# Résumé: Chapitres 1 à 3 du livre "Scientific Computing with Python"

**Page 1 – Introduction et environnement de travail (Chapitre 1)**

Ce premier chapitre sert d’introduction à Python dans un contexte de calcul scientifique. L’idée générale est de créer un environnement de travail stable et performant. J’ai découvert que l’usage d’un environnement isolé comme **Anaconda** est non seulement recommandé, mais presque indispensable. Cela permet d’installer Python 3.x ainsi que les bibliothèques nécessaires telles que **NumPy**, **SciPy** et **Matplotlib** sans perturber le système principal.

L’éditeur **Spyder**, fourni avec Anaconda, offre une interface complète pour programmer : à gauche l’éditeur de code, à droite la console IPython et des onglets pour la documentation. C’est l’outil parfait pour un débutant qui souhaite coder, tester, visualiser.

La configuration de base recommande d’importer dès le départ les bibliothèques principales :

Python

CopierModifier

from numpy import \*

from matplotlib.pyplot import \*

Ce n’est pas très propre sur le long terme (cela importe tout), mais pratique pour les exemples du livre.

**Page 2 – Premiers pas avec Python**

La suite du chapitre explore la syntaxe de base du langage Python :

* Les **types de données** : entiers, flottants, complexes, booléens, chaînes.
* Les **structures de contrôle** : boucles for, instructions conditionnelles if, et les break/else.
* L’usage des **fonctions** pour encapsuler du code.
* Les **scripts** Python (.py) et leur exécution via %run dans IPython.

Un point que j’ai particulièrement retenu : Python structure ses blocs via **l’indentation**, ce qui change des langages à accolades. Cela force une certaine rigueur.

Enfin, l’introduction à **Jupyter Notebook** m’a donné envie d’explorer davantage cet outil. Il permet de combiner texte, équations, code et graphiques dans un document interactif : idéal pour les devoirs ou rapports de projet.

**Page 3 – Variables et types fondamentaux (Chapitre 2)**

Ce chapitre détaille les **types de données élémentaires** en Python. Il commence par rappeler que Python est un langage **dynamiquement typé** : pas besoin de déclarer le type, il est déduit automatiquement.

**Variables**

Les variables sont des **références vers des objets**. Elles peuvent être redéfinies à tout moment, et partagent parfois le même objet (alias). Il faut donc faire attention à la mutabilité des objets manipulés.

**Nombres**

Python distingue trois types numériques :

* int pour les entiers (non bornés)
* float pour les réels (précision machine, avec arrondis)
* complex pour les nombres complexes (notés a + bj)

J’ai appris que les opérations sur les flottants ne donnent pas toujours le résultat attendu à cause des **erreurs d’arrondi**. Par exemple, 0.4 - 0.3 == 0.1 retourne False. Il faut utiliser isclose() ou allclose() pour comparer des nombres flottants.

**Valeurs spéciales**

Python (et NumPy) introduisent des constantes utiles :

* inf pour l’infini (utile en calculs limites)
* nan pour “not a number”, souvent issu d’opérations indéfinies

**Page 4 – Booléens, chaînes et conversions**

**Booléens**

Le type bool n’a que deux valeurs : True et False, mais il est **hérité de int**, ce qui permet par exemple True + 1 == 2.

Les opérateurs logiques (and, or, not) peuvent retourner des objets non booléens selon les cas. Ce comportement est parfois surprenant mais puissant :

Python

CopierModifier

result = x or y # retourne x si x est "True", sinon y

**Cast implicite**

De nombreux objets peuvent être **convertis en booléen** :

* [], '', 0, None deviennent False
* Les autres sont True

Cela permet de tester si une liste est vide ou si une chaîne contient du texte sans comparaison explicite.

**Chaînes de caractères**

Les str sont **immutables**, donc on ne peut pas modifier un caractère directement. Plusieurs opérations sont disponibles :

* Indexation, slicing
* Concaténation + ou répétition \*
* Méthodes : split(), join(), find()

Le livre insiste aussi sur le **formatage** moderne avec les f-strings :

python

CopierModifier

name = "Alice"

print(f"Bonjour {name}")

**Page 5 – Exercices de conversion et opérations avec les types**

Les exercices du chapitre m’ont permis de manipuler les fonctions float(), int(), str(), et de mieux comprendre le **comportement implicite** de Python quand il traite les types.

Un point que j’ai retenu : attention à la **conversion des types** dans les listes ou tableaux. Par exemple, si on insère un flottant dans un tableau d’entiers, il est tronqué (sans avertissement).

**Page 6 – Listes et conteneurs (Chapitre 3)**

Les **listes** sont des collections dynamiques très puissantes :

python

CopierModifier

L = [1, 2, 3]

On peut y insérer n’importe quel type, même mélangés. Les opérations incluent :

* Indexation, slicing, concaténation
* Méthodes (append, extend, pop, remove)
* Insertion par slicing
* List comprehension : [x\*2 for x in L if x > 1]

Un piège est que certaines méthodes comme sort() ou reverse() **modifient la liste en place** et ne retournent rien. Il faut bien distinguer les opérations in-place de celles qui retournent un nouveau résultat.

**Tuples**

Les tuples sont comme des listes, mais **immutables**. Ils servent souvent pour :

* des retours multiples de fonctions
* des clés de dictionnaires

Un tuple à un seul élément doit avoir une virgule : (1,).

**Page 7 – Dictionnaires et ensembles**

**Dictionnaires**

Ce sont des **collections non ordonnées de paires clé-valeur** :

python

CopierModifier

d = {'nom': 'Alice', 'age': 30}

Très utiles pour organiser des données. Les méthodes clés :

* keys(), values(), items()
* accès par clé : d['nom']
* insertion d’une nouvelle clé simplement : d['ville'] = 'Paris'

**Ensembles (sets)**

Comme en mathématiques : des collections **sans doublons** et **non ordonnées** :

Python

Copier Modifier

s = {1, 2, 3}

Ils permettent les opérations classiques : union, intersection, difference. Très utiles pour filtrer des données ou éviter les répétitions.

**Conversion entre types**

Python facilite la **conversion entre conteneurs** :

* list(), tuple(), set(), dict()
* Vérification de type avec isinstance()

**Conclusion**

Ces trois premiers chapitres posent des **bases solides** pour la programmation scientifique en Python. J’ai particulièrement apprécié la clarté des explications sur les types de données et l’attention portée à la rigueur numérique (erreurs d’arrondi, comparaisons). La manipulation des listes, des dictionnaires et des tuples est incontournable pour tout projet Python, et ce livre offre une introduction bien structurée.