TP2

Compte-rendu de *votre nom & votre groupe*

Notebook, Markdown et IPython

ENSICAEN 1A MC 2022

Eric Ziad-Forest

Enseignants des groupes de TD/TP

- Groupe 1: Philippe Lefebvre [lefebvre@ensicaen.fr]
- Groupe 2: Tanguy Gernot [tanguy.gernot@ensicaen.fr]
- Groupe 3: Christine Porquet [chris@ensicaen.fr]
- Groupe 4: Eric Ziad-Forest [ziad@ensicaen.fr]

Auteurs :

- JeremyLaforet ((mailto: jeremy.laforet@utc.fr)
- Eric Ziad-Forest (mailto: ziad@ensicaen.fr))

Contenu sous licence [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)](https://creativecommons.org/licenses/by-sa/4.0)

Objectif

- Pratique du notebook de *Jupyter* ainsi que des rudiments de *IPython*
- Rudiments également de *Markdown*
- Utilisation de *IPython*
- Révision des commandes Shell (TP1)
- Savoir utiliser les principales méthodes et primitives sur les listes

Avant de commencer

Ouvrez un terminal (`alt+ctrl+T`) afin de créer un dossier `TP2` (commande linux `mkdir`) dans votre espace de travail Initiation Python vu lors du TP1. Puis aller sur Moodle https://foad.ensicaen.fr/ (https://foad.ensicaen.fr/) et dans InitProgMC cliquer pour télécharger le dossier TP2.

Aller dans le dossier `Download` sur votre espace de travail et cliquer sur le fichier .zip et enfin faite une extraction dans TP2 (si besoin placer tous les fichiers dans le dossier TP2...

Placer vous alors dans TP2 à l'aide du terminal (commande `cd`).

Puis toujours dans le terminal lancer la commande:

```
jupyter notebook
```

Choisissez TP2_notebook_python_2021.ipynb

Vous voilà près pour la suite

Rendu

> **Vous allez au cours de ce TP, compléter ce document qui servira de compte-rendu en fin de séance et sera déposé sur la plateforme Moodle.**
> N'oubliez pas d'inscrire votre nom et prénom à côté de compte-rendu, juste en dessous de TP2.

Introduction

Les 4 premières parties sont essentiellement à lire pour comprendre et apprendre les différents outils utiles pour élaborer vos comptes rendus. C'est assez long, mais sans grande difficulté.

`Jupyter` est une application Web qui regroupe intimement deux fonctionnalités très différentes :

- Un outil qui permet de créer des documents multimédia intégrant du texte, des formules mathématiques, des graphiques, des images, voire des animations et des vidéos.
- Une interface qui permet d'exécuter du code informatique. Pour cela `Jupyter` s'appuie sur des programmes indépendants capablent d'interpréter le langage dans lequel est écrit ce code.
  Dans la terminologie de `Jupyter` ces interpréteurs sont appelés des *noyaux* (*kernel* en anglais).

Les documents `Jupyter` sont appelés `notebook` .
Un fichier `notebook IPython` est reconnaissable par son extension `.ipynb` .

Le document que vous lisez actuellement est un notebook, vous pouvez bien évidemment vous en inspirer.

Enfin n'oubliez pas d'évaluer toute les cellules `Entrée[ ]` (`In[ ]` en anglais) durant votre lecture.

PARTIE I

Manipulation du Notebook de *Jupyter*

*À l'usage, il se révèle plus efficace d'utiliser les **raccourcis clavier** que de naviguer dans les menus avec la souris.*
Prenez-en l'habitude.

> Dans un premier temps cliquez dans le menu sur l'onglet `Aide` puis `Visite de l'interface utilisateur` (en anglais `Help` puis `User Interface Tour`)

Cellules

Le notebook est constitué d'une succession de cellules.
Les cellules peuvent être dans 2 modes

- ***mode commande***
- ***mode édition***

Un liseré de couleur repère la cellule actuellement sélectionnée.
Le liseré est:

- **bleu** si la cellule est en mode commande
- **vert** si elle est en mode édition. (c'est-à-dire que vous pouvez écrire dedans ou modifier ce qui s'y trouve)

Le mode édition

Pour entrer dans le ***mode édition*** de la cellule sélectionnée, il suffit de presser la touche **`Entrer`** ou de cliquer à l'intérieur de la cellule. Quand une cellule est en édition vous pouvez saisir du texte comme dans un éditeur classique.

Lorsque le curseur est en début de ligne ou lorsque vous avez sélectionné du texte, l'appui sur la touche **`Tab`** (respectivement **`Shift+Tab`**) indente (respectivement désindente) les lignes correspondantes.

Voici d'autres raccourcis clavier :

- **`Ctrl + A :`** Sélectionne tout le texte de la cellule.
- **`Ctrl + Z :`** Annule les dernières saisies de texte.

- **`Ctrl + Entrer] :`** Exécute la cellule.
- **`Maj + Entrer :`** Exécute la cellule et sélectionne la cellule suivante. (L'appui répété de cette touche permet ainsi d'exécuter pas à pas toutes les cellules du notebook).
- **`Alt + Entrer :`** Exécute la cellule et insère une nouvelle cellule juste en dessous.

- **`ESC :`** Passe dans le ***mode commande***.

Le mode commande

Quand vous êtes dans le **mode commande** (ligne bleu vertical à gauche de la cellule), vous pouvez ajouter ou supprimer des cellules mais vous ne pouvez pas saisir de texte dans une cellule. Notez bien que dans ce mode, certaines touches du clavier sont associées à des actions sur le notebook. Tenter de saisir du texte dans ce mode peut donc produire des effets inattendus.

Voici les raccourcis principaux disponibles en mode commande :

- **`A`** et **`B`** : Insèrent une nouvelle cellule, respectivement au-dessus ou au-dessous de la cellule sélectionnée. (`A` pour `Above` et `B` pour `Below`
- **`M`** : Transforme la cellule en une cellule de type `Markdown` (Cf. ci-dessous).
- **`Y`** : Transforme la cellule en une cellule de type `Code` .
- **`C`** et **`V`** : Copie et colle des cellules.
- **`DD`** : Supprime la cellule sélectionnée.
- **`Z`** : Annule la dernière suppression de cellule.
- **`II`** : Interrompt l'exécution du code.
- **`00`** : Redémarre l'interpréteur. Il se retrouve alors dans son état initial.

> **Remarque**.
> Et pour avoir à chaque instant tous ces raccourcis
> **`Esc + H`** : Affiche la liste de tous les raccourcis clavier.

On peut utiliser aussi la barre de menu!

Par exemple pour diviser la cellule à partir du curseur (en anglais) Edit - Split - Cell (traduiser pour les menus en français)

Chercher pour fusionner des cellules... (en anglais merge)

> **Exercice** 1
>
> 1. Diviser la cellule en 3 cellules distinctes: Cellule 1, Cellule 2 et Cellule 3
> 2. Fusionner les 3 cellules
> 3. Supprimer les 2 cellules suivantes

> Réponse à la question 1 sur la cellule ci-desous

In [ ]:
```
Cellule 1

Cellule 2

Cellule 3
```

> Réponse à la question 2 sur les cellules ci-desous

In [ ]:
```
Cellule 1
```

In [ ]:
```
Cellule 2
```

In [ ]:
```
Cellule 3
```

> Réponse à la question 3 sur la cellule ci-desous

In [ ]:
```
# cette cellule est à effacer !
```

PARTIE II

Le langage *Markdown*

L'appui sur les touches `esc` et ensuite `M` transforme la cellule sélectionnée en type `Markdown` . Vous pouvez alors rédiger du texte enrichi (titres, sous-titres, gras, italique, alinéas, tableaux, liens hypertextes, etc). Le texte de la cellule doit être rédigé en langage *Markdown* qui est un *langage de balisage léger*.
La *syntaxe markdown* est facile à apprendre.
Le plus simple est d'ailleurs de regarder des exemples de documents tels que celui que vous lisez actuellement.

Il est également possible d'insérer des morceaux de texte en langage $\LaTeX$ pour composer des expressions mathématiques.

$$g(x) = f\left(8\int_{-\infty}^{+\infty}(x-1)\left[\frac{x}{h^2+1}\right]dh\right)$$

Pour en savoir plus: https://fr.wikipedia.org/wiki/Aide:Formules_TeX (https://fr.wikipedia.org/wiki/Aide:Formules_TeX)

Cliquer 2 fois dans cette cellules pour voir l'editeur `Markdown` .

Édition et exécution d'une cellule de code

Un notebook jupyter consiste en **cellules** individuelles pouvant contenir du **code** Python, du **texte** (avec formatage Markdown, HTML et LaTeX) ou des **graphiques**. Commençons par les cellules de type Code.

> **Exercice** 2
> Placez le curseur dans la cellule ci-dessous suivante puis `Entrer` *Enter* (Retour Chariot).
> Que se passe-t-il ?

In [ ]:
```
print(2020+1)
```

> Rien d'autre que l'insertion d'un saut de ligne...
> C'est normal, car dans une cellule de type `Code` (c'est-à-dire préfixée par **Entrée[ ]:**) on est par défaut en mode **édition de code** Python !

Pour **exécuter le code** `Python` d'une cellule, il faut en effet frapper les touches `Maj + Entrer` (`Shift + Enter` en anglais) (ou encore utiliser la souris ;-(et cliquer sur le bouton  `Exécuter` , ou encore utiliser le menu `Cellule>Exécuter les cellules...`).
Essayez !

In [ ]:
```
1j*(1+1j)
```

Tout ce que l'on peut insérer dans un *notebook*

Graphiques dans un notebook

Si vous exécutez la **fonction magique** `%pylab inline` (nous verrons prochainement les **fonctions magiques** de `IPython`) , les packages **Numpy** et **Matplotlib** sont importés et il sera possible de dessiner des graphiques de façon `inline` (intégrés au notebook).

In [ ]:
```
%pylab inline

#L'exemple de code suivant sera alors exécutable._

#Fait appel à numpy (linspace et pi)
x = linspace(0, 3*pi, 500)

#Fait appel à matplotlib (plot et title)
plot(x, sin(x))
title('Graphique sin(x)')
```

Documents Multimedia dans un notebook

Evaluer toutes les entrées suivantes!

Intégrer une image :

In [ ]:
```
from IPython.display import Image
Image("http://api.si.lycee.ecmorlaix.fr/APprentissagePythonJupyter/img/jupyter.png",width=100, height=200)
```

Ou simplement en lien `Markdown` :

```
![Logo de jupyter](https://jupyter.org/assets/main-logo.svg)
```

Logo de jupyter

Intégrer une vidéo youtube :

In [ ]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('qxMcGDnT8uc') # qxM...8uc est le code youtube dans l'url
```

Intégrer un fichier pdf :

In [ ]:
```
from IPython.display import IFrame
IFrame("https://phychim.ac-versailles.fr/IMG/pdf/tuto_python_matplotlib.pdf", width=900, height=700)
```

Intégrer une page web :

In [ ]:
```
from IPython.display import HTML
HTML('<IFrame src="https://fr.wikipedia.org/wiki/Jupyter" width=700 height=350>')
```

Sauvegarder un notebook

Lorsque vous avez terminé de travailler dans un notebook, **n'oubliez pas de le sauvegarder** avec le bouton  (ou le menu `Fichier>Créer une nouvelle sauvegarde`). Sachez que *Jupyter* effectue par défaut des sauvegardes à intervalle régulier, et que `Fichier>Restaurer la sauvegarde` permet de revenir à un état enregistré.
De plus vous pouvez, exporter votre travail dans toutes sortes de formats (HTML, PDF, python, LaTeX ...)

Imprimer

Si vous désirez imprimer un notebook, n'utilisez pas la commande d'impression de votre navigateur web depuis la fenêtre de notebook, le résultat sera très mauvais.
La bonne méthode consiste à faire au préalable depuis le menu Fichier>Imprimer l'aperçu. Mais il est peut-être préférable de télécharger une version numérique du notebook sur l'ordinateur local dans le format qui vous convient comme le HTML pour obtenir une page web statique avec le menu File>Download as...

Bloqué?

Si jupyter Notebook semble bloqué, c'est surement qu'il tourne indéfiniment en raison par exemple d'une boucle mal contrôlée (consommant 100% de CPU d'1 core de votre machine).
Vous devriez voir le message `Kernel busy` au haut de l'écran.
Pour l'interrompre, cliquez sur ii ou le bouton (ou menu Noyau>Interrompre).
Si ça ne fonctionne vraiment pas, il faudra se résoudre à tuer le serveur web avec `Ctrl + C`

Des questions

Si vous ne trouvez pas les réponses à un problème posé chercher-la sur Le site. *(c'est une référence dans le monde informatique...)*

https://stackoverflow.com/questions/tagged/jupyter-notebook (https://stackoverflow.com/questions/tagged/jupyter-notebook)

PARTIE III

Programme IPython

L'aide en ligne IPython

Lorsque vous sélectionnez *Python 3* comme *kernel*, le programme lancé par *Jupyter* n'est pas directement l'interpréteur *Python* mais le programme *IPython*.

Le texte soumis par l'utilisateur est d'abord analysé par *IPython* qui utilise l'interpréteur *Python* chaque fois qu'il est nécessaire d'évaluer du code *Python*.

> **Attention** Ce principe permet d'ajouter des fonctionnalités très pratiques, mais qui ne font pas partie du langage *Python*.

IPython offre par ce biais plusieurs mécanismes d'assistance dont il faut user sans modération.

L'auto-complétion

Lorsque vous commencez à saisir un nom connu de l'interpréteur, l'appui sur la touche **`Tab`** complète le nom automatiquement (Comme sous `Linux`. Si plusieurs noms sont possibles un menu contextuel vous propose de choisir. Ceci économise de la frappe tout en évitant les erreurs d'orthographe dans les noms des variables.

La documentation d'un objet

Pour lire la documentation en ligne concernant un objet *python* (module, fonction, classe, etc), il suffit d'ajouter un **`?`** juste après le nom et d'appuyer sur la touche **`Entrer`**. Un *pager* s'ouvre alors avec la dite documentation. En doublant le **`?`** le *pager* donne une documentation plus complète si disponible.

In [ ]:
```
# Exemple
complex?
```

Un autre moyen est d'utiliser `help()`

In [ ]:
```
help(complex)
```

L'état de l'interpréteur

Lorsqu'on ouvre un `terminal`, un programme interpréteur de commandes système est lancé et attend les commandes pour les exécuter.
De la même façon, en arrière-plan d'un `notebook`, un interpréteur de code a été lancé et attend lui aussi qu'on lui donne du code. Dans le coin supérieur droit vous pouvez voir qu'il s'agit ici de l'interpréteur *Python 3*.

Cellules codes

Un notebook est constitué de cellules successives.
Les cellules pour écrire du code sont repérables par le prompt `Entrée[ ]:`.
Essayons :

In [ ]:
```
2 / 3
```

Le texte `2 / 3` a été transmis à l'interpréteur *Python3*. Il s'agit d'une expression *Python3* valide. L'interpréteur a donc pu l'évaluer. La valeur de l'expression (dont le type est ici un nombre flottant) est alors récupérée et représentée sous la forme du texte affiché à droite du prompt de sortie `Out[ ]`

N'oubliez pas que vous êtes dans un notebook. Vous pouvez donc modifier l'expression ci-dessus et la ré-exécuter en appuyant sur `Maj-Entrer` (`Ctrl-Entrer`, `Alt-Entrer`). Faites-le ! Le numéro entre crochet est un compteur. Il est incrémenté à chaque sollicitation de l'interpréteur.

In [ ]:
```
hauteur = 2 + 2
```

L'exécution de l'instruction ci-dessus n'a produit aucun résultat en sortie. Cette instruction a cependant eu un effet. Elle a modifié l'état de l'interpréteur. En interne, l'interpréteur a associé la valeur de l'expression `2 + 2` (i.e. le type entier 4) au nom *hauteur*.

On peut alors exécuter :

In [ ]:
```
5 * hauteur
```

In [ ]:
```
print()
```

In [ ]:
```
print(7 * hauteur)
```

> **Remarque**
> Dans les 2 dernières instructions la 1ère donne une *sortie* `Out[]` , tandis que la 2d imprime le *résultat* (il n'y a pas de *sortie* pour cette instruction)

L'état de l'interpréteur est constitué de toutes les associations valeurs/noms qu'il mémorise en interne.
Les séquences de code qui contiennent des affectations (opérateur `=`) modifient l'état de l'interpréteur. Un même code peut produire des effets différents selon l'état dans lequel se trouve l'interpréteur.

Observez les deux cellules de code ci-dessous. Selon que l'on exécute d'abord la premiere cellule ou d'abord la seconde cellule, l'état de l'interpréteur ne sera pas le même à l'issue. Ce n'est pas la même chose d'ajouter 1 à un nombre puis de doubler le résultat ou bien de doubler un nombre et d'ajouter le 1 ensuite.
Exemple:

In [ ]:
```
hauteur = 2 + 2
```

In [ ]:
```
hauteur = hauteur + 1
hauteur = 2 * hauteur
print(hauteur)
```

In [ ]:
```
hauteur = 2 + 2
```

In [ ]:
```
hauteur = 2 * hauteur
hauteur = hauteur + 1
print(hauteur)
```

Lorsque vous ouvrez un `notebook` , vous le visualisez tel qu'il a été sauvegardé. Vous voyez en particulier les résultats des séquences de code qui ont été exécutées. Cependant, à l'ouverture du `notebook` , un nouvel interpréteur tout neuf est lancé. Vous devez donc exécuter à nouveau les cellules. Mais dans quel ordre ? La réponse naturelle est : *dans l'ordre où apparaissent les cellules*. Mais cela suppose que la personne qui a réalisé le `notebook` a fait correctement les choses.

Les exceptions

Dans certaines situations, l'interpréteur peut s'interrompre en *levant une exception*. Les exceptions n'ont rien d'exceptionnelles.
On en rencontre souvent, en particulier lorsque l'on commet des erreurs de programmation.

In [ ]:
```
T = [18, 5, 3]
T[3]
```

Pour comprendre ce qui a produit l'exception il faut :

- identifier le nom de l'exception, ici `IndexError` ,
- lire le message associé, ici ' `list index of range` ',
- identifier l'instruction qui l'a provoqué, ici l'évaluation de `T[3]` à la ligne 2.

Il se peut aussi que le code que l'on exécute ne se termine pas :

```
k = 1
while k > 0:
    k = k + 1
```

```
---------------------------------------------------------------------------
_
KeyboardInterrupt                         Traceback (most recent call las
t)
<ipython-input-6-a60500a6252c> in <module>
      1 k = 1
----> 2 while k > 0:
      3     k = k + 1

KeyboardInterrupt:
```

Lorqu'une cellule de code s'exécute le prompt comporte une étoile `Entrée[ *]` .
Pour interrompre l'interpréteur, il suffit d'appuyer deux fois sur la touche `I` (i) (voir `Mode Commande` ci-dessus). Si cela s'avère nécessaire, il est également possible de redémarrer l'interpréteur. Pour cela il faut appuyer deux fois sur la touche `0` . L'interpréteur se retrouve alors dans son état initial.

> **Exercice** 3
> Lancer les instructions *Python* dans la cellule ci-dessous et intérrompez l'évaluation comme expliqué ci-dessus.

In [ ]:
```
k = 1
while k > 0:
    k = k + 1
```

> Pourquoi, selon vous cela n'a pas fonctionné ?

L'entrée et la sortie standard

Les programmes, quels qu'ils soient, lisent des données en provenance de sources variées (fichiers, réseaux, ports usb, etc). Ils envoient eux-mêmes des données vers diverses destinations. Dans tous les cas, cela se traduit par la lecture ou l'écriture de flux d'octets.

Tout programme peut lire un flux d'octets particulier que l'on appelle l'*entrée standard*. Lorsqu'il s'agit d'un programme lancé par un utilisateur, l'entrée standard est généralement le flux d'octets généré par le clavier. Tout programme peut aussi écrire sur un flux particulier que l'on appelle la *sortie standard*.

Dans un programme *Python* la fonction `print` permet d'écrire sur la sortie standard.

In [ ]:
```
# 1er boucle for
print("Accueil s'écrit :")
for x in "accueil":
    print(x)
```

> **Remarquez** de nouveau que l'exécution du code ci-dessus n'a retourné aucune valeur (il a juste imprimé à l'écran le résultat). Il n'y a pas de prompt `Out[ ]` . Remarquez aussi que chaque fonction `print` a également envoyé un caractère *saut de ligne* sur la sortie standard.

Dans un programme *Python* la lecture de l'entrée standard se fait au moyen la fonction `input` :

In [ ]:
```
texte = input("Veuillez entrer un texte : ")
texte.upper() # méthode upper vu en cours
```

L'interpréteur a écrit le texte *Veuillez entrer un texte :* sur la sortie standard. Les caractères générés par l'utilisateur sur l'entrée standard ont également été réémis en écho sur la sortie standard.

L'expression `texte.upper()` a été évaluée et sa valeur retournée comme résultat pour `Out[ ]` .

De façon générale, si la dernière ligne d'une séquence de code est une expression non affectée à une variable, alors la valeur de cette expression est retournée comme résultat pour `Out[ ]` .

PARTIE IV

Les commandes systèmes via IPython

Révision du TP1 mais au lieu du `terminal` utilisons `IPython` .

Lorsque le texte de la cellule débute par un `!` alors `IPython` en déduit que le reste du texte n'est pas du code *python* mais une commande système qui doit être exécutée par le *Shell*.
Autrement dit `IPython` peut se substituer au `terminal` .

Vous allez taper les commandes `linux` vous permettant de créer dans votre espace de travail le sous-dossier `/CommandeLinux` , comme dans le TP1 (avec les mêmes commandes!). Mais cette fois-ci, ce n'est pas dans le `terminal` , mais uniquement dans `IPython notebook` .

> **Attention : Après chaque instruction de code ecrit en Python, vous devrez taper `Maj + Entrée` pour évaluer votre code.**

Pour les commandes `shell` , si cela ne fonctionnait pas (directement), précédez alors votre commande de la touche (point d'exclamation) : `!`

In [ ]:
```
# essayez pwd par exemple
```

Vous allez constater que les commandes du `shell` fonctionnent parfaitement dans cet environnement `ipython` .
Ainsi avec les commandes : `pwd` , `ls` , `cd` , `mkdir` ... , déplacez-vous dans le répertoire de travail créé lors du TP1, puis une fois dans celui-ci, créez un dossier TP2.

In [ ]:
```
# path working directory
pwd
```

In [ ]:
```
# Liste le contenu du dossier courant
!ls -l
```

In [ ]:
```
cd Documents/
```

In [ ]:
```
# Poursuivez dans les cellules suivantes
```

In [ ]:
```

```

A vous de continuer.
Pour insérer une cellule de code en dessous taper `Esc + B` **B**elow) (au dessus `Esc + A` **A**bove)

Placer vous dans votre dossier de travail TP2 et exercez-vous avec les commandes suivantes :

In [ ]:
```
mkdir essai
```

In [ ]:
```
%mkdir essai2
!ls
```

In [ ]:
```
!rmdir essai2/
!ls
```

Attention si vous répétez 2 fois l'instruction précédente cela entrainera une errreur puisqu'alors le fichier après la 1ère fois sera effacer donc n'existera plus !!!

L'instruction `range`

In [ ]:
```
list(range(3,8)) # range(a,b) donne la suite des entiers de a à b-1
```

In [ ]:
```
#équivalent à:
l=[] # déclaration et initialisation d'une liste à liste vide
i=3 # déclaration et initialisation d'un entier à 3
while (i<8):
    l.append(i) # méthode ajout d'un élément à une liste
    i=i+1 # ou i+=1
l
```

In [ ]:
```
for i in range(10): # ici suite de 10 entiers de 0 à 9
    !mkdir 'essai'$i # ici $i est la variable i pour le shell
```

In [ ]:
```
ls
```

Exercice 4

1. Effacer les dossiers impairs. *(dans **range(a,b,c)** que signifie le c?)*
2. Créer les dossiers suivants :
   essai0/ essai00/ essai01/ essai02/ essai1/ essai10/ essai11/ essai12/ essai2/ essai20/ essai21/
   à la racine de TP2/ .
3. Effacer tous ces dossiers.
4. (Pour les plus avancés)
   Comment ferriez-vous pour inclure les dossiers essaiV0/ essaiV1/ essaiV2/ dans le dossier essaiV/ pour V variant de 0 à 4?

In [ ]:
```
for i in range(10):
    !rmdir 'essai'$i  # Attention cette instruction
                      # lancée 2 fois crée une erreur.
```

In [ ]:
```
ls
```

Pour les plus avancé

Importer dans un premier temps le fichier `arithmetique.py` dans votre dossier `TP2` .
Par exemple, la commande dans la cellule ci-dessous exécutée dans un terminal permet de visualiser les premiers octets du fichier `arithmetique.py` .

```
xxd arithmetique.py | head
```

Le programme `xxd` affiche les octets sur la sortie standard. Mais ici le flux de sortie (xxd arithmetique.py) est redirigé (dû au pipe `|` vu à la fin du TP1) vers l'entrée du programme `head` . Celui-ci envoie sur la sortie standard uniquement les 10 premières lignes du texte qu'il lit en entrée.

Pour exécuter cette commande système depuis le `notebook` , il suffit de placer un point d'exclamation au tout début :

In [ ]:
```
!xxd arithmetique.py | head # remarquez bien le point d'exclamatio
n.
```

Voici d'autres exemples :

In [ ]:
```
# Envoie le texte du fichier sur la sortie standard
!cat arithmetique.py
```

In [ ]:
```
# Affiche les 8 premières lignes du fichier
!head -n8 arithmetique.py
```

In [ ]:
```
# Copie les 8 premières lignes du fichier arithmetique.py
# dans le fichier arithmetique2.py qui est crée...
!head -n8 arithmetique.py > arithmetique2.py
```

In [ ]:
```
ls
```

In [ ]:
```
!cat arithmetique2.py
```

PARTIE IV

Input, print

Exercice 5

1. Ecrire vos premières instructions avec `input()` et `print()` :
   - l'une avec la commande *input("Quel est votre prénom ? ")* que vous placerez dans une variable **a**
   - l'autre qui renverra à l'écran en utilisant `print` : "Bonjour" et la variable **a**
     par exemple si **a**="Eric" le résultat devrait être :

     `"Bonjour Eric"`

   - Afficher alors les deux phrases suivantes:
     - `"Je viens de dire Bonjour"`
     - `"Bonjour, je fais des choses étranges aujourd'hui."`

     cette fois-ci en plaçant "Bonjour" dans une variable **b**
2. Comment faire dans la première phrase pour enlever la lettre majuscule.

Opérations et méthodes sur les listes

Quelques méthodes sur les listes :

```
L.count(x)    #compte le nombre d'éléments égaux à x dans L
L.reverse()   #inverse les éléments de la liste
s.join (L)    #joint les éléments de L séparées par la chaine s
L.append(x)   #ajoute x à la liste L
L.sort()      #trie la liste L
```

Voir d'autres encore : extend, insert, remove, pop...

> **Remarque**.
> Pour obtenir celle-ci dans `IPython` , une fois une liste entrée et suivie d'un point utiliser la touche `Tab` de complétion automatique. Vous verrez alors toutes les possibilités de fonctions et de méthodes disponibles avec les listes.
> Vous pouvez aussi utiliser l'instruction:
>
> ```
> print(dir(list))
> ```
>
> comme vu en cours.
> Et avoir des renseignements sur leur utilisation en tapant:
>
> ```
> help(list)
> ```

Les primitives

Il existe aussi des primitives (fonctions) telles que: `len` , `min` , `max` , `del`

Exemple :

```
len(L)        # donne la taille (le nombre d'éléments dans L).
del(L[i:j])   #supprime les éléments de i à j exclus.
```

Je vous propose 3 enchainements d'instructions pour mieux appréhender ce que l'on peut faire avec les listes.
Evaluez les et commentez-les. (Un commentaire commence par `#`)

In [ ]:
```
#Liste de caractères
id = ['Elève', 'prof', 'MM314']
'prof' in id
```

In [ ]:
```
len(id)
```

In [ ]:
```
min(id)
```

In [ ]:
```
max(id)
```

In [ ]:
```
#Liste d'entiers
nums = [5, 102, 13, 2, 99, 154, 7]
```

In [ ]:
```
min(nums)
```

In [ ]:
```
max(nums)
```

In [ ]:
```
sorted(nums)
```

In [ ]:
```
nums
```

In [ ]:
```
nums.sort()
```

In [ ]:
```
nums
```

Quelle différence faites-vous entre **sorted(nums)** et **nums.sort()**?

In [ ]:
```
#Une liste peut contenir une liste
mylist = ['a', 'b', ['c', 'd', 'e']]
```

In [ ]:
```
len(mylist)
```

In [ ]:
```
mylist[1]
```

In [ ]:
```
len(mylist[1])
```

In [ ]:
```
mylist[2]
```

In [ ]:
```
len(mylist[2])
```

Attention
Les seuls guillemets simples acceptés par python sont : ' ' mais les guillemets en virgule, accents graves ou aigüs (seul) ne fonctionnne pas...

In [ ]:
```
a='bonjour'
a
```

In [ ]:
```
b=`bonjour`
b
```

Exercice 6

Ci-dessous, la séquence d'instructions suivantes :

```
x = [1,2,3]
y=x
print(y)
y[1] = "hello"
print(y)
print(x)
```

1. Qu'obtient-on pour les listes **x** et **y**?
2. Cela vous semble-t-il cohérent?
3. Corrigez donc la séquence précédente pour que le changement d'un élément de la liste **y** ne change pas la liste **x**.
   indice : penser à faire une copie de liste.

Exercice 7

1. Définir la liste:

   ```
   liste=[17,38,10,83,25,72]
   ```

2. Triez et affichez cette liste.
3. Ajoutez l'élément 12 à la liste et affichez la liste.
4. Renversez et affichez la liste.
5. Affichez l'indice de l'élément 17.
6. Enlevez l'élément 38 et affichez la liste.
7. Affichez la sous-liste du 2 au 5ème éléments (inclus). *Attention les listes commencent à 0!*
8. Affichez la sous-liste du début au 2ème élément (inclus).
9. Affichez le dernier élément en utilisant un indiçage négatif.
10. Nombre d'éléments de la liste en utilisant une primitive du langage.
11. Quels sont le plus grand et le plus petit nombre de la liste?
12. Copier liste dans liste2. Remplacer le 2ème élément par 0 dans liste. Y a-t-il eu des changements dans liste2? Comment remédier au problème ?

In [ ]:

Attention : Bien remarquer que certaines méthodes de listes ne retournent rien (mais transforment néanmoins la liste originale, vérifiez-le).

Exercice 8 print,list,for
Soit la liste des prénoms suivants :

```
['Jean-Michel', 'Marc', 'Vanessa','Mounir' ,'Maximilien', 'Alexandre-Benoit', 'Louise', 'Amarilla']
```

Ecrivez une instruction qui affiche chacun de ces noms avec le nombre de caractères correspondant.

In [ ]:

Pour les plus avancés

Exercice 9 Calcul de PGCD
Soit la séquence d'instructions suivantes :

```
A=[246,124]
A.sort()
A= [A[1]%A[0], A[0]]
```

1. Que donne A?
2. En utilisant la question précédente et en appliquant l'algorithme d'Euclide, trouvez le PGCD de 21608 et 51319.
3. Créer une fonction qui calcul le pgcd de 2 nombres entrés en paramètre.
4. Une fonction qui calcule les coefficients de Bezout.(voir https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide_%C3%A9tendu (https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide_%C3%A9tendu))

In [ ]: