# Introduction à Python

M. L. Sahari
Département de Mathématiques
Université Badji Mokhtar-Annaba

## Table des matières

# 1 Introduction

Python est un langage de programmation de haut niveau, particulièrement adapté au calcul scientifique et à l'analyse de données. Sa syntaxe claire et ses bibliothèques puissantes (NumPy, SciPy, Matplotlib) en font un outil incontournable pour les mathématiciens.

## 1.1 Pourquoi Python pour les mathématiques ?

— Syntaxe intuitive et proche de la notation mathématique
— Bibliothèques spécialisées pour le calcul numérique
— Gratuit et open source
— Très utilisé en recherche et en industrie

# 2 Installation et configuration sous Windows

Cette section vous guidera pas à pas dans l'installation de Python et de l'éditeur Visual Studio Code (VS Code) sur Windows.

## 2.1 Installation de Python

1. **Télécharger Python**
   — Rendez-vous sur le site officiel : `https://www.python.org/downloads/`
   — Téléchargez la dernière version stable (Python 3.12 ou supérieur)
   — Choisissez l'installateur Windows (fichier .exe)
2. **Installer Python**
   — Lancez l'installateur téléchargé
   — **IMPORTANT :** Cochez la case `"Add Python to PATH"` en bas de la fenêtre
   — Cliquez sur `"Install Now"` pour une installation standard
   — Attendez la fin de l'installation
   — Cliquez sur `"Close"` une fois terminé
3. **Vérifier l'installation**
   — Ouvrez l'invite de commandes (Cmd) : tapez `cmd` dans la barre de recherche Windows
   — Tapez la commande suivante et appuyez sur Entrée :

```
python --version
```

   — Vous devriez voir s'afficher la version de Python installée (ex : Python 3.12.0)

## 2.2 Installation de Visual Studio Code

Visual Studio Code est un éditeur de code gratuit, léger et puissant, particulièrement adapté pour Python.

1. **Télécharger VS Code**
   — Allez sur : `https://code.visualstudio.com/`
   — Cliquez sur `"Download for Windows"`
   — L'installateur (.exe) se télécharge automatiquement

2. **Installer VS Code**
   — Lancez l'installateur
   — Acceptez les conditions d'utilisation
   — Choisissez le dossier d'installation (laissez par défaut)
   — **Cochez les options suivantes** (recommandé) :
      — `"Add to PATH"`
      — `"Create a desktop icon"`
      — `"Add 'Open with Code' action to Windows Explorer file context menu"`
   — Cliquez sur `"Install"` puis `"Finish"`
3. **Installer l'extension Python**
   — Lancez VS Code
   — Cliquez sur l'icône `"Extensions"` dans la barre latérale gauche (ou appuyez sur `Ctrl+Shift+X`)
   — Dans la barre de recherche, tapez : `Python`
   — Trouvez l'extension `"Python"` développée par Microsoft
   — Cliquez sur `"Install"`

## 2.3 Installation des bibliothèques scientifiques

Pour le calcul scientifique, vous aurez besoin d'installer NumPy, Matplotlib et d'autres bibliothèques.

1. **Ouvrir un terminal dans VS Code**
   — Dans VS Code, allez dans le menu `Terminal > New Terminal`
   — Ou utilisez le raccourci `Ctrl+‘` (accent grave)
2. **Installer les bibliothèques**
   Tapez les commandes suivantes une par une dans le terminal :

   ```
   pip install numpy
   pip install matplotlib
   pip install scipy
   ```

   Ou installez-les toutes en une seule commande :

   ```
   pip install numpy matplotlib scipy
   ```

3. **Vérifier l'installation**
   Dans le terminal, lancez Python :

   ```
   python
   ```

   Puis testez l'importation :

   ```
   >>> import numpy as np
   >>> import matplotlib.pyplot as plt
   >>> print(np.__version__)
   >>> exit()
   ```

   Si aucune erreur n'apparaît, tout est correctement installé !

## 2.4 Créer et exécuter votre premier programme Python

1. **Créer un nouveau fichier**
   - Dans VS Code, allez dans `File > New File`
   - Sauvegardez le fichier : `File > Save As...`
   - Nommez-le `test.py` (l'extension `.py` est importante)
2. **Écrire du code**

   Copiez ce code simple dans votre fichier :

```python
import numpy as np
import matplotlib.pyplot as plt

# Creation d'un tableau de valeurs
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

# Affichage
print("Premier programme Python !")
print(f"Maximum de sin(x): {np.max(y)}")

# Graphique
plt.plot(x, y)
plt.title("Fonction sinus")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(True)
plt.show()
```

3. **Exécuter le programme**
   - Cliquez sur le bouton `"Run"` (triangle) en haut à droite
   - Ou appuyez sur `F5`
   - Ou utilisez le terminal : `python test.py`

## 2.5 Conseils et astuces pour VS Code

- **Raccourcis utiles :**
  - `Ctrl+/` : Commenter/décommenter une ligne
  - `Ctrl+S` : Sauvegarder
  - `F5` : Exécuter en mode debug
  - `Ctrl+Space` : Auto-complétion
- **Extensions recommandées :**
  - `Python` (Microsoft) - déjà installée
  - `Pylance` - améliore l'auto-complétion
  - `Jupyter` - pour les notebooks
- **Mode interactif :**

  Vous pouvez aussi utiliser Python en mode interactif directement dans le terminal :

```
python
>>> 2 + 2
4
>>> import math
```

```
>>> math.sqrt(16)
4.0
```

# 3 Les bases de Python

## 3.1 Variables et types de données

En Python, on n'a pas besoin de déclarer le type des variables. Le type est déterminé automatiquement.

```python
# Entiers
n = 42
m = -17

# Flottants (nombres reels)
x = 3.14159
y = 2.71828

# Complexes
z = 3 + 4j
w = complex(1, 2)

# Booleens
vrai = True
faux = False

# Chaines de caracteres
message = "Bonjour"
```

Listing 1 – Types de base

## 3.2 Opérations arithmétiques

```python
# Addition, soustraction, multiplication
a = 10 + 5
b = 10 - 5
c = 10 * 5

# Division reelle et division entiere
d = 10 / 3       # Resultat: 3.333...
e = 10 // 3      # Resultat: 3

# Modulo et puissance
f = 10 % 3       # Resultat: 1
g = 2 ** 10      # Resultat: 1024
```

Listing 2 – Operations de base

## 3.3 Structures de données

### 3.3.1 Listes

Les listes sont des séquences ordonnées et modifiables.

```
# Creation
L = [1, 2, 3, 4, 5]
vide = []

# Acces aux elements (indexation commence a 0)
premier = L[0]       # 1
dernier = L[-1]      # 5

# Slicing (extraction de sous-listes)
sous_liste = L[1:4]   # [2, 3, 4]

# Ajout d'elements
L.append(6)          # [1, 2, 3, 4, 5, 6]

# Longueur
n = len(L)           # 6
```

Listing 3 – Manipulation de listes

### 3.3.2 Tuples

Les tuples sont similaires aux listes mais **non modifiables**.

```
point = (3, 4)
x, y = point   # Decomposition: x=3, y=4
```

Listing 4 – Tuples

### 3.3.3 Dictionnaires

Les dictionnaires associent des clés à des valeurs.

```
etudiant = {
    "nom": "Dupont",
    "age": 21,
    "notes": [15, 17, 14]
}

nom = etudiant["nom"]
etudiant["filiere"] = "Mathematiques"
```

Listing 5 – Dictionnaires

# 4 Structures de contrôle

## 4.1 Conditions

```python
x = 10

if x > 0:
    print("x est positif")
elif x < 0:
    print("x est negatif")
else:
    print("x est nul")
```

Listing 6 – Instructions conditionnelles

## 4.2 Boucles

### 4.2.1 Boucle for

```python
# Iteration sur une liste
for i in [1, 2, 3, 4, 5]:
    print(i ** 2)

# Utilisation de range
for i in range(5):       # 0, 1, 2, 3, 4
    print(i)

for i in range(1, 10, 2):  # 1, 3, 5, 7, 9
    print(i)
```

Listing 7 – Boucle for

### 4.2.2 Boucle while

```python
n = 1
while n <= 10:
    print(n)
    n += 1
```

Listing 8 – Boucle while

## 4.3 List comprehension

Syntaxe compacte pour créer des listes.

```python
# Carres des nombres de 0 a 9
carres = [i**2 for i in range(10)]

# Nombres pairs
pairs = [i for i in range(20) if i % 2 == 0]

# Application d'une fonction
import math
racines = [math.sqrt(i) for i in range(1, 11)]
```

Listing 9 – List comprehension

# 5 Fonctions

## 5.1 Définition de fonctions

```python
def factorielle(n):
    """Calcule n! de maniere iterative"""
    resultat = 1
    for i in range(2, n + 1):
        resultat *= i
    return resultat

def pgcd(a, b):
    """Calcule le PGCD par l'algorithme d'Euclide"""
    while b != 0:
        a, b = b, a % b
    return a

# Fonction recursive
def fibonacci(n):
    """Calcule le n-ieme terme de Fibonacci"""
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

Listing 10 – Definition de fonctions

## 5.2 Arguments par défaut

```python
def puissance(x, n=2):
    """Calcule x^n (n=2 par defaut)"""
    return x ** n

# Utilisation
a = puissance(5)       # 25
b = puissance(5, 3)    # 125
```

Listing 11 – Arguments par defaut

# 6 NumPy : calcul numérique

NumPy est la bibliothèque fondamentale pour le calcul scientifique en Python.

```python
import numpy as np

# Creation de tableaux
```

```
a = np.array([1, 2, 3, 4, 5])
b = np.zeros(5)             # [0, 0, 0, 0, 0]
c = np.ones(5)              # [1, 1, 1, 1, 1]
d = np.linspace(0, 1, 11)  # 11 points entre 0 et 1
e = np.arange(0, 10, 0.5)  # De 0 a 10 par pas de 0.5

# Operations vectorisees
x = np.array([1, 2, 3, 4])
y = np.array([2, 3, 4, 5])

somme = x + y          # Addition element par element
produit = x * y        # Multiplication element par element
carre = x ** 2         # Carre de chaque element

# Fonctions mathematiques
z = np.exp(x)          # Exponentielle
w = np.sin(x)          # Sinus
```

Listing 12 – Bases de NumPy

### 6.1 Matrices

```
import numpy as np

# Creation de matrices
A = np.array([[1, 2], [3, 4]])
B = np.eye(3)              # Matrice identite 3x3
C = np.zeros((3, 4))       # Matrice 3x4 de zeros

# Operations matricielles
D = A + A                  # Addition
E = A @ A                  # Multiplication matricielle
F = A.T                    # Transposee

# Algebre lineaire
det = np.linalg.det(A)           # Determinant
inv = np.linalg.inv(A)           # Inverse
valp, vecp = np.linalg.eig(A)    # Valeurs et vecteurs propres
```

Listing 13 – Matrices avec NumPy

## 7 Matplotlib : visualisation

```
import numpy as np
import matplotlib.pyplot as plt

# Trace d'une fonction
x = np.linspace(-2*np.pi, 2*np.pi, 1000)
y1 = np.sin(x)
```

```
y2 = np.cos(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y1, label='sin(x)', color='blue')
plt.plot(x, y2, label='cos(x)', color='red', linestyle='--')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fonctions trigonometriques')
plt.legend()
plt.grid(True)
plt.axhline(y=0, color='k', linewidth=0.5)
plt.axvline(x=0, color='k', linewidth=0.5)
plt.show()
```

Listing 14 – Graphiques avec Matplotlib

# 8 Exemples mathématiques

## 8.1 Méthode de Newton

```
def newton(f, df, x0, epsilon=1e-10, max_iter=100):
    """
    Trouve une racine de f par la methode de Newton
    f: fonction
    df: derivee de f
    x0: point initial
    """
    x = x0
    for i in range(max_iter):
        fx = f(x)
        if abs(fx) < epsilon:
            return x
        x = x - fx / df(x)
    return x

# Exemple: racine de f(x) = x^2 - 2 (calcul de sqrt(2))
f = lambda x: x**2 - 2
df = lambda x: 2*x

racine = newton(f, df, 1.0)
print(f"Approximation de sqrt(2): {racine}")
```

Listing 15 – Recherche de racine par la methode de Newton

## 8.2 Intégration numérique

```
import numpy as np

def integration_trapezes(f, a, b, n=1000):
```

```
    """
    Calcule l'integrale de f sur [a,b] par la methode des trapezes
    """
    x = np.linspace(a, b, n+1)
    y = f(x)
    h = (b - a) / n
    return h * (0.5*y[0] + np.sum(y[1:-1]) + 0.5*y[-1])

# Exemple: integrale de sin(x) sur [0, pi]
f = lambda x: np.sin(x)
resultat = integration_trapezes(f, 0, np.pi)
print(f"Integrale: {resultat}")  # Devrait etre proche de 2
```

Listing 16 – Methode des trapezes

### 8.3 Résolution de systèmes linéaires

```
import numpy as np

# Systeme: 2x + y = 5
#          x + 3y = 7

A = np.array([[2, 1], [1, 3]])
b = np.array([5, 7])

# Resolution
x = np.linalg.solve(A, b)
print(f"Solution: x = {x[0]}, y = {x[1]}")

# Verification
verification = A @ x
print(f"Verification: {verification}")
```

Listing 17 – Systeme lineaire Ax

## 9 Exercices

1. Écrire une fonction qui détermine si un nombre est premier.
2. Calculer la somme $\sum_{k=1}^{n} \frac{1}{k^2}$ pour différentes valeurs de $n$ et observer la convergence vers $\frac{\pi^2}{6}$.
3. Implémenter l'algorithme d'Euclide étendu pour trouver les coefficients de Bézout.
4. Créer une fonction qui calcule les $n$ premiers termes de la suite de Syracuse.
5. Tracer la fonction $f(x) = \frac{\sin(x)}{x}$ et observer sa limite en 0.
6. Résoudre numériquement l'équation $e^x = x + 2$ par la méthode de Newton.
7. Créer une fonction qui calcule le déterminant d'une matrice par développement récursif (sans utiliser NumPy).
8. Approximer $\pi$ par la méthode de Monte-Carlo.

# 10 Ressources complémentaires

— Documentation officielle Python : `https://docs.python.org/fr/3/`
— NumPy : `https://numpy.org/doc/`
— Matplotlib : `https://matplotlib.org/`
— SciPy (calcul scientifique avancé) : `https://scipy.org/`

# 11 Conclusion

Python est un outil puissant qui vous accompagnera tout au long de vos études et de votre carrière en mathématiques. La maîtrise de ce langage vous permettra d'explorer des concepts mathématiques complexes, de visualiser vos résultats et de résoudre des problèmes numériques de manière efficace.

N'hésitez pas à expérimenter, à modifier les exemples et à créer vos propres programmes !