

---

## Chapitre 1-4

# Installer son environnement de travail

### 1. Introduction

Il ne s'agit ici que de CPython, l'implémentation de référence de Python, et non de PyPy ou Jython.

Quel que soit votre système d'exploitation, vous pouvez installer Python en lisant ce chapitre puis, dans un second temps, installer des bibliothèques tierces au gré de vos besoins (cf. section Installer une bibliothèque tierce) et vous pourrez créer des environnements virtuels (cf. section Créer un environnement virtuel).

Si vous souhaitez installer d'un seul coup Python ainsi que Jupyter (anciennement IPython) et la plupart des bibliothèques scientifiques ou d'analyse de données, vous pouvez aller directement à la section Installer Anaconda, pour installer celui-ci en lieu et place de Python. Vous disposerez alors d'autres méthodes pour gérer les environnements virtuels et pour installer des bibliothèques tierces.

### 2. Installer Python

#### 2.1 Pour Windows

Le système d'exploitation Windows requiert usuellement l'utilisation d'un installateur pour pouvoir installer un logiciel quel qu'il soit. Si vous disposez de Windows, vous devriez en avoir l'habitude. Python ne déroge pas à la règle.

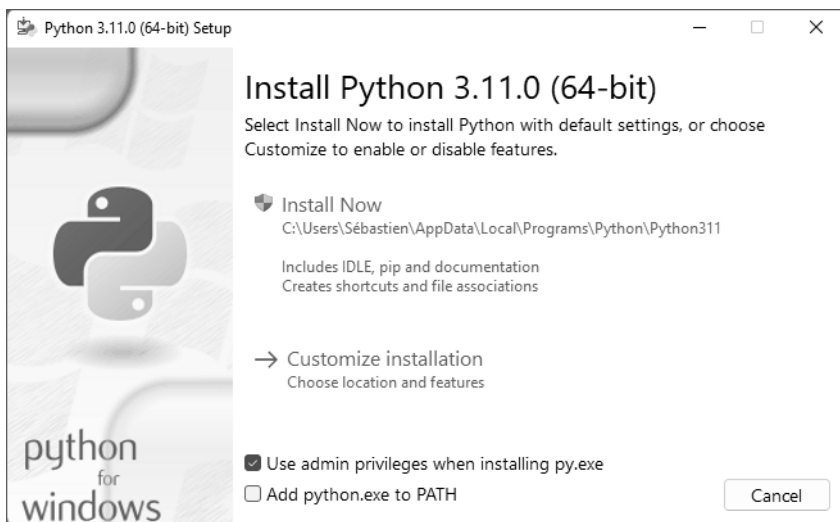
Pour installer Python, vous devez donc aller sur le site officiel (<https://www.python.org/downloads/>) pour télécharger l'installateur adéquat. Comme vous pourrez le constater, on vous met en avant un accès rapide à la dernière version (au moment où ces lignes sont écrites, la 3.11.0), puis un accès aux dernières versions encore actives (actuellement la version 3.10 qui reçoit encore des corrections d'anomalies, puis les versions 3.9 à 3.7 qui reçoivent des corrections de sécurité uniquement).

Il est également possible de télécharger la toute dernière version de la branche 2.7 qui est en fin de vie (elle n'est plus mise à jour), car il existe encore de nombreux projets n'ayant pas encore migré.

Le support correctif dure 2 ans après la première sortie de la version et le support de sécurité dure 5 ans.

Pour notre part, nous vous conseillons la dernière 3.x, mais vous êtes libre d'installer celle que vous souhaitez ou même d'en installer plusieurs suivant vos contraintes, il n'y a pas d'objection à cela.

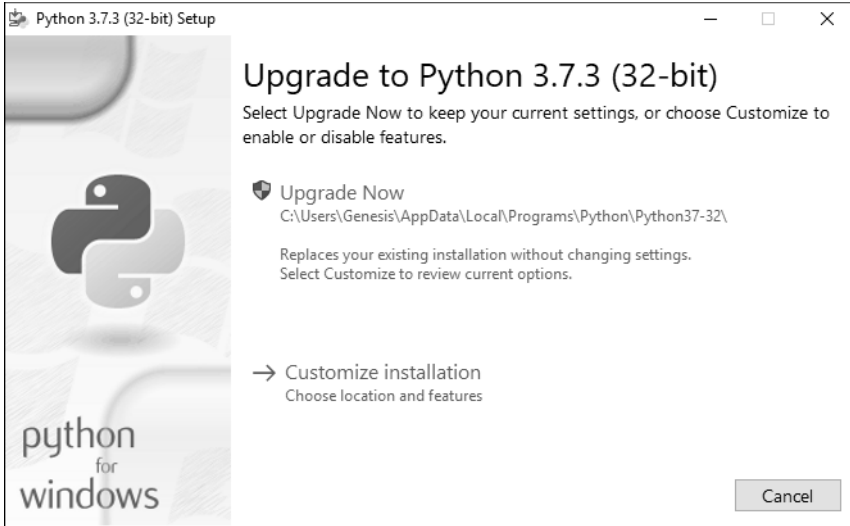
Une fois le téléchargement effectué, vous devez lancer l'installateur (et éventuellement passer quelques protections de votre système qui vous demande d'accorder votre confiance à cet installateur), pour observer l'écran suivant :



Comme vous pouvez le constater, il est possible de personnaliser l'installation en choisissant le chemin d'installation du logiciel ou en choisissant de ne pas sélectionner quelques fonctionnalités, mais nous ne le conseillons pas.

Nous vous recommandons en revanche de cocher la case **Add python.exe to PATH** afin de configurer la variable PATH du terminal pour rendre Python accessible plus facilement.

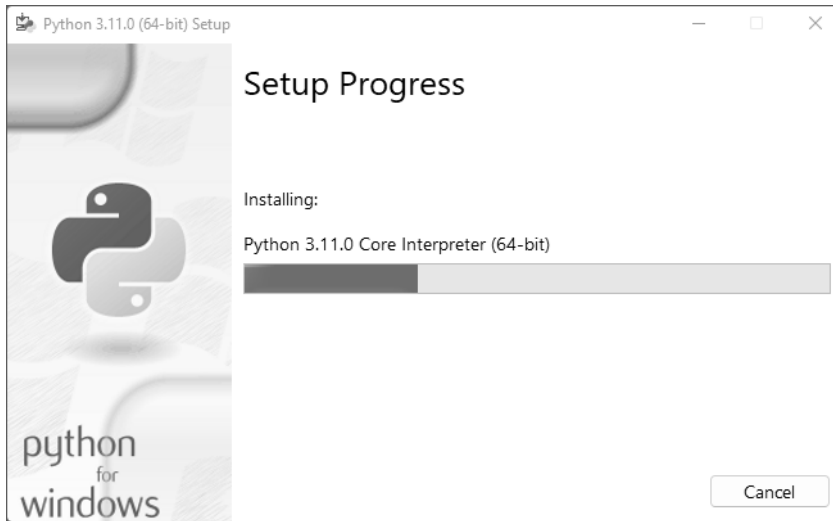
Si vous avez déjà une ancienne version de Python installée de la même branche (dans cet exemple, Python 3.7.2 est déjà installé), vous pourrez la mettre à jour à l'aide du même installateur :



Par contre, si vous avez déjà la version 3.7.1 et que vous installez la version 3.11, cette dernière ne viendra pas remplacer la précédente, mais s'installera à côté. Si vous souhaitez remplacer, il vous faudra donc désinstaller proprement la toute dernière version installée, ce qu'il est possible de faire en relançant l'installateur d'origine.

Nous vous encourageons à garder les installateurs sur votre PC, car ils pourraient devenir indisponibles au téléchargement si trop vieux.

Quel que soit le scénario, vous arriverez devant un écran vous montrant la progression de l'installation et vous n'aurez qu'à fermer la fenêtre une fois celle-ci terminée :



Vous êtes maintenant prêt à utiliser Python.

## 2.2 Pour Mac

Il faut savoir qu'une version de Python est déjà préinstallée sur Mac, car Mac OS X l'utilise pour ses propres besoins et Python est intégré à son propre cycle de développement. Cependant, si vous souhaitez une version différente de celle qui est déjà présente, vous pouvez l'installer, sachant qu'il n'y a pas de contre-indication à posséder plusieurs versions de Python sur la même machine.

Pour installer Python sur Mac OS X, la procédure à suivre est similaire à celle pour Windows. Il faut donc se rendre sur le site officiel (<https://www.python.org/downloads/mac-osx/>), télécharger un installateur correspondant à sa configuration et suivre les étapes.

Pour les utilisateurs de Mac, sachez que Python dispose d'une bonne intégration de ses spécificités, en particulier vis-à-vis de Objective-C, le langage de programmation avec lequel est développé Mac OS X, et Cocoa, interface de programmation de Mac OS X.

## 2.3 Pour GNU/Linux et BSD

Les différentes distributions libres utilisent nativement Python, notamment pour des parties sensibles. Python y est donc tout naturellement déjà installé, généralement sous la dernière version de la branche 2.x. Cependant, ici comme ailleurs, il n'y a pas d'objections à utiliser plusieurs versions de Python.

Le plus simple reste d'utiliser votre gestionnaire de paquets, ce qui peut se faire via un outil graphique, comme Synaptic pour Debian :



Il suffit alors de faire une recherche sur le mot-clé **python** pour voir les différentes versions (sur une ancienne Debian, ce sont Python 2.6, 2.7 et 3.2).

Par contre, tous les paquets `python3-xxxxx` que vous pouvez voir ici sont des bibliothèques tierces et non Python lui-même. Nous en parlerons plus tard dans ce chapitre.

Une fois les paquets souhaités sélectionnés, il ne manque plus qu'à les installer en cliquant sur le bouton **Appliquer**.

Notez que tout ceci peut se faire par la simple ligne de commande, toujours en utilisant votre gestionnaire de paquets qui peut être apt-get, aptitude, yum, emerge, pkg\_add ou autre.

Voici par exemple pour une distribution Debian ou Ubuntu :

```
$ sudo aptitude install python3
```

Ceci ne permet cependant pas de choisir la version que l'on souhaite, à moins d'aller trouver des sources alternatives. Si l'on veut avoir la toute dernière version de Python, il faudra la plupart du temps passer par la compilation.

## 2.4 Par la compilation

Compiler Python n'est pas en soi une tâche très complexe. C'est par contre souvent une tâche imposée lorsque l'on ne travaille pas avec des conteneurs. En effet, en entreprise, on développe souvent des applications qui sont destinées à être hébergées. Il est alors impératif de travailler sur votre propre poste avec une version de Python qui soit la même que celle existante sur la machine de production.

Sous GNU/Linux, mais aussi sous d'autres systèmes, il est possible de compiler la version de Python que l'on souhaite. Après tout, Python n'est rien d'autre qu'un programme écrit en C. Pour ce faire, il faut aller télécharger le code source (<https://www.python.org/downloads/source/>), qui prend la forme d'une archive, puis décompresser celle-ci, se placer dans le répertoire ainsi obtenu et taper ces quelques commandes :

```
$ ./configure --prefix=/path/to/my/python/directory
$ make
$ sudo make altinstall
```

Notez que dans cette dernière ligne, nous n'utilisons pas la commande **make install**, qui aurait pour effet de remplacer votre Python système par le Python que vous compilez, ce qui pourrait avoir des conséquences indésirables voire désastreuses.

Notez également que vous choisissez lors de la configuration le chemin dans lequel vous placerez vos bibliothèques Python. En général, l'usage veut que l'on utilise **/opt**, mais il n'y a pas de règle, tout dépend des pratiques de votre entreprise ou votre expérience en la matière.

Si vous venez d'installer Python 3.5 par cette méthode, vous aurez alors maintenant accès à ce programme en l'appelant ainsi, depuis votre terminal :

```
$ python3.5
```

Par cette même méthode, vous pouvez installer les dernières versions (<https://www.python.org/download/pre-releases/>) de Python qui ne sont pas encore sorties (alphas ou betas), ce qui vous permet de les tester en avant-première !

Notons que, par cette méthode, toutes les bibliothèques de Python ne fonctionneront pas. En effet, lorsqu'elles ont besoin d'autres bibliothèques C, il faut effectuer des compilations croisées et utiliser les différents en-têtes de ces bibliothèques. C'est le cas par exemple pour faire fonctionner Curses, ReportLab (génération de fichiers PDF) ou encore PyUSB (accès aux ports matériels USB).

Dans ce cas-là, la commande **./configure** devra recevoir des arguments supplémentaires et vous devrez trouver un tutoriel en ligne pour vous indiquer la démarche, laquelle peut être plus ou moins complexe.

## 2.5 Pour un smartphone

Installer une machine virtuelle Python sur un smartphone est possible. Pour Android, la procédure est assez simple puisqu'il existe un produit dédié (<http://qpython.com/>), tout comme sur Windows Phone (<https://apps.microsoft.com/store/detail/python-39/9P7QFQMJRFP7>). Pour iOS, c'est une autre paire de manches (<https://github.com/linusyang/python-for-ios>) étant donné que l'utilisateur est enfermé dans un système sur lequel il n'a aucun contrôle.

## 3. Installer une bibliothèque tierce

### ■ Remarque

*Si vous abhorrez le terminal, sachez que vous pouvez installer une bibliothèque tierce depuis votre IDE, ce qui sera probablement plus aisé pour vous.*

### 3.1 À partir de Python 3.4

Pour installer une bibliothèque tierce, vous devez simplement connaître son nom. Celui-ci est généralement assez intuitif. Par exemple, la bibliothèque permettant de communiquer avec un serveur Redis s'appelle `redis`.

Il peut y avoir des variations. Par exemple, la bibliothèque de référence pour traiter du XML est `lxml` et, plus complexe, celle pour BeautifulSoup est `bs4`. En recherchant comment répondre à un besoin sur le Net ou sur PyPi (<https://pypi.python.org/pypi>), vous trouverez rapidement une bibliothèque de référence.

Sur des sujets plus confidentiels, il vous arrivera de trouver plusieurs petites bibliothèques. Vous pourrez alors les tester et choisir celle que vous utiliserez pour votre projet.

Sachez que vous pouvez aussi conduire une recherche directement depuis votre terminal :

```
■ $ pip search xml
■ $ pip search soup
```

Cela vous donnera une liste de bibliothèques accompagnée d'une courte description, à la manière de ce que font les gestionnaires de paquets sous Linux (lesquels sont écrits en Python, au passage).

Sachez que **pip** existe quel que soit votre système d'exploitation (vous devez être familier avec le terminal de votre système, cependant) et que depuis la version 3.4 de Python, il est installé automatiquement avec celui-ci. Si ce n'est pas votre cas, consultez la section suivante : Pour une version inférieure à Python 3.4.

**pip** est un outil formidable. Si vous utilisez une version de Python qui est celle du système, vous utiliserez alors la commande **pip** pour gérer les bibliothèques. Si vous utilisez une autre version, telle que Python 3.5, alors vous utiliserez la commande **pip-3.5**. Pour Python 3.3, ce sera **pip-3.3**. Dans les exemples suivants, il vous faudra prendre en compte cette particularité.

Cet outil vous permettra d'installer une bibliothèque à sa dernière version ainsi que toutes les bibliothèques dépendantes. En effet, il n'est pas rare qu'une bibliothèque de Python ait besoin d'une autre bibliothèque (ou de plusieurs) pour fonctionner. Par exemple, l'installation de `redis` se fait par cette commande :

```
■ $ pip install redis
```

On peut aussi choisir la version à installer :

```
■ $ pip install -Iv redis==2.10.5
```

Ou mettre à jour la bibliothèque à une version précise :

```
■ $ pip install -U redis==2.10.5
```

Ou à la dernière version :

```
■ $ pip install -U redis
```

Et on peut la désinstaller :

```
■ $ pip uninstall redis
```

Une fonctionnalité très importante permet d'obtenir la liste des bibliothèques déjà installées (quelle que soit la manière dont elles ont été installées) :

```
■ $ pip freeze
```

Ce que l'on peut mettre dans un fichier :

```
■ $ pip freeze > requirements.txt
```

Pour installer tous les paquets ainsi listés, il faut procéder ainsi :

```
■ $ pip install -r requirements/base.txt
```

Cette méthode est particulièrement utile dans le cadre d'un environnement virtuel ; nous y reviendrons.

Il est possible de retrouver des informations sur un paquet déjà installé :

```
■ $ pip show django-redis
---
Name: django-redis
Version: 4.3.0
Location: /path/to/my/env/lib/python3.4/site-packages
Requires: redis
```

On voit ici que le paquet **django-redis** a une dépendance vers **redis** : en l'installant, on installe automatiquement **redis**.

Mettre à jour ce paquet met à jour automatiquement les dépendances :

```
■ $ pip install -U django-redis
```

Si on ne veut pas mettre à jour les dépendances, on peut procéder ainsi :

```
■ $ pip install -U --no-deps django-redis
```

On peut aussi installer plusieurs bibliothèques en même temps :

```
■ $ pip install django-redis==4.3.0 bs4 lxml
```

Cette commande installera donc automatiquement redis s'il n'est pas installé, car il est déclaré comme dépendance.

Cette commande a cependant des limites. En effet, si vous installez une bibliothèque tierce qui utilise une bibliothèque C, vous devrez disposer des en-têtes C correspondants (paquets **dev** pour Debian ou **devel** pour Fedora). Il faut donc avoir un peu de pratique dans ce genre de situation pour savoir déjouer ces pièges.

## Chapitre 3

# Préparer les données avec Pandas et Numpy

### 1. Pandas, la bibliothèque Python incontournable pour manipuler les données

Nous sommes désormais prêts à explorer et analyser nos données. Pour ce faire, nous nous appuyerons sur l'un des modules Python les plus essentiels : Pandas.

Pandas est la bibliothèque de Python permettant de manipuler et analyser les données. Elle a été créée en 2008 par Wes McKinney, un statisticien et développeur Python. Cette bibliothèque s'est progressivement imposée comme un élément incontournable pour gérer des données et a contribué à faire de Python une référence en la matière.

#### 1.1 Installation

Pour commencer, assurons-nous qu'elle est bien installée. Si tel n'est pas le cas, validons la ligne suivante dans une invite de commande :

```
■ pip install pandas
```

Une fois installée, il suffit de l'importer :

```
■ import pandas as pd
```

L'alias `pd` est couramment utilisé pour simplifier les commandes liées à Pandas. Il permet ensuite de s'assurer que les commandes pandas que nous lançons s'y réfèrent bien.

## 1.2 Structure et type de données

Avant de l'utiliser, prenons le temps d'expliquer les différentes structures que peut prendre pandas. Celle que tout le monde a en tête est le `DataFrame` à deux dimensions se présentant comme un tableur Excel, mais il existe en réalité un type par dimension :

Nom de la structure	Nombre de dimensions	Principe
Séries	1	Données unidimensionnelles indexées
DataFrames	2	Feuille de calcul avec lignes et colonnes
Panels	3	Collection de DataFrames
DataFrames multi-index	4	DataFrames avec multi-index pour les lignes ou les colonnes

### ■ Remarque

*Il existe aussi une structure `pane14D` de quatre dimensions mais elle est dépréciée et il est déconseillé de l'utiliser.*

Dans l'immense majorité des cas, seuls les séries et les `DataFrames` seront rencontrés, mais il est utile de connaître l'existence des autres structures.

L'indexation est au cœur de ces structures, de manière beaucoup plus présente que dans les tableurs. Elle permet d'accéder facilement aux lignes, colonnes ou cases. Pandas propose d'ailleurs plusieurs façons en offrant la possibilité de s'y référer soit par leur indice soit par leur nom.

Petite précision utile : l'indexation commence à 0 et non à 1.

Le tableau suivant indique les quatre façons de faire :

Action	<b>iloc</b>	<b>loc</b>	<b>at</b>	<b>query</b>
Accès à une case	<code>df.iloc[2,1]</code>	<code>df.loc[2,'Col2']</code>	<code>df.at[2,'Col2']</code>	Inapproprié
Accès à une ligne	<code>df.iloc[2, :]</code>	<code>df.loc[2, :]</code>	Inapproprié	<code>df.query("index==2")</code>
Accès à une colonne	<code>df.iloc[:,1]</code>	<code>df.loc[:, 'Col2']</code>	Inapproprié	Inapproprié
Filtrage avec condition	<code>masque = df['A'] &gt; 2</code> <code>df.iloc[masque.values, 1]</code>	<code>masque = df['A'] &gt; 2</code> <code>df.loc[masque.values, "B"]</code>	Inapproprié	<code>df.query("A&gt;2")</code>

### ■ Remarque

*La grande différence entre `iloc` et `loc`, qui sont les fonctions les plus utilisées, réside dans le fait que tout est numérique avec `iloc` alors que `loc` oblige à préciser le nom de la variable.*

## 1.3 Possibilités offertes

Passé le côté structurel, regardons ensemble ce qui a fait le succès de Pandas : sa capacité à répondre à tous les besoins inhérents aux manipulations de données.

Dès le départ, Pandas offre l'assurance de pouvoir lire quasiment tous les types de fichiers. Outre les formes les plus classiques comme les fichiers texte, CSV, Excel ou JSON, nous est aussi offerte la possibilité de lire du SQL, des fichiers propriétaires comme SAS ou SPSS et d'autres formats que nous rencontrerons plus tard comme HDF5, Parquet ou Pickle.

Une fois les données acquises, nous pouvons accéder simplement à différentes informations comme les dimensions, le type des variables, le nombre d'observations non nulles par colonne ou des statistiques descriptives de base.

Pandas nous offre ensuite toute la panoplie de fonctions pour préparer les données : supprimer des observations ou des colonnes, imputer, remplacer, fusionner d'autres fichiers ou créer de nouvelles variables.

À cela s'ajoutent des capacités de visualisations graphiques, très pratiques pour appréhender les données. Cependant, pour des besoins plus avancés, nous pourrions préférer les fonctionnalités offertes par des modules dédiés tels que Matplotlib ou Seaborn.

Cette interopérabilité avec les autres modules est d'ailleurs un des autres piliers qui fait la force de Pandas. En plus des bibliothèques graphiques mentionnées précédemment, Pandas interagit parfaitement avec tous les modules Python dédiés à la data science comme Scikit-Learn ou Numpy. Nous allons justement présenter ce dernier qui agit dans l'ombre de Pandas.

## 2. Numpy, le pilier du calcul numérique

Numpy est une bibliothèque fondamentale pour le calcul scientifique en Python agissant comme une infrastructure de base pour de nombreuses autres bibliothèques. Prenons le temps de voir ensemble ce qui fait sa force et pourquoi elle est omniprésente dans les modules Python.

### 2.1 La structure ndarray

Le ndarray, abréviation de N-dimensionnal array signifiant tableau à n dimensions, est vraiment au cœur de la bibliothèque Numpy. Deux autres structures, les matrix et scalars, sont également proposées mais nous polariserons notre attention sur le ndarray tant son rôle est central.

#### ■ Remarque

*Attention, quantité de noms différents peuvent renvoyer à un ndarray. Ainsi, le terme tableau est la forme générique pour les qualifier mais nous pouvons rencontrer le terme vecteur pour des ndarray à une dimension ou matrice pour ceux en comprenant deux. Au-delà de deux dimensions, il est courant de rencontrer les appellations : array, NDArray, tenseur ou tensor.*

Commençons par étudier les caractéristiques de cette structure fondamentale de Numpy.

### 2.1.1 Une structure homogène

Avant tout, c'est un tableau multidimensionnel homogène, c'est-à-dire qu'il peut prendre autant de dimensions que souhaité avec la contrainte que toutes les données soient du même type. Voici un moyen simple de créer un vecteur ndarray à partir d'une simple liste Python :

```
import numpy as np
a = [1, 2, 3, 4, 5]
array_numpy = np.array(a)
print(array_numpy.dtype)
```

```
# Output: int64
```

Quant à l'obligation d'homogénéité mentionnée plus haut, notons que l'introduction d'un membre de type float entraîne de facto le changement de type de l'ensemble du ndarray :

```
array_numpy_2 = np.array([1.0, 2, 3, 4, 5])
print(array_numpy_2.dtype)
```

```
# Output: float64
```

Cet exemple n'avait pour but que de créer un ndarray à partir d'une structure familière. Maintenant que nous avons pris nos marques, voici la façon de le créer directement :

```
debut = 0
fin = 10 #valeur non incluse
pas = 2
array_numpy = np.arange(debut, fin, pas) # Le pas peut être décimal

print(array_numpy)
# Output: [0 2 4 6 8]
```

Et l'éventail des possibilités ne s'arrête pas là. La commande `linspace`, par exemple, permet de créer un vecteur avec un certain nombre de valeurs d'intervalles identiques, très pratique lors de la création de graphiques :

```
array_linspace = np.linspace(0,100,15)
```

Cette simple ligne de code va ainsi générer un vecteur entre 0 et 100 inclus possédant 15 valeurs en tout à intervalles égaux.

# 66 \_\_\_\_\_ Maîtrisez la Data Science

avec Python

Voyons maintenant par quels moyens créer des matrices à deux dimensions.

Cela peut se faire directement depuis une table Pandas de la façon la plus simple possible :

```
# df est un DataFrame pandas
array_from_pandas = df.values
```

Ici, nous avons récupéré tout le DataFrame mais nous n'aurions pu obtenir qu'une seule variable sous forme de vecteur de cette façon :

```
array_col_pandas = df['nom_colonne'].values
```

Ou plus :

```
array_cols_pandas = df[['nom_colonne1', 'nom_colonne2',
                        'nom_colonne3']].values
```

Il nous est aussi offert la possibilité de créer une matrice en fixant les dimensions et le contenu. Comme une matrice nulle, remplie de zéros, à l'aide de la commande `zeros` :

```
nb_rows = 10
nb_columns = 10

shape = (nb_rows, nb_columns)
matrice_zeros = np.zeros(shape)
```

Les commandes `ones` et `full` procèdent de la même façon mais avec respectivement des 1 ou une valeur spécifique définie pour toutes les cases :

```
# Matrice remplie de 1
matrice_1 = np.ones(shape)

# Matrice remplie de 100 (Tous les nombres sont possibles) :
matrice_100 = np.full(shape, 100)
```

Enfin, signalons les différentes possibilités offertes pour générer des nombres aléatoires, qu'ils soient entiers ou décimaux.

Pour les nombres décimaux, nous pouvons demander une matrice de nombres uniformes entre 0 et 1 grâce à `random.rand` :

```
# Exemple de matrice de 3 lignes sur 4 colonnes :
matrice_rand = np.random.rand(3,4)
```

L'utilisation nous permettra d'obtenir des nombres gaussiens, c'est-à-dire qui ont une moyenne de 0 et un écart-type de 1 :

```
# Exemple de matrice de 3 lignes sur 4 colonnes :
matrice_randn = np.random.randn(3,4)
```

La création des nombres entiers va nécessiter une opération supplémentaire : l'usage du mot-clé `reshape` car il n'est pas possible de définir directement les dimensions. Nous indiquerons ainsi dans `reshape` le nombre de lignes et de colonnes souhaitées :

```
# Création d'une liste de nombres entiers aléatoires
# random.randint(low, high=None, size=None)
randint_numpy = np.random.randint(0,50,100)

# Transformation en matrice de 10 lignes sur 10 colonnes
randint_numpy = randint_numpy.reshape(10,10)
```

Cette matrice aurait pu être créée directement en une ligne en chaînant les commandes :

```
randint_numpy = np.random.randint(0,50,100).reshape(10,10)
```

### Remarque

*Attention à ce que le produit des dimensions de la nouvelle forme produite avec `reshape` corresponde exactement à la longueur de la liste de nombres aléatoires, sinon nous obtiendrons une erreur.*

```
array([[ 0, 39, 24, 36, 35,  5,  6,  3, 34, 40],
       [33, 28,  4, 26, 32, 45,  9,  5, 33,  7],
       [30,  8, 20,  7,  3, 21, 27, 44,  3, 38],
       [20,  7, 19, 31,  0,  5, 27, 43, 30,  9],
       [19,  7, 21, 37, 28,  8, 43, 46,  0, 40],
       [38, 25, 10, 34, 23, 32, 19, 26, 14, 32],
       [ 6, 33, 44, 45, 41,  4, 29, 27, 17, 35],
       [ 2, 20, 45, 15, 36, 41,  4, 49, 13, 30],
       [45, 23, 34, 35,  9, 26, 26, 22, 12, 15],
       [34, 26, 38, 46, 16, 47, 40,  0, 10, 11]])
```

Avant de clôturer ce point, signalons une possibilité commune à toutes les structures Numpy : la possibilité de définir le `dtype` qui va impacter les performances. Ainsi, plus la place allouée en bits va être élevée, plus grande sera la précision mais au détriment du temps de calcul et vice versa.

Regardons ensemble la conséquence du changement de type sur une matrice :

```
# Définition de la graine d'aléa pour retrouver le même résultat
np.random.seed(0)

m1 = np.random.randn(3,4)
m2 = m1.astype(np.float16) # Modification du type de 64 à 16 bits

print("m1:\n",m1)
print("m2:\n",m2)
```

```
m1:
[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]
m2:
[[ 1.764   0.4001  0.9785  2.24   ]
 [ 1.867   -0.977   0.95   -0.1514]
 [-0.1032  0.4106  0.144   1.454  ]]
```

Après cette exploration des différentes structures, regardons comment accéder aux données.

## 2.1.2 L'indexation

À l'instar des listes Python, les éléments des tableaux sont indexés. L'accès y est ainsi facilité. Mais la comparaison avec les simples listes Python s'arrête là car les ndarray offrent beaucoup plus de vitesse et d'options.

Commençons par découvrir comment accéder à nos données.

L'accès à un élément spécifique, dans le cadre d'une matrice à deux dimensions, consiste simplement à fournir le numéro de ligne et de colonne :

```
print(m2[0,1]) # Affichage de l'élément ligne 0 / colonne 1

# Output: 0.4001
```

Si nous souhaitons accéder à une ligne ou une colonne en particulier, il faut simplement recourir aux « : » indiquant d'inclure tous les éléments concernés :

```
print(m2[0, :]) # Affichage de la première ligne (d'index 0)

# Output: array([1.764 , 0.4001, 0.9785, 2.24   ], dtype=float16)
```

```
print(m2[ :, 1]) # Affichage de la deuxième colonne (d'index 1)

# Output: array([ 0.4001, -0.977 ,  0.4106], dtype=float16)
```

Nous pouvons aussi utiliser les « : » comme dans le cadre des listes pour afficher plusieurs lignes ou colonnes comme ici :

```
print(m2[ :, 1:3]) # Affichage des colonnes d'index 1 et 2

# Output:
array([[ 0.4001,  0.9785],
       [-0.977 ,  0.95  ],
       [ 0.4106,  0.144 ]], dtype=float16)
```

Pour finir, l'accès à des colonnes ou des lignes discontinues se fait en passant par une liste comme ici :

```
print(m2[:, [0,3]] # Affichage des colonnes d'index 0 et 3

# Output:
array([[ 1.764 ,  2.24  ],
       [ 1.867 , -0.1514],
       [-0.1032,  1.454 ]], dtype=float16)
```

Nous invitons celles et ceux qui ne sont pas encore très familiers avec ce type d'indexation à bien prendre le temps de les pratiquer car Numpy va revenir constamment au cours des différentes étapes.

Profitons de ce point sur les indexations pour aborder le sujet des manipulations des tableaux que nous allons rencontrer fréquemment.

## 2.1.3 La modification des structures

Il est courant de devoir agir sur la structure des données. En effet, certaines étapes nécessitent d'assembler des tables, d'autres de remettre le vecteur à plat ou de jouer sur ses dimensions. Une mise au point rapide s'impose en commençant par ce qu'il vaut mieux éviter : la fusion.

La fusion, également appelée merging, consistant à combiner deux tables en utilisant une clé primaire d'assemblage, est mieux réalisée avec Pandas, qui offre toutes les fonctionnalités nécessaires à cet effet. Nous verrons plus loin comment accomplir cette opération dans les meilleures conditions.