PRISE EN MAIN DE CLASSEURS JUPYTER

(Emmanuel DUBOIS 2023)

Un classeur Jupyter est créé dans votre environnement de développement par les boutons

- + pour créer un nouveau classeur. Vous serez alors interrogé sur le type de document à utiliser et le type Notebook utilisant un Kernel (moteur de calcul sous-jacent) Python 3 sera votre point de départ.
- Soit en récupérant un classeur depuis votre poste de travail par le bouton . Lors d'un double clic sur le fichier, vous le verrez s'ouvrir dans l'écran de l'application.

Un tel document est formé de 2 types de cellules :

- des cellules de *Texte* destinés à recevoir l'explication rédigée de concepts.
- des cellules de *Code* destinées à recevoir le code informatique appliquant les traitements sur des données. Lors de l'exécution des cellules de code, ces dernières présentent leur résultat immédiatement sous le code.

PREMIERE PARTIE : Cellules de Texte

Sous *JupyterHub* (autre nom de l'interface originale *Jupyter notebooks*) et *JupyterLab*, les cellules de texte en sont créées en cliquant, dans la barre d'outil au sommet du document, sur l'outil + sachant le type (dans la liste déroulante de cette même barre d'outil) fixé sur *Markdown*.

Sous *Visual Studio code (vscode)* et *Google Collab*, les cellules de texte sont créées en cliquant sur le bouton **+ Texte** ou **+ Markdown** dans la barre d'outils au sommet du document.

Elles peuvent ensuite être manipulées par les commandes apparaissant lors d'un clic droit ou secondaire.

Leur rendu est obtenu par :

- un clic sur le bouton ▶ sous *JupyterHub* ou *Google Collab* et sur le bouton ✓ situé en haut et à droite de la cellule sous *JupyterLab*.
- la séquence de touche `Maj + Entrée`

Un double-clic sur une cellule permet l'édition de son source markdown.

1 - Enrichissement de texte

italique et *italique* en faisant précéder et suivre le mot par un `*` ou `_` **gras** et **gras** en faisant précéder et suivre le mot par 2 `*` ou `_`
__gras et italique__ en faisant précéder et suivre le mot par 3 `*` ou `_`
mise en exposant en encadrant le terme par `<sup>` et `</sup>`
<u>soulignement</u> en encadrant le terme par `<ins>` et `</ins>`
~~barré~~ en encadrant le terme par des `~~` .
`police de taille fixe (code)` en encadrant par des apostrophes inverses ‘.

Pour éviter l'interprétation Markdown d'un caractère on peut le préfixer par `\`.

Texte **Markdown** de la cellule :

```
1 - Enrichissement de texte
===========================

*italique* et _italique_ en faisant précéder et suivre le mot par un  `*` ou `_`
**gras** et __gras__ en faisant précéder et suivre le mot par 2  `*` ou `_`
***gras et italique* ** en faisant précéder et suivre le mot par 3  `*` ou `_`
mise en <sup>exposant</sup> en encadrant le terme par  `<sup>` et `</sup>`
<ins>soulignement</ins> en encadrant le terme par ` <ins>` et `</ins>`
~~barré~~ en encadrant le terme par des `~~`.
`police de taille fixe (code)` en encadrant par de  s apostrophes inverses $`$.

Pour éviter l'interprétation Markdown d'un  caractère on peut le préfixer par  `` .
```

2 - Gestion des paragraphes

Toute ligne se terminant par 2 espaces ou 2 sauts de ligne impose le début d'un nouveau paragraphe pour la suite.

Une ligne de séparation entre deux paragraphes peut être réalisés en mettant 3 tirets (`---`), 3 caractères de souligbnement(`___`) ou 3 apostrophes (`***`) en début de ligne :

Un paragraphe de code peut être encadré par des suites de trois apostrophes inverses ‘‘‘ :

```
#Ceci est un paragraphe en formatage de code
```

Si le nom du langage débute le bloc de code, immédiatement après les ‘‘‘, le code bénéficiera de la mise en couleur illustrant la syntaxe du langage cible.

Un paragraphe peut être mis en citation (ou en retrait) en faisant débuter la ligne par un `>` (ou plusieurs `>>` pour des citations imbriquées) :

> Ceci est une citation ou en texte en retrait
>
> > Ceci est une citation imbriquée.

Les lignes correspondant à des éléments de listes à puces débutent par un `*` suivi d’un espace. Les listes peuvent être imbriquées en faisant précéder le caractère `*` par des doubles espaces en nombres progressifs.

- liste à puces
- élément n°2
  - élement 2.1
  - élement 2.2
    - élement 2.2.1
    - élement 2.2.2

Les lignes correspondant à des éléments de listes numérotées débutent par un numéro suivi d’un point et d’un espace.

1. liste ordonnée
2. élément n°2

Texte **Markdown** de la cellule :

```
2 - Gestion des paragraphes
==========================

Toute ligne se terminant par 2 espaces ou 2 sauts   de ligne impose le début d'un nouveau paragraphe   pour la suite.

Une ligne de séparation entre deux paragraphes   peut être réalisés en mettant 3 tirets ( `---`), 3 caractères de
souligbnement( `___`) ou 3 apostrophes ( `***`) en début de ligne :

---
___
***

Un paragraphe de code peut être encadré par des   suites de trois apostrophes inverses $ ```$ :
```
#Ceci est un paragraphe en formatage de code
```
Si le nom du langage débute le bloc de code,   immédiatement après les $ ```$, le code bénéficiera de la mise en
couleur illustrant la syntaxe du langage cible.

Un paragraphe peut être mis en citation (ou en   retrait) en faisant débuter la ligne par un   `>`(ou plusieurs  `>>`
pour des citations imbriquées) :
> Ceci est une citation ou en texte en retrait
>> Ceci est une citation imbriquée.

Les lignes correspondant à des éléments de listes   à puces débutent par un  `*` suivi d'un espace. Les listes peuvent
être imbriquées en faisant précéder le caractère   `*` par des doubles espaces en nombres progressifs.
* liste à puces
* élément n°2
  * élement 2.1
  * élement 2.2
    * élement 2.2.1
    * élement 2.2.2

Les lignes correspondant à des éléments de listes   numérotées débutent par un numéro suivi d'un point   et d'un
espace.
1. liste ordonnée
2. élément n°2
```

3 - Structuration

Les titres sont soit soulignés avec des `-` ou des `=` :

Titre 2

soit précédés par des `#` et le nombre de `#` indique leur niveau :

Titre 1

Titre 2

Titre 3

Titre 4

Titre 5

Titre 6

Texte **Markdown** de la cellule :

```
3 - Structuration
=================

Les titres sont soit soulignés avec des  `-` ou des `=` :

Titre 2
--------

soit précédés par des  `#` et le nombre de  `#` indique leur niveau :

# Titre 1
## Titre 2
### Titre 3
#### Titre 4
##### Titre 5
###### Titre 6
```

4 - Tableaux

Les tableaux sont simulés en utilisant le `|` comme séparateur de colonne et le `-` comme séparateur de ligne. Il n'est pas nécessaire que les colonnes aient le même nombre de caractères. Par défaut les cellules sont alignées à gauche, toutefois, lors des séparations de lignes l'indications des seuls `-` impliquent l'alignement à gauche, les caractères `:-:` impliquent le centrage et les caractères `-:` l'alignement à droite.

Entête de tableau	Second entête	Troisième entête
Cellule n°1	Cellule n°2	Cellule n°3
Cellule n°4	Cellule n°5	Cellule n°6

Texte **Markdown** de la cellule :

```
4 - Tableaux
============

Les tableaux sont simulés en utilisant le  `|` comme séparateur de colonne et le  `-` comme séparateur de ligne.  Il n'est pas nécessaire que les colonnes aient le  même nombre de caractères. Par défaut les cellules  sont alignées à gauche, toutefois, lors des séparations de lignes  l'indications des seuls  `-` impliquent l'alignement à gauche, les caractères  `:-:` impliquent le centrage et les caractères  `-:` l'alignement à droite.

Entête de tableau|Second entête|Troisième entête|
-|:-:|-:|
Cellule n°1|Cellule n°2|Cellule n°3|
Cellule n°4|Cellule n°5|Cellule n°6|
```

5 - Liens hypertexte

On peut insérer un lien hypertexte affiché dont le texte est mis entre crochets et l'adresse entre parenthèses .

Le syntaxe complète est **[** *Texte du lien* **](** *URL* "*titre*" **)**. Le *titre* est facultatif et apparaît lorsque le lien est survolé.

Texte **Markdown** de la cellule :

```
5 - Liens hypertexte
================

On peut insérer un [ lien hypertexte ](www.universite-paris-saclay.fr ) dont le texte est mis entre crochets et l'adresse entre parenthèses .

Le syntaxe complète est  **[ ** _Texte du lien_ **](  ** _URL_ "_titre_" **) **. Le _titre_ est facultatif et apparaît lorsque le lien est survolé.
```

In [ ]:

5 - Images

Une image sera présenté par la notation **![** *texte alternatif* **](** *URL* "*titre*" **)**. Le *texte alternatif* et le *titre* sont recommandés mais pas obligatoires.

Texte **Markdown** de la cellule :

```
5 - Images
==========
Une image sera présenté par la notation  **![  ** _texte alternatif_  **](  ** _URL_ "_titre_" **) **. Le _texte alternatif_  et le _titre_  sont recommandés mais pas obligatoires.
![](https://www.python.org/static/community_logos/python-logo-master-v3-TM.png  "Logo du langage Python" )
```

7 - Caractères étendus

Cases cochées ou pas :

- ☑ Choix n°1
- ☐ Choix n°2

Formes géométriques, caractères décoratifs et émoticones :

&#*numéro***;** où *numéro* est un numéro décimal
&#x*numéro***;** où *numéro* est un numéro hexadécimal
Ces numéros sont issus de la référence des caractères unicodes

■ ■ ☔ ☔ ✂ ✂ ⏳ ⏳ 😀 😀

Ils peuvent par ailleurs être transférés par copier/coller.

Texte **Markdown** de la cellule :

```
7 - Caractères étendus
======================

<ins>Cases cochées ou pas :  </ins>
- [x] Choix n°1
- [ ] Choix n°2

<ins>Formes géométriques, caractères décoratifs et  émoticones :  </ins>

**&#**_numéro_**;** où _numéro_ est un numéro décimal
**&#x**_numéro_**;** où _numéro_ est un numéro hexadécimal
Ces numéros sont issus de la [ référence des caractères unicodes ](https://www.w3schools.com/charsets/
ref_utf_geometric.asp )

&#9632;   &#x25a0;
&#9730;   &#x2602;
&#9984;   &#x2700;
&#8987;   &#x231B;
&#128512;   &#x1F600;

Ils peuvent par ailleurs être transférés par  copier/coller.
```

8 - Mathématiques

Les symboles mathématiques peuvent être placés en notation LateX entre des caractères `$` . Ile peuvent également être indiqués ous forme de références unicode comme les formes géométriques, élements décoratifs et emoticones.

Les équations mathématiques peuvent être placées en notation Latex :

- entre des simples caractères `$` pour un afichage dans le texte : $\sum_{n=1}^{10} n^2$
- entre des doubles caractères `$` (`$$`) pour un affichage dans un paragraphe isolé du texte:

$$\sum_{n=1}^{10} n^2$$

Pour en connaitre les possibilités, quelques sources :
Mathématiques en Markdown sous R
Mathématiques sous Latex

Texte **Markdown** de la cellule :

```
Les symboles mathématiques peuvent être placés en  [notation LateX ](https://oeis.org/wiki/
List_of_LaTeX_mathematical_symbols ) entre des caractères  `$`. Ile peuvent également être indiqués ous forme de
[références unicode ](https://www.w3schools.com/charsets/ref_html_symbols.asp  ) comme les formes géométriques,
élements décoratifs et emoticones.


Les équations mathématiques peuvent être placées  en notation Latex :
-  entre des simples caractères  `$` pour un afichage dans le texte : $  sum_{n=1}^{10} n^2$
-  entre des doubles caractères  `$`(`$$`) pour un affichage dans un paragraphe isolé du  texte: $$ sum_{n=1}^{10}
n^2$$

Pour en connaitre les possibilités, quelques  sources :
[Mathématiques en Markdown sous R ](https://github.com/RubyRong/edav_CC27/blob/
8c8b8d8210d306163eeb21859f901597ba75eeb1/mathmd_cheatsheet.pdf )
[Mathématiques sous Latex ](https://quickref.me/latex )
```

DEUXIEME PARTIE : Cellules de Code

Principes généraux

Sous *JupyterHub* et *JupyterLab*, les cellules de code en sont créées en cliquant, dans la barre d'outil au sommet du document, sur l'outil + sachant le type (dans la liste déroulante de cette même barre d'outil) fixé sur *Code*.

Sous *Visual Studio Code (vscode)* et *Google Collab*, les cellules de code sont créées en cliquant sur le bouton **+ Code** dans la barre d'outil au sommet du document.

Elles peuvent ensuite être manipulées par les commandes apparaissant lors d'un clic droit ou secondaire.

Leur exécution et rendu (s'affichant immédiatement sous la cellule de code) sont obtenus :

- par un clic sur le bouton ▶ .
- par la séquence de touche `Maj + Entrée`

Un double-clic sur une cellule permet l'édition de son source Python.

Une cellule de code peut également recevoir des commentaires, toutefois ces derniers suivront la norme du langage : en Python les commentaires suivent le symble `#` .

In [1]:
```
# Ceci est un exemple de cellule de code ayant recours aux opérateurs arithmétiques de calcul
# l'opérateur puissance s'écrit ** en Python
1-2+3*(4/5)**6
```

Out[1]:
```
-0.21356799999999976
```

Il est possible d'obtenir l'exécution de toutes les cellules d'un classeur sous *JupyterHub* et *JupyterLab* par l'appui sur le bouton ▶▶ de la barre d'outils. Sous *Visual Studio Code (vscode)* il s'agit du bouton ▷ Run All et sous *Google Colab* vous aurez la commande `Exécution/Tout exécuter CTRL+F9` .

Dans ce cas, l'exécution peut être arrêtée par l'appui sur le bouton ■ . En cas d'erreur d'exécution sur une cellule, l'exécution s'arrête sur cette dernière qui présente la nature de l'erreur comme résultat.

Accès à la documentation

En complément des entrées du menu `Aide` , une cellule de code peut retourner la documentation d'un commande : il suffit pour cela de la faire suivre du symbole `?` , comme ci dessous pour obtenir de l'aide sur la commande `print` du langage Python :

In [2]:
```
print?
```

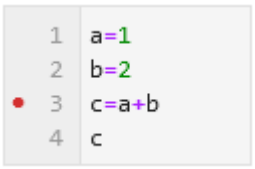
```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

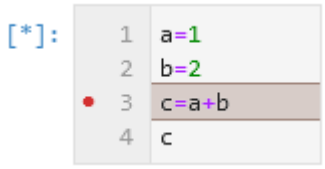
Débogage

Les instructions multilignes d'une cellule de code peuvent être exécutées en pas-à-pas, c'est à dire en marquant une pause après chaque ligne. Pour cela il faut activer le mode de débogage par le bouton à

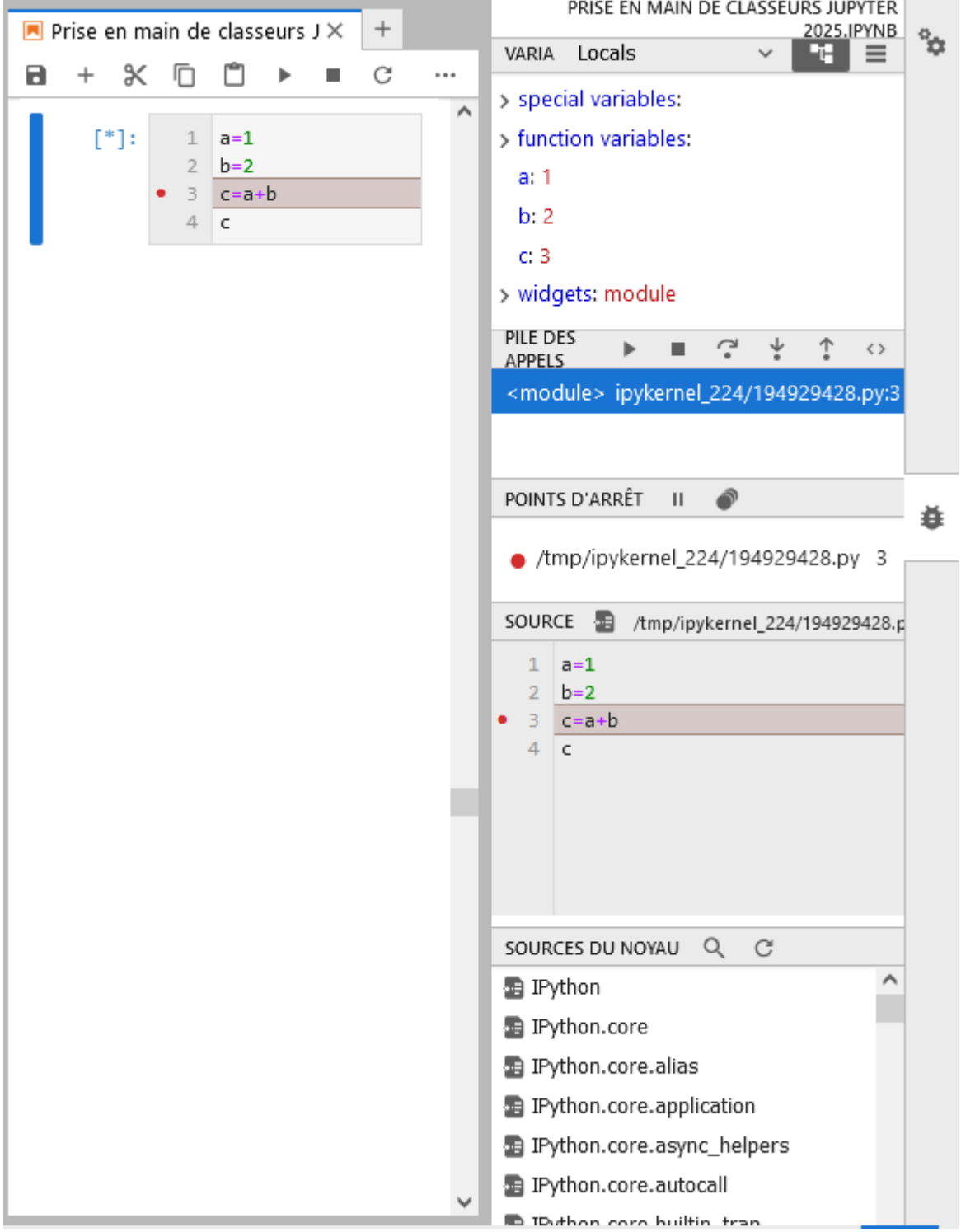
bascule  situé en haut à droite. Lorsqu'il est activé il prend l'apparence  . Dès lors on peut mettre un point d'arrêt, matérialisé par un disque rouge ● par un clic à gauche d'un numéro de ligne. Un clic ultérieur permettra de le retirer.



Lorsque la cellule de code est exécutée par par un clic sur le bouton  ou par la séquence de touche `Maj + Entrée`, l'exécution s'arrête sur la première ligne présentant un point d'arrêt.



Dès lors, un clic sur le bouton sur la barre de droite de la fenêtre fera apparaitre le volet de débogage présentant la valeur des variables et la pile d'exécution (qui indique l'arborescence d'appels de fonctions conduisant à la ligne courante).



Il est ensuite possible de passer en pas à pas à la ligne suivante en cliquant sur la touche `F10` (sur certains claviers d'ordinateur portable par `Fn` + `F10`) ou d'utiliser la barre de boutons du volet de débogage

- Continuer à vitesse rapide `F9` ou `Fn` + `F9`
- Terminer le débogage `Maj ⇧` + `F9` ou `Maj ⇧` + `Fn` + `F9`
- Instruction ou ligne suivante `F10` ou `Fn` + `F10`
- Entrer dans la fonction présente sur la ligne `F11` ou `Fn` + `F11`
- Sortir de la fonction courante `Maj ⇧` + `F11` ou `Maj ⇧` + `Fn` + `F11`
- Evaluer le code : donne la valeur de la variable ou de l'expression saisie dans une fenêtre popup.

Vous pouvez expérimenter le débogage dans la cellule ci-dessous.

In [3]:
```
a=1
b=2
c=a+b
c
```

Out[3]:
```
3
```

Widgets

Les Widgets permettent de réaliser des cellules de code interactives en intégrant des éléments de réglages du résultat par l'utilisateur. Les différents widgets sont disponible à l'adresse : https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html

Les widgets sont initialisés par les commandes :

In [4]:
```
import ipywidgets as widgets
from IPython.display import display
```

Dès lors il est possible de créer un élement d'interface utilisateur retournant une valeur choisie par l'utilisateur

In [20]:
```
element=widgets.IntSlider(
    value=7,
    min=0,
    max=10,
    step=1,
    description='Test:',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=True,
    readout_format='d')
display(element)
```

```
IntSlider(value=7, description='Test:', max=10)
```

Désormais chaque exécution des lignes suivantes renverra le dernier choix de l'utilisateur.

In [23]:
```
valeur=element.value
print(valeur)
```

```
6
```

Il est possible d'appeler interactivement des fonctions en permettant à l'utilisateur de choisir tous ou certains de leurs paramètres avec la fonction *interactive*.

Exemple avec la veleur intrinsèque d'une option :

In [57]:
```
def valeurIntrinsèqueOption(sens,S,X):
    if (sens=="Call"):
        print(max(0,S-X))
    else:
        print(max(0,X-S))

interactive(valeurIntrinsèqueOption,sens=["Call","Put"],S=(0,100,1),X=(0,100,1))
```

Out[57]:
```
interactive(children=(Dropdown(description='sens', options=('Call', 'Put'), value='Call'), In
tSlider(value=50,…
```

Exemple avec un graphique interactif de sinusoïde :

In [50]:
```
%matplotlib inline
from ipywidgets import interactive,fixed
import matplotlib.pyplot as plt
import numpy as np

def f(amplitude,fréquence,phase,décalage,légende):
    plt.figure(2)
    x = np.linspace(-10, 10, num=1000)
    plt.plot(x, décalage + amplitude*np.sin(fréquence*x+phase),label=légende )
    plt.ylim(min(-5,-amplitude), max(5,amplitude))
    plt.legend()
    plt.show()

interactive(f, amplitude=(0.1,10,0.1),fréquence=(0.1,10,0.1),phase=(-6.4,6.4,0.1),décalage=fi
```

Out[50]:
```
interactive(children=(FloatSlider(value=5.0, description='amplitude', max=10.0, min=0.1), Flo
atSlider(value=5.…
```

Il est également possible de réaliser des visualisations cartographiques :

In [7]:
```
from ipyleaflet import Map
Map(center=[48.78221, 2.28440], zoom=17)
```

Out[7]:
```
Map(center=[48.78221, 2.2844], controls=(ZoomControl(options=['position', 'zoom_in_text', 'zo
om_in_title', 'zo…
```

TROISIEME PARTIE : *Magic commands*

Ces commandes particulières permettent d'intéragir avec le système informatique sous-jacent et sont notamment utiles pout importer un module étandant les possiblités du langage Python. Elles peuvent également interagir avec le moteur de rendu pour créer des résultats difficiles à réaliser en *markdown*.

1 - Commandes en interaction avec le système d'exploitation

Leur particularité est d'être précédés d'un point d'exclamation (`!`).

En effet, le python de base est enrichi de nombreux modules qu'il faut installer avant de les utiliser.

Exemples de modules complémentaires utilisés en finance

Ces commandes utiliseront généralement l'installateur de la distribution de python sous-jacente.

`!pip install` *`nom du module/package à installer`* si distribution classique
`!conda install` *`nom du module/package à installer`* si distribution Anaconda
`!apt-get install` *`nom du module/package à installer`* si distribution debian/ubuntu linux

Certaines commandes peuvent être utiles mais dépendent du système d'exploitation :

Commande	Windows	Linux, OSX
Lister les fichiers courants	`!ls`	`!dir`
Lister les commandes disponibles	`!cmd /?`	`!ls /bin`

In [8]:
```
# Exemple de commande
!echo "Bonjour depuis le système d'exploitation"
```

```
Bonjour depuis le système d'exploitation
```

In [9]:
```
# Installation de la librairie numpy si nécessaire
!pip install numpy
```

```
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages (1.24.3)
```

In [10]:
```
#Vérification de la version de Curl, outil permettant de télécharger des fichiers et pages wel
!curl --version
```

```
curl 7.86.0 (x86_64-conda-linux-gnu) libcurl/7.86.0 OpenSSL/3.2.1 zlib/1.2.13 libssh2/1.10.0
nghttp2/1.51.0
Release-Date: 2022-10-26
Protocols: dict file ftp ftps gopher gophers http https imap imaps mqtt pop3 pop3s rtsp scp s
ftp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS GSS-API HSTS HTTP2 HTTPS-proxy IPv6 Kerberos Largefile libz NTLM
NTLM_WB SPNEGO SSL threadsafe TLS-SRP UnixSockets
```

2 - Commandes de rendu de cellules

Ces commandes modifient les rendus sous Jupyter et débutent par un symbole pourcent (`%`). Plus d'informations sont disponibles dans des classeurs dédiés :

- Cell Magics
- Scripts Magics
- Rich Display System

Documentation Jupyter

Les classeurs Jypyter peuvent même devenir interactifs interactifs.

In [11]:
```
# obtenir une liste des commandes
%lsmagic
```

Out[11]:
```
Available line magics:
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cat  %cd  %cl
ear  %colors  %conda  %config  %connect_info  %cp  %debug  %dhist  %dirs  %doctest_mode  %ed
%edit  %env  %gui  %hist  %history  %killbgscripts  %ldir  %less  %lf  %lk  %ll  %load  %load
_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %lx  %macro  %
magic  %man  %matplotlib  %mkdir  %more  %mv  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc
%pfile  %pinfo  %pinfo2  %pip  %popd  %pprint  %precision  %prun  %psearch  %psource  %pushd
%pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %rep  %rerun  %r
eset  %reset_selective  %rm  %rmdir  %run  %save  %sc  %set_env  %store  %sx  %system  %tb  %
time  %timeit  %unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%javascript  %%js  %%latex
%%markdown  %%perl  %%prun  %%pypy  %%python  %%python2  %%python3  %%ruby  %%script  %%sh  %
%svg  %%sx  %%system  %%time  %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

In [12]:
```
# Aide sur une commande
%sx?
```

```
Docstring:
Shell execute - run shell command and capture output (!! is short-hand).

%sx command

IPython will run the given command using commands.getoutput(), and
return the result formatted as a list (split on '\n').  Since the
output is _returned_, it will be stored in ipython's regular output
cache Out[N] and in the '_N' automatic variables.

Notes:

1) If an input line begins with '!!', then %sx is automatically
invoked.  That is, while::

  !ls

causes ipython to simply issue system('ls'), typing::

  !!ls

is a shorthand equivalent to::

  %sx ls

2) %sx differs from %sc in that %sx automatically splits into a list,
like '%sc -l'.  The reason for this is to make it as easy as possible
to process line-oriented shell output via further python commands.
%sc is meant to provide much finer control, but requires more
typing.

3) Just like %sc -l, this is a list with special attributes:
::

  .l (or .list) : value as list.
  .n (or .nlstr): value as newline-separated string.
  .s (or .spstr): value as whitespace-separated string.

This is very useful when trying to use such lists as arguments to
system commands.
File:      /opt/conda/lib/python3.10/site-packages/IPython/core/magics/osm.py
```

In [13]:
```
%%html
<marquee style='width: 50%; color: blue;'><b>Bonjour!</b></marquee>
```

Bonjour!

In [14]:
```
%%html
<svg id="layer" width="50%" height="auto" version="1.1" viewBox="0 0 600 203.3" xml:space="pr
<style type="text/css">
        .st0{fill:#62003C;}
</style>
<g transform="translate(-21.3,-219.1)">
        <circle class="st0" cx="604.4" cy="236" r="16.9"/>
        <circle class="st0" cx="567.7" cy="268.1" r="10.4"/>
        <path class="st0" d="m145.7 394.1c0-8.6-5.8-13.7-15.5-13.7h-18.8v41.1h9v-13h7.2l8.8 1
</g>
</svg>
```

université
PARIS-SACLAY

QUATRIEME PARTIE : EXERCICE

Rédigez l'énoncé ci-dessous dans une cellule de texte puis détaillez le calcul dans une cellule de code afin d'obtenir le résultat numérique demandé, sachant que 3% s'écrit 0.03 en décimal (le langage python ne tolérant pas de postfixer un nombre par le symbole `%`).
Le mot obligation porte un lien hypertexte renvoyant vers la page web `https://fr.wikipedia.org/wiki/Obligation_(finance)`. Le rendu peut s'écarter du modèle en fonction de la plateforme que vous utilisez, du moment que vous utilisez les polices, tableaux et équations en vigueur sur celle-ci.

La formule de calcul de la valeur de marché V à l'émission d'une obligation **au pair** et à remboursement *in fine* sachant un nominal n, un taux de coupon c et une maturité m, sachant un taux d'actualisation constant r est :

$$V = n \times \left(\frac{1}{(1+r)^m} + \sum_{i=1}^{m} \frac{c}{(1+r)^i} \right)$$

Soit une obligation ayant les caractéristiques suivantes :

- un nominal de 100 €,
- une maturité de 2 ans
- un taux de coupon de 5%

et présentant ainsi les revenus suivants :

terme	cashflow
1 an	5 €
2 ans	105 €

Dans le cas d'un taux d'actualisation de 3%, cette obligation aura une valeur à l'émission de :

```
103.8269
```

In [ ]:

La formule de calcul de la valeur de marché V à l'émission d'une obligation **au pair** et à remboursement *in fine* sachant un nominal n, un taux de coupon c et une maturité m, sachant un taux d'actualisation constant r est :

$$V = n \times \left(\frac{1}{(1+r)^m} + \sum_{i=1}^{m} \frac{c}{(1+i)^i} \right)$$

Soit une obligation ayant les caractéristiques suivantes :

- un nominal de 100 €

- une maturité de 2 ans
- un taux de coupon de 5%

présentant les revenus suivants :

terme	cashflows
1 an	5€
2 ans	105€

Dans le cas d'un taux d'actualisation de 3%, cette obligation aura la valeur à l'émission de

Source markdown :

```
La formule de calcul de la valeur de marché $V$ à l'émission d'une
[obligation](https://fr.wikipedia.org/wiki/Obligation_(finance)) **au pair**
et à remboursement _in fine_ sachant un nominal $n$, un taux de coupon $c$
et une maturité $m$, sachant un taux d'actualisation <ins>constant</ins> $r$
est :
$$V=n \times \left( \frac{1}{(1+r)^{m}} +\sum_{i=1}^{m}{\frac{c}{(1+i)^i}}
\right)$$
Soit une obligation ayant les caractéristiques suivantes :
* un nominal de 100 €
* une maturité de 2 ans
* un taux de coupon de 5%
présentant les revenus suivants :

|terme|cashflows|
|:-:|:-:|
|1 an|5€|
|2 ans|105€|

Dans le cas d'un taux d'actualisation de 3%, cette obligation aura la valeur
à l'émission de
```

In [15]:
```
5/(1+0.03)+105/(1+0.03)**2
```

Out[15]:
```
103.82693939108304
```