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

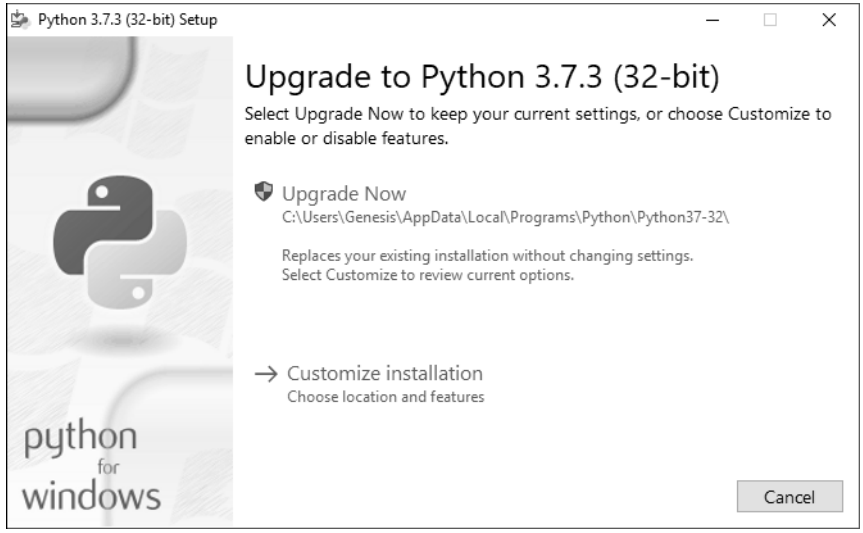
2.1 Pour Windows

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.

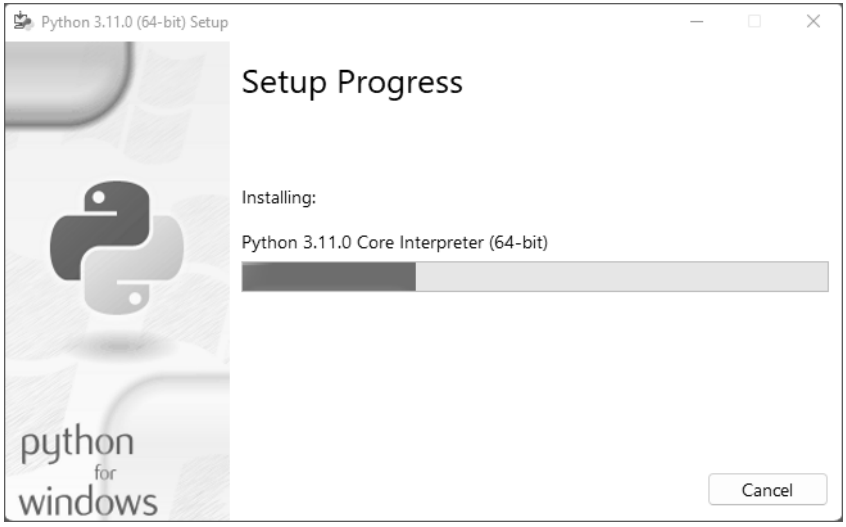
Si vous avez déjà une ancienne version de Python installée de la même branche (dans cet exemple, Python 3.7.2 est déjà installé), vous pourrez la mettre à jour à l'aide du même installeur :



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'installeur d'origine.

Nous vous encourageons à garder les installeurs 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
La pile technologique en Python

1. Les outils de la Data Science

Un bon artisan a besoin de bons outils. C'est également vrai en Data Science et en Machine Learning.

Il existe trois grandes catégories d'outils :

– Des outils « intégrés » qui contiennent ce qui est nécessaire pour un projet : charger les données, les analyser, créer des modèles, les évaluer, les déployer, créer des rapports…

– Des outils « Auto ML » qui simplifient le processus pour les non-experts, en automatisant au maximum les différentes phases.

– Des outils de « développement » avec lesquels il est possible de faire plus de choses, à condition de tout coder, par exemple en Python.

1.1 Les outils intégrés

Il existe de nombreux outils dans la première catégorie, et chacun nécessite un apprentissage particulier permettant de s'en servir au mieux. De plus, ayant chacun leurs points forts et points faibles, il est important de pouvoir choisir le bon logiciel selon le but recherché.

Il est possible de citer : Dataiku DSS, SAS, QlikView, Tableau, Power BI, Snowflake, etc. Certains sont très orientés statistiques (SAS), d'autres visualisation de données (Tableau), mais tous couvrent au moins une partie du processus. Il s'agit principalement d'applications téléchargeables et accessibles via une interface web.

Remarque

Attention, la plupart de ces logiciels sont payants. Il existe bien souvent des versions gratuites, mais qui sont bridées dans leurs fonctionnalités et qui limitent souvent leur utilisation à un usage personnel. Pour être en règle, il est donc important de se rapprocher des principaux éditeurs.

1.2 L'auto ML

Les outils d'auto ML sont des outils généralement accessibles via une interface web. L'utilisateur y rentre ses données brutes et la tâche voulue : classification, régression, etc. L'application se charge ensuite du reste : choix de la préparation des données à effectuer, choix des modèles, choix des paramètres. Le meilleur modèle créé est alors retourné à l'utilisateur.

Ces outils ont l'énorme avantage d'être utilisables sans connaissance en Machine Learning. Cependant, ils manquent de souplesse et ne sont souvent pas très aimés des Data Scientists qui préfèrent avoir le contrôle de leur processus de modélisation.

Ils permettent en revanche d'avoir un premier modèle très rapidement, et un Data Scientist peut ensuite consacrer du temps à d'autres modèles et à l'optimisation des résultats de l'auto ML. La majorité des acteurs du cloud ont leurs propres outils.

1.3 Les outils de développement

Il est tout à fait possible de coder les algorithmes de Machine Learning dans tous les langages de programmation. De nombreux frameworks ou librairies existent, prêts à être incorporés dans des projets.

Les outils de développement classiques sont donc tout à fait utilisables. Tous les éditeurs de code comme Atom, Visual Studio Code ou encore Sublim Text peuvent servir pour un projet.

De plus, le code a l'avantage de plus facilement être partagé et synchronisé entre plusieurs personnes grâce aux gestionnaires de versions comme Git. Par ailleurs, les outils de CI/CD (Intégration continue/Déploiement continu) comme GitLab CI, Jenkins ou Ansible peuvent faciliter le déploiement et la mise à jour. Là encore, chaque cloud provider propose ses outils supplémentaires, comme la série des services Amazon Web Services (AWS) dont le nom commence par « Code » : CodeCommit, CodeBuild... ou encore Azure DevOps.

Ils sont de plus en plus utilisés aujourd'hui.

2. Langage Python

2.1 Présentation

Python est un langage de programmation dont la première version est sortie en 1991. C'est donc un langage mature, connu depuis longtemps des développeurs.

Il possède plusieurs caractéristiques :

– **Interprété** : cela signifie qu'il n'y a pas de compilation avant de pouvoir exécuter le code. Il est donc facile à déboguer mais cela le rend moins efficace en termes de temps d'exécution que des langages compilés comme C/C++. Cependant, la majorité des librairies dites « bas niveau » étant codée en C, le code reste performant.

- **Multiplateforme** : un code en Python ne dépend pas de la plateforme sur laquelle il s'exécute, ce qui le rend très portable et facile à partager. Il est en particulier compatible avec Windows, Unix, Mac OS, Android, iOS.
- **Multiparadigme** : Python permet différentes formes de programmation et s'adapte donc à la majorité des développeurs. Il est ainsi possible d'écrire :
  - En programmation impérative, forme de programmation la plus ancienne
  - En programmation orientée objet, forme aujourd'hui préférée des développeurs tous langages confondus
  - Et en programmation fonctionnelle, permettant une évaluation de fonctions mathématiques et la manipulation de listes de manière simplifiée
- **Lisible** : le langage Python ne contient pas de délimiteurs de blocs contrairement à d'autres langages, comme des accolades en Java ou des mots-clés begin – end en Pascal. La structure se définit par l'indentation des blocs. De cette façon, il est plus court et lisible et, avantage non négligeable, forcément bien indenté.
- **Facile à apprendre** : que vous soyez débutant en programmation ou que vous connaissiez un autre langage, il est facile d'apprendre à coder en Python, même si maîtriser complètement le langage demande plusieurs années d'expérience.

Toutes ces caractéristiques en font un très bon langage pour la Data Science, bien qu'il ne soit pas le seul utilisable.

2.2 Brève présentation de R

R est un langage de programmation destiné aux statistiques et à la Data Science. Il s'agit d'un langage libre et open source. Ce langage a été fortement développé depuis 1997. C'est la force de sa communauté et les nombreux packages disponibles (plus de 15000) qui en font un langage si apprécié.

En effet, quel que soit l'analyse ou le calcul souhaité sur des données, il existe sûrement un package R permettant de le faire.

Sa syntaxe est cependant particulière, avec par exemple pour l'affectation le symbole <-, mais les tableaux et matrices sont très bien gérés et le code optimisé.

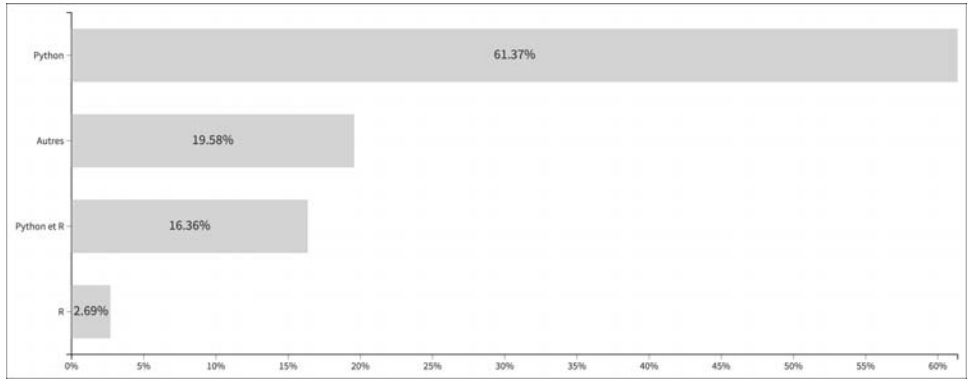
Tout comme Python, il est interprété, multiplateforme et lisible.

2.3 Python ou R ?

Python et R sont devenus les deux langages préférés des Data Scientists. Les deux permettent de coder facilement la majorité des algorithmes de Machine Learning et disposent de librairies pour accéder aux principaux formats ou sources de données.

En tant que langage de programmation, Python est particulièrement apprécié des Data Engineers, et tous les algorithmes existent dans des frameworks Python, souvent codés en C pour des questions d'optimisation.

D'après l'enquête de Kaggle sur les langages utilisés par les Data Scientists (« Data Science and Machine Learning Survey »), Python devance R depuis 2017. Le sondage réalisé fin 2022 donne les résultats suivants, ce qui fait de Python le langage préféré par trois quarts des Data Scientists :



2.4 Python 2 vs Python 3

Il existe deux versions majeures de Python :

– Python 2, supportée jusqu'à fin 2019, et dont la dernière version est la version 2.7, sortie en 2010

– Python 3, actuellement en version 3.12, sortie en octobre 2023

Les différences entre la version 2 et la version 3 sont importantes, rendant souvent la compatibilité entre du code 2.7 et 3.x impossible. De plus, de nombreuses librairies ne sont compatibles qu'avec une seule des versions.

Depuis début 2020, la version 2 n'est plus supportée et ne doit donc plus être utilisée. Il existe cependant quelques cas exceptionnels, comme du code commencé en 2.7 qui doit être maintenu en attendant une migration vers 3.x, ou du code faisant appel à une librairie non disponible en 3.x (ce qui est de moins en moins le cas).

Attention aussi au choix de la version en 3.x. La version 2.7 ayant été très longtemps la version mise en production, les éditeurs de librairies n'ont pas forcément tous créé des versions compatibles avec toutes les versions 3.x. Par exemple TensorFlow, librairie la plus utilisée pour le Deep Learning, n'est compatible avec Python 3.8 que depuis Tensorflow 2.2 et Python 3.9 que depuis sa version 2.5. À l'heure où nous écrivons ces lignes, Tensorflow n'est officiellement pas supporté pour les versions de Python supérieures à 3.9.

De même, il peut exister des contraintes sur les versions disponibles sur un système donné en particulier si le déploiement doit avoir lieu sur du matériel de faible puissance, comme des objets connectés.

Il est donc important de définir les librairies externes et les capacités des systèmes utilisés pour le déploiement en amont du projet, afin de ne pas faire de choix bloquants au moment du déploiement.

3. Jupyter

3.1 Caractéristiques de Jupyter

Jupyter est un logiciel qui propose de créer des « notebooks ». Chaque notebook est en réalité une page de taille non fixée, dans laquelle vont s'enchaîner des cellules.

Chaque cellule peut être de différents types :

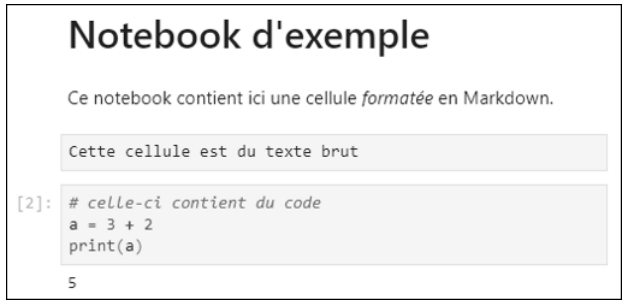
– Code, qui pourra être exécuté et dont le résultat s'affichera immédiatement sous la cellule à l'exécution.
– Texte brut, non formaté et affiché tel quel.
– Texte en Markdown, pour de la mise en page et même des formules en LaTeX.

Remarque

*Le Markdown est un langage de formatage léger créé en 2004 et utilisé dans de nombreux logiciels. Par exemple, les titres de niveau 1 commencent par #, ceux de niveau 2 par ##, etc. Les textes sont mis en italique et en gras en les entourant d'astérisques : *italique*, **gras**.*

Le LaTeX est quant à lui un langage de composition de documents. Fortement utilisé dans les milieux académiques, il permet de dissocier l'écriture du texte de sa mise en page. En effet, celle-ci est calculée lors d'une compilation du texte brut, en respectant les contraintes typographiques et le modèle voulu. Le langage LaTeX est très réputé pour sa capacité à écrire facilement des équations complexes.

Voici un exemple de rendu d'un notebook :



Chaque notebook doit être associé à un kernel. Il s'agit d'un environnement contenant le langage de programmation principal du notebook ainsi que les packages installés. Cela permet d'avoir en parallèle différentes versions d'un même langage.

Il est ainsi possible d'avoir dans le même Jupyter un kernel en Python 3.6 et un en Python 3.9, avec différents modules. En effet, chaque kernel peut avoir ses propres librairies ou des versions différentes de celles-ci, ce qui évite en partie les conflits potentiels.

Actuellement, Jupyter supporte plus de quarante langages dont Python, R, Julia, Scala… De plus, il s'intègre avec Spark, qui permet de gérer des gros volumes de données partitionnés sur plusieurs nœuds.

Le format d'enregistrement des notebooks est un fichier .ipynb qui est en réalité un fichier JSON contenant toutes les informations nécessaires : contenu des cellules et des retours s'ils sont enregistrés. Ce notebook peut ensuite être exporté dans les principaux formats d'échange, dont PDF, HTML, EPS…

Le code JSON du notebook montré précédemment est le suivant :

```
{
"cells": [
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Notebook d'exemple\n",
```