Semaine 1 : Maîtriser l'Ordinateur, les Données et Python

Manuel de l'élève • Aucun prérequis en programmation nécessaire • Du 23 au 29 novembre 2026

> **Bienvenue dans l'aventure IAIO 2027 !** Avant d'entraîner une intelligence artificielle, tout chercheur doit savoir trois choses essentielles : organiser son travail sur un ordinateur, comprendre et nettoyer des données brutes, et donner des instructions précises à la machine en langage **Python**. Ce manuel t'explique chaque notion pas à pas avec des exemples concrets de la vie quotidienne.

1. Comprendre son ordinateur et organiser ses fichiers

Un ordinateur ne « devine » jamais ce que tu veux faire : il suit exactement l'endroit où tu ranges tes fichiers et les commandes que tu lui donnes. La première compétence d'un bon informaticien est de garder un espace de travail propre et ordonné.

- **Dossier (ou répertoire) :** C'est comme une chemise cartonnée dans laquelle on range des feuilles.
- **Fichier :** C'est un document précis (un texte, un tableau de chiffres, un programme Python).
- **Extension de fichier :** Ce sont les lettres après le point final qui indiquent à l'ordinateur comment ouvrir le fichier : `.py` pour un programme Python, `.csv` pour un tableau de données, `.txt` pour un texte simple.
- **Chemin d'accès (Path) :** C'est l'adresse exacte d'un fichier dans l'ordinateur, par exemple : `Documents/IAIO_Jacmel/Semaine_1/notes.csv`.

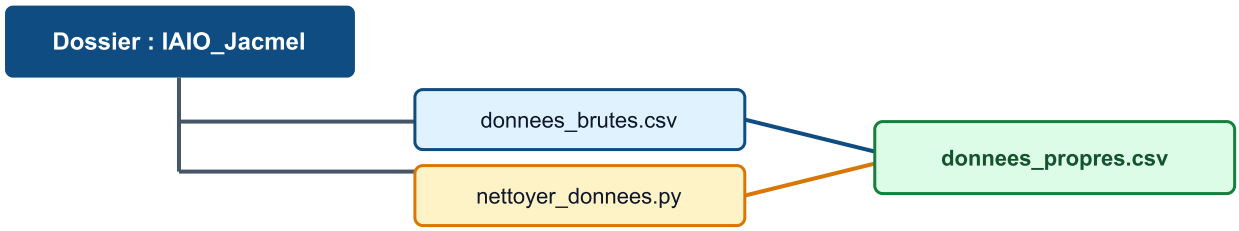

Figure 1 : Organisation recommandée — on ne modifie jamais le fichier original directement ; le script Python lit le fichier brut et produit un fichier propre.

Règles d'or à appliquer dès le premier jour :

1. Évite les espaces et les accents dans les noms de fichiers informatiques : écris `exercice_1.py` plutôt que `Exercice numéro 1.py`.
2. Fais toujours une copie de sauvegarde (`backup`) avant de modifier un fichier de données.
3. Enregistre ton travail toutes les 5 minutes (raccourci clavier `Ctrl + S`) pour ne rien perdre en cas de coupure de courant à Jacmel.

2. Les fichiers CSV : comment les données sont stockées

En intelligence artificielle, les tableaux de chiffres et de textes sont très souvent enregistrés au format **CSV** (*Comma-Separated Values*, c'est-à-dire « valeurs séparées par des virgules »). Chaque ligne représente un élément (un élève, une mesure météo, un trajet), et chaque colonne est séparée par une virgule.

Vue dans un tableur (Excel / LibreOffice)	Contenu réel du fichier texte (`eleves.csv`)

Colonne 1 : nom | **Colonne 2 :** age | **Colonne 3 :** note
Marie | 17 | 18
Jean | 16 | 15
	`nom,age,note`
	`Marie,17,18`
	`Jean,16,15`

3. Premiers pas avec Python : Variables et Types de données

Python est le langage de programmation le plus utilisé en intelligence artificielle parce qu'il se lit presque comme de l'anglais courant. Pour commencer, nous utilisons le **terminal** (une fenêtre noire ou blanche où l'on tape des commandes comme `python mon_fichier.py`) et un éditeur de code.

> **Image mentale — Qu'est-ce qu'une variable ?** Imagine une boîte étiquetée dans une cuisine. Sur l'étiquette, tu écris `score`. À l'intérieur de la boîte, tu mets le nombre `0`. Plus tard, tu peux ouvrir la boîte, changer le contenu pour `10`, ou lire ce qu'il y a dedans. En Python, on écrit simplement : `score = 0`.

Chaque donnée en Python possède un **type** précis. Mélanger les types sans faire attention provoque des erreurs :

Type Python	Signification	Exemples	À quoi ça sert ?
`int`	Nombre entier (*integer*)	`17`, `0`, `-5`	Compter des élèves, des étapes, des points.
`float`	Nombre à virgule flottante	`15.5`, `3.14`, `0.0`	Mesurer des moyennes, des distances, des probabilités.
`str`	Chaîne de caractères (*string*)	`"Jacmel"`, `"Marie"`	Stocker du texte (toujours entre guillemets).
`bool`	Booléen (Vrai ou Faux)	`True`, `False`	Prendre des décisions logiques (majuscule obligatoire).

Exemple commenté (opérations de base et affichage) :

```
ville = "Jacmel"          # Texte (str)
eleves_inscrits = 12      # Nombre entier (int)
places_max = 5            # Nombre entier (int)
places_restantes = eleves_inscrits - places_max

print("Atelier de :", ville)
print("Candidats non retenus :", places_restantes)  # Affiche : 7
```

Attention au piège classique : `"17"` (entre guillemets) est du texte, alors que `17` (sans guillemets) est un nombre. On ne peut pas faire `"17" + 3` directement : il faut d'abord convertir le texte en nombre entier avec `int("17")` ou en nombre décimal avec `float("17.5")`.

4. Prendre des décisions : les conditions (`if`, `elif`, `else`)

Un programme intelligent doit s'adapter aux situations. En Python, on utilise `if` (si), `elif` (sinon si) et `else` (sinon). **Point très important :** en Python, les instructions qui dépendent d'un `if` doivent être décalées vers la droite de **4 espaces** (c'est ce qu'on appelle l'**indentation**).

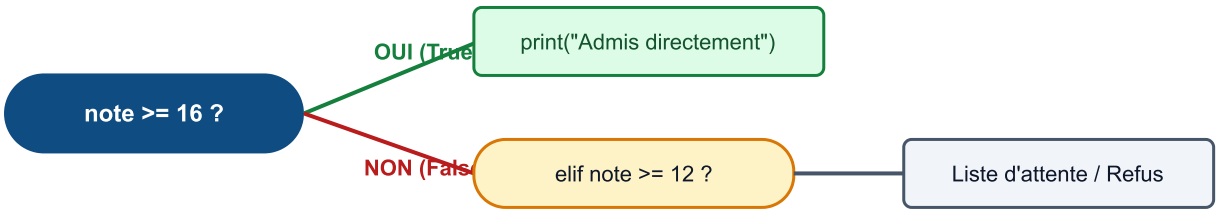

Figure 2 : Fonctionnement d'un test conditionnel en Python — l'ordinateur suit une seule branche selon que la condition est vraie ou fausse.

Les opérateurs de comparaison et de logique à connaître :

- `==` : égal à (attention : deux signes `=` pour comparer, un seul `=` pour ranger dans une variable !).
- `!=` : différent de ; `>` : strictement supérieur ; `>=` : supérieur ou égal ; `<` : inférieur.
- `and` : vrai seulement si les **deux** conditions sont vraies.
- `or` : vrai si **au moins une** des deux conditions est vraie.

```
note = 14
habite_jacmel = True

if note >= 16 and habite_jacmel == True:
    print("Sélectionné !")
elif note >= 12 and habite_jacmel == True:
    print("Convoqué à l'entretien oral.")
else:
    print("Non retenu pour cette session.")
```

5. Répéter des actions automatiquement : les boucles (`for` et `while`)

La grande force d'un ordinateur est de répéter une opération 10 fois ou 100 000 fois sans se fatiguer ni se tromper.

- **La boucle `for` (pour chaque élément) :** On l'utilise quand on sait à l'avance combien de fois répéter ou quand on parcourt une liste de données.
- **La boucle `while` (tant que) :** On l'utilise quand on répète une action tant qu'une condition reste vraie.

```
# Exemple 1 : Parcourir une liste de notes avec "for"
notes = [14, 18, 12, 16]
total = 0

for n in notes:
    total = total + n

moyenne = total / len(notes)   # len(notes) donne la taille de la liste (ici 4)
print("Moyenne du groupe :", moyenne)  # Affiche : 15.0

# Exemple 2 : Compter de 1 à 3 avec range(1, 4)
for numero in range(1, 4):
    print("Exercice numéro", numero)
```

6. Organiser plusieurs données : Listes et Dictionnaires

Quand on manipule des jeux de données, on regroupe les valeurs dans deux structures fondamentales :

- **La liste (`list`) :** Une suite ordonnée d'éléments entre crochets `[ ]`. **Attention :** en Python, on commence toujours à compter à partir de **0** ! Dans `villes = ["Jacmel", "Cayes", "Port-au-Prince"]`, `villes[0]` vaut `"Jacmel"` et `villes[1]` vaut `"Cayes"`. On ajoute un élément à la fin avec `villes.append("Léogâne")`.
- **Le dictionnaire (`dict`) :** Une fiche d'identité entre accolades `{ }` qui associe une **clé** (le nom du champ) à une **valeur**.

```
# Exemple de dictionnaire représentant un candidat
candidat = {
    "nom": "Claudine",
    "age": 17,
    "score": 21
}

print(candidat["nom"])          # Affiche : Claudine
candidat["score"] = 23          # Met à jour le score
```

7. Créer ses propres outils : les Fonctions (`def`)

Une **fonction** est un mini-programme réutilisable auquel on donne un nom. Elle reçoit des données en entrée (les *paramètres*), effectue un calcul, et renvoie un résultat grâce au mot-clé `return`.

```
def convertir_sur_100(score_sur_25):
    resultat = (score_sur_25 / 25) * 100
    return resultat

note_finale = convertir_sur_100(20)
print("Pourcentage obtenu :", note_finale, "%")  # Affiche : 80.0 %
```

8. Projet fil rouge de la Semaine 1 : Nettoyer un fichier CSV réel

Dans la réalité, les fichiers de données contiennent toujours des erreurs de saisie : des espaces en trop, des âges négatifs, des cases vides ou des mots écrits à la place d'un chiffre. En intelligence artificielle, si l'on donne des données sales à un modèle, il apprendra n'importe quoi (*« Garbage in, garbage out »*).

Voici comment lire un fichier `candidats_brut.csv` en Python, vérifier chaque ligne, ignorer les lignes invalides et écrire un fichier propre `candidats_propre.csv` :

```
# Contenu supposé de candidats_brut.csv :
# nom,age,score
# Alice,17,22
# Bob,-3,19
# Chloé,seize,20
# David,16,24

fichier_entree = open("candidats_brut.csv", "r", encoding="utf-8")
fichier_sortie = open("candidats_propre.csv", "w", encoding="utf-8")

entete = fichier_entree.readline()          # Lit la 1re ligne ("nom,age,score\n")
fichier_sortie.write(entete)                # Recopie l'en-tête dans le fichier
propre

lignes_valides = 0

for ligne in fichier_entree:
    ligne = ligne.strip()                   # Enlève les espaces et retours à la
ligne
    morceaux = ligne.split(",")             # Découpe la ligne à chaque virgule
    nom = morceaux[0]
    age_texte = morceaux[1]
    score_texte = morceaux[2]

    # Vérifier si l'âge et le score sont bien composés uniquement de chiffres
    if age_texte.isdigit() and score_texte.isdigit():
        age = int(age_texte)
        score = int(score_texte)
        if age >= 12 and age <= 22 and score >= 0 and score <= 25:
            fichier_sortie.write(nom + "," + str(age) + "," + str(score) + "\n")
            lignes_valides = lignes_valides + 1

fichier_entree.close()
fichier_sortie.close()
print("Nettoyage terminé ! Lignes valides conservées :", lignes_valides)
```

Éthique et responsabilité des données (exigence IAIO) : Lorsque tu nettoies un fichier, demande-toi toujours : *« Est-ce que ma règle de nettoyage risque d'exclure injustement certaines personnes ? »* Par exemple, si un élève a écrit `" 17 "` avec des espaces autour, utiliser `age_texte.strip().isdigit()` permet de récupérer sa ligne au lieu de l'éliminer à tort. De plus, ne partage jamais publiquement les noms ou informations personnelles d'un fichier d'élèves.

9. Fiche d'exercices pratiques — Semaine 1

Exercice 1 — Manipulation de variables et calculs (Débutant)

Écris un petit programme Python qui crée trois variables : `heures_par_jour = 2`, `jours_par_semaine = 5`, et `nombre_semaines = 13`. Calcule le nombre total d'heures d'entraînement jusqu'à l'IAIO 2027 et affiche une phrase claire avec `print()`.

Exercice 2 — Conditions et logique de sélection (Débutant)

Écris un programme qui vérifie deux variables : `score_test = 19` et `disponible_jacmel = True`. Si le score est supérieur ou égal à 18 **et** que `disponible_jacmel` est vrai, affiche `"Qualifié pour l'atelier"`. Si le score est supérieur ou égal à 18 mais que `disponible_jacmel` est faux, affiche `"Score suffisant mais présence requise à Jacmel"`. Sinon, affiche `"À encourager pour la prochaine édition"`.

Exercice 3 — Boucles et listes : Statistiques d'une classe (Intermédiaire)

On te donne la liste suivante : `scores = [12, 22, 19, 8, 24, 17, 21]`. Sans utiliser la fonction `max()` de Python, écris une boucle `for` qui : (1) compte combien de candidats ont obtenu au moins 18 points, et (2) trouve le meilleur score de la liste.

Exercice 4 — Fonctions et dictionnaires (Intermédiaire)

Écris une fonction `evaluer_candidat(fiche)` qui reçoit un dictionnaire comme `{"nom": "Jean", "test": 20, "oral": 4}` (où `test` est sur 25 et `oral` est sur 5). La fonction doit calculer la note totale sur 30 et retourner `"Retenu"` si le total est supérieur ou égal à 22, ou `"En attente"` sinon.

Exercice 5 — Défi Fil Rouge : Nettoyage d'un fichier CSV météo de Jacmel (Avancé)

Tu reçois un fichier `meteo_jacmel.csv` contenant des lignes au format `jour,temperature,pluie_mm`. Certaines lignes contiennent des erreurs (température négative comme `-10` ou supérieure à `50`, ou texte `"erreur"`). Écris un script Python qui lit `meteo_jacmel.csv`, ne garde que les lignes où la température est un entier compris entre `15` et `42` degrés, calcule la température moyenne des jours valides, et écrit les lignes valides dans `meteo_propre.csv`.