1  Introduction au langage Python – Traitement d'images satellites avec Python

# 1  Introduction au langage Python

Dans ce chapitre, nous allons présenter quelques éléments essentiels du langage Python qui nous seront utiles dans ce manuel. Python est un langage très riche et peut aboutir à des projets logiciels très sophistiqués. Il est important de comprendre que la programmation Python n’est pas une fin en soit ici. Python est pour nous principalement un outil de ‘scriptage’ et de manipulation de la donnée.

Python, créé par [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) en 1991, est un langage de programmation polyvalent et facile à apprendre, souvent comparé à un couteau suisse numérique pour sa simplicité et sa polyvalence. Comme un outil multifonction, Python peut être utilisé pour une variété de tâches, du développement web à l’analyse de données, en passant par l’intelligence artificielle.

## 1.1 Les distributions

Il existe plusieurs [distributions](https://wiki.python.org/moin/PythonDistributions) du langage Python, ces distributions sont comme différentes saveurs de votre glace préférée - chacune a ses propres caractéristiques uniques, mais elles sont toutes fondamentalement Python. Voici un aperçu des principales distributions :

* [CPython](https://www.python.org/downloads/) : C’est la distribution “vanille” officielle, comme la recette originale de Python. C’est le choix idéal pour la compatibilité et la conformité aux standards.
* [Anaconda](https://www.anaconda.com/download) : Pensez-y comme à un sundae tout garni. Il vient avec de nombreuses bibliothèques scientifiques préinstallées, idéal pour l’analyse de données et le machine learning.
* [Miniconda](https://docs.anaconda.com/miniconda/miniconda-install/) : est une distribution légère de Python qui vous permet d’ajouter les librairies au besoin.
* PyPy : C’est comme une version turbo de Python, optimisée pour la vitesse.

Chaque distribution a ses forces, que ce soit la simplicité, la vitesse ou des fonctionnalités spécifiques. Le choix dépend de vos besoins, comme choisir entre une glace simple ou un banana split élaboré.

## 1.2 Les styles de programmation en Python

Il existe plusieurs approches pour programmer en Python. La plus directe est en version interactive en tapant python et de rentrer des commandes ligne par ligne.

### 1.2.1 Les outils de programmation

Un code python prend la forme d’un simple fichier texte avec l’extension .py et peut être modifié avec un simple éditeur de texte. Cependant, il n’y aura pas de rétroactions immédiates de l’interpréteur Python ce qui rend la correction d’erreurs (débogage) beaucoup plus laborieux.

Un IDE (*Integrated Developement Environnement*) est comme une boîte à outils complète pour les programmeurs, vous trouverez :

* Un éditeur de texte amélioré pour écrire votre code, avec des fonctionnalités comme la coloration syntaxique qui rend le code plus lisible.
* Un compilateur qui transforme votre code en instructions que l’ordinateur peut comprendre.
* Un débogueur pour trouver et corriger les erreurs, tel un détective numérique.
* Des outils d’automatisation qui effectuent des tâches répétitives, comme un assistant virtuel pour le codage.
* L’accès à la documentation des différentes librairies.

Ces outils intégrés permettent aux développeurs de travailler plus efficacement, en passant moins de temps à jongler entre différentes applications et plus de temps à produire du code.

Voici quelques options populaires :

* [PyCharm](https://www.jetbrains.com/pycharm/) : C’est un des outils les plus utilisés dans l’industrie. Il offre une multitude de fonctionnalités comme l’autocomplétion intelligente et le débogage intégré, idéal pour les grands projets. Cepednant, cet outil peut être assez gourmand en mémoire et en CPU.
* [Visual Studio Code](https://code.visualstudio.com/) : Gratuit, léger mais puissant, il est personnalisable avec des extensions pour Python.
* [Spyder](https://www.spyder-ide.org/) : Logiciel libre et gratuit, orienté vers les applications scientifiques.
* [Jupyter Notebooks](https://jupyter.org/) : Imaginez un cahier interactif pour le code. Idéal pour l’analyse de données et l’apprentissage, il permet de mélanger code, texte et visualisations. Des services gratuits dans le **cloud** sont disponibles comme Google Colab et Kaggle. Ces environnements sont néanmoins moins appropriées pour des grands projets et le débogage.
* Sublime Text : C’est comme un stylo élégant et rapide. Léger et rapide, il est apprécié pour sa simplicité et sa vitesse. Le choix dépend de vos besoins, que vous soyez débutant ou développeur chevronné. L’important est de trouver l’éditeur qui vous convient le mieux pour coder confortablement.

## 1.3 Bonnes pratiques

Python est un langage très dynamique, qui évolue constamment. Cela pose certains défis pour la gestion du code à long terme. Il est fortement conseillé d’utiliser des environnements virtuels pour gérer vos différentes librairies. Voici quelques bonnes pratiques à suivre :

1. **N’installez par la toute dernière version de Python** : installez toujours une version ou deux qui précède [la dernière version](https://www.python.org/downloads/). Les versions trop récentes peuvent être instables. La version de python désirée peut être spécifiée au moment de la création d’un environnement virtuel (voir plus bas). Vous pouvez afficher la liste des versions de python avec la commande conda search --full-name python. Il est recommandé d’installer 1 ou 2 version antérieure, par exemple si 3.13 est la version plus récente, installer plutôt la version 3.11.
2. **N’utilisez pas de version obsolète de Python** : cela peut sembler contradictoire avec le point 1 mais c’est l’excès inverse. Si vous utilisez une version trop ancienne alors toutes vos librairies vont cessez d’évoluer et peuvent devenir obsolète.
3. **Utilisez des environnements virtuels** : Pensez-y comme à des compartiments séparées pour chaque projet. Cela évite les conflits entre les différentes versions de bibliothèques et garde votre système propre. Par exemple, si vous souhaitez vérifier une nouvelle version de Python, utilisez un environnement : conda create --name test python=3.11
4. **Vérifiez l’installation** : Après l’installation, ouvrez un terminal et tapez python --version pour vous assurer que tout fonctionne correctement.

### 1.3.1 Création d’un environnement virtuel

Il y a deux façons d’installer un environnement virtuel selon votre distribution de Python:

1. **Option 1** : vous utilisez [Anaconda](https://www.anaconda.com/download) ou [Miniconda](https://docs.anaconda.com/miniconda/miniconda-install/), dans ce cas la commande conda est utilisée pour créer un environnement test avec Python 3.10:

conda env -n test python=3.10  
conda activate test

1. **Option 2** : vous utilisez [CPython](https://www.python.org/downloads/)

conda env -n test python=3.10  
conda activate test

### 1.3.2 Création d’un environnement de travail local (avancé)

**Note**: les notebooks peuvent fonctionner localement uniquement sous Linux ou avec WSL2.

Les notebooks Python fonctionnent par défaut dans l’environnement [Google Colab](https://colab.google/). Si vous souhaitez faire fonctionner ces notebook localement, vous pouvez installer un environnement local avec un serveur [Jupyter](https://jupyterlab.readthedocs.io/en/stable/getting_started/starting.html). Il suffit de suivre les étapes suivantes: 1. Installer WSL2 sous [Windows](https://learn.microsoft.com/en-us/windows/wsl/install) 2. Installer [vscode](https://code.visualstudio.com/docs/setup/windows) 3. Installer [Miniconda](https://docs.anaconda.com/miniconda/install/#quick-command-line-install) 4. Faire une installation du contenu du livre soit en utilisant une commande git clone ou en récupérant le .zip du livre 5. Ouvrir WSL2 et placer vous dans le répertoire du livre TraitementImagesPythonVol1. Assurez vous que vous avez accès à conda en tapant conda --version 6. Lancer la commande conda env create -f jupyter\_env.yaml 7. Activer le nouvel environnement: conda activate jupyter\_env 8. Le serveur jupyter peut ensuite être lancé avec la commande suivante: jupyter lab --ip='\*' --NotebookApp.token='' --NotebookApp.password='' Une fenêtre devrait alors apparaître dans votre fureteur. Dans le menu de gauche vous pouvez accéder aux notebooks dans le répertoire notebooks:

|  |
| --- |
|  |

Figure 1.1: La librairie NumPy est le fondement de nombreuses librairies scientifiques (d’après ([Harris 2020](references.html#ref-NumpyNature))).

## 1.4 Les structures de base en Python

Il y a essentiellement deux structures de données que Python manipule : les listes et les dictionnaires.

### 1.4.1 Les listes

Les listes sont comme des boites extensibles où vous pouvez ranger différents types d’objets :

* Représentées par des crochets : [1, 2, 3, "python"].
* Ordonnées et modifiables (mutables), vous pouvez récupérer une valeur par sa position avec [].
* Permettent les doublons (deux fois la même valeur).
* Idéales pour stocker des collections d’éléments que vous voulez modifier

### 1.4.2 Les tuples

Les tuples sont similaires aux listes, mais les boîtes sont scellées :

* Représentés par des parenthèses : (1, 2, 3, "python").
* Ordonnés mais non modifiables (immutables).
* Permettent les doublons.
* Souvent utilisé pour stocker des données qui ne doivent pas changer (comme des paramètres).

### 1.4.3 Les ensembles (Sets)

Les ensembles sont comme des boites magiques qui ne gardent qu’un exemplaire de chaque objet :

* Représentés par des accolades : {1, 2, 3}.
* Non ordonnés et modifiables.
* N’autorisent pas les doublons.
* Utiles pour éliminer les doublons et effectuer des opérations mathématiques sur des ensembles.

## 1.5 Dictionnaires

Les dictionnaires sont comme des boites avec des étiquettes sur chcune d’elle :

* Représentés par des accolades avec des paires clé-valeur : {"nom": "Python", "année": 1991}.
* Non ordonnés et modifiables.
* Les clés doivent être uniques, mais les valeurs peuvent être dupliquées
* Utiles pour stocker des données associatives ou pour créer des tables de recherche rapide

## 1.6 Programmation objet

La programmation orientée objet (POO) en Python est comme construire avec des blocs LEGO. Chaque objet est un bloc LEGO avec ses propres caractéristiques (attributs) et capacités (méthodes). Les classes sont les plans pour créer ces blocs. Par exemple, une classe “Voiture” pourrait avoir des attributs comme “couleur” et “vitesse”, et des méthodes comme “démarrer” et “accélérer”.

Python rend la POO accessible avec des fonctionnalités conviviales :

1. **Encapsulation** : Comme emballer un cadeau, elle cache les détails internes d’un objet.
2. **Héritage** : Permet de créer de nouvelles classes basées sur des classes existantes, comme un enfant héritant des traits de ses parents.
3. **Polymorphisme** : Permet à différents objets de répondre au même message de manière unique, comme si différents animaux répondaient différemment à “fais du bruit”.

Ces caractéristiques font de Python un excellent choix pour apprendre et appliquer les concepts de la POO, rendant le code plus organisé et réutilisable

**Liste des *packages* utilisés dans ce chapitre**

* Pour importer et manipuler des fichiers géographiques :
  + numpy pour manipuler des données matricielles.
  + rasterio pour importer et manipuler des données matricielles.
* Pour construire des cartes et des graphiques :
  + tmap est certainement le meilleur *package* pour la cartographie.
  + ggplot2 pour construire des graphiques.

## 1.7 Cahier de révision (notebook)

Harris, Millman, C. R. 2020. « Array programming with NumPy. » *Nature*: 357‑362. <https://doi.org/10.1038/s41586-020-2649-2>.