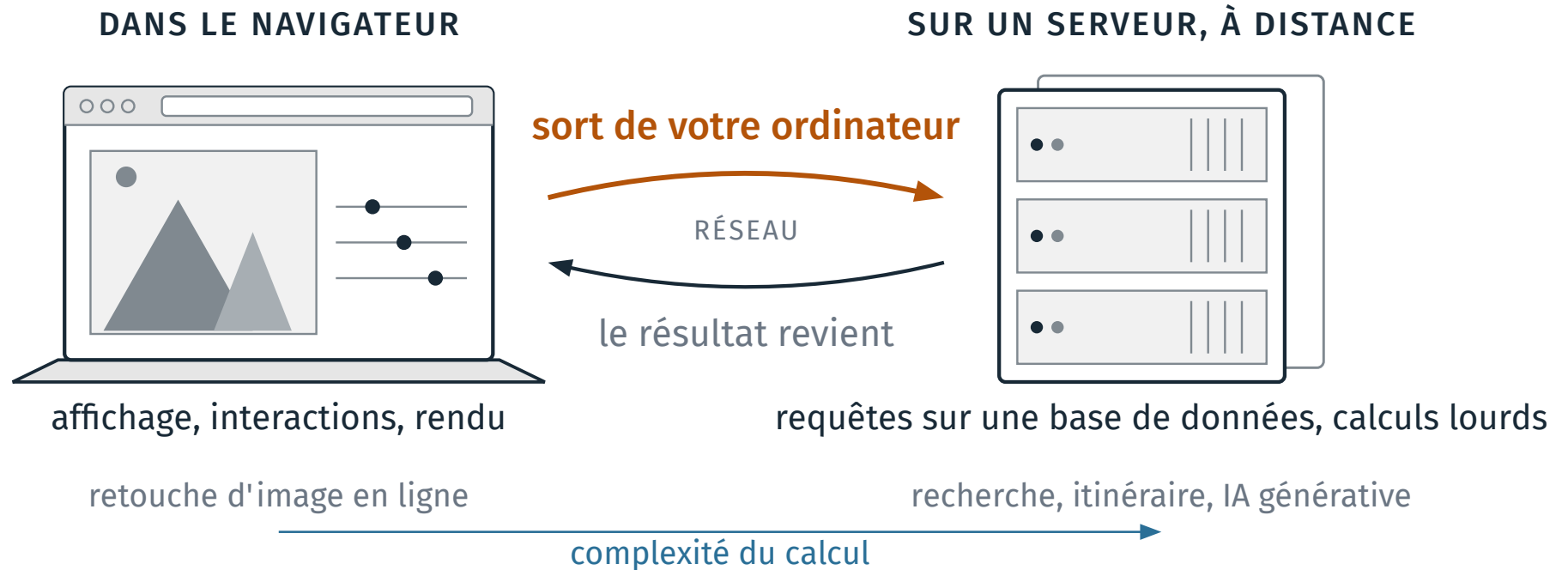
Entrées et sorties d'un logiciel

Ce qu'un logiciel reçoit et ce qu'il produit sont de deux natures : un fichier, qui se conserve, ou un flux vers un périphérique, qui ne garde rien.



Un fichier sert à conserver un résultat et à l'échanger : avec un autre logiciel, avec une autre machine, ou avec quelqu'un d'autre.

Où s'exécute une application web ?



Utilisation / utilité d'un fichier

Un fichier conserve un résultat après l'arrêt du programme : un état de travail à reprendre, ou un document final à transmettre.

Ce qu'il permet	Quand cela vous servira
Conserver un résultat	relire dans une semaine ce que le programme a produit
Passer d'un logiciel à l'autre	le tableau écrit par l'un, ouvert par l'autre
Changer de machine	de votre poste à celui de la salle, et retour
Le remettre à quelqu'un	un rendu, ou le dépôt partagé du cours 2

D'où la suite de cette partie : nommer un fichier, reconnaître son type, et savoir le désigner par son chemin.

Quizz : vocabulaire associé aux chemins de fichier

Un chemin dit où trouver un fichier dans l'arborescence des dossiers. Plusieurs mots en désignent les parties : dites à quoi chacun correspond dans cet exemple.

`C:\Users\alice\Documents\raven.odt`

Le mot	Ce qu'il désigne dans l'exemple
la racine, ou le disque	
un nom de dossier	
le nom du fichier	
le chemin du fichier	
le dossier parent	

Quizz : vocabulaire associé aux chemins de fichier - Réponse

`C:\Users\alice\Documents\raven.odt`

la racine, ou le disque

trois noms de dossier

le nom du fichier, extension comprise

Le mot	Ce qu'il désigne dans l'exemple
la racine, ou le disque	c:\, le point de départ ; / sous macOS et Linux
un nom de dossier	Users, alice, Documents : trois, du plus large au plus précis
le nom du fichier	raven.odt, extension comprise
le chemin du fichier	tout, de la racine au fichier
le dossier parent	C:\Users\alice\Documents, le dossier qui le contient

Un chemin se lit de gauche à droite, de la racine au fichier ; chaque séparateur descend d'un dossier.

Le chemin d'un fichier

Un chemin **absolu** part de la racine, un chemin **relatif** du dossier où l'on se trouve.

	Windows	macOS et Linux
Le séparateur de dossiers	la barre inversée \, <i>backslash</i>	la barre /, <i>slash</i>

<pre>C:\Users\alice\ └─ cours1\ └─ 1a_formats\ └─ raven.odt └─ 1d_erreurs\ └─ chemin.py</pre>	Le chemin de raven.odt		Depuis
	Absolu	C:\Users\alice\cours1\1a_formats\raven.odt	la racine
	Relatif	1a_formats\raven.odt	cours1
	Relatif qui remonte	..\1a_formats\raven.odt	1d_erreurs

Sous macOS et Linux, l'autre séparateur : /home/alice/cours1/1a_formats/raven.odt. Un projet qui n'écrit que des chemins relatifs se copie, se déplace et s'envoie sans rien changer : C:\Users\alice n'existe que sur un poste.

Fichier, extension et type de fichier

Le type d'un fichier (texte, vidéo...) est indiqué par son **extension**, la fin du nom après le dernier point. Le système s'en sert pour choisir le logiciel à lancer, mais elle reste une indication sur le contenu, pas une garantie.

releve_2026.csv
le nom, que vous choisissez l'extension

Attention. Windows masque les extensions qu'il connaît : raven.odt s'affiche raven.
Réglage à changer une fois, avant le TD 1a.

L'extension	Ce qu'elle décide
ce qu'elle fait	le système choisit le logiciel à lancer au double-clic
ce qu'elle ne fait pas	elle ne modifie aucun octet du fichier
comment on la change	en renommant le fichier, comme le reste du nom

Fichiers binaires et fichiers texte

Tout fichier est une suite de **bits**, 0 ou 1, comptés par **octets** de huit : 256 valeurs, représentées par deux chiffres hexadécimaux. Fichier **texte** si chaque octet est un caractère, **binaire** sinon.



	Fichier texte	Fichier binaire
Ses octets	des caractères, tous	ce que le format décide
Qui le lit	n'importe quel éditeur de texte	le logiciel qui connaît le format
Ce qu'on en fait	lire, comparer, versionner	l'ouvrir dans son logiciel

$2^8 = 256$; seize chiffres, de 0 à F. Le binaire est repris au cours 3.

Le fichier texte

Un fichier texte est une suite de caractères, chacun rangé dans un octet selon une table d'**encodage**. Il s'ouvre avec un éditeur de texte, qui montre tout ce qu'il contient.

raven_une_ligne.txt, les huit premiers octets

Ce qui l'ouvre

```
4F 6E 63 65 20 75 70 6F
O n c e _ u p o
```

Bloc-notes, Notepad++, l'éditeur de code du module — pas un traitement de texte, qui ajoute sa mise en forme

La table	Ce qu'elle code	Où on la voit
ASCII	128 caractères, l'anglais sans accent : 0 vaut 4F	en entier, au TD 1a
UTF-8	toutes les langues, ASCII inclus ; é prend deux octets	la suite du cours

Octets réels, relevés sur le fichier du TD 1a.

Reconnaître un format à son extension

Pour chacune de ces extensions, dites de quel type de contenu il s'agit, et si le fichier est lisible dans un éditeur de texte.

.mp3	.mp4	.jpg	.png
.svg	.tif	.pdf	.odt
.xlsx	.csv	.zip	.exe
.py	.md	.json	.yaml

Reconnaître un format à son extension — réponses

.mp3 son, avec perte	.mp4 vidéo, la plus courante	.jpg photo, avec perte	.png image, sans perte
.svg image vectorielle : texte	.tif image, y compris GeoTIFF	.pdf document mis en page	.odt LibreOffice, archive ZIP
.xlsx Excel, archive ZIP	.csv tableau : du texte	.zip archive de fichiers	.exe programme Windows
.py code Python : du texte	.md documentation : du texte	.json données, réglages : texte	.yaml réglages : du texte

Six de ces seize formats sont du texte : ceux qu'on peut ouvrir dans un éditeur, comparer ligne à ligne et versionner. En bleu, les quatre que vous écrirez vous-mêmes dans ce module.

TRAVAUX DIRIGÉS

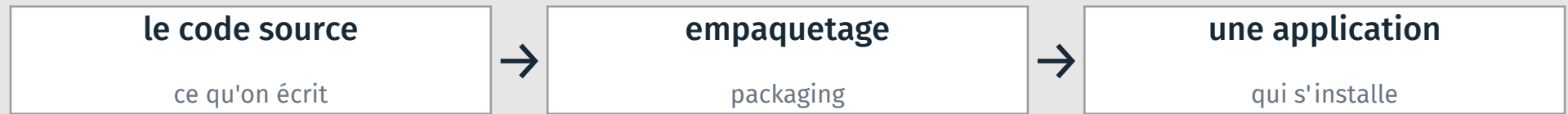
- | | | |
|--------------|--|-------|
| TD 1a | Fichiers, formats et extensions
cours1/1a_formats/ | ≈ 20' |
| TD 1b | Un .odt est une archive ZIP — facultatif
cours1/1b_archive_odt/ | ≈ 12' |

Programmation et éditeur de code

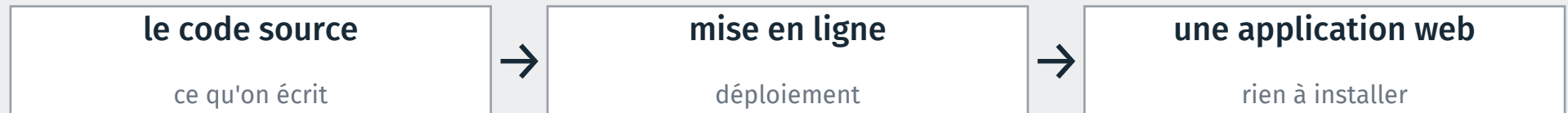
D'un programme à une application

Une distinction entre un programme et un logiciel, au sens classique, est la façon dont ils sont distribués et ce que doit faire un utilisateur pour arriver à s'en servir.

Distribution



Déploiement



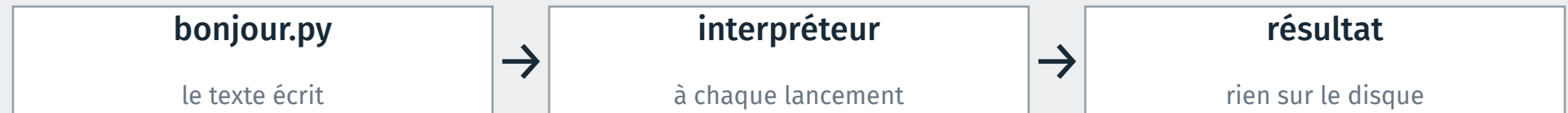
C'est l'empaquetage qui change, pas le programme : le même code se distribue en application à installer, ou se déploie en application web.

Deux chemins du texte à l'exécution

Compilé : le texte est traduit une fois pour toutes. À chaque lancement il n'y a plus rien à comprendre, donc c'est plus rapide.



Interprété : le texte est lu et exécuté à chaque lancement. Comprendre le code est donc refait à chaque fois.



Lancer un programme Python ne crée rien sur le disque : il n'y a pas d'exécutable à produire.

Du code source aux instructions machine

Compiler enchaîne quatre étapes. Au bout, des instructions que le processeur exécute telles quelles.



Ce qu'a écrit un humain

```
int largeur = 1920;  
int hauteur = 1080;  
return largeur * hauteur;
```

Ce que le processeur exécute

```
mov    DWORD PTR -8[rbp], 1920  
mov    DWORD PTR -4[rbp], 1080  
mov    eax, DWORD PTR -8[rbp]  
imul   eax, DWORD PTR -4[rbp]
```

Une instruction machine fait une opération élémentaire : `mov` copie une valeur, `imul` en multiplie deux. Le processeur les exécute telles quelles, là où un interpréteur doit d'abord lire et analyser la ligne.

La place de l'interpréteur

L'interpréteur lit le texte du programme, et c'est lui qui s'adresse au système. Un programme compilé s'en passe : il est déjà en instructions machine.



Programme interprété

bonjour.py, une page web

son texte ↓ ↑ le résultat



Interpréteur : traduit en bytecode, puis l'exécute

python, le navigateur

un appel système : open, read ↓ ↑ des octets



Système d'exploitation

Windows, macOS, Linux

Le bytecode n'est pas des instructions machine : l'interpréteur l'exécute lui-même.

Les fonctions d'un IDE

IDE, pour *integrated development environment* : un logiciel qui réunit des fonctions pour aider à l'écriture, test et partage de code.

La fonction	Ce que l'éditeur en fournit
Écrire le code	coloration, indentation, complétion, soulignement des fautes
Le lancer et le tester	un terminal intégré et un bouton d'exécution, sans quitter la fenêtre
Naviguer dans le projet	l'arborescence à gauche, la recherche dans tous les fichiers
Déboguer	exécuter pas à pas, arrêter sur une ligne, lire les variables
Connaître le langage	certain IDE n'en servent qu'un ; d'autres s'étendent par extensions

Attention. Un IDE ne contient pas forcément l'interpréteur ni le compilateur. Ils s'installent à part, et se configurent pour chaque langage et chaque système.

Les fonctions d'édition de texte d'un IDE

Programmer nécessite d'éditer des fichiers texte sans “faute”. L'éditeur sert à rendre cela plus simple et rapide.

Dans un éditeur de texte ordinaire	Dans un éditeur de code
une faute de frappe se découvre à l'exécution	elle est soulignée pendant la frappe
on cherche un fichier dans l'explorateur	l'arborescence et la recherche sont dans la fenêtre
une indentation fausse ne se voit pas	les espaces s'affichent

La colonne de droite est ce que cette partie détaille, ligne après ligne.

Texte brut et règles du langage

Un langage a une syntaxe définie. L'éditeur la connaît, par une extension, et colore chaque catégorie de mot sans rien ajouter au fichier.

Enregistré par un traitement de texte

```
<text:p text:style-name="P1">
altitude = 128.4</text:p>
<text:p>print(altitude)</text:p>
```

Le fichier d'un programme

```
altitude = 128.4
print(altitude)
```

Affiché par l'éditeur de code

```
altitude = 128.4
print(altitude)
```

Il souligne de même ce qui ne suit pas ces règles, sans rien lancer.

```
altitudes = [128.4, 131.0]
for altitude in altitudes
    print(altitude)
```

Le deux-points manque. Sans l'éditeur, la faute n'apparaîtrait qu'au lancement : `SyntaxError: expected ':'`.

Chasse fixe et chasse proportionnelle

Un éditeur de code emploie une police à chasse fixe, où toutes les lettres ont la même largeur. Le même programme dans la police d'un traitement de texte :

Chasse fixe (éditeur de code)

```
def surface(longueur, largeur):  
    aire = longueur * largeur  
    if aire > 250:  
        categorie = "grande"  
    else:  
        categorie = "petite"  
    return aire, categorie
```

Chasse proportionnelle (traitement de texte)

```
def surface(longueur, largeur):  
    aire = longueur * largeur  
    if aire > 250:  
        categorie = "grande"  
    else:  
        categorie = "petite"  
    return aire, categorie
```

Même texte, même taille, deux polices. À droite, rien ne dit de combien chaque ligne est décalée.

L'indentation, en espaces ou en tabulation

Un espace et une tabulation sont deux caractères différents. Une tabulation vaut le nombre de colonnes que l'éditeur lui donne, et ce réglage change d'un éditeur à l'autre.

Tabulation réglée sur 4 colonnes

```
def surface(longueur, largeur):  
    ....aire = longueur * largeur  
→    return aire
```

les deux lignes semblent alignées

· marque un espace, → une tabulation, comme l'éditeur les dessine. Les octets du fichier sont les mêmes des deux côtés : seul le réglage de l'éditeur change.

Le même fichier, tabulation sur 8

```
def surface(longueur, largeur):  
    ....aire = longueur * largeur  
→        return aire
```

le décalage apparaît

TRAVAUX DIRIGÉS

- | | | |
|--------------|---|-------|
| TD 1c | Configurer l'éditeur de code, et lancer un programme
<code>cours1/1c_vscode_python/</code> | ≈ 25' |
| TD 1d | Trois programmes fautifs
<code>cours1/1d_erreurs/</code> | ≈ 10' |
| TD 1e | Le même programme en C++ — facultatif
<code>cours1/1e_hello_cpp/</code> | ≈ 10' |

Markdown et notebook

Le format de texte dans lequel s'écrit la documentation d'un projet, et le document qui en fait un carnet exécutable

Les fichiers texte d'un projet

Le code n'est pas le seul texte d'un projet. Ses réglages et sa documentation s'écrivent aussi en texte, dans le même éditeur.

Fichier	Ce qu'il porte	Qui le lit
.py	les instructions du programme	l'interpréteur
.md	la documentation, les notes, le README	un humain
.toml, .yaml	les réglages du projet et ses dépendances	un outil
.csv	un petit jeu d'essai, pour vérifier que le programme marche	un programme

Attention. Les données de travail ne sont pas des fichiers du projet. Elles sont stockées ailleurs, et le programme y accède par un chemin de fichier. Un projet ne contient qu'un petit jeu de données, pour le tester.

Le format de la documentation

Un `.txt` n'a aucune mise en forme, un `.odt` en a mais se prête mal aux outils du code. Markdown est du texte brut où quelques signes portent la mise en forme, que l'éditeur sait rendre.

Ce qu'on écrit, `README.md`

Trajet

Trace le trajet de la gare à l'école.

Lancer

```
python trajet.py
```

Le résultat est **trajet.png**.

Ce que l'aperçu montre

Trajet

Trace le trajet de la gare à l'école.

Lancer

```
python trajet.py
```

Le résultat est *trajet.png*.

Moins de possibilités qu'un traitement de texte. En échange : l'éditeur, la comparaison ligne à ligne, le versionnement, et une conversion quand il en faut une.

L'intention de Markdown

John Gruber, 2004 : un format de texte facile à lire et à écrire, convertible en HTML, et publiable tel quel sans avoir l'air balisé.

Le fichier .md

Crêpes

1 heure de repos.

1. Mélanger la farine
2. Casser les ****œufs****

Les deux produisent le même affichage. Seul celui de gauche se lit sans être converti.

Le même contenu en HTML

```
<h1>Crêpes</h1>
<p><em>1 heure de repos.</em></p>
<ol><li>Mélanger la farine</li>
<li>Casser les <strong>œufs</strong>
</li></ol>
```

La syntaxe de Markdown

Une dizaine de marques suffisent, et le texte reste lisible sans être rendu : le dièse annonce un titre, le tiret une puce, les astérisques une emphase.

Ce qu'on écrit

Un titre

Un sous-titre

Du texte, de l'**emphase**,
du ****gras****.

- une puce
- 1. une étape

[un lien](<https://typst.app>)
![une photo](poele.jpg)

Syntaxe complète, publiée par Gruber : daringfireball.net/projects/markdown/syntax.

Ce qui s'affiche

Un titre

Un sous-titre

Du texte, de l'*emphase*, du **gras**.

- une puce
- 1. une étape

un lien

☐ une photo

Programmation littérale

Le texte, le code et son résultat tiennent dans un seul document.

trajet.ipynb

Longueur du trajet

texte

[1] import numpy as np
points = np.loadtxt("trajet.csv")
print(points.shape)

code

[1]: (128, 2)

résultat

et ainsi de suite

Trois sortes de blocs, dans l'ordre où on les écrit. Le terme est de Donald Knuth, 1984.

Le bloc de texte : du Markdown

Le bloc de texte s'écrit en Markdown, et s'affiche mis en forme.

Ce qu'on tape dans le bloc

Longueur d'un trajet

Les points du trajet sont donnés en
****coordonnées projetées****, en mètres.

Le # fait un titre, les deux astérisques mettent en gras : rien de neuf depuis le TD 1f, Markdown.

Ce que le notebook affiche

Longueur d'un trajet

Les points du trajet sont donnés en **coordonnées projetées**, en mètres.

Un notebook dans JupyterLab

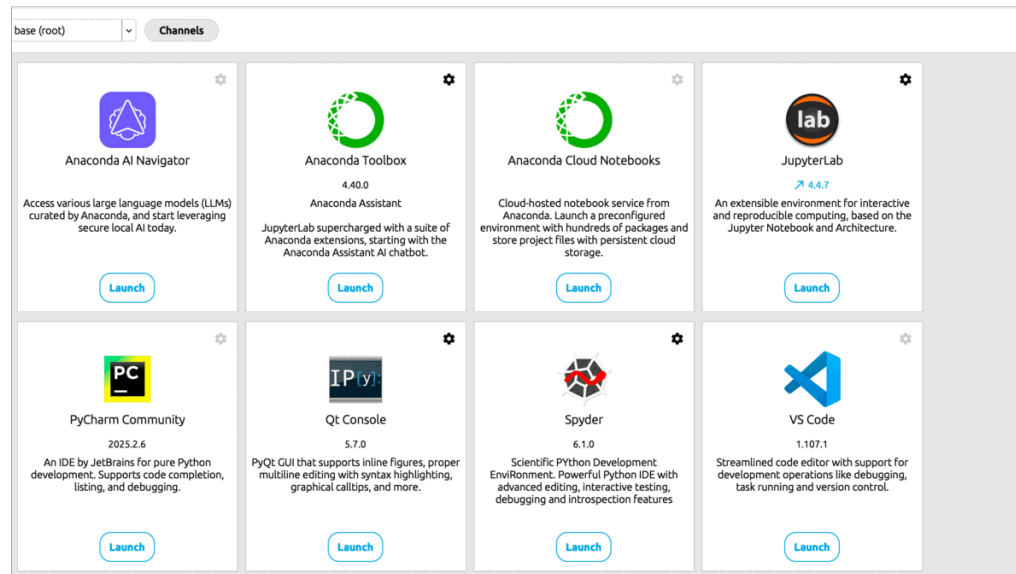
Les trois sortes de blocs dans une vraie fenêtre.



Capture réelle. Le code y calcule la longueur d'un trajet de quatre points.

Lancer JupyterLab depuis Anaconda

JupyterLab est livré avec Anaconda, et se lance depuis la page d'accueil de Navigator, comme VS Code au TD 1c.



La même page qu'au TD 1c, une fiche plus loin. Le numéro sous le nom est la version installée : il n'y a rien à installer.

TRAVAUX DIRIGÉS

- | | | |
|--------------|--|-------|
| TD 1f | Mettre en forme une recette en Markdown
cours1/1f_markdown/ | ≈ 20' |
| TD 1g | Le notebook, ouvert de trois façons
cours1/1g_notebooks/ | ≈ 12' |

Bibliothèques et environnements Python

Ce dont le code d'un projet dépend, où cela s'installe, et comment un notebook désigne le Python qui l'exécute

Un programme s'appuie sur des bibliothèques

Un programme ne réécrit pas tout à partir de zéro. Il appelle des fonctions déjà écrites, regroupées en **bibliothèques**, qui ont trois origines.

Livrées avec Python

la bibliothèque standard : math,
csv, pathlib, json...

Écrites pour le projet

vos propres fichiers, importés les
uns par les autres

Publiées par d'autres

numpy, pillow, markdown... à
installer depuis un dépôt de
paquets

Ce qu'on écrit	Ce que cela demande
<code>import csv</code>	rien : installé avec l'interpréteur
<code>from recette.calculs import adapter</code>	que le projet ait été installé
<code>import markdown</code>	que la bibliothèque ait été installée avant

Les diapositives suivantes le montrent sur un programme réel.

L'import d'une bibliothèque et ses conditions

Une bibliothèque se déclare par `import`, en tête du programme, avant tout appel à ses fonctions. Si elle n'est pas disponible, l'import échoue et le programme s'arrête sur un `ModuleNotFoundError`.

Livrées avec Python

```
import math
print(math.sqrt(2))
```

1.4142135623730951

`math`, `csv`, `pathlib`, `json` : environ deux cents modules, installés en même temps que l'interpréteur.

Sorties réelles dans un environnement réduit à Python. C'est la colonne de droite qui rend nécessaire tout ce qui suit : installer, puis décrire ce qu'on a installé.

Publiées par d'autres

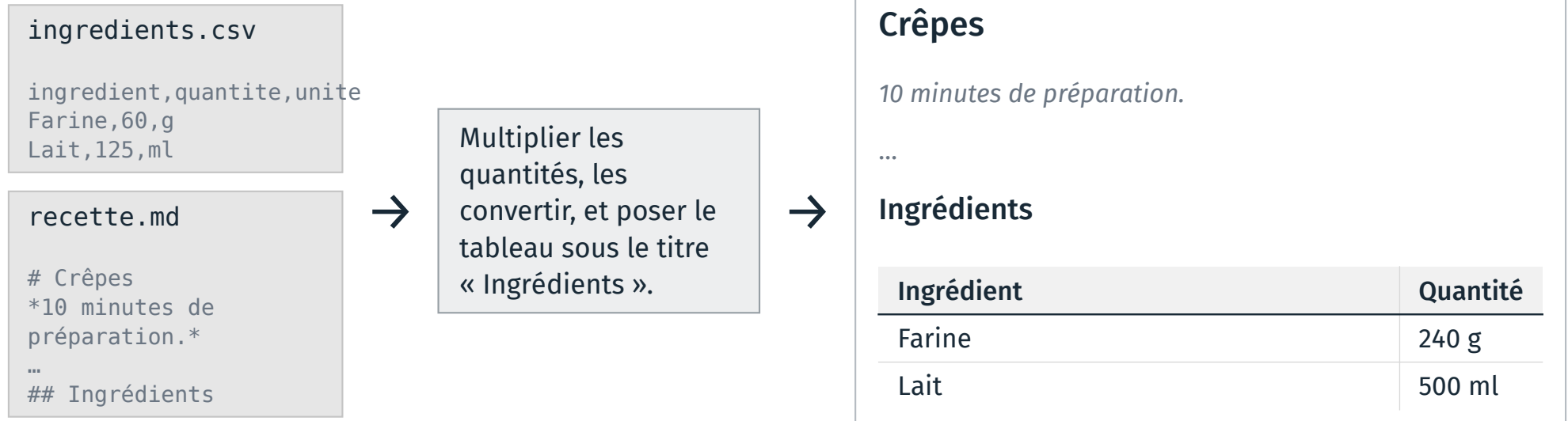
```
import numpy
```

`ModuleNotFoundError:`
`No module named 'numpy'`

`numpy`, `pillow`, `markdown` : il faut les installer, et savoir lesquelles.

Exemple de programme : l'objectif

Deux fichiers en entrée, la description de la recette et les quantités pour une personne en unités SI ; en sortie, une page de recette HTML avec les quantités adaptées au nombre de personnes voulu et au type d'unités de mesure.



Pour quatre personnes : les 60 g du fichier en font 240. Le tableau de droite n'existe dans aucun des deux fichiers d'entrée, il est calculé.

Exemple de programme : le code

En s'appuyant sur des bibliothèques existantes, le programme tient en neuf lignes. Le commentaire de chaque ligne en donne l'origine.

```
import markdown                                # installée
from tabulate import tabulate                 # installée
from recette.calculs import adapter, en_table, lire_ingredients # le projet

ingredients = lire_ingredients(DONNEES / "ingredients.csv") # le projet
ingredients = adapter(ingredients, PERSONNES, UNITES)      # le projet
lignes = en_table(ingredients)                          # le projet

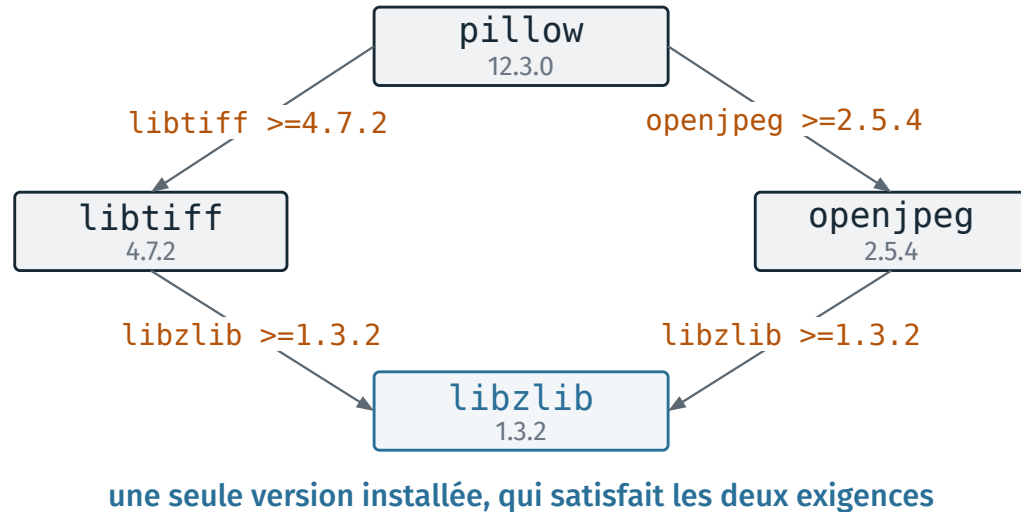
tableau = tabulate(lignes, headers=EN_TETE)               # installée
source = (DONNEES / "recette.md").read_text(encoding="utf-8") # avec Python
source = source.replace("## Ingrédients", tableau)        # Python seul

corps = markdown.markdown(source, extensions=["tables"])  # installée
page = GABARIT.format(titre="Crêpes", corps=corps)        # Python seul
sortie.write_text(page, encoding="utf-8")                 # avec Python
```

Deux lignes seulement demandent une bibliothèque installée. `source.replace` est resserrée ici : elle garde le titre.

Installation de bibliothèques : les dépendances

Une bibliothèque en réclame d'autres. Les installer demande d'abord la liste complète, puis une version par paquet qui convienne à tous.



À gauche : les paquets écrits dans le fichier sont les dépendances *directes*, celles qu'ils entraînent sont *transitives*. À droite : `pillow` et ses deux dépendances réclament le même `libzlib`, dont une seule version sera installée. Sept bibliothèques demandées, 293 paquets installés, relevés les 8 et 10 septembre 2026.

Le numéro de version d'une dépendance

D'une version à l'autre, une bibliothèque gagne des fonctions et en retire d'autres. Le même code ne passe donc pas partout.

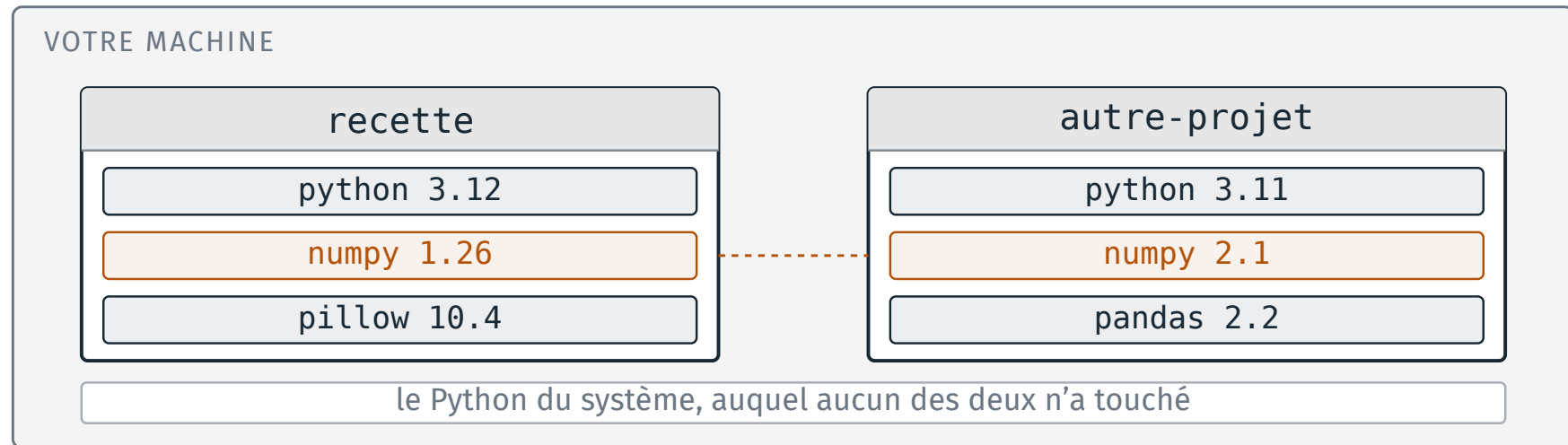
Le code	avec <code>numpy 1.26.4</code>	avec <code>numpy 2.5.2</code>
<code>np.trapezoid([0, 1, 2])</code>	<code>AttributeError: module 'numpy' has no attribute 'trapezoid'</code>	<code>2.0</code>
<code>np.NaN</code>	<code>nan</code>	<code>AttributeError: np.NaN was removed in the NumPy 2.0 release</code>

La valeur n'a pas disparu. En 1.26, `np.nan`, `np.NaN` et `np.NAN` désignent la même chose ; la version 2.0 n'en garde qu'une.

Sorties réelles, relevées le 10 septembre 2026 dans deux environnements. D'où deux exigences opposées : `numpy>=2` pour la première ligne, `numpy<2` pour la seconde tant que le code n'est pas repris.

Environnement : un moyen de gérer les conflits de dépendances

Un projet suit `numpy 1.26`, l'autre `numpy 2.1` : installées ensemble, la seconde chasse la première. D'où un dossier par projet, un **environnement**.



la même bibliothèque, en deux versions, sans qu'elles se croisent

Les deux versions coexistent sans se croiser, et le Python du système n'est pas touché.

Comment installer des librairies python

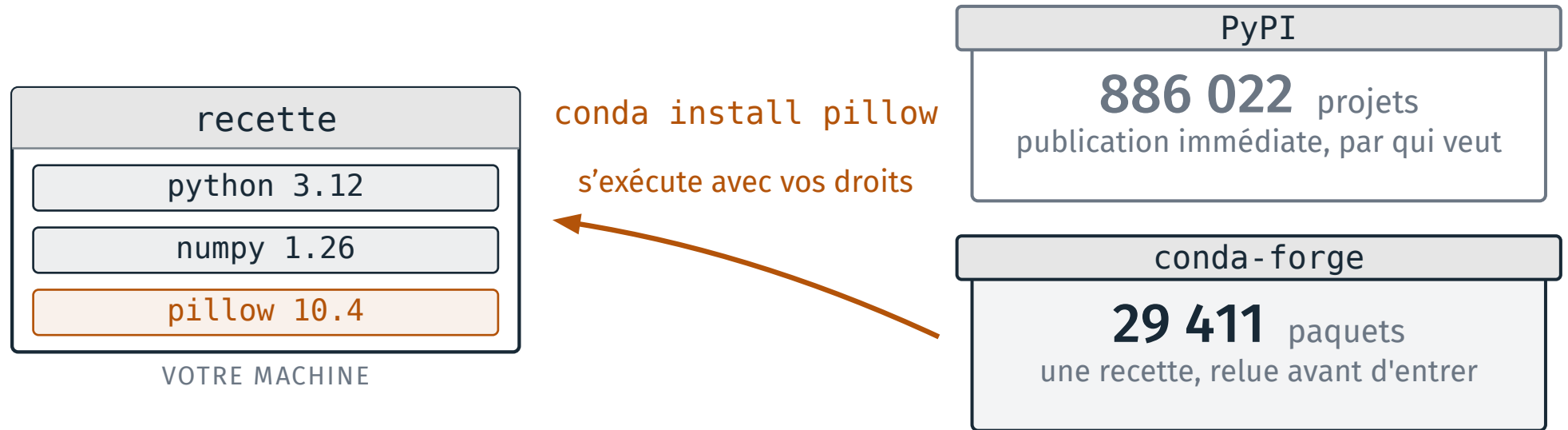
Plusieurs outils existent et l'écosystème évolue encore. Ici on utilise `conda` pour l'environnement et les dépendances, et `pip` pour installer le projet lui-même en local. Vocabulaire : en Python, une librairie est appelée package.



`pyenv`, né la même année que `conda`, n'est pas sur l'axe : il choisit la version de Python et n'installe aucun paquet. Dates de création des dépôts, relevées le 8 septembre 2026 par l'API de GitHub.

D'où viennent les paquets

Une commande d'installation va chercher le paquet dans un dépôt. Les deux que vous croiserez n'y laissent pas entrer la même chose.



Les deux ne s'opposent pas : un paquet conda-forge est le plus souvent construit à partir de PyPI, quelques jours plus tard. Projets relevés le 8 septembre 2026 dans l'index de PyPI, recettes conda-forge le même jour.

Créer un environnement, et le décrire

Deux façons d'obtenir le même environnement. Pour qu'un projet se refasse ailleurs, c'est le fichier qui compte : lui seul voyage avec le code.

Composer l'environnement

```
conda create -n recette \
    -c conda-forge python=3.12
conda install -n recette \
    -c conda-forge markdown tabulate
conda activate recette
```

Rien n'en garde trace : ailleurs, il faut se souvenir de ces lignes.

Le décrire dans un fichier

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

à la racine du projet : `environment.yml`

```
name: recette
channels:
  - conda-forge
dependencies:
  - python=3.12
  - markdown
  - tabulate
```

À défaut de fichier, écrire les commandes de gauche dans le `README`.

Ce que l'activation d'un environnement change

Activer un environnement n'installe rien et ne déplace rien : cela pose un dossier de plus en tête de la liste où le terminal cherche les commandes.

vous tapez `python` et le terminal parcourt `PATH` de gauche à droite

sans environnement actif

`C:\Windows\System32`

→

`C:\Windows`

introuvable, ou un autre python

après `conda activate recette`

`...\envs\recette`

→

`C:\Windows\System32`

→

`C:\Windows`

`...\envs\recette\python.exe`

posé en tête par l'activation

Chemins d'un poste Windows, avant et après `conda activate` ; sous macOS et Linux, `...\envs/recette/bin` passe devant `/usr/bin`. La liste parcourue est la variable `PATH`.

Projet : déclaration des informations & métadonnées

Un projet se décrit dans un fichier texte : son nom, sa version, et ce dont son code a besoin.

`pyproject.toml`, à la racine du projet

```
[project]
name = "recette"
version = "0.1.0"
requires-python = ">=3.10"
dependencies = ["markdown>=3.5"]
```

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

Contenu réel de `pyproject.toml`, écrit en TOML : des sections entre crochets, une valeur par nom. `environment.yml` est en YAML, où l'indentation porte la structure.

Ce que chaque ligne déclare

Ce qui est écrit	Ce que c'est
name, version	les métadonnées
dependencies	ce que le code importe
requires-python	les versions de Python
[project.scripts]	la commande installée

Le terminal de l'éditeur de code

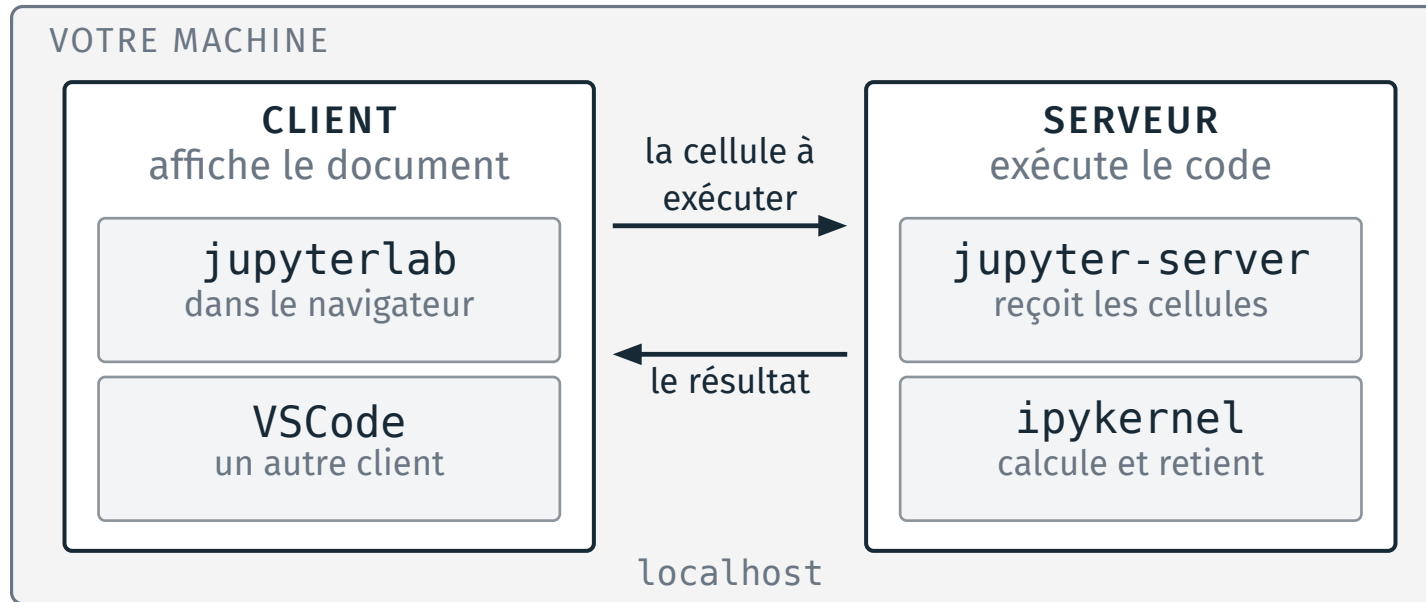
L'éditeur ouvre un terminal dans sa fenêtre, déjà placé dans le dossier du projet.

L'action	Ce qu'elle règle
Terminal → Nouveau terminal	un terminal dans le dossier ouvert
le sélecteur, à droite du panneau	l'interpréteur de commandes : PowerShell, bash, zsh
Ctrl + Maj + P, Python: Select Interpreter	l'environnement activé dans chaque nouveau terminal
la barre d'état, en bas	l'environnement en cours

Le dossier du projet est le dossier courant : c'est de lui que partent les chemins relatifs des commandes.

Le client et le serveur d'un notebook

Un notebook est une application web : un client qui affiche, un serveur qui exécute.



Changer de client ne change pas le noyau : JupyterLab et l'éditeur de code ouvrent le même fichier et parlent au même serveur.

Les trois emplacements du serveur

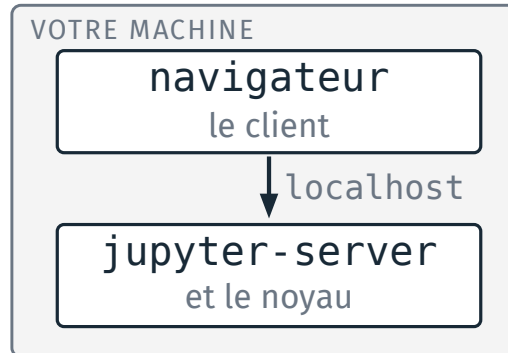
Client et serveur sont deux rôles, pas deux machines.

SUR UN ORDINATEUR DISTANT



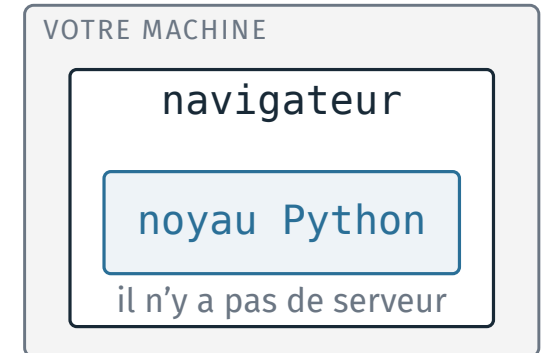
Colab, un serveur du laboratoire

SUR VOTRE ORDINATEUR



jupyter lab, ou l'éditeur de code

DANS LE NAVIGATEUR

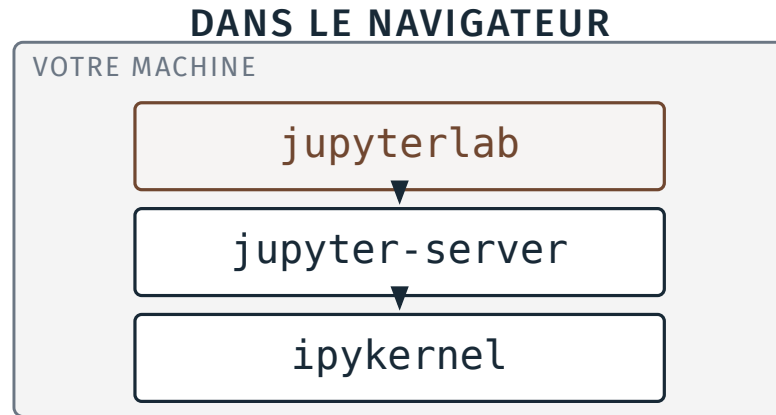


JupyterLite : rien ne sort de l'onglet

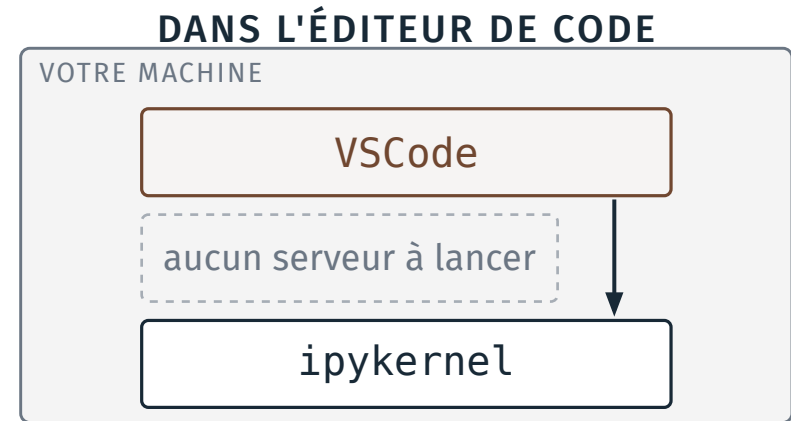
Seul le premier cas fait sortir quelque chose de votre machine. Dans le troisième, le noyau Python est exécuté par le navigateur lui-même.

Les clients d'un notebook

Le même fichier s'ouvre par plusieurs clients. Tous ont besoin du même noyau.



jupyter lab, puis une adresse localhost



l'éditeur démarre le noyau lui-même

L'éditeur de code n'a pas besoin de `jupyterlab` : il démarre `ipykernel` lui-même. `ipykernel` est le noyau Python ; il en existe pour d'autres langages, et le nom *Jupyter* vient de Julia, Python et R.

TRAVAUX DIRIGÉS

- | | | |
|--------------|--|-------|
| TD 1h | Installer un projet Python, et décrire son installation
<code>cours1/1h_recette/</code> | ≈ 20' |
| TD 1i | Le client, le noyau, et où ils sont installés — facultatif
<code>cours1/1i_noyaux/</code> | ≈ 20' |
| TD 1j | Installer un projet en lisant son README — facultatif
<code>cours1/1j_trajet/</code> | ≈ 15' |

À retenir

Un logiciel	transforme une entrée en sortie ; son traitement part d'un texte
Une application	est un logiciel destiné à une tâche ; « app » en est l'abréviation
Une interface	décide de ce qu'il reste du travail, pas de ce qu'il produit
Une extension	nomme le fichier, elle ne dit pas ce qu'il contient
Un format	décide de ce qu'on peut relire, comparer et versionner
Une dépendance	du code écrit par d'autres, réutilisé, qu'il faut installer et déclarer
Un environnement	rend l'outillage reproductible d'un poste à l'autre
Un notebook	un client qui affiche, un noyau qui exécute et qui retient