# Cours 1 – Premiers pas avec Python

Du 0/1 au script : comprendre ce que Python permet de faire

Etienne Dagorn

Université de Lille – L2 Économie

2026–2027

# Pourquoi ce cours existe

## But du cours

Vous rendre capables de produire un petit script fiable, lisible et exécutable dans les TD.

- Pas besoin de tout connaître pour commencer.
- Il faut surtout savoir organiser son dossier, exécuter le bon fichier et lire les erreurs simples.
- Le TD1 transforme ces idées en gestes obligatoires.

# Le parcours du semestre

| Séance | Support | Compétence principale |
|---|---|---|
| Cours 1 | CM | Script, variables, erreurs |
| TD1 | Livret | Installation et premiers scripts |
| TD2 | Livret | Listes, dictionnaires, fonctions |
| Cours 2 | CM | Données, tableaux, description |
| TD3 | Livret | Statistiques et graphiques |
| TD4 | Livret | Graphiques et simulation simple |
| TD5 | Livret | CSV, pandas et mini-projet descriptif |

# La règle de travail

## Un bon script de débutant

Un bon script ne fait pas beaucoup de choses. Il fait peu de choses, mais dans le bon ordre, avec des résultats que l'on peut vérifier.

1. Créer ou charger les données.
2. Calculer une valeur.
3. Afficher ou exporter le résultat.
4. Ajouter un commentaire qui explique l'interprétation.

# Pourquoi une histoire avant le code ?

## Le problème pédagogique

Si l'on commence directement par `print()`, Python semble magique. L'objectif est de comprendre ce que Python cache, et pourquoi ce langage nous aide.

- Les machines ne manipulent pas des "idées" : elles manipulent des représentations.
- Programmer consiste à transformer une méthode en instructions exécutables.
- L'histoire de l'informatique aide à voir que code, machine et calcul sont liés depuis longtemps.

# Repères historiques : calculer, coder, programmer

| Période | Exemple | Idée utile pour nous |
|---|---|---|
| 1804 | Métier Jacquard | des cartes perforées commandent une machine textile |
| 1843 | Ada Lovelace | un algorithme peut être écrit pour une machine générale |
| 1890 | Herman Hollerith | des données administratives sont codées sur cartes perforées |
| 1930–1940 | Enigma et cryptanalyse | une règle mécanique transforme un message |
| 1991 | Python | un langage lisible rend la programmation plus accessible |

## Fil conducteur

À chaque étape : représenter une information, appliquer une règle, produire un résultat.

# Enigma : une machine pour transformer un message

## Idée simple

Enigma n'est pas "intelligente" : elle applique une transformation déterminée par un réglage.

- Entrée : une lettre du message.
- Réglage : position des rotors.
- Traitement : substitutions successives.
- Sortie : une autre lettre.

# Enigma en pseudo-code simplifié

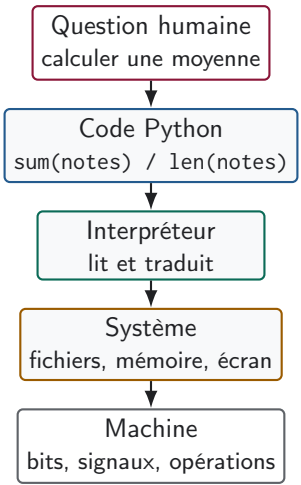
Ici, on ne reproduit pas Enigma. On garde seulement la logique informatique.

```
# Objectif : transformer un message
# Entrees : message, reglage_secret
# Pour chaque lettre du message :
#     appliquer une regle de transformation
#     ajouter la nouvelle lettre au message code
# Sortie : message code
```

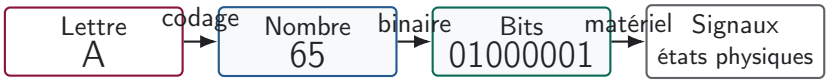
## Lien avec Python

Une boucle, une règle, une sortie : c'est déjà la forme de nombreux scripts de TD.

# Schéma : de l'humain à la machine



# Schéma : une donnée change de forme




## Conclusion

Python nous laisse écrire "A", pendant que les couches inférieures gèrent les nombres, les bits et les signaux.

# Du 0/1 à Python

## Idée de départ

Un ordinateur ne comprend pas directement les phrases humaines. Au niveau matériel, tout est représenté par des états simples : présence ou absence de signal.

| Niveau | Vocabulaire | Exemple |
|---|---|---|
| État physique | signal | courant / pas courant |
| Bit | 0 ou 1 | `0`, `1` |
| Octet | 8 bits | `01000001` |
| Donnée codée | nombre ou caractère | `65` ou `"A"` |
| Programme Python | instruction lisible | `print("A")` |

# Bits, octets, conventions

- Un **bit** est une information minimale : deux possibilités, souvent notées `0` et `1`.
- Un **octet** regroupe 8 bits, donc il peut coder 256 valeurs différentes.
- La même suite de bits peut être interprétée selon une convention : nombre, lettre, couleur, son, instruction.
- Un fichier est essentiellement une longue suite de bits organisée selon un format.

## Exemple

`01000001` peut être lu comme un nombre ou comme la lettre `A`, selon la table de codage utilisée.

# Ce que fait vraiment la machine

## Processeur

Le processeur exécute des instructions très simples : lire une valeur, écrire une valeur, additionner, comparer, sauter à une autre instruction.

- La machine exécute très vite des opérations très petites.
- Les humains veulent exprimer des idées plus générales : calculer une moyenne, ouvrir un fichier, tracer un graphique.
- Un langage de programmation sert à faire le pont entre ces deux mondes.

# Les couches entre la machine et nous

| Couche | Rôle | Exemple |
|---|---|---|
| Matériel | exécute des signaux | mémoire, processeur |
| Système d'exploitation | organise ressources et fichiers | macOS, Windows, Linux |
| Terminal | envoie des ordres textuels au système | `cd`, `pwd` |
| Interpréteur Python | lit le code Python | `python3` |
| Script | garde nos instructions | `tp1.py` |
| Bibliothèques | ajoutent des outils | pandas, `matplotlib` |

## À retenir

Quand quelque chose bloque, il faut identifier la couche : fichier introuvable, commande introuvable, erreur Python ou erreur de logique.

# À quoi sert un langage de programmation ?

1. **Exprimer une méthode** : transformer une question en étapes.
2. **Nommer les objets** : variables, fonctions, fichiers.
3. **Automatiser** : refaire le même calcul sans copier-coller.
4. **Vérifier** : relancer exactement le même script.
5. **Partager** : permettre à quelqu'un d'autre de comprendre et reproduire.

## Pour économistes

Python sert surtout à rendre une analyse calculable, reproductible et discutable.

# Langage : syntaxe et sens

## Syntaxe

Les règles d'écriture du langage.

```
print("Bonjour")
print("Bonjour"
```

## Sémantique

Ce que l'instruction veut faire.

```
age = 20
age = age + 1
```

## Conséquence

Une erreur de syntaxe empêche Python de lire. Une erreur de logique produit un résultat faux mais parfois sans message d'erreur.

# Python est un langage de haut niveau

```
notes = [12, 15, 9, 14]
moyenne = sum(notes) / len(notes)
print(moyenne)
```

- Nous écrivons des instructions lisibles par des humains.
- L'interpréteur Python lit ces instructions et demande au système d'exécuter les opérations nécessaires.
- On perd un peu de proximité avec la machine, mais on gagne énormément en lisibilité.

# Exercice : replacer les niveaux

Remettre ces éléments du plus proche de la machine au plus proche de l'utilisateur.

```
# A. print("Bonjour")
# B. 01000001
# C. processeur
# D. script tp1.py
# E. python3
# F. systeme d exploitation
```

# Exercice : replacer les niveaux

Remettre ces éléments du plus proche de la machine au plus proche de l'utilisateur.

```
# A. print("Bonjour")
# B. 01000001
# C. processeur
# D. script tp1.py
# E. python3
# F. systeme d exploitation
```

## Ordre attendu

C puis B puis F puis E puis D puis A.

# Exercice : coder une règle à la main

On utilise un chiffrement très simple : remplacer chaque lettre par la suivante.

```
# Exemple : A -> B, B -> C, C -> D
# Message : BAC
# Sortie attendue : CBD
```

1. Quelle est l'entrée ?
2. Quelle est la règle ?
3. Quelle est la sortie ?
4. Quelle partie deviendrait une boucle en Python ?

# Informatique : six mots utiles

| Mot | Sens simple | Exemple Python |
|---|---|---|
| Donnée | une valeur disponible | `22`, `"France"` |
| Instruction | une action demandée | `print(age)` |
| Algorithme | méthode en étapes | calculer une moyenne |
| Programme | algorithme écrit dans un langage | `tp1_nom_prenom.py` |
| Exécution | moment où la machine lit le programme | `python3 fichier.py` |
| Erreur | désaccord entre attente et réalité | `NameError` |

## Idée centrale

Programmer, ce n'est pas "parler ordinateur" : c'est écrire une suite d'actions assez précise pour être répétée sans ambiguïté.

# Algorithme : entrée, traitement, sortie

## Modèle mental

1. Entrée : ce que l'on connaît.
2. Traitement : ce que l'on calcule.
3. Sortie : ce que l'on affiche ou exporte.

```
# Objectif : calculer un cout
# Entrees : prix, quantite
# Traitement : prix * quantite
# Sortie : phrase lisible
```

# Mémoire : l'état d'un programme

```
age = 20
age = age + 1
print(age)
```

- Une variable est un nom attaché à une valeur.
- L'état du programme change quand une variable reçoit une nouvelle valeur.
- Lire une ligne demande de connaître l'état créé par les lignes précédentes.

## Point important

Dans `age = age + 1`, Python lit d'abord l'ancienne valeur de age, puis écrit la nouvelle.

# Chemins : où sont les fichiers ?

```
TD1/
  scripts/
    hello.py
  notes/
    essai.txt
```

| Expression | Sens |
|---|---|
| Dossier courant | endroit depuis lequel la commande est lancée |
| Chemin relatif | chemin depuis le dossier courant : `scripts/hello.py` |
| Chemin absolu | chemin complet depuis la racine du disque |
| `pwd` | commande qui affiche le dossier courant |

# PATH : ne pas confondre deux idées

```
python3 scripts/hello.py
```

- `scripts/hello.py` est le chemin du fichier à exécuter.
- `python3` est une commande que le système doit retrouver.
- La variable système `PATH` contient les dossiers où le terminal cherche les commandes.

## À retenir

Quand `python3` n'est pas reconnu, le problème est souvent le `PATH`. Quand `hello.py` n'est pas trouvé, le problème est souvent le dossier courant.

# Pourquoi pas seulement Excel ou une calculatrice ?

| Outil | Très utile pour | Limite |
|---|---|---|
| Calculatrice | un calcul isolé | difficile à relancer et vérifier |
| Tableur | regarder un petit tableau | manipulations parfois invisibles |
| Python | écrire une méthode complète | demande une syntaxe rigoureuse |

## Raison du cours

Python force à expliciter les étapes : cela rend les résultats plus reproductibles et les erreurs plus faciles à discuter.

# Terminal, interpréteur, script

| Outil | Sert à | Exemple |
|---|---|---|
| Terminal | parler au système | `pwd`, `cd`, `python3 fichier.py` |
| Interpréteur | tester une ligne | `>> 2 + 2` |
| Script | garder un travail | `td1_nom_prenom.py` |

## À retenir

Un TD se rend sous forme de script, pas sous forme d'une suite de tests perdus dans l'interpréteur.

# Créer, sauvegarder, exécuter

## Rituel de début de TD

Avant de chercher une erreur Python, vérifier que le bon fichier est bien sauvegardé et exécuté depuis le bon dossier.

1. Créer un fichier dont le nom finit par `.py`.
2. Écrire deux lignes très simples.
3. Sauvegarder le fichier.
4. L'exécuter depuis le dossier qui le contient.
5. Modifier une ligne, sauvegarder, relancer.

## Signal d'alerte

Si l'ancienne sortie apparaît encore, le fichier n'est probablement pas sauvegardé ou ce n'est pas le bon fichier qui est lancé.

# Un script minimal

```
print("Bonjour Python")
print("Je sais executer un fichier .py")
```

- Chaque ligne est exécutée de haut en bas.
- Le résultat doit pouvoir être relancé demain.
- C'est cette reproductibilité qui rend Python utile pour l'économie appliquée.

# Variables : donner un nom à une valeur

```
nom = "Alice"
age = 22
revenu = 1450.75
etudiant = True
```

- Une variable évite de répéter une valeur.
- Le nom doit aider à comprendre le rôle de la valeur.
- = signifie affecter une valeur, pas tester une égalité.

# Types de base

| Type | Exemple | Usage typique |
|---|---|---|
| `int` | `22` | âge, quantité, compteur |
| `float` | `1450.75` | revenu, prix, taux |
| `str` | `"France"` | nom, catégorie, code pays |
| `bool` | `True` | condition vraie/fausse |

## Réflexe

Quand une opération échoue, regarder les types avant de réécrire tout le code.

# Calculer

```
prix = 2.10
quantite = 3
cout = prix * quantite

print(cout)
print(f"Cout total : {cout:.2f} euros")
```

- Le calcul seul ne suffit pas : il faut afficher un résultat lisible.
- Les unités doivent apparaître dans le texte ou dans un commentaire.

# Listes : plusieurs valeurs dans un ordre

```
notes = [12, 15, 9, 14]

print(notes[0])
print(notes[-1])
print(sum(notes) / len(notes))
```

- Une liste permet de regrouper des observations.
- Les indices commencent à zéro.
- Une moyenne est un calcul sur une collection.

# Chaînes : du texte manipulable

```
mot = "Python"

print(mot[0])
print(mot[-1])
print(mot[:3])
print(mot[::-1])
```

- Le texte est indexé comme une séquence.
- Les tranches `debut:fin` excluent la borne de fin.

# Booléens : comparer

```
x = 10

print(x > 5)
print(x == 10)
print(x != 8)
print(x < 5)
```

## Différence clé

`x = 10` affecte une valeur. `x == 10` pose une question à Python.

# Lire une erreur

| Erreur | Question à se poser |
|---|---|
| `SyntaxError` | ai-je oublié une parenthèse, un guillemet, deux-points ? |
| `IndentationError` | un bloc après : est-il bien décalé ? |
| `NameError` | ai-je utilisé une variable non créée ? |
| `TypeError` | ai-je mélangé texte et nombre ? |
| `IndexError` | l'indice demandé existe-t-il dans la liste ? |
| `FileNotFoundError` | suis-je dans le bon dossier ? |

## Méthode

Lire la dernière ligne de l'erreur, puis remonter vers la ligne indiquée.

# Mini-débogage : lire avant de corriger

```
Traceback (most recent call last):
  File "tp1.py", line 3, in <module>
    print(esprance_vie)
NameError: name 'esprance_vie' is not defined
```

1. Dernière ligne : le type d'erreur est `NameError`.
2. Ligne indiquée : `line 3`.
3. Hypothèse : le nom de variable est mal orthographié.
4. Correction : comparer avec le nom créé plus haut.

# Exercice en cours : prédire puis tester

Avant d'exécuter, prédire les résultats :

```
age = 20
age = age + 1
print(age)

texte = "2"
print(texte * 3)
print(int(texte) + 3)
```

- Où y a-t-il une conversion de type ?
- Quel résultat est du texte, quel résultat est un nombre ?

# Pseudo-code : écrire avant Python

## Principe

Le pseudo-code décrit la logique avant la syntaxe. Il sert à vérifier l'ordre des étapes avant de demander à Python d'exécuter.

```
# PSEUDO-CODE
# Objectif : calculer le cout total d un achat
# Entrees : prix, quantite
# Etapes :
# 1. Creer les variables
# 2. Multiplier prix par quantite
# 3. Afficher le resultat avec une unite
```

# Du pseudo-code au code

```
# PSEUDO-CODE
# prix unitaire
# quantite achetee
# cout = prix * quantite
# afficher cout
```

```
prix = 2.10
quantite = 3
cout = prix * quantite
print(f"Cout : {cout:.2f} euros")
```

## Lecture

La partie gauche est une promesse logique ; la partie droite est sa traduction exécutable.

# Moment interactif : compléter

Objectif : calculer une moyenne et afficher une phrase interprétable.

```
# PSEUDO-CODE A COMPLETER
# Entree : ...
# Etape 1 : compter les observations
# Etape 2 : ...
# Etape 3 : afficher ...
```

# Moment interactif : compléter

Objectif : calculer une moyenne et afficher une phrase interprétable.

```
# PSEUDO-CODE A COMPLETER
# Entree : ...
# Etape 1 : compter les observations
# Etape 2 : ...
# Etape 3 : afficher ...
```

```
notes = [12, 15, 9, 14]
n = len(notes)
moyenne = sum(notes) / n
print(f"Moyenne : {moyenne:.1f}/20")
```

# Pseudo-code de débogage

```
# PSEUDO-CODE POUR UNE ERREUR
# 1. Lire la derniere ligne du message
# 2. Identifier le type d erreur
# 3. Aller a la ligne indiquee
# 4. Verifier nom de variable, type, chemin
# 5. Relancer le script complet
```

## Question à poser en direct

Quelle étape utiliseriez-vous pour une `NameError` ? Et pour une `FileNotFoundError` ?

# Exercice pseudo-code : condition

Compléter le pseudo-code avant de regarder la traduction Python.

```
# Objectif : afficher une categorie d age
# Entree : age
# Si age est superieur ou egal a 18 : ...
# Sinon : ...
```

# Exercice pseudo-code : condition

Compléter le pseudo-code avant de regarder la traduction Python.

```
# Objectif : afficher une categorie d age
# Entree : age
# Si age est superieur ou egal a 18 : ...
# Sinon : ...
```

```python
age = 20
if age >= 18:
    print("majeur")
else:
    print("mineur")
```

# Exercice pseudo-code : chemin de fichier

On est dans le dossier `TD1`. Quel chemin permet d'exécuter `hello.py` ?

```
TD1/
  scripts/
    hello.py
  notes/
    essai.txt
```

```
# Choisir puis tester :
python3 hello.py
python3 scripts/hello.py
python3 TD1/scripts/hello.py
```

# Ce que le TD1 rend obligatoire

1. Vérifier Python et créer un dossier de travail.
2. Exécuter un script depuis le terminal.
3. Manipuler nombres, chaînes, listes et booléens.
4. Calculer une moyenne.
5. Ajouter trois commentaires explicatifs.

## Ce qui compte

Le rendu doit s'exécuter. Une réponse juste mais impossible à relancer n'est pas encore un résultat reproductible.

# Transition vers TD2

Le TD2 ajoute une idée centrale :

## Structure de données

Choisir une forme adaptée pour représenter une information.

- Liste : plusieurs valeurs dans un ordre.
- Dictionnaire : une clé associée à une valeur.
- Fonction : un calcul que l'on peut réutiliser.

# Résumé

- Tout est codé en bits au niveau matériel, mais nous travaillons avec des couches d'abstraction.
- Un langage de programmation sert à exprimer une méthode lisible, reproductible et partageable.
- Python est un langage de haut niveau lu par un interpréteur.
- Un algorithme relie une entrée, un traitement et une sortie.
- Les variables décrivent l'état du programme à un moment donné.
- Les chemins relatifs, le dossier courant et le PATH ne désignent pas la même chose.