PAPY_TP1_eleves

September 8, 2023

1 TP 1 : Environnement de programmation et prise en main

Durée: 4h

> **Objectifs:** - Mise en place d'un environement commun * Mise en place d'un système unix de développement * Installation et déploiement conda * Prise en main IDE/Git * installation jupyter - S'exercer sur le python de base

Remarque: une part importante des activités de développement consiste à rechercher de l'information (principalement en ligne) et de l'appliquer à son problème. Dans la pratique, on évitera de copier coller une solution incomprise depuis stackoverflow. Par contre certaines explications sont de très bonnes qualités, il faut savoir trouver un bon équilibre! Enfin, un grand nombre d'outils et bibliothèques possèdes des documentation de très bonne qualité comme nous allons le voir avec les docs *microsoft, anaconda, miniconda, jupyter, python, etc.*

1.1 Système d'exploitation (OS)

Windows permet l'utilisation native de python mais ce n'est pas très pratique

On va s'équiper d'un environemement UNIX (mac/linux) si ce n'est déjà fait

1.1.1 Windows Subsytem Linux (WSL)

Suivre https://docs.microsoft.com/fr-fr/windows/wsl/install

ou https://lecrabeinfo.net/installer-wsl-windows-subsystem-for-linux-sur-windows-10.html

Pour un plus grand confort, installer le terminal windows https://apps.microsoft.com/store/detail/windows-terminal/9N0DX20HK701?hl=fr-fr&gl=FR

A partir de maintenant il est vivement recommandé d'effectuer les manipulations dans la WSL ou sur votre systeme linux.

> **Nous n'assurons pas le support windows**

1.1.2 Les machines virtuelles (VM)

Vous pouvez aussi installer très facilement des VM, en utilisant par exemple `VirtualBox` pour virtualiser un système de test. **Ce n'est pas recommandé pour nos TPs puisque les performances sont assez mauvaises.**

1.1.3 Docker (optionnel)

Une autre approche est d'utiliser des *container* comme docker. Si vous êtes administrateur de votre machine, je vous recommande de tester cette approche. Elle permet d'invoquer un environnement d'exécution isolé et contenant tous les outils nécessaires au bon fonctionnement d'un programme. Cette approche est très utilisée pour le déploiement de services sur des fermes de serveur. En local, elle permet par exemple l'utilisation simultanée de logiciels incompatibles entre eux (par exemple lié à problèmes de version) et surtout un contrôle fin et pérenne des versions de logiciels/bibliothèques nécessaire au bon fonctionnement d'une application.

> **Remarque** Il n'est pas nécessaire d'installer Docker pour ce TP, vous pouvez passer à la suite si vous le souhaitez.

Installation https://docs.docker.com/get-docker/

Quelques bases https://docs.docker.com/get-started/overview/ propose une introduction très complète.

On peut par exemple lancer un système ubuntu très simplement depuis windows en exécutant la commande suivante:

```
$ docker run -it ubuntu
```

Qui recherchera l'image d'ubuntu si celle-ci n'est pas déjà présente en local, puis exécutera l'environnement en mode interactif.

Pour information, le dockerfile ne contient que 3 lignes :

```
FROM scratch #l'image minimale de docker
ADD ubuntu-jammy-oci-amd64-root.tar.gz / #l'archive contenant ubuntu
CMD ["bash"] # la commande que l'on exécutera par défaut avec l'option -it
```

> Pour les TPs de ce cours, une très bonne série d'images est proposée par https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html. Dans un premier temps, `scipy-notebook` est un choix très complet.

1.2 Anaconda

```
<img src="
```

```
<ul>
    <li><b>Anaconda</b> : distribution de python</li>
    <li>gestionnaire d'environement</li>
    <li> <a src="https://www.anaconda.com/products/distribution">Distribution officielle avec c
    <li>miniconda : version minimale</li>
    <li>mamba : installation rapide de paquet </li>
</ul>
```

Installation (Windows)

Suivre les instructions en fonction de votre machine: https://docs.anaconda.com/anaconda/install/

[18]:
```
%%html
<iframe src="https://docs.anaconda.com/anaconda/install/" width="1000"␣
 ↪height="800"></iframe>
```

```
<IPython.core.display.HTML object>
```

Anaconda peut être livré avec une version graphique

Mais contient à minima un moteur pour la gestion des paquets *via* un interface en ligne de commande, soit `Ancaconda prompt` (qui émule un shell linux) soit directement dans le shell du système en l'activant ou non.

1.2.1 Installation (miniconda sous linux)

Miniconda est une version "allégée" et surtout libre de conda. On peut télécharger le script d'installation)(`Miniconda3 Linux 64-bit`) directement https://docs.conda.io/en/latest/miniconda.html ou exécuter la série de commande suivante dans le dossier de votre choix.

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ chmod 711 Miniconda3-latest-Linux-x86_64.sh
$ ./Miniconda3-latest-Linux-x86_64.sh
```

Puis on répond aux questions posées par le script. Une fois celui-ci complété, `conda` est accessible et l'environement de base est actif par défaut dans les nouveaux terminaux.

> miniconda est aussi disponible sous windows.

1.2.2 Création d'environement via la CLI (Command Line Interface)

But: - Créer un environnement autonome pour ce TP : `TP1` - Installer quelques paquets pour un usage scientifique/AI

> Selon la méthode d'installation, les paquets installés dans l'environnement de base diffèrent. On va donc créer un environement distinct qui nous permettra d'avoir la même configuration sur toute les machines.
>
> On verra dans un TP ultérieur que ces environements peuvent etre exportés (plus ou moins) facilement pour des questions de portabilité.
>
> Pour toutes les questions suivantes, vous trouverez les informations nécessaires dans les documentations conda/anaconda en ligne.

```
<ol>
    <li> Lancer un terminal/anaconda prompt</li>
    <li> Créer un environement conda appelé <code>TP1</code> </li>
    <li> L'activer avec la commande <code>conda activate TP1</code>. L'entête du terminal comme
    <li> Quel version de python est appelée par les commandes <code>python</code> et <code>pytl
    <li> Installer la version 3.10.6 de python précisément.
    <li> Tester à nouveau la version de python</li>
    <li> Installer et tester le paquet <code>ipython</code> grace à conda. Il propose entre au
    <li> Installer les paquets suivants : <code>scipy, matplotlib, sympy</code>. <i>Vérifier à
    <br> <i>L'installation peut prendre du temps puisqu'elle nécessite de vérifier qu'il n'y a
```

```
    </li>
    <li> Installer le paquet <code>pygame</code> et tester la commande suivante :  <code>pytho
    <li> Dans le terminal interractif de votre choix, calculez les 10 premiers termes de la su
    <li> Idem pour les 100 premiers termes.</li>
</ol>
```

1.3 IDE : l'exemple VS Code

L'éditeur intégré de développement (IDE) s'est imposé comme l'outil indispensable pour l'élaboration de projet de code moderne. C'est particulièrement vrai en python, avec les possibilités d'introspection, exécution et débuggage de code.

1.3.1 Installation et documentation

Télécharger le logiciel et l'installer. Profitez en pour survoler la doc et les tutoriels. - https://code.visualstudio.com/

Exercice:

Lancer VS Code

Créer un fichier hello-world.py qui affichera "hello, world" suivit de la version de python en cours d'utilisation.

Exécuter le script dans le terminal (usuel) et celui inclu dans l'IDE.

1.3.2 Installation des extensions python

L'IDE natif propose la coloration syntaxique mais guère plus à ce stade. Il faut donc ajouter des extensions en fonction de vos projets. Pour ce cours, installer les extensions `Python` et `Pylance` qui offrent grand nombre de *features*.

1.3.3 Connection à une machine hote

Une des grandes forces de VS code est la facilité d'utilisation sur des hotes distants. Moyennant une connection SSH (locale pour la WSL) à une machine équipée de `VS code` (au moins dans sa partie serveur), on peut interragir avec le code de la même manière que s'il était sur sa propre machine.

Pour faire cela on utilsera les paquets `remote-WSL` et `remote-SSH` comme décrit ici.

Pour nos TP cela permet d'accéder à l'environement de développement que l'on a conçu dans l'exercice pércédent. VS code sera capable de repérer les paquets installés ou non et d'afficher les docstrings.

1.3.4 Git

En plus de votre IDE vous aurez **ABSOLUMENT** besoin d'un logiciel de versionnage, dont `git` est le leader incontesté. A installer et activer dans votre environnement!

Quelques bases : https://training.github.com/downloads/github-git-cheat-sheet.pdf

Et la doc officielle (trés/trop) détaillée

Exercice: création d'un dépot git

Déplacer le fichier hello-world.py dans un dossier TP1

Créer un nouveau dépot git

Y ajouter le fichier et `commit` les changements.

Il s'agit la d'un dépot local. On pourrait aussi l'uploader sur un plateforme type github.

Exercice: clonage et analyse de sources

Trouver et cloner les sources de la bibliothèque numpy dans un dossier temporaire?

Quel est le tag du commit 7d4349e332fcba2bc3f266267421531b3ec5d3e6?

Ouvrir les sources de la bibliothèque numpy et trouver de quelle façon est calculé la norme d'un array. Trouver la ligne où cette fonction est définie.

Quelles sont les arguments de cette fonction?

Quel cas (ligne et valeur de l'argument) peuvent être appelés si ord est laissé vide (`None`)?

A la fin de cette section, vous devez disposer de VS code, avec un systeme de controle de version.

1.4 Les jupyter notebooks

Ce document est en fait un notebook, nous allons l'ouvrir pour poursuivre ce TP.

Avant cela, il faudra installer la suite d'applications avec la commande

```
$ conda install -c conda-forge jupyter jupyterlab nb_conda_kernels jupyter_contrib_nbextensions
```

Que l'on peut décomposer ainsi : `conda` appelle le gestionnaire d'environnement, en mode `install`. On choisit un "channel" particulier ici avec l'option `-c conda-forge` qui contient plus de paquets que le dépôt standard puisqu'il est maintenu par la communauté. Enfin on liste les paquets que l'on souhaite installer.

> La documentation principale : https://docs.jupyter.org/en/latest/
>
> Pour plus de détail sur les extensions : https://towardsdatascience.com/jupyter-notebook-extensions-517fa69d2231
>
> De même, pour le setup d'un environnement IA : https://towardsdatascience.com/how-to-set-up-anaconda-and-jupyter-notebook-the-right-way-de3b7623ea4a

Deux interfaces s'offrent à vous, 1. les notebooks "purs" avec les extensions pour une approche assez textuelle mais fournie avec les nombreux outils des `nbextensions`. 1. `jupyterlab`, qui réplique un environnement de type IDE, pour des notebooks. On a une barre de navigation, un explorateur de variable, etc. C'est entrain de devenir le nouveau standard.

D'un point de vue technique. L'utilisation de jupyter se passe sur deux plans, un coté serveur, que l'on lance dans le terminal (ou sur un serveur) et de l'autre une visualisation à partir du navigateur de son choix. L'adresse par défaut est https://localhost:8888 avec 8888 le port par défaut (on peut le modifier) ou remplacer localhost par une adresse sur le web.

Attention: fermer le navigateur ne suffit pas à interrompre le process. Il faut le faire dans le terminal avec un double CTRL+C. On peut par contre terminer individuellement les notebooks pour libérer la mémoire occupée par celui-ci.

1.4.1 Un peu de pratique

Il existe 3 types de cellules dans les notebook. Des cellules de texte comme celle-ci. Une fois validée, le moteur interprète le texte avec le balisage Markdown (et HTML) pour sa mise en forme. Par exemple: **Du texte gras markdown** suivit d'une formule mathématique en LaTeX. On peut intégrer des images ou des liens très simplement : https://www.markdownguide.org/getting-started/. On peut aussi stocker du texte brut, pour par exemple commenter une cellule de code entière

[19]:
```
#Enfin il y a les blocks de code interprétés. Par défaut en python avec le␣
↪dernier output qui est affiché. Comme dans un interpréteur python standard.
import sys
sys.version
```

[19]:
```
'3.11.5 | packaged by conda-forge | (main, Aug 27 2023, 03:34:09) [GCC 12.3.0]'
```

[ ]:
```
#On peut aussi créer des fonctions ou tout type d'objet python
def mafonction(arg1):
    print(arg1)
    return None
```

Dans ce cas, attention à la portée des variables et à l'ordre d'exécution. En effet, il est tentant de réexécuter des cellules, parfois dans le désordre. Il faudra donc s'efforcer, une fois la solution trouvée, d'avoir un notebook qui s'exécute d'une traite sans erreur. Pour vérifier cela, on utiliser le bouton noyau *"redémarrer et tout exécuter"* .

L'interpréteur ipython et par conséquent jupyter est livré avec un nombre important de **magics** qui vont soit augmenter un cellule python soit interragir directement avec le system.

[ ]:
```
%run hello-world.py
#permet d'exécuter dans le notebook (et surtout l'interpréteur attaché) un␣
↪fichier externe
```

[ ]:
```
# Evalue le temps d'exécution d'une ligne
%timeit for i in range(1000): i*2.5
```

[20]:
```
%%timeit
#pour la cellule entière
for i in range(1000):
    i*2.5
```

```
24.3 µs ± 351 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

Quelques commandes du terminal sont disponibles avec les magics comme `cd`, `pwd`, `ls`.

Se déplacer dans le dossier TP1 et afficher son contenu

Les deux magics suivantes sont à appliquer dans la majorité des notebooks. Elle permettent le rechargement automatique des modules que vous utilisez ce qui est particulièrement utile lors de l'utilisation d'un projet en cours de développement.

[ ]:
```
%reload_ext autoreload
%autoreload 2
```

1.5 Un peu de python pour finir

On va s'appuyer sur l'aspect dynamique du notebook pour la fin de ce TP. En particulier, on va travailler avec du texte et des fichiers puisque cela nécessite des fonctions basiques de python.

Exercice: Ci-dessous un certain nombre de problèmes sont décrit, ouvrez un notebook jupyter et répondez directement dedans.

Remarque: n'oubliez pas d'accéder à la documentation et autres pense bête pour répondre aux questions.

1.5.1 Manipulation d'un texte assez long.

On va travailler avec le poème archi classique de Jean de La Fontaine.

```
LA CIGALE ET LA FOURMI
La Cigale, ayant chanté
Tout L'Été,
Se trouva fort dépourvue
Quand la Bise fut venue.
Pas un seul petit morceau
De mouche ou de vermisseau.
Elle alla crier famine
Chez la Fourmi sa voisine,
La priant de lui prêter
Quelque grain pour subsister
Jusqu'à la saison nouvelle.
« Je vous paierai, lui dit-elle,
Avant l'Août, foi d'animal,
Intérêt et principal. »
La Fourmi n'est pas prêteuse :
C'est là son moindre défaut.
« Que faisiez-vous au temps chaud ?
Dit-elle à cette emprunteuse.
- Nuit et jour à tout venant
Je chantais, ne vous déplaise.
- Vous chantiez ? j'en suis fort aise :
Eh bien ! dansez maintenant. »
```

Derrière se texte anodin se cache un nombre important de caractères spéciaux qu'il faut pouvoir manipuler si l'on veut extraire des informations de celui-ci.

Copier le poème dans un fichier `fable.txt` situé dans le même dossier que le notebook.

Intérraction avec les fichiers

- **Ouvrir le fichier `fable.txt`** et stocker son contenu dans une variable `fable`.
- **Afficher le contenu de fable. S'agit-il d'un texte ascii?**

Les strings python `str` sont encodés par défaut avec la norme utf-8 ce qui permet de traites les caractères du français, et de la plupart des autres langues. De plus un certain nombre d'"escape character" sont interprétés à l'affichage. C'est le cas du retour à la ligne. Ce dernier est codé par `\n`. Ainsi, si l'on affiche le contenu de fable explicitement dans le notebook, il n'y plus de retour à la ligne mais une seule ligne contenant tout le texte.

[ ]:
```
fable
```

- **Ecrire un script qui compte les dialogues prononcés par les deux personnages.** On tiendra compte des "-" lorsqu'ils indiquent un changement d'orateur.
- **Supprimer tous les "e" du texte**
- **Enregistrer le nouveau fichier sous le nom `fabl.txt`**

1.5.2 Un parseur simple

A l'image du référencement Google, on souhaite extraire les titres d'une page web : https://fr.wikipedia.org/wiki/Python_(langage).

Comme la plupart des sites, celui-ci est basé sur un code html que l'on va importer directement avec python et la bibliothèque native `urllib` (cf doc.) et sa fonction `urlopen`. - **Lire le code html et en afficher quelques lignes (`readlines`).** L'output doit ressembler à ceci

```
b'<!DOCTYPE html>\n'
b'<html class="client-nojs" lang="fr" dir="ltr">\n'
b'<head>\n'
b'<meta charset="UTF-8"/>\n'
b'<title>Python (langage) \xe2\x80\x94 Wikip\xc3\xa9dia</title>\n'
...
```

Combien y a-t-il de lignes de code? et de caractères? > *Remarque: les lignes extraites par `readlines` sont du bytecode et non des strings. Pour notre analyse, on doit convertir et puis chercher les balises de texte comme précédemment.*

Le code est vide (une fois lu) on va donc stocker celui ci dans un `str`.

Extraire les titres de niveau 1 à 3 (balise <h*> et </h*>) dans une structure de donnée permettant de les trier par importance. *On peut utiliser des **regex** pour faciliter l'analyse.*

Afficher ces titres de manière hiérarchique (en non chronologique).

On constate qu'il n'y a pas simplement les balises mais aussi tout un tas de détails que l'on ne souhaite pas conserver pour cet exercice. On peut reprendre l'idée de l'exercice précédent pour résoudre celui-ci.

*En fait, comme pour pratiquement tout, il existe une librairie python qui sait faire le parsing pour nous : `beautifulsoup`

Extraire les mêmes informations à l'aide de `beautifulsoup`

Pour l'installer : `mamba  install -c anaconda beautifulsoup4 -y`

1.5.3 Bonus

Le jeu "plus petit/plus grand" est un des classiques dans l'apprentissage de la programmation. L'ordinateur génère un nombre aléatoire et le joueur essaye de le retrouver. À chaque étape, l'ordinateur indique si le nombre proposé est plus petit ou plus grand que le nombre à trouver. Ici, l'exercice proposé est de programmer la position inverse : le joueur choisit un nombre et l'ordinateur essaye de le retrouver selon la même approche.

La vraie difficulté de l'exercice sera que le programme doit détecter la tricherie (celle du joueur, car le programme, lui, ne triche jamais). Ce cas se produit quand l'ordinateur propose (par exemple) 50 et que le joueur répond "+". Puis plus tard, il propose 51 et le joueur répond "-". Et bien entendu, une situation symétrique si l'ordinateur propose (toujours) 50 et que le joueur répond "-". Puis plus tard, il propose 49 et le joueur répond "+".

Exemple de résultat honnête

```
Mémorisez un nombre entre 1 et 100, je vais essayer de le retrouver
Appuyez sur <enter> quand vous serez prêt. Et ne trichez pas ensuite...
Je propose 57... +, - ou G ?-
Je propose 37... +, - ou G ?-
Je propose 19... +, - ou G ?+
Je propose 25... +, - ou G ?G
J'ai trouvé 25 en 4 coups !!!
```

Exemple de tricherie

```
Mémorisez un nombre entre 1 et 100, je vais essayer de le retrouver
Appuyez sur <enter> quand vous serez prêt. Et ne trichez pas ensuite...
Je propose 44... +, - ou G ?-
Je propose 29... +, - ou G ?-
Je propose 17... +, - ou G ?+
Je propose 25... +, - ou G ?+
Je propose 27... +, - ou G ?-
Je propose 26... +, - ou G ?-
Tricheur !!! A la réponse 4 il avait été proposé 25 et répondu "+" - En proposant 26 la réponse
J'ai gagné par forfait en 6 coups !!!
```

[ ]: