Introduction à Python et à l'environnement de travail

par :
Dr. Bilal Dendani

جـامعة بـاجـي مختـار - عنـابـة
BADJI MOKHTAR - ANNABA UNIVERSITY

Plan de travail

- Prise en main de Jupyter notebok
  - Installation de Jupyter notebook
  - Présentation de l'environnement Jupyter notebook
- Premiers pas avec python
  - Introduction
  - Syntax de base
  - Structures de contrôle en Python
  - Conditions (if/else)
  - Boucles (for, while)
- Manipulation de données avec numpy et pandas
  - Numpy
  - Pandas

Objectif du TP

- Se familiariser avec l'environnement Jupyte Notebook.
- Apprendre les bases de la programmation en Python (variables, types de données, conditions, boucles).
- Découvrir les structures de données de base (listes, tuples, dictionnaires).
- Introduction à Numpy et Pandas.

Partie 1: Prise en main de Jupyter Notebook

Objectifs

- Comprendre le rôle de Jupyter Notebook en data science.
- Installer et lancer Jupyter Notebook.
- Se familiariser avec l'interface de Jupyter Notebook.
- Apprendre à exécuter des cellules de code et à utiliser les raccourcis.
- Créer et enregistrer un premier notebook.

Qu'est-ce que Jupyter Notebook ?

- Jupyter Notebook est un environnement interactif pour écrire et exécuter du code Python.
- l'outil préféré pour la data science car il permet de combiner du code, des explications (texte) et des visualisations dans un seul document.
- Supporte plusieurs langages de programmation (Python, R, Julia, Scala, Java, Java script, PHP...). Mais principalement utilisé avec Python.

Installation de Jupyter Notebook

- **Option 1 :** Installer via Anaconda (recommandé pour les débutants)
  - Téléchargez et installez Anaconda.
  - Jupyter est préinstallé dans Anaconda, ainsi que d'autres outils utiles comme Spyder, Pandas, et NumPy.
- **Option 2 :** Installer via pip (Dans le cas ou vscode existe au niveau de ta machine)
  - Ouvrez un terminal et tapez :

```bash
pip install jupyter
```

Lancer Jupyter notebook:

```bash
jupyter notebook
```

Interface de Jupyter Notebook

- **Barre d'outils :** Contient des options comme enregistrer, insérer des cellules, et exécuter du code.
- **Cellules :** Les blocs où vous écrivez votre code ou texte.
- **Onglet "Kernel" :** Permet de redémarrer ou réinitialiser l'environnement d'exécution.
- **Onglet "Files" :** Naviguez dans les fichiers de votre projet

Créer un nouveau notebook

- Cliquez sur "**New**" dans l'interface et sélectionnez "**Python 3**".
- Un nouveau notebook s'ouvre avec une cellule de code prête à être utilisée.
- **Enregistrez votre notebook** avec un nom explicite en utilisant l'extension .ipynb.

Types de cellules

- **Deux types de cellules principaux :**
  - **Cellule de code :** Pour écrire du code Python qui sera exécuté.
  - **Cellule de texte (Markdown) :** Pour écrire du texte, des formules ou des explications formatées.
    - Markdown permet d'ajouter des titres, du texte en gras, italique, des liens, etc.
    - Exemple de syntaxe Markdown :

```markdown
# Ceci est un titre
**Ceci est en gras**
```

Cellule de type Markdown

Titres

- `# Titre de niveau 1`

Titre de niveau 1

- `## Titre de niveau 2`

Titre de niveau 2

- `### Titre de niveau 3`

Titre de niveau 3

- `#### Titre de niveau 4`

Titre de niveau 4

Gras et Italique

- `**Gras**` **Gras**
- `*Italique*` *Italique*
- `***Gras et Italique***` ***Gras et Italique***

Exécution d'une cellule

- Cliquez sur la cellule et appuyez sur **Shift + Enter** pour exécuter.
- Les résultats s’affichent immédiatement sous la cellule.
- L’ordre d'exécution est visible à gauche de la cellule (ex. In [1]).

Sauvegarde et exportation des notebooks

- **Sauvegarder automatiquement** : Jupyter enregistre régulièrement les modifications.
- **Exportation** : Vous pouvez exporter votre notebook au format PDF, HTML, ou script Python (.py).

    - Allez dans **"File" > "Download as"** pour choisir le format d'exportation souhaité.

Exercice pratique : Créer et exécuter un notebook

- Créez un nouveau notebook.
- Dans une cellule, affichez le message : Bienvenue au Jupyter
- Ajoutez une cellule Markdown de type titre pour expliquer le but du notebook.
- Sauvegardez et renommez le fichier.

Partie 2: Premiers pas avec Python

Objectives

- Comprendre la syntaxe de base de Python.
- Se familiariser avec les variables, types de données, conditions et boucles.
- Découvrir les structures de données de base (listes, tuples, dictionnaires)
- Utiliser Jupyter Notebook pour écrire et exécuter du code Python.

Pourquoi Python ?

- Langage open source
- Python est un langage populaire pour la data science, apprécié pour sa simplicité et sa lisibilité.
- Utilisé pour le traitement de données, l'intelligence artificielle, le développement web, etc.
- Python fonctionne dans plusieurs environnements (Jupyter Notebook, IDEs comme Vscode, PyCharm, Spyder).
- L'un des langages les plus utilisé dans les industrés.
- Large bibliothèque pour la manipulation, l'analyse et la visualization de données (Pandas, Numpy, Matplotlib, Seaborn, Sickit-learn).
- Grande communauté active avec de nombreuses ressources disponibles.
- Utilisé dans de nombreux domaines : finance, santé, marketing, etc.

Syntaxe de base de Python

- **Commentaires :** Utilisation de # pour ajouter des commentaires.
  - Exemple : # Ceci est un commentaire

- **Affichage de texte :** La fonction print() permet d'afficher du texte.
  - Exemple : print("Bonjour tout le monde !")
  - Print(10, “Un text”, True) # Sur la meme ligne afficher plusieur informations

Exercice pratique

- Afficher votre nom & prenom et ajouter un commentaire en utilisant jupyter notebook

  - Écrire un script qui affiche votre nom et prénom :
    - Ajouter un commentaire “mon nom et prénom est :”
    - Votre nom
    - Votre prénom.

Variables et types de données

- **Variables :** Stockent des valeurs, pas besoin de déclarer leur type.

```python
nom = "Jean"
age = 20
est_etudiant = True
```

- **Types de données :**
  - Nombres (entiers, flottants).
  - Chaînes de caractères.
  - Booléens (True ou False).

Exercice pratique : Variables

- Déclarez des variables pour :
  - Votre nom (chaîne).
  - Votre âge (entier).
  - Votre statut étudiant (booléen).
- Affichez ces variables avec la fonction print().
- Ajouter un titre niveau 1 pour cette cellule

Opérations sur les variables

- **Opérations mathématiques :**
  - Addition : a + b
  - Multiplication : a * b

```python
a = 5
b = 3
resultat = a + b
print(resultat)
```

- Concaténation de chaînes

```python
nom_complet = "Jean" + " " + "Dupont"
print(nom_complet)
```

Conditions (if, elif, else)

- Permettent de tester des conditions.
- Syntaxe

```python
if condition:
    # exécuter ce bloc
elif autre_condition:
    # exécuter ce bloc
else:
    # exécuter ce bloc
```

- Exemple

```python
age = 20
if age < 18:
    print("Vous êtes mineur.")
elif age == 18:
    print("Vous avez 18 ans.")
else:
    print("Vous êtes majeur.")
```

Les boucles : for

- La boucle for permet de répéter des actions un nombre défini de fois.
- Syntaxe :

```python
for i in range(n):
    # exécuter ce bloc n fois
```

- Exemple

```python
for i in range(5):
    print(i)
```

Les boucles : While

- La boucle while répète un bloc de code tant qu'une condition est vrai
- Syntaxe :

```python
compteur = 0
while compteur < 5:
    print(compteur)
    compteur += 1
```

- Exercice pratique :
  - Afficher des nombres pairs.
    - Écrivez une boucle for qui affiche tous les nombres pairs entre 1 et 10.
    - Écrivez une boucle while qui affiche les nombres de 0 à 5.

Exercice 01:

Programme de vérification d'âge

- Écrire un programme python qui demande à l'utilisateur son âge et qui affiche un message en fonction de l'âge.

1. Demandez à l’utilisateur de saisir son âge.
2. Si l’âge est inférieur à 18, affichez : "Vous êtes mineur".
3. Si l’âge est compris entre 18 et 60, affichez : "Vous êtes un adulte".
4. Si l’âge est supérieur à 60, affichez : "Vous êtes un senior".

Python code pour Ex 01:

```python
age = int(input("Quel est votre âge ? "))

if age < 18:
    print("Vous êtes mineur")
elif 18 <= age <= 60:
    print("Vous êtes un adulte")
else:
    print("Vous êtes un senior")
```

Exercice 02:

- Utiliser une boucle pour calculer la somme des nombres pairs dans un intervalle donné par l'utilisateur.
- **Instructions :**
  1. Demandez à l'utilisateur de saisir deux nombres : un début et une fin d'intervalle.
  2. Utilisez une boucle for pour calculer et afficher la somme des nombres pairs dans cet intervalle.
  3. Assurez-vous que le programme fonctionne quel que soit l'ordre des nombres saisis.

Code python pour l’exo 2

```python
debut = int(input("Entrez le début de l'intervalle : "))
fin = int(input("Entrez la fin de l'intervalle : "))

# Assurer que debut est plus petit que fin
if debut > fin:
    debut, fin = fin, debut

somme_pairs = 0
for i in range(debut, fin + 1):
    if i % 2 == 0:
        somme_pairs += i

print("La somme des nombres pairs dans l'intervalle est :", somme_pairs)
```

Structures de données en Python : LISTES

- Une **liste** est une structure de données qui permet de stocker plusieurs valeurs dans un seul objet. Elle est mutable, ce qui signifie qu'on peut modifier ses éléments après sa création.

```
# Créer une liste
fruits = ['pomme', 'banane', 'cerise']

# Accéder aux éléments
print(fruits[0])  # affiche 'pomme'

# Ajouter un élément
fruits.append('orange')

# Supprimer un élément
fruits.remove('banane')

# Longueur de la liste
print(len(fruits))
```

Structures de données en Python : TUPLES

- Similaire à une liste, mais il est immuable. Cela signifie qu'une fois créé, on ne peut pas modifier les éléments d’un tuple.

```
# Créer un tuple
coordonnees = (4, 5)

# Accéder aux éléments
print(coordonnees[0])  # affiche 4

# Impossible d'ajouter ou de supprimer des éléments
# coordonnees[0] = 10  # ceci générerait une erreur
```

Structures de données en Python : Dictionnaires

- Un dictionnaire est une collection de paires clé-valeur, utilisée pour associer des clés uniques à des valeurs. C'est très pratique pour stocker des données structurées comme un enregistrement ou une entrée de base de données.

```python
# Créer un dictionnaire
etudiant = {
    'nom': 'Alice',
    'age': 21,
    'cours': ['math', 'informatique']
}

# Accéder aux valeurs
print(etudiant['nom'])  # affiche 'Alice'

# Ajouter ou modifier une entrée
etudiant['age'] = 22

# Supprimer une entrée
del etudiant['cours']
```

Exercice 03:

- **Recherche dans une liste**
- Utiliser une boucle while pour chercher un élément dans une liste donnée par l'utilisateur.
- **Instructions :**

1. Créez une liste d’au moins 5 éléments (nombres ou chaînes de caractères).
2. Demandez à l’utilisateur de saisir un élément à rechercher dans la liste.
3. Utilisez une boucle while pour parcourir la liste et afficher si l’élément a été trouvé ou non.

Code python pour Exercice 03:

```python
liste = ["pomme", "banane", "orange", "fraise", "ananas"]
element = input("Entrez un élément à rechercher dans la liste : ")

trouve = False
i = 0
while i < len(liste):
    if liste[i] == element:
        trouve = True
        break
    i += 1

if trouve:
    print(f"L'élément '{element}' a été trouvé dans la liste.")
else:
    print(f"L'élément '{element}' n'est pas dans la liste.")
```

Exercice 04

1. Créez un dictionnaire nommé notes_etudiant avec les matières suivantes comme clés et les notes correspondantes comme valeurs :

Mathématiques : 14

Physique : 16

Chimie : 12

Informatique : 18

Histoire : 10

2. Ajoutez une nouvelle matière Anglais avec la note 15 au dictionnaire.

3. Écrivez un code pour afficher la note obtenue en **Physique**.

4. Corrigez la note de **Histoire** et changez-la à 12.

5. Calculez et affichez la moyenne des notes de l'étudiant en utilisant les valeurs du dictionnaire.

6. Supprimez la matière **Chimie** du dictionnaire (supposons que la matière n'est plus prise en compte pour cette année).

7. Affichez le dictionnaire final pour vérifier toutes les modifications.

```python
notes_etudiant = {
    "Mathématiques": 14,
    "Physique": 16,
    "Chimie": 12,
    "Informatique": 18,
    "Histoire": 10
}

# 2. Ajouter une matière
notes_etudiant["Anglais"] = 15

# 3. Accéder à une note
print("Note en Physique :", notes_etudiant["Physique"])

# 4. Modifier une note
notes_etudiant["Histoire"] = 12

# 5. Calculer la moyenne des notes
moyenne = sum(notes_etudiant.values()) / len(notes_etudiant)
print("Moyenne des notes :", moyenne)

# 6. Supprimer une matière
del notes_etudiant["Chimie"]

# 7. Afficher le dictionnaire final
print("Dictionnaire final :", notes_etudiant)
```

Partie 3 : Biblithèques pour la minpulation de données

Bibliothèque Numpy

- Crée par le chercheur Travis Oliphant au début des années 2000
- **NumPy**, diminutif de Numerical Python, est une bibliothèque de base pour le calcul scientifique en Python.
- Elle offre un support pour les **tableaux multidimensionnels** et **matrices** de grande taille, ainsi qu'une collection de **fonctions mathématiques** permettant d'opérer sur ces tableaux.
- NumPy introduit une nouvelle structure de données avec des capacités **hautes performances** pour manipuler de grands volumes de données.
- Elle surpasse les structures de données Python par défaut en termes de cas d'utilisation complexes de manipulation de données et de **performance**.

Bibliothèque Pandas

- **Dévelopé par le développeur américain Wes McKinney en Janvier 2008**
- **Pandas** est une bibliothèque puissante et populaire pour la manipulation de données en Python.
- Elle est construite au-dessus de la bibliothèque **NumPy** et fournit des structures de données flexibles qui facilitent la manipulation efficace des données structurées.
- Elle propose deux principales structures de données : **Series** (tableau étiqueté à une dimension) et **DataFrame**.
- La forte dépendance avec NumPy la rend extrêmement flexible et efficace pour les tâches d'analyse de données de grande envergure.

- Sa structure de données supporte des données homogènes (même type).
- Structure de données multidimensionnelle.
- Empreinte mémoire réduite -> Plus rapide (consomme moins de ressources).
- Nécessite une pratique supplémentaire pour se familiariser avec ses fonctionnalités

- Fournit une structure de données qui peut stocker des types de données différents.
- Structures de données avec jusqu'à 2 dimensions (DataFrame).
- Empreinte mémoire plus importante -> Plus lent (consomme plus de ressources).
- intuitif et facile à utiliser avec des fonctionnalités avancées.

Déclarer une structure de données Séries avec pandas

Python

```python
import pandas as pd

# Créer une Series à partir d'une liste
data = [10, 20, 30, 40, 50]
series = pd.Series(data)

# Afficher la Series
print(series)
```

Déclarer une structure de données Dataframe avec pandas

Python

```python
import pandas as pd

# Créer un DataFrame à partir d'un dictionnaire
data = {'Nom': ['Alice', 'Bob', 'Charlie'],
        'Âge': [25, 30, 35],
        'Ville': ['Paris', 'Lyon', 'Marseille']}

df = pd.DataFrame(data)

# Afficher le DataFrame
print(df)
```

Importer numpy et pandas avec affichage de version des packages :

Python Copy

```python
import numpy as np
import pandas as pd

print("Version de NumPy :", np.__version__)
print("Version de pandas :", pd.__version__)
```

Créer DataFrame à partir de la lecture d'un fichier ".csv"

Python

```python
import pandas as pd

# Chemin vers votre fichier CSV
file_path = 'nom_du_fichier.csv'

# Lire le fichier CSV et créer un DataFrame
df = pd.read_csv(file_path)

# Afficher les premières lignes du DataFrame
print(df.head())
```

Références

- OpenAI. (2024). ChatGPT [AI language model]. Retrieved from https://chat.openai.com