

Python et VS Code

1 Pourquoi VS Code ?

Laissons **ChatGPT** répondre à la question (Figure 1), avec une première possibilité en tête de liste des réponses.

Quel est l'environnement de développement le plus pratique pour développer des programmes en Python ainsi que des Notebooks ?

Bonne question 👍

Pour développer en **Python** et travailler aussi avec des **Notebooks (Jupyter)**, tu as plusieurs environnements possibles, chacun avec ses avantages. Voici un résumé des plus pratiques :

♦ Environnements tout-en-un

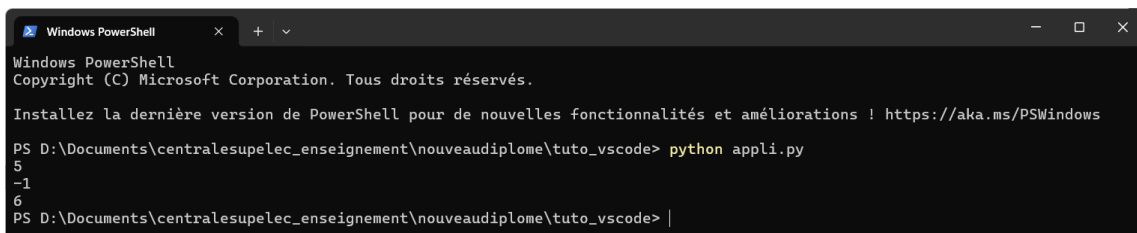
1. VS Code (Visual Studio Code)

- **Points forts :**
 - Léger, très populaire et gratuit.
 - Extension **Python** officielle (linting, debugging, auto-completion, venv).
 - Extension **Jupyter** → ouverture et exécution directe des notebooks `.ipynb`.
 - Intégration Git, gestion d'environnements virtuels, Docker, etc.
- **Idéal si :** tu veux un éditeur polyvalent pour coder en Python **et** manipuler des notebooks dans la même interface.

Figure 1

Remarquez par ailleurs (Figure 2) que l'environnement le plus basique pour développer des programmes en Python correspond à :

- un terminal de commande (on peut y lancer une commande « `python xxxx.py` » pour exécuter un programme ou une commande « `pip install yyyy` » pour installer des packages supplémentaires),
- un simple éditeur de texte (pour créer le fichier « `xxxx.py` »).



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS D:\Documents\centralesupelec_enseignement\nouveaudiplome\tuto_vscode> python appli.py
5
-1
6
PS D:\Documents\centralesupelec_enseignement\nouveaudiplome\tuto_vscode> |
```

Figure 2

L'application **VS Code** rajoute juste des fonctionnalités complémentaires à cette configuration basique pour faciliter le développement. Elle convient parfaitement pour tester des méthodes telles que celles décrites en TD ou en TP.

2 Installation

2.1 Python

La version conseillée est Python 3.10.6 :

<https://www.python.org/downloads/release/python-3106/>

Veillez à cocher l'option « Add Python to PATH » lors de l'installation.

Si vous avez déjà installé une autre version de Python, vous pouvez la conserver sans que cela pose un problème.

2.2 VS Code

2.2.1 Téléchargement

Lien pour l'installation sous Windows :

<https://code.visualstudio.com/>

Lien pour l'installation sous d'autres systèmes d'exploitation :

<https://code.visualstudio.com/Download>

2.2.2 Les « plugins » (extensions)

Lors de la première édition d'un fichier Python (fichier avec extension **.py**), **VS Code** propose d'installer une extension dédiée à l'édition et l'exécution des programmes écrits en Python (Figure 3).

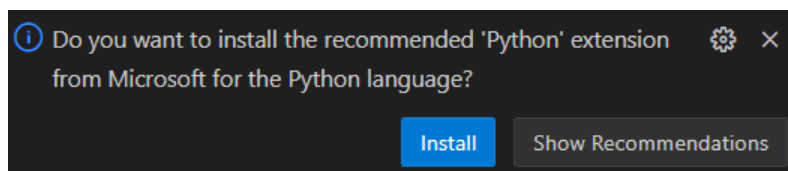


Figure 3

Lors de la première édition d'un Notebook Python et d'une première tentative d'exécution (fichier avec extension **.ipynb**), **VS Code** propose d'installer une extension dédiée à l'édition et l'exécution des Notebooks (Figure 4).

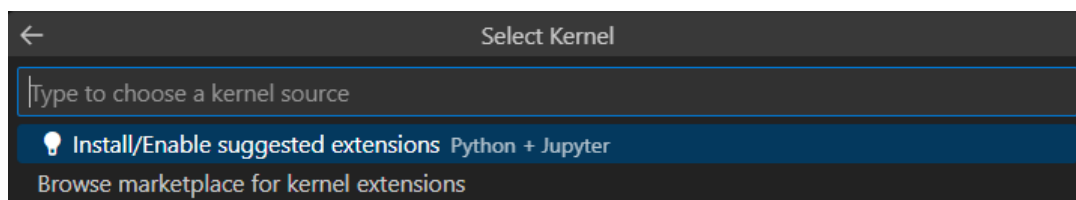


Figure 4

Après cette première série d'installations, on peut consulter la liste des extensions installées avec le bouton « damier » dans le menu à gauche de l'écran (Figure 5).

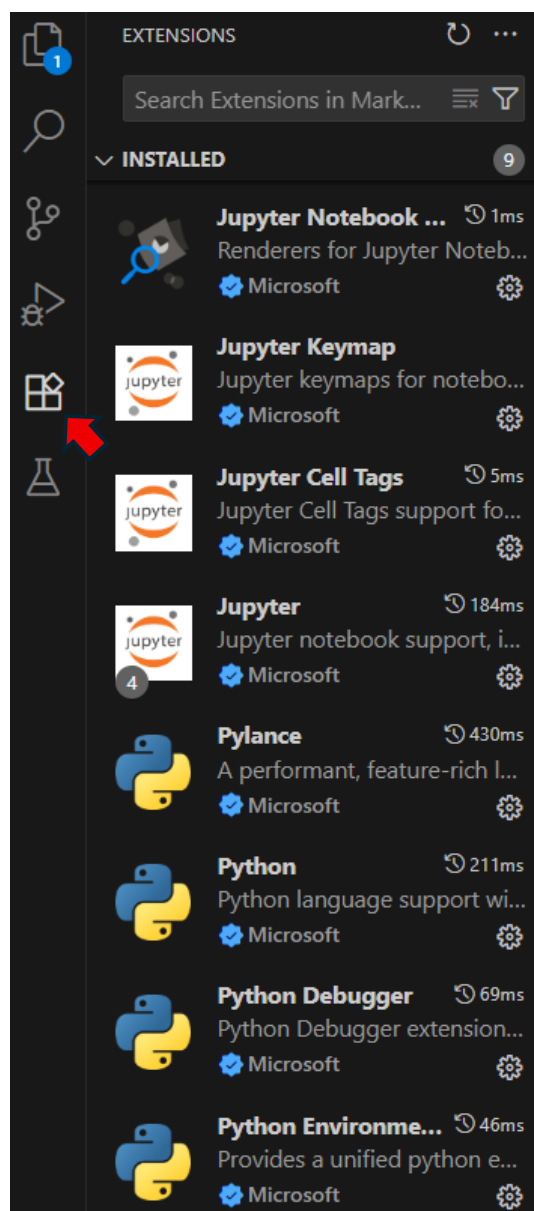


Figure 5

On peut envisager ensuite d'installer d'autres extensions facilitant l'utilisation de VS Code et la mise en forme correcte des documents (Figure 6 et Figure 7).

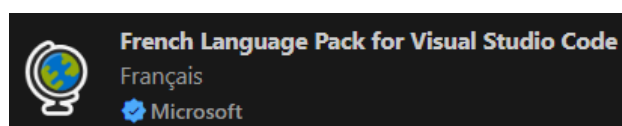


Figure 6

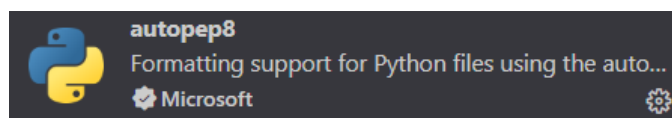


Figure 7

3 Conseils pour le développement

3.1 Python

Plutôt que de rassembler les fonctions et les instructions qui les appellent dans un unique fichier Python, il est préférable de procéder comme indiqué ci-dessous.

Dans le même répertoire, on peut générer un fichier ne contenant que les fonctions (**ope.py** dans l'exemple) et un fichier ne contenant que l'appel à ces fonctions et d'autres instructions (**appli.py** dans l'exemple).

Fichier **ope.py**

```
def add(a, b):  
    return a+b  
  
def sub(a, b):  
    return a-b  
  
def mul(a, b):  
    return a*b
```

Fichier **appli.py**

```
import ope  
  
c, d = 2, 3  
e = ope.add(c, d)  
print(e)  
e = ope.sub(c, d)  
print(e)  
e = ope.mul(c, d)  
print(e)
```

3.2 Jupyter

De la même façon, avec un fichier Notebook, on peut affecter une cellule de code à la définition des fonctions et d'autres cellules de code à l'appel de ces fonctions (Figure 8). Pour s'assurer de la cohérence du Notebook, il est préférable, pour finir, d'utiliser la commande « Exécuter tout » (menu en haut de la fenêtre) et vérifier qu'il n'y a aucune erreur à l'exécution. Il pourra alors être repris par d'autres utilisateurs qui retrouveront ainsi les mêmes résultats.



Figure 8

4 Cellules « Marquage » (*Markdown*)

4.1 Principe

Dans un Notebook, il est possible d'utiliser aussi des cellules « Marquage » en complément des cellules de code. Dans ces cellules, le langage [Markdown](#) est utilisé pour la mise en forme de ce qui apparaîtra ensuite à l'écran.

4.2 Exemple dans VS Code

En Figure 9 apparaît le texte brut qui est tapé dans la cellule « Marquage ». Le résultat final est visualisé en Figure 10 après interprétation de ce texte brut. On peut basculer facilement entre les deux modes, via le bouton « check » ou « crayon » en haut à droite de la cellule.

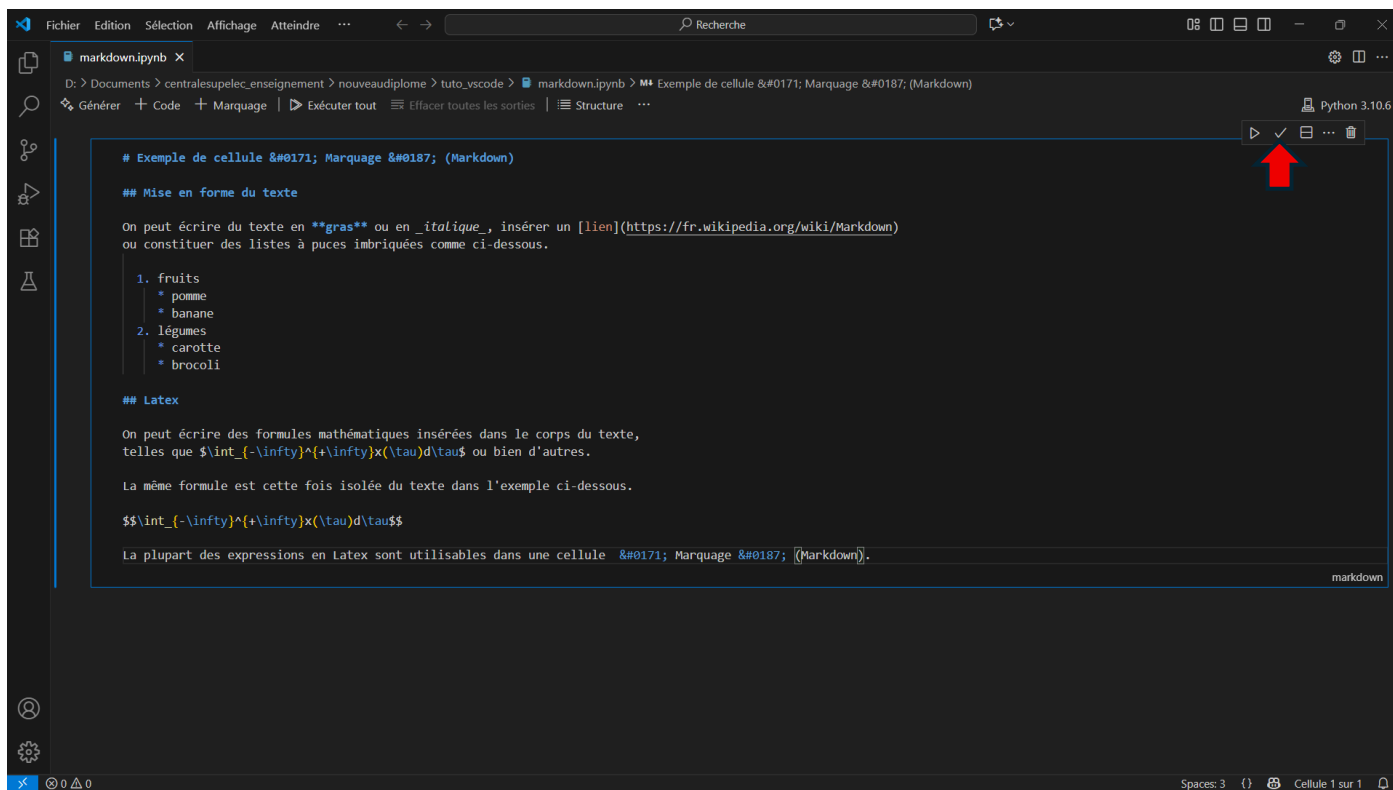


Figure 9

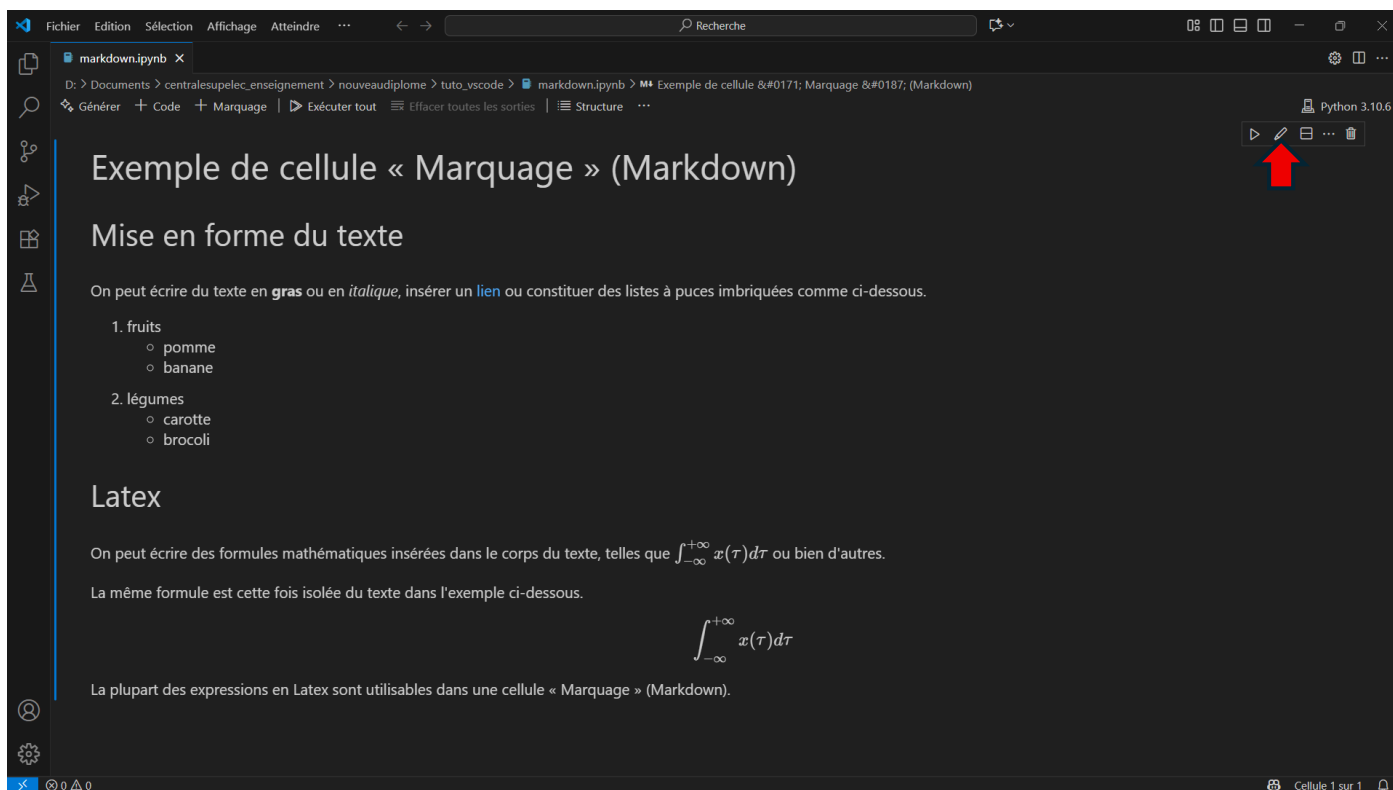


Figure 10