

Installation et utilisation de Python et outils associés

par **Jean-Daniel Bonjour** et **Samuel Bancal**

version 2020-09-03 - © Creative Commons BY-SA

1. Version de Python
2. Outils de développement Python
 - 2.1 Interpréteurs Python
 - 2.1.1 L'interpréteur de base CPython
 - 2.1.2 L'interpréteur interactif IPython
 - 2.2 Éditeurs de programmation
 - 2.3 IDE's pour Python
 - 2.3.1 Spyder
 - 2.3.2 Autres IDE's
 - 2.4 Jupyter Notebook
 - 2.4.1 Installation et utilisation de Jupyter Notebook
 - 2.4.2 Réalisation de slide-shows avec Jupyter
3. Installation d'un environnement Python v3 complet
 - 3.1 Installations recommandées, selon votre système d'exploitation
 - 3.1.1 Anaconda pour Windows
 - 3.1.2 Anaconda pour GNU/Linux Ubuntu
 - 3.1.3 Anaconda pour macOS
4. Première utilisation d'Anaconda
 - 4.1 Anaconda dans les salles EPFL/ENAC/SSIE GR B0 01 et GR C0 02 sous Ubuntu
 - 4.2 Anaconda Navigator, section *home*
 - 4.3 Anaconda Navigator, section *Environments*
 - 4.4 Anaconda depuis un terminal
 - 4.5 L'outil `conda` depuis le terminal
5. Autres distributions ou outils Python
 - 5.1 Installation alternative pour GNU/Linux (sans Anaconda)
 - 5.2 Distributions

1. Version de Python

De par le passé, Python a évolué à travers différentes versions. Chaque passage à une version supérieure s'est faite avec une compatibilité ascendante. Cela signifie p. ex. que du code Python s'exécutant avec *Python 3.2*, marchera également avec *Python 3.3* ou versions suivantes. Il y a cependant une exception à cela : le passage de **Python 2.x** (2000) à **Python 3.x** (dès 2008) !

Pour corriger certains défauts de jeunesse du langage, il a en effet été décidé de faire quelques changements importants qui rendent le code *Python 2* non exécutable avec *Python 3* sans certaines adaptations.

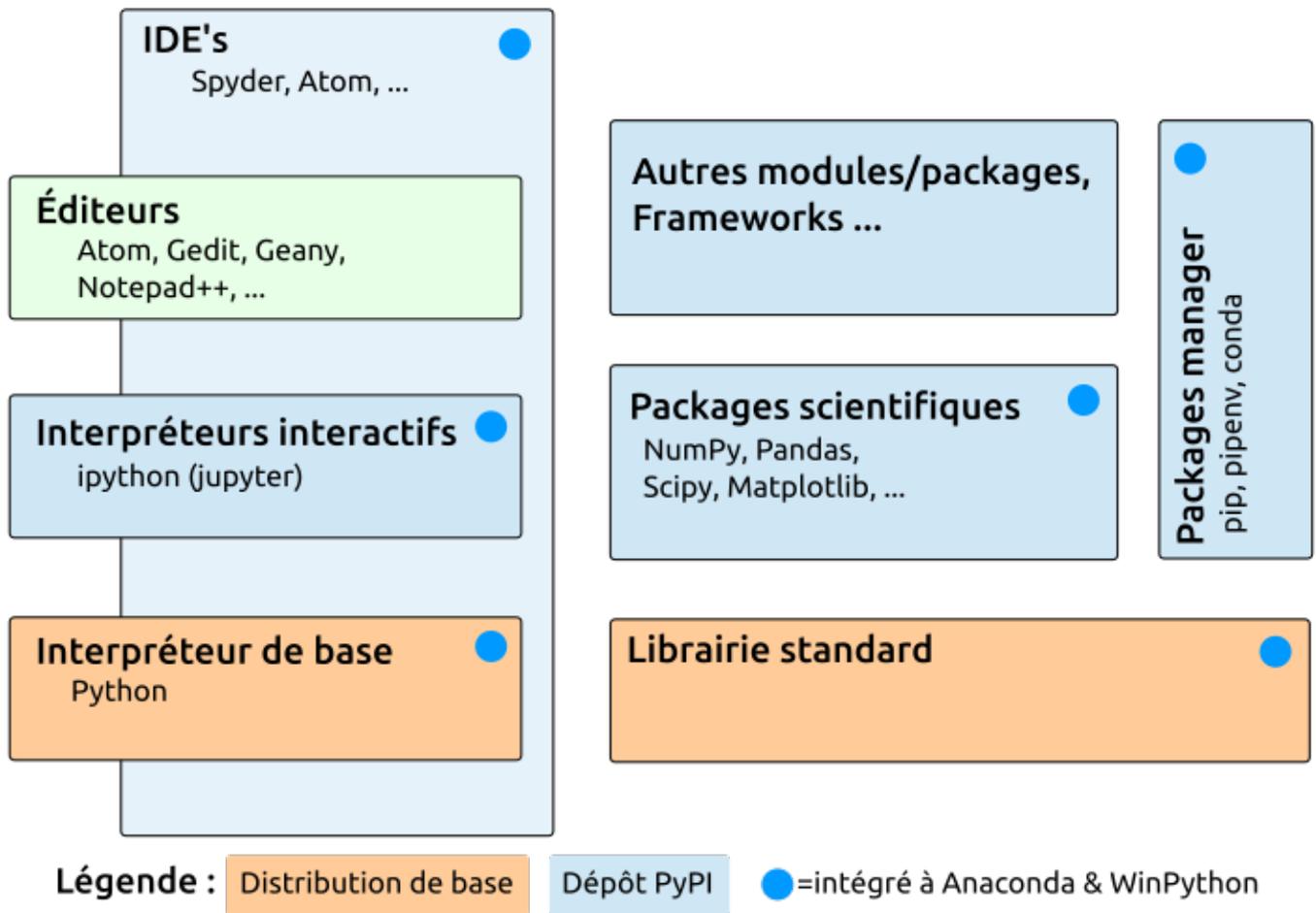
Ce changement a été fait sur plusieurs années (puisque *Python 3* est apparu en 2008 et que *Python 2* n'est plus supporté que depuis le 1er janvier 2020). Cela a donné à chacun ayant écrit du code le temps de migrer ses projets vers la nouvelle version.

Aujourd'hui, *Python 2* n'existe que d'un point de vue historique, et nous allons donc faire du **Python 3** !

2. Outils de développement Python

Un **environnement de développement** Python est constitué de différents outils et composants présentés dans la **figure** ci-dessous. Les chapitres qui suivent décrivent le rôle et l'utilisation de ceux-ci.

S'agissant de l'**installation** de Python et de ces outils sur votre machine, voyez plus bas le chapitre "[Installation d'un environnement Python v3 complet](#)".



2.1 Interpréteurs Python

Il existe de nombreuses implémentations de Python... et par conséquent d'interpréteurs. Les plus connus sont :

- l'*interpréteur de base* : interpréteur par défaut intégré à toute distribution Python, écrit en C et ainsi parfois dénommé **CPython**
- **IPython** : interpréteur interactif très évolué
- **IDLE** : mini IDE s'appuyant sur l'interpréteur de base et le toolkit graphique tkinter
- **Jython** : interpréteur écrit en Java (donc tournant sur la VM Java), permettant l'utilisation d'objets Java dans du code Python
- **PyPy** : interpréteur écrit lui-même en Python et qui a vocation d'être très rapide (JIT compiler)
- **IronPython** : implémentation du langage Python dans les environnements Microsoft .NET et Mono ; notez que cet interpréteur est resté à la version 2.7 ... ce qui est un signe de retard, que le projet n'est plus à jour.

Il faut noter que l'interpréteur de base est l'interpréteur de référence du langage Python, et que les autres interpréteurs n'implémentent souvent pas la toute dernière version du langage.

Nous n'allons présenter ici que les deux plus courants : l'interpréteur de base CPython, et l'interpréteur interactif IPython à travers Jupyter.

2.1.1 L'interpréteur de base CPython

Lorsque Python est distribué avec le système d'exploitation (ce qui est le cas sous GNU/Linux et macOS), on dispose alors de l'*interpréteur de base* et de la *librairie standard*.

 L'interpréteur **CPython** permet d'**exécuter des scripts/programmes** en frappant, depuis une fenêtre terminal : `python script.py`.

L'option `-i` est particulièrement intéressante : en frappant `python -i script.py`, juste après l'exécution du *script*, on entre dans le mode interactif de l'interpréteur, ce qui donne la possibilité d'examiner les variables globales où tracer le stack d'erreurs.

L'interpréteur de base peut donc aussi être utilisé en **mode interactif**, et c'est ce qui se passe lorsqu'on lance la commande `python` sans passer de *script* en argument. Le mode interactif est alors signalé par le prompt `>>>`. On quittera l'interpréteur en passant la commande `quit()` ou `exit()`, ou en frappant `<ctrl-D>` sous Linux et macOS, ou `<ctrl-Z>` sous Windows (qui envoie un **EOF**, c-à-d. un caractère de fin de fichier).

Le mode interactif donne accès à une aide en ligne :

- `help("topics")` : énumère les différents "concepts" Python au sujet desquels on peut demander de l'aide ; frapper ensuite `help("SUJET")` pour avoir de l'aide sur le *SUJET* souhaité
- `help(objet)` : affiche des informations sur l'*objet* Python spécifié (variable, fonction...) tel que son type, ses méthodes...
- `help("module")` : affiche l'aide relative au *module* spécifié ; si le module est importé on peut directement faire `help(module)` (sans

guillements)

- `help("module.objet")` : affiche directement l'aide relative à l'objet spécifié du *module*
- `help()` : entre dans l'aide en mode interactif

Pour exécuter un *script* lorsqu'on est dans l'interpréteur interactif, on passera la commande : `exec(open('script.py').read())` (hum... pas très convivial !).

Il est possible de se définir un prologue Python, c'est à dire un fichier de code Python qui sera exécuté lors de chaque démarrage interactif de Python (p.ex. pour charger des modules...). On le nomme en général `.pythonrc` et on le place dans le répertoire personnel de l'utilisateur. L'activation de ce prologue s'effectue par la définition de la variable d'environnement suivante dans son prologue shell (`~/bashrc`) : `export PYTHONSTARTUP=~/.pythonrc`

Mais pour un usage interactif fréquent, les possibilités de l'interpréteur de base sont très limitées, et on a bien meilleur temps d'utiliser l'interpréteur IPython présenté ci-après.

2.1.2 L'interpréteur interactif IPython

Bien que l'utilisation de IPython soit la majorité du temps faite à travers l'application **Jupyter** (décrite [plus bas](#)), il est également possible de lancer IPython de façon native.

 **IPython** est l'interpréteur Python interactif le plus utilisé actuellement en raison de sa convivialité, de ses vastes possibilités interactives et graphiques.

Pour démarrer IPython, passez la commande : `ipython`. Pour connaître les options de lancement, frappez : `ipython --help` ou `ipython --help-all`.

Pour quitter IPython, passez la commande `exit` ou `quit`, ou frappez `<ctrl-D>` et confirmez.

 La rédaction de la suite de ce chapitre (*magic fonctions*, etc...) est en cours, merci de patienter ! Mais vous pouvez déjà découvrir par vous même ces *magic fonctions* en tapant `%magic` dans IPython.

2.2 Éditeurs de programmation

Avant de faire votre choix et d'installer un éditeur, prenez connaissance du chapitre suivant sur les [IDE](#), ainsi que le chapitre sur [l'installation d'un environnement Python3](#).

N'importe quel éditeur de texte brut permet de programmer en Python. Il est cependant utile de disposer au minimum de la *coloration syntaxique* Python, ce qu'offrent la plupart des éditeurs de programmation. Nous vous recommandons notamment :

-  multiplateforme : **Atom** Excellent Éditeur *Libre*, développé par GitHub. Parmi les extensions recommandables pour Python, nous proposons :
 - `kite` : outil offrant une completion intelligente des mots en cours de frappe. Les propositions sont faites grâce à une inspection du code et des bibliothèques importées dans le code. Il offre également une documentation instantannée pour chaque symbole lorsque le curseur se place dessus.
 - `Hydrogen` : outil permettant d'exécuter du code directement depuis l'éditeur (*Jupyter* doit être installé)
 - `atom-beautify` : outil qui propose une ré-écriture du code selon un standard esthétique favorisant la lecture
 - `highlight-selected` : outil surlignant les autres occurrences du mot (variable/fonction/...) sélectionné
 - `linter-flake8` : outil de validation de la qualité de rédaction du code
 - `minimap` : outil offrant une aperçu de l'entier du code en mini sur la droite. Utile lorsqu'on commence à avoir de longs fichiers
-  sous GNU/Linux (à installer avec votre gestionnaire de packages habituel) : **Gedit**, **Geany**, **Kate**, etc...
-  sous Windows : **Notepad++**, etc...
-  sous macOS : **Sublime Text** (nécessite une licence à partir de la version 4), **TextWrangler**, etc...

Pour une liste plus complète d'éditeurs, voyez [ce lien](#).

2.3 IDE's pour Python

Utiliser un bon éditeur pour programmer en Python c'est bien. Recourir à un environnement de développement (**IDE**, *Integrated Development Environment*) c'est encore plus confortable et puissant. Pour un usage scientifique de Python, Spyder figure parmi les plus répandus.

2.3.1 Spyder

 **Spyder** (Scientific Python Development EnviRonment) est un IDE orienté vers un usage scientifique de Python et doté de fonctionnalités avancées d'édition, debugging, introspection et profiling. L'utilisateur MATLAB ou GNU Octave GUI se retrouvera dans un environnement familier, avec notamment des fenêtres Console, Editor, Variable explorer, File explorer, History, Online help... Disponible sur tous les systèmes d'exploitation, cet IDE est lui-même écrit en Python et s'appuie sur le toolkit graphique multiplateforme [Qt](#) (nécessitant les packages PyQt et PySide).

Cet IDE s'installe sous GNU/Linux de façon standard (voir [ce chapitre](#)). S'agissant des autres OS, il est notamment intégré aux *bundles* *WinPython* et *Anaconda* présentés plus loin.

 La rédaction de la suite de ce chapitre est en cours, merci de patienter !

2.3.2 Autres IDE's

Parmi les IDE's bien adaptés à l'écriture de code Python, mentionnons en outre :

-  **PyCharm**, un IDE Python de plus en plus en vogue, existant en versions Community (libre) et Professional (payante, incluant support du développement HTML, JavaScript et SQL)
-  **Eclipse**, l'historique IDE libre et multiplateforme, se prête bien entendu aussi au développement Python, complété par l'extension **PyDev**
-  **Pyzo**, IDE libre interagissant bien avec *Anaconda*
-  **Eric Python IDE**, IDE libre basé Qt/Scintilla et architecture de plugins
-  **PyScripter**, IDE Python libre spécifiquement Windows

Pour une liste plus complète d'IDE's orientés Python, voyez [ce lien](#).

2.4 Jupyter Notebook

L'interpréteur IPython versions 0.12 à 3.x offrait une fonctionnalité appelée *Ipython Notebook* qui permettait, à la façon d'autres logiciels scientifiques (Mathematica, Maple...), de créer des documents interactifs composés de code Python vivant, de texte formaté (Markdown, HTML et LaTeX) et de graphiques.

Cette fonctionnalité a été sortie du projet IPython pour constituer un outil indépendant dénommé  **Jupyter**. L'objectif de cette scission a été de rendre cette technologie de *notebooks* accessible à d'autres langages de programmation que Python. Bien entendu Jupyter Notebook est toujours utilisable avec le langage Python, mais il supporte désormais des dizaines d'autres langages (par exemple GNU Octave). En outre la plupart des fonctionnalités IPython décrites au chapitre précédent sont utilisables dans les *cellules* de code du Notebook.

2.4.1 Installation et utilisation de Jupyter Notebook

Le chapitre [Installation](#) vous indique, selon votre système d'exploitation, comment installer Jupyter et le démarrer en mode console, Notebook ou Lab.

Pour découvrir les principes de base d'**utilisation de Jupyter** (création et manipulation de cellules, texte Markdown...), vous pouvez ensuite télécharger [ce notebook](#) (avec `<clic-droite> Save link as`). Ouvrez-le dans Jupyter et exercez ce qui est indiqué.

Nous vous mettons aussi à disposition un [autre notebook](#) d'introduction rapide aux **bases du langage Python**.

2.4.2 Réalisation de slide-shows avec Jupyter

Jupyter offre en standard des fonctionnalités permettant de présenter un notebook sous forme de slide-show. Voyez [cet exemple](#) ainsi que le [notebook](#) Jupyter associé.

Pour créer un slide-show sur la base d'un notebook existant, il faut commencer par définir quelles cellules seront affichées en mode slide-show, et quel sera l'enchaînement de la présentation. Pour cela, activer le mode : **View > Cell Toolbar > Slideshow** (bascule). Un menu déroulant "Slide Type" apparaît alors au haut de chaque cellule, offrant les options suivantes :

- Slide : la cellule débutera un nouveau slide
- - (ou vide) : la cellule apparaîtra sur le slide courant en même temps que la précédente cellule (défaut)
- Fragment : la cellule apparaîtra sur le slide courant au prochain clic sur flèche `▼` ou `>` (ou touche clavier correspondante)
- Sub-slide (sous-slide) : cellule apparaissant avec un clic sur flèche `▼` (mais pas avec `>`)
- Skip : la cellule restera masquée en mode présentation
- Note : définit une cellule de notes de conférencier

Voyons maintenant les 2 techniques de présentation proprement dites.

A) On peut générer la présentation sous forme d'un fichier HTML complètement stand-alone. Pour cela, commencer par sauvegarder et quitter le notebook, puis passer la commande : `jupyter nbconvert notebook.ipynb --to slides --post serve`. Cela fabrique un fichier nommé `notebook.slides.html`, et l'option `--post serve` provoque son ouverture automatique dans une fenêtre de navigateur web. Le contrôle du déroulement du slide-show s'effectue avec les actions suivantes :

- avancer au point/slide suivant avec les touches `>` et `▼` (ou clic sur les symboles correspondants)
- bascule en mode plein écran (et vice-versa) avec `<F11>`
- aperçu de tous les slides (planche de contact) avec `<esc>`

B) Avec la technique ci-dessus, en mode présentation les cellules ne peuvent pas être éditées et exécutées. Cela est cependant possible en recourant à l'extension **RISE** (Reveal.js Jupyter/IPython Slideshow Extension) :

- cette extension est déjà intégrée à la distributions WinPython sous Windows
- avec la distribution Anaconda sous macOS, il faudra préalablement installer cette extension avec la commande : `pip install rise`
- sous Linux, il faudra passer les 3 commandes :

- installation de l'extension : `pip3 install --user rise`
- mise en place des fichiers JavaScript et CSS : `jupyter nbextension install rise --user --py`
- activation/validation de l'extension : `jupyter nbextension enable rise --user --py`

Lorsque vous ouvrirez un notebook, vous verrez dès lors apparaître un nouveau bouton `[Enter/Exit RISE Slideshow]` sur la droite de la barre d'outils du notebook. Équivalent au raccourci `<alt-r>`, ce bouton fonctionne comme une bascule et permet de passer interactivement du mode d'affichage normal (notebook) au mode slide-show et vice versa, sans qu'il soit nécessaire de "convertir" le notebook en présentation. Le contrôle du déroulement du slide-show s'effectue ensuite avec les actions suivantes :

- avancer au point/slide suivant avec les touches `>` ou `<espace>`
- bascule en mode plein écran (et vice-versa) avec `<F11>`
- aperçu de tous les slides (planche de contact) avec `w`
- ouverture d'une seconde fenêtre de navigateur pour l'affichage des notes du conférencier avec `t`
- affichage/masquage des boutons Exit et Aide avec `,`

3. Installation d'un environnement Python v3 complet

Python n'est pas installé par défaut sous Windows. Sous GNU/Linux et macOS par contre, l'*interpréteur de base* Python ainsi que la *bibliothèque standard* sont intégrés au système :

- 🐍 depuis GNU/Linux Ubuntu 14.04 on disposait en parallèle des deux versions Python v2 et v3 ; Ubuntu 20.04 n'offre désormais plus que Python v3 par défaut.
- 🍏 sous macOS 10.11 (*El Capitan*, 2015), 10.12 (*Sierra*, 2016), 10.13 (*High Sierra*, 2017), 10.14 (*Mojave*, 2018), 10.15 (*Catalina*, 2019), Apple est très conservateur et n'offre en standard que Python 2.7

🐍 🍏 🐧 **Anaconda** est un *bundle* Python3 très en vogue dans le monde scientifique. Développé par la société Anaconda Inc., libre dans sa version de base (nommée "Individual Edition"), il a l'avantage d'être multiplateforme (i.e. disponible sous Windows, macOS et GNU/Linux) et d'intégrer une grande quantité d'outils et packages Python, notamment : IPython, PIP, Jupyter, Spyder, NumPy, SciPy, Matplotlib, Pandas, Sympy... (voir la [liste complète](#)).

Ce support de cours étant orienté Python3, les chapitres qui suivent ont pour objectif de vous aider à installer sur votre propre machine, selon votre système d'exploitation et de façon non intrusive, l'environnement de développement **Anaconda** qui comportera la bibliothèque standard, ainsi que les packages de base couramment utilisés pour du calcul scientifique (Numpy, Scipy, Matplotlib et bien plus encore) ainsi que IPython, Jupyter et Spyder.

3.1 Installations recommandées, selon votre système d'exploitation

3.1.1 Anaconda pour Windows

Il serait possible d'installer l'interpréteur Python v3 de base (incluant sa bibliothèque standard) sous Windows en utilisant l'installateur proposé par le [site Python](#), mais l'ajout d'autres outils n'est ensuite pas évidente. Nous nous orientons donc plutôt vers un bundle complet.

1. Rendez-vous sur la page <https://www.anaconda.com/distribution/>
2. Cliquez sur *Download* ce qui vous rendra en bas de la page
3. Votre système d'exploitation (Windows) devrait être automatiquement détecté
4. Deux possibilités sont offertes : Choisissez l'installateur graphique 64 bits.
5. Le fichier `Anaconda3-version-Windows-x86_64.exe` (env. 466 MB) est alors téléchargé (notez le `3` qui correspond à la version 3 de Python)
6. Passez ensuite à l'installation proprement dite en double-cliquant sur cet outil d'installation. Acceptez les propositions par défaut (la licence, l'installation au niveau système donc multi-utilisateur, l'emplacement d'installation par défaut qui est `C:\ProgramData\anaconda3`, qui pèsera env. 5.0 GB)
7. Une fois l'installation terminée, vous pouvez jeter le fichier d'installation que vous aviez téléchargé.

Vous pouvez à présent lancer **Anaconda Navigator** depuis le menu Démarrer.

3.1.2 Anaconda pour GNU/Linux Ubuntu

S'agissant tout d'abord de l'*interpréteur Python3* et sa *bibliothèque standard* : rien à faire de spécial, car Ubuntu embarque déjà Python 3 depuis sa version 14.04.

Si vous souhaitez investiguer d'autres façon d'installer votre environnement Python sur Ubuntu sans Anaconda (ce qui sera plus léger), référez-vous au chapitre [Installation alternative pour GNU/Linux](#).

1. Rendez-vous sur la page <https://www.anaconda.com/distribution/>
2. Cliquez sur *Download* ce qui vous rendra en bas de la page
3. Votre système d'exploitation (Linux) devrait être automatiquement détecté
4. Deux possibilités sont offertes : Choisissez l'installateur graphique 64 bits.
5. Le fichier `Anaconda3-version-Linux-x86_64.sh` (env. 550 MB) est alors téléchargé (notez le `3` qui correspond à la version 3 de Python)
6. Rendez ce fichier téléchargé exécutable avec la commande `chmod +x Anaconda3-version-Linux-x86_64.sh`

7. Passez ensuite à l'installation proprement dite en l'exécutant : `./Anaconda3-version-Linux-x86_64.sh`. Acceptez les propositions par défaut (la licence, l'installation au niveau système donc multi-utilisateur, l'emplacement d'installation par défaut qui est `/home/username/anaconda3`, qui pèsera env. 3.6 GB)
8. Une fois l'installation terminée, vous pouvez jeter le fichier d'installation que vous aviez téléchargé.

Notez que l'installateur a ajouté un bloc d'intégration à votre fichier `$HOME/.bashrc`. Cela permet dans tout terminal `bash` d'avoir accès aux outils *Anaconda* sans devoir se soucier où il se trouve. Si vous utilisez un autre shell que `bash`, il vous sera nécessaire de reporter ce bloc dans le fichier correspondant (`$HOME/.zshrc` pour `zsh` par exemple).

Vous pouvez à présent lancer `anaconda-navigator` depuis une console.

3.1.3 Anaconda pour macOS

Il serait possible d'installer l'interpréteur Python v3 de base (et sa librairie standard) sous macOS en utilisant l'installateur proposé par le [site Python](#). Une autre alternative assez courante et bien documentée sur Internet serait d'utiliser le gestionnaire de paquets [Homebrew](#). Nous nous orientons cependant ici plutôt vers l'installation du bundle Anaconda complet, très répandu.

1. Rendez-vous sur la page <https://www.anaconda.com/distribution/>
2. Cliquez sur *Download* ce qui vous rendra en bas de la page
3. Votre système d'exploitation (mac OS) devrait être automatiquement détecté
4. Deux possibilités sont offertes : Choisissez l'installateur graphique 64 bits.
5. Le fichier `Anaconda3-version-MacOSX-x86_64.pkg` (env. 650 MB) est alors téléchargé (notez le `3` qui correspond à la version 3 de Python)
6. Passez ensuite à l'installation proprement dite en double-cliquant sur cet outil d'installation. Acceptez les propositions par défaut (la licence, l'installation au niveau système donc multi-utilisateur, l'emplacement d'installation par défaut qui est `/anaconda3`, qui pèsera env. 2.6 GB)
7. Une fois l'installation terminée, vous pouvez jeter le fichier d'installation que vous aviez téléchargé.

Si vous désirez utiliser Anaconda depuis un autre compte que celui sous lequel il a été installé, cela est possible en ajoutant simplement, dans votre fichier `~/.bash_profile`, la ligne suivante : `export PATH="/anaconda3/bin:$PATH"`

Vous pouvez à présent lancer `Anaconda-Navigator` depuis le dossier *Applications*.

4. Première utilisation d'Anaconda

4.1 Anaconda dans les salles EPFL/ENAC/SSIE GR B0 01 et GR C0 02 sous Ubuntu

L'installation dans ces salles d'enseignement comprend par défaut un Python3 qui fait partie du système d'exploitation. Celui-ci peut être utilisé tel quel.

Pour utiliser Anaconda dans ces salles, il vous faut l'activer depuis le terminal avec la commande :

```
./opt/anaconda3/activate_env.sh
```

Cela vous permet ensuite d'utiliser Anaconda. Vous voyez d'ailleurs apparaître le préfixe (`base`) qui en est le témoin. Sachez encore que cet environnement *base* appartient à l'utilisateur `root` (administrateur de la machine), donc il vous faudra créer un environnement supplémentaire si vous souhaitez installer des nouveaux packages.

4.2 Anaconda Navigator, section *home*

Une fois lancé l'outil `anaconda-Navigator`, vous pouvez, depuis depuis la section "home", lancer différents outils, tel que :

1. **JupyterLab** qui s'ouvrira dans une fenêtre de votre navigateur web par défaut. Vous pouvez cliquer sur l'icône du Notebook Python3 ce qui créera un nouveau notebook. Testez que cela marche avec un simple `2+2` ou `print('Hello World!')`.
2. **Spyder**. Au premier lancement, Jupyter vous proposera d'installer **Kite**. Il s'agit d'un module qui vous aidera bien dans la rédaction de votre code. Donc Acceptez l'installation de celui-ci. Si vous deviez choisir de l'installer plus tard, vous pouvez vous rendre sur <https://www.kite.com/integrations/spyder/>. Vous pouvez également tester cet outil en écrivant un nouveau fichier à gauche avec pour simple instruction `print('Hello World!')`, puis l'exécuter avec l'icône **Run** (Raccourcis F5). Vous pouvez également lancer des commandes Python dans la partie en bas à droite (`2+2` sera un bon moyen de voir que ça marche).

4.3 Anaconda Navigator, section *Environments*

Dans la le menu sur la gauche, en choisissant *Environments*, vous pouvez

1. Créer / Cloner / Importer / Supprimer des environnements virtuels
2. Pour chacun de ceux-ci, installer / désinstaller des packages et même choisir leur version

4.4 Anaconda depuis un terminal

-  sous GNU/Linux et Mac OS, il suffit d'ouvrir un terminal et l'environnement Anaconda y est à disposition.
-  sous Windows, il vous faut lancer *Anaconda Prompt* depuis le menu démarrer

Dans ce terminal, l'*invite* (c-à-d. le texte marqué à gauche de la commande que l'utilisateur peut saisir) est préfixée maintenant par un `(base)`. Cela signifie que nous utilisons l'environnement virtuel nommé *base*. Comme décrit dans le chapitre des [environnements](#) ou juste ci-dessous, il est possible de fonctionner avec plusieurs de ces environnements et pour chacun d'eux, choisir quel package utiliser.

Ainsi, vous pouvez lancer :

- `spyder` pour lancer l'IDE **Spyder**
- `jupyter-lab` pour lancer **Jupyter Lab** dans un navigateur web
- `anaconda-navigator` pour lancer l'outil de gestion Navigateur Anaconda
- `python` pour utiliser l'interpréteur Python v3
- `ipython` pour avoir un interpréteur IPython dans la fenêtre terminal
- `jupyter notebook` pour manipuler des Jupyter notebooks (préférez Jupyter Lab)

4.5 L'outil `conda` depuis le terminal

Vous pouvez créer et gérer vos environnements virtuels avec la commande `conda` :

- `conda create -n env` : crée un nouvel environnement Python nommé *env*
- `conda install (-n env) packages` : installe les *packages* spécifiés dans l'environnement *env* si spécifié, sinon dans l'environnement actuel
- `conda list (-n env)` : affiche la liste des packages installés et leur version courante
- `conda update (-n env) packages` : met à jour les *packages* spécifiés
- `conda remove (-n env) packages` : supprime les *packages* spécifiés
- `conda search query` : affiche tous les packages (installés ou disponibles) correspondant à *query*
- `conda env list` : affiche la liste de tous les environnements présents
- `source activate env` : bascule dans l'environnement choisi (modification du PATH du shell courant)

Voici encore 2 commandes spécifiques qui sont utiles de façon générale :

- `conda update conda` : mettre à jour Python et le gestionnaire de packages Conda
- `conda update anaconda` : mettre à jour tous les packages Anaconda (dans cette commande, `anaconda` représente un *méta-package* rassemblant l'ensemble des packages proposés par Anaconda)

5. Autres distributions ou outils Python

5.1 Installation alternative pour GNU/Linux (sans Anaconda)

On vous présente ici 2 autres façons d'installer les outils et bibliothèques Python sous Ubuntu, en comparaison avec la solution Anaconda :

A. Anaconda, tel que documenté [plus haut](#)

- avantages :
 - très bonne intégration globale de tous les packages scientifiques et IDE Spyder
 - même application que pour les autres Systèmes d'exploitation (Windows et Mac OS)
- inconvénients :
 - prend plus de place que d'utiliser ce qui est déjà installé au niveau système
 - utilisation d'une surcouche logicielle dont on pourrait se passer
 - risque de changement de licence à venir Les 2 avantages cités ci-dessus justifient le choix de cette solution pour l'enseignement les cours *ENG-270 Outil Informatique pour l'ingénieur en environnement* et *ENG-274 Programmation Matlab.*

B. Soit une installation dans le **compte utilisateur** (c-à-d. dans `~/.local`) basée sur le dépôt **PyPI** en utilisant le gestionnaire de paquets `pip3` :

- avantages :
 - offre le plus vaste choix de packages
 - accès aux dernières versions de packages, ou libre choix des versions
 - possibilité de créer des `virtualenv` (et conserver avec chaque appli tout son environnement)
 - possibilité d'automatiser la gestion des `virtualenv` avec `pipenv`
- inconvénients :
 - consomme de l'espace-disque dans le répertoire de l'utilisateur
 - l'installation de Spyder est problématique

C. Soit une installation **au niveau système** et basée sur le **packaging Ubuntu/Canonical** :

- avantages :
 - simplicité (usage des procédures d'installation standards Ubuntu)
 - sécurité (packaging réalisé par Canonical, mises à jour automatiques en cas de failles de sécurité)

- outils accessibles depuis tous les comptes utilisateur de la machine
- ne consomme pas d'espace disque utilisateur
- inconvénients :
 - moins de choix de packages qu'avec PIP (accès uniquement à ce qui est packagé par Canonical)
 - plus figé (en terme de versions)

Méthode B, un peu orientée développeur, mais offrant le plus de flexibilité :

1. installation du gestionnaire de paquets **PIP3** (pour Python v3) : `sudo apt install python3-dev python3-pip`
2. installation de **IPython** : `pip3 install --user ipython`
3. installation de **Jupyter** : `pip3 install --user jupyter`
4. installation de **JupyterLab** : `pip3 install --user jupyterlab`
5. installation des librairies **Scientific Python** v3 de base : `pip3 install --user numpy scipy matplotlib`
6. insertion dans votre fichier `~/.bashrc` de la ligne suivante : `export PATH=$PATH:~/local/bin`
7. note : l'installation ultérieure de toute autre librairie/package **PyPI** se fera avec : `pip3 install --user package`

Méthode C, plus simple, mais ne donnant pas accès aux dernières versions/fonctionnalités :

1. installation de **IPython** pour Python v3 : `sudo apt install ipython3 ipython3-qtconsole` (env. 280 MB)
2. installation de **Jupyter** : `sudo apt install jupyter-qtconsole jupyter-console jupyter-notebook jupyter-client` (env. 130 MB)
3. installation des librairies **Scientific Python** v3 de base : `sudo apt install python3-numpy python3-scipy python3-matplotlib` (env. 95 MB)
4. éventuelle installation de l'IDE **Spyder** : `sudo apt install spyder3` (env. 160 MB)
5. note : l'installation de toute autre librairie/package packagé par Canonical se fera avec : `sudo apt install python3-package`

Depuis une fenêtre terminal, vous pouvez maintenant utiliser les commandes suivantes :

- `ipython3` pour avoir un interpréteur IPython dans une fenêtre terminal
- `jupyter notebook` pour manipuler des Jupyter notebooks Python v3 dans un navigateur web
- `jupyter-lab` pour démarrer JupyterLab dans un navigateur web
- `spyder3` pour démarrer l'IDE Spyder

5.2 Distributions

L'inventaire ci-dessous n'est pas exhaustif.

 **WinPython** est un *bundle* Scientific Python très complet, également beaucoup utilisé par la communauté scientifique Python. Il est cependant uniquement disponible sous Windows.

   **Enthought Canopy** (ex- Enthought Python Distribution) est un *bundle* Scientific Python multiplateforme. Commercial, il est cependant gratuit pour un usage académique ou dans la version de base Canopy Express. Il n'inclut pas l'IDE Spyder.

   **Active Python** (de la société ActiveState Software Inc.) est une distribution Python multiplateforme importante, mais moins orientée vers un usage scientifique. Proposée en deux versions : Community (gratuite) et Business (payante).

 **Python(x,y)** est un *bundle* Scientific Python libre pour Windows. Plus ancien que WinPython, il tourne sous Windows 64bit mais en mode 32bit. Basé sur l'IDE Eclipse.

 **Cygwin**, l'environnement libre bien connu d'émulation Unix pour Windows, permet l'installation et utilisation de Python. Mais cette solution n'a de sens que pour ceux utilisent/maîtrisent déjà Cygwin.

 Sous macOS il est aussi possible d'installer Python via l'un des systèmes de packaging **Homebrew**, **Fink** ou **MacPorts**. Mais c'est relativement lourd (peut nécessiter l'environnement de développement Apple XCode) et moins évident lorsque l'on veut intégrer certains outils Python annexes (Qt, IDE, etc...)



Jean-Daniel Bonjour et Samuel Bancal, 2015-2020