



Recommandations pour la programmation scientifique.

Cocopyshs – 24 novembre, 2022





Matthias Bussonnier

Ingénieur Logiciel chez Quansight-Labs ou je
développe de l'Open Source Scientifique 99% du temps.

Jupyter/IPython Core Developer, Fondateur.



bussonniermatthias@gmail.com

@Carreau sur GitHub



Comment suis-je arrivé ici ?

- 2010-2014 M2 Physique et Doctorat en Biophysique – Actin Gel Mechanics
- 2011 – Début de contribution à l'Open Source Python
- 2012 – Core Développeur de IPython qui deviendra en partie Jupyter.
- 2015 – PostDoc à University of California Berkeley sur IPython – IPython devient Jupyter .
- 2018 – Research Facilitator à University of California Merced.
- 2020 – Open Source developer à QuanSight Labs.



De l'Académique vers développeur professionnel.

Dans mon cas, développer des outils, et enseigner comment les utiliser a un impact plus important sur la recherche.




Qu'est-ce que programmation Scientifique

- Exploratoire
- Le but principal est la compréhension (des données), pas l'acte de programmer en tant que tel.
- Le code fait partie intégrante de la narration.
- Le processus d'interrogation et de dialogue avec l'ordinateur est critique.

En opposition à l'ingénierie logicielle.

- L'état de départ et le but sont connus.
- On cherche à produire un logiciel/code qui nous amène du point de départ à notre but.

- 
- Les deux ne sont pas en opposition.
 - Nous passons constamment de l'un à l'autre.
 - Explorer un jeu de données CSV.
 - Charger un CSV et visualiser sur une carte (programmation Scientifique).
 - Dataframe pandas -> Carte Choroplèthe avec matplotlib (ingénierie logicielle)
 - Interpréter la carte pour savoir quelle opération effectuer ensuite , itérer/filter le jeu de données (programmation Scientifique)



La Place de Python

Python est relativement bien placé car:

- Simple à écrire,
- Interactif
- Suffisamment performant grâce aux bonnes bibliothèques.

Polyglotisme ?

- N'hésitez pas à aller voir d'autre langage (et autres domaines scientifiques)
 - R/Julia/Javascript ...



Pourquoi utiliser des outils de programmation/code

- Le code n'est qu'un outil / levier qui amplifie les capacités du chercheur l'utilisant.
- La compréhension des outils change le domaine des possibilités accessibles.
 - Mais attention au "si tout ce que vous avez est un marteau, tout ressemble à un clou"
- Il y a un grand nombre d'outils liés à la programmation pour pouvoir s'assurer que les fondations sont stables.
 - Le plus stables les fondations sont, le plus loin, et le plus rapidement vous pourrez explorer votre domaine.



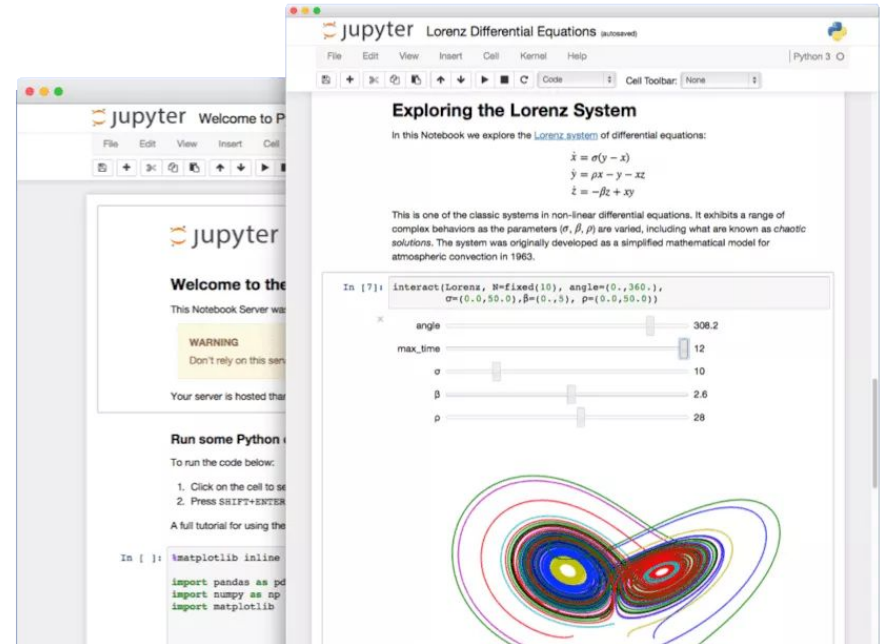
Mini Pause – Questions/Commentaires dans le Chat?

Le Dialogue Humain/Ordinateur

Jupyter

Jupyter Notebooks – Alternier Narrative et Code

- IPython a été créé en 2011 par Fernando Pérez.
- Jupyter Notebook : vers ~2012, 6ème prototype.
 - Utiliser les technologies du web pour avoir une visualisation riche.
- Interaction / dialogue avec l'ordinateur.
- Exposer cette interaction au lecteur, le laisser potentiellement interagir.





Bonnes pratique pour utiliser Jupyter

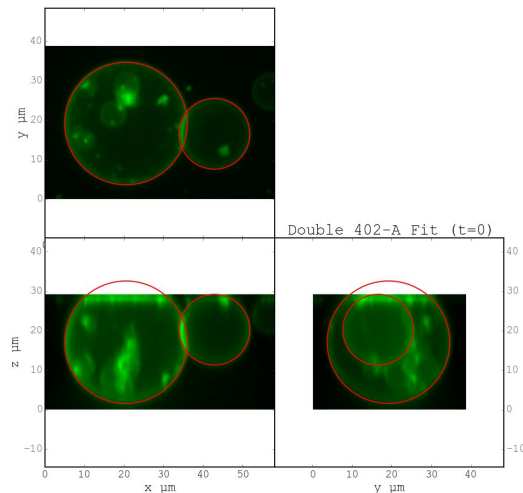
- Apprenez bien l'interface et les raccourcis claviers.
 - Utiliser completions et aide intégrée.
 - Utilisez les cellules markdown pour décrire le pourquoi, comment et vos observations.
 - Paramétrez vos notebooks pour les rendre réutilisables sur d'autres jeux de données.
 - "Redémarrer le noyau et Exécuter toutes les cellules".
-
- Sachez quand ne pas utiliser de notebook, et mettre les les fonction utilisés dans des fichiers .py

Premier pas vers la science ouverte dès conception.

Utiliser des outils comme Jupyter permet les premières étapes de science ouverte.

Un code et des analyses déjà écrites, (et sous contrôle de versions) sont faciles partager. Si possible les dépôts devraient déjà être sur une plateforme (type github GitLab), et n'ont plus qu'à être rendus publiques.

Souvent le code peut lui-même être public dès le début, ce qui permet de collaborer, ou d'avoir des retours plus rapidement.



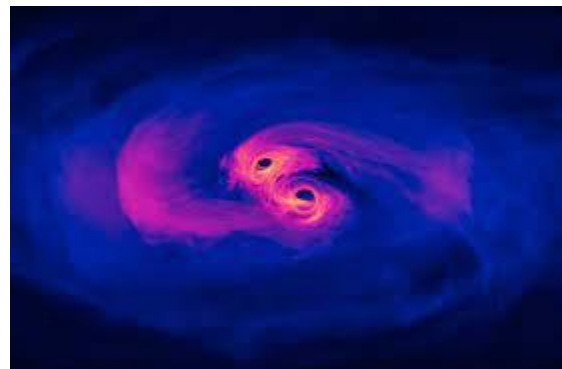
Premier pas vers la science ouverte dès conception

Reproduction des résultats et Modification Facile en utilisant des outils supplémentaires, comme Binder.

Par exemple, <https://github.com/minrk/ligo-binder>,

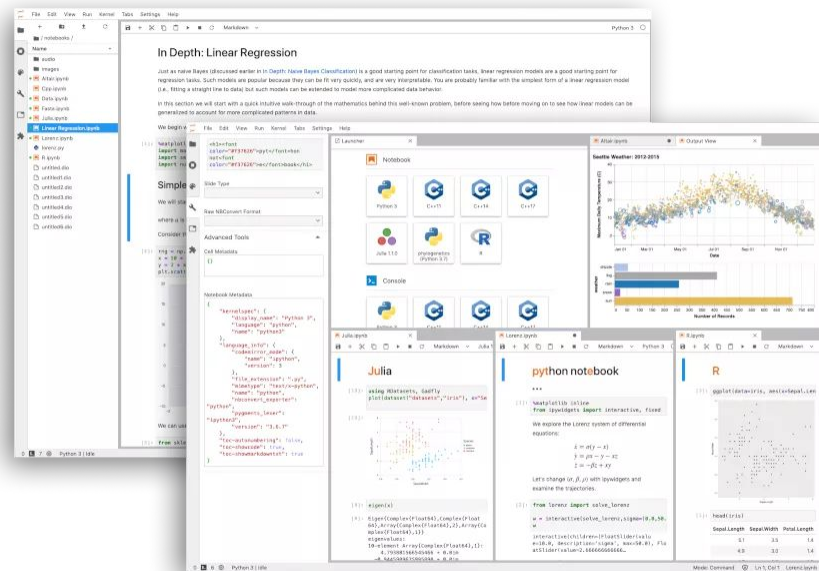
Or <https://github.com/villano-lab/galactic-spin-W1>

- Nombre infini de matériel supplémentaire.
- Plus facile à reproduire et modifier permet une itération plus rapide
- plus de citations ?



Ne pas utiliser que des notebooks

- Les notebooks sont là pour l'explication haut niveau.
- Les briques de base devraient être réutilisables.
- Contribuer aux bibliothèques open-source existantes.
 - Apprendre et découvrir d'autres façons de faire.
 - Les auteurs de bibliothèques sont très souvent académiques.
 - Numpy, SciPy, SymPy, Dask, ...





Mini Pause – Questions/Commentaires dans le Chat?



Plus loin que la programmation elle-même

Un grand nombre d'outils et de pratiques peuvent être trouvés en dehors de la programmation elle-même.

Il ne faut pas hésiter à aller voir dans le milieu d'ingénierie logiciel et adapter les outils.

Écrire des programmes est une activité sociale, avec des conventions, suivre les conventions permet une meilleure collaboration et un apprentissage plus rapide.



Exemple d'outil en dehors:

- Analyse_v1.m
- Analyse_v1.5.m
- Analyse_v2.m
- Analyse_v2_modified.m
- Analyse_v2_final.m
- Analyse_v3.m

Gestionnaire de Version, par exemple : Git

Fait initialement pour la collaboration et l'écriture de logiciel, mais est d'une grande aide pour la recherche.

- Historique, qui a fait quoi quand.
- Équivalent de Cahiers de laboratoire
 - Utile pour répondre à des revues potentielles / y avait t-il un bug pour certains jeux de données
- Sauvegarde/Partage.
- Tester différentes approches (branches/ stash)



git



Demo – Perte de données ?

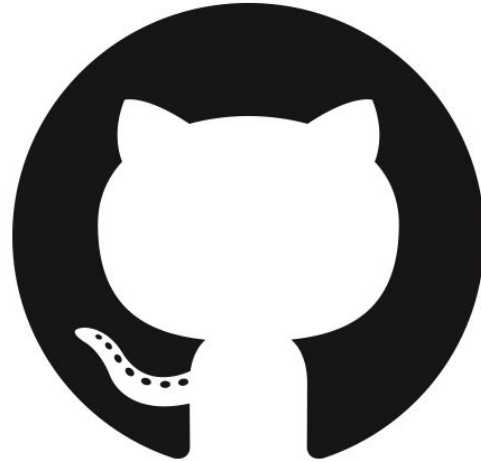
Avoir confiance dans son environnement permet d'expérimenter avec plus de sûreté.



Mini Pause – Questions/Commentaires dans le Chat?



Les plateformes de partage de code. GitLab/GitHub





Collaborations avec Humains et Robots

- intégrés avec un très grand nombre d'outils.
 - Revues, discussions, questions références.
- Vérifications Automatiques :
 - Mon code est-il conforme aux conventions ? (Pep8)
 - Y a t'ils des bugs potentiels ? (flake8, mypy)
 - ...
- Corrections Automatique:
 - Black (auto-formatting)
 - isort
- Tests, tests, tests.
 - Pytest, hypothesis, ...
 - Ais-je toujours les mêmes résultats pour mes analyses précédentes?



Conclusion

La programmation n'est pas une activité solitaire. De la même façon qu'on n'apprends pas une langue sans parler à des personnes natives.

- Partager avec vos collègues, des chercheurs du domaines, des chercheurs d'autres domaines, et aller voir un peu les bonnes pratiques d'ingénierie logicielle.
- Venez à des conférences de programmation Scientifique: [EuroSciPy](#), [JupyterCon](#)(2023 in Paris), SciPy (Texas in general).

Faites de la Science ouverte par défaut. Pas besoin d'un code parfait pour être publié.

Utilisez les outils de l'ingénierie logicielle pour diminuer votre charge mentale et faire plus de Science.



Questions?



Misc:

- Journal of Open Source Software <https://joss.theoj.org/>
- <https://docs.pytest.org/en/7.2.x/>
- <https://black.readthedocs.io/en/stable/>
- <http://mypy-lang.org/>
- <https://github.com/features/actions>
- <https://mybinder.org/>
- <https://github.com/medialab/ipysigma>