

Utilisation de VSCode pour le projet MSLD

1. Télécharger et installer VSCode

1. Télécharger l'installateur approprié en ligne: <https://code.visualstudio.com/>.
2. Installer et lancer VSCode.

2. Ouvrir le projet

On peut ouvrir le dossier du projet en cliquant sur "Open Folder..." puis en navigant jusqu'au dossier du projet.

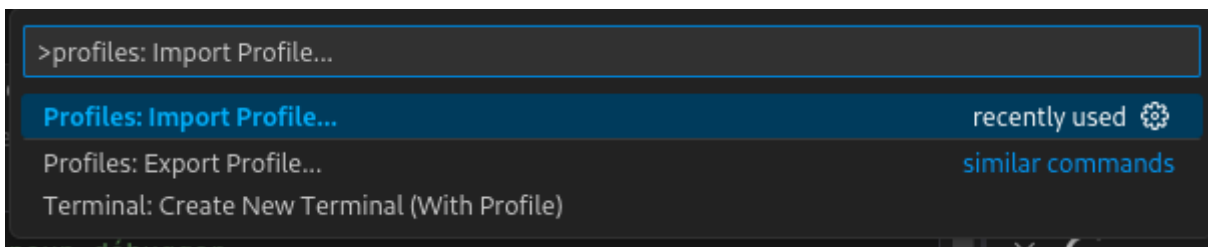
3. Installer les extensions

Pour utiliser les outils de développement Python dans VSCode, il faut installer quelques extensions. Les plus importantes sont:

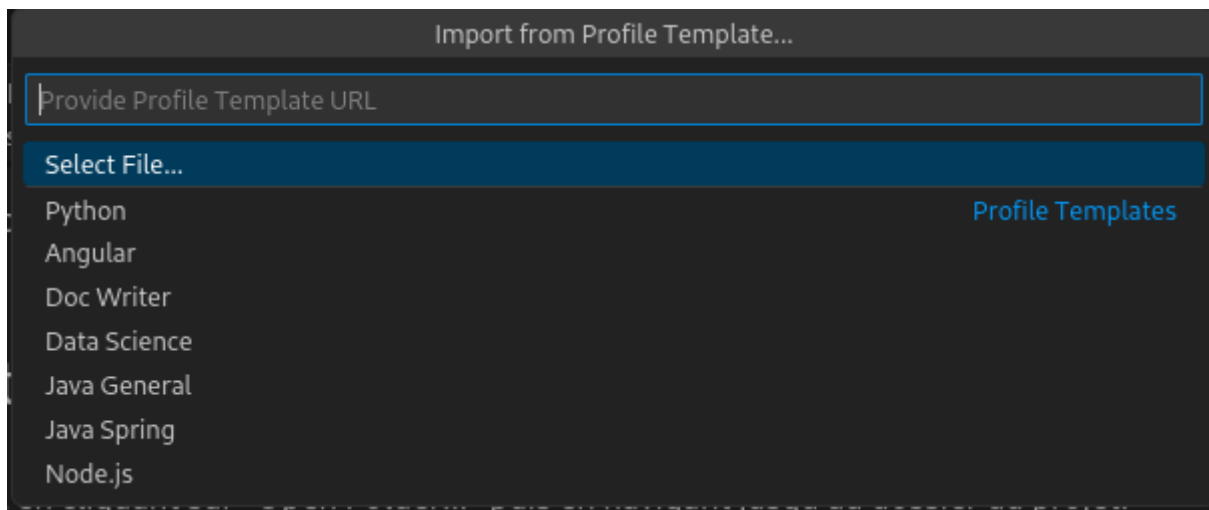
- ["Python"](#)
- ["Jupyter"](#)

Vous pouvez utiliser le profil fourni qui installera toutes les extensions:

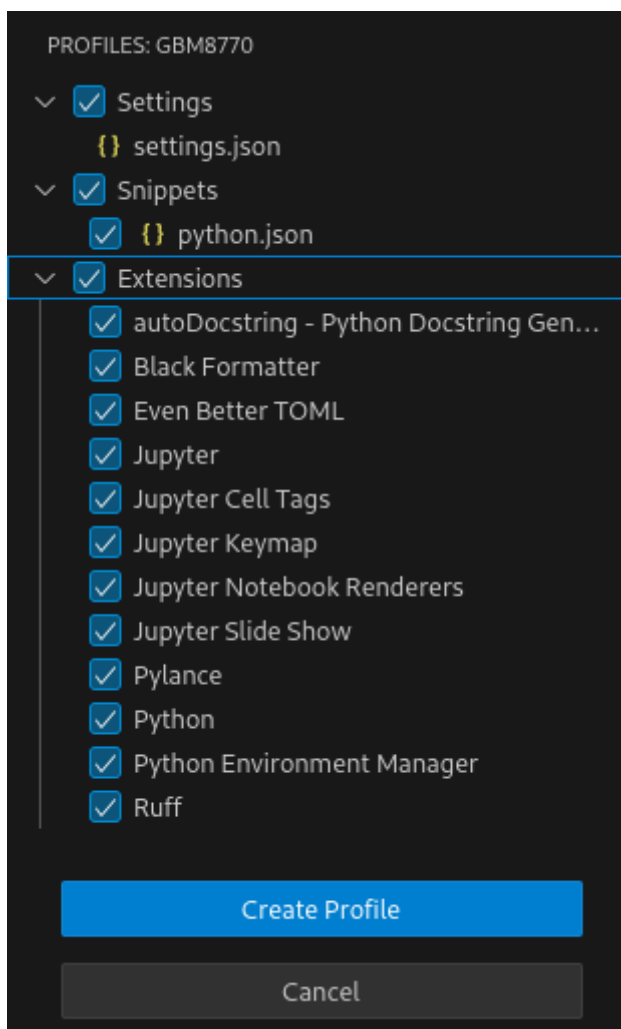
1. Ouvrir la palette de commande avec `Ctrl+Shift+P` (ou `Cmd+Shit+P` pour les utilisateurs de MacOS)
2. Sélectionner la commande "Profiles: Import Profile..."



3. Cliquer sur "Select File..." et naviguer jusqu'au fichier `GBM8770.code-profile`.

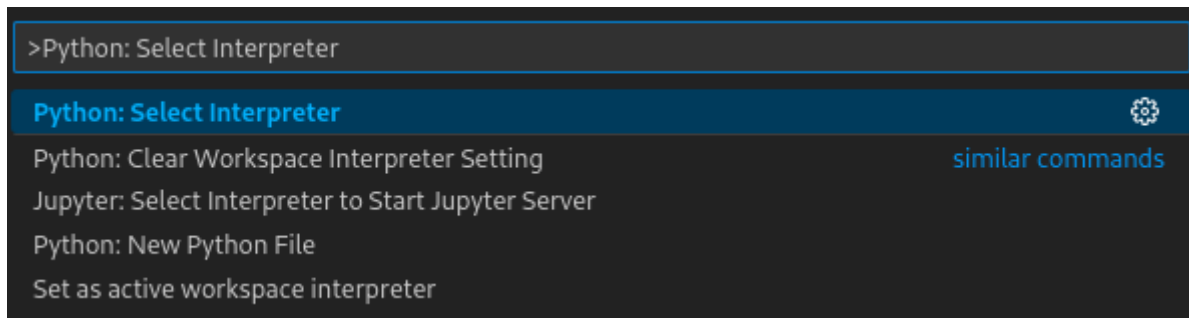


4. Dans l'onglet "Profiles" qui s'ouvre, cliquer sur "Create Profile"

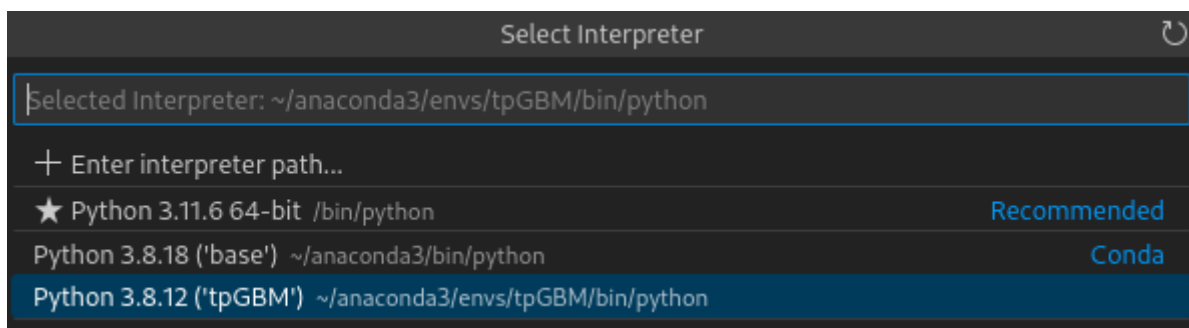


4. Choisir l'interpréteur Python

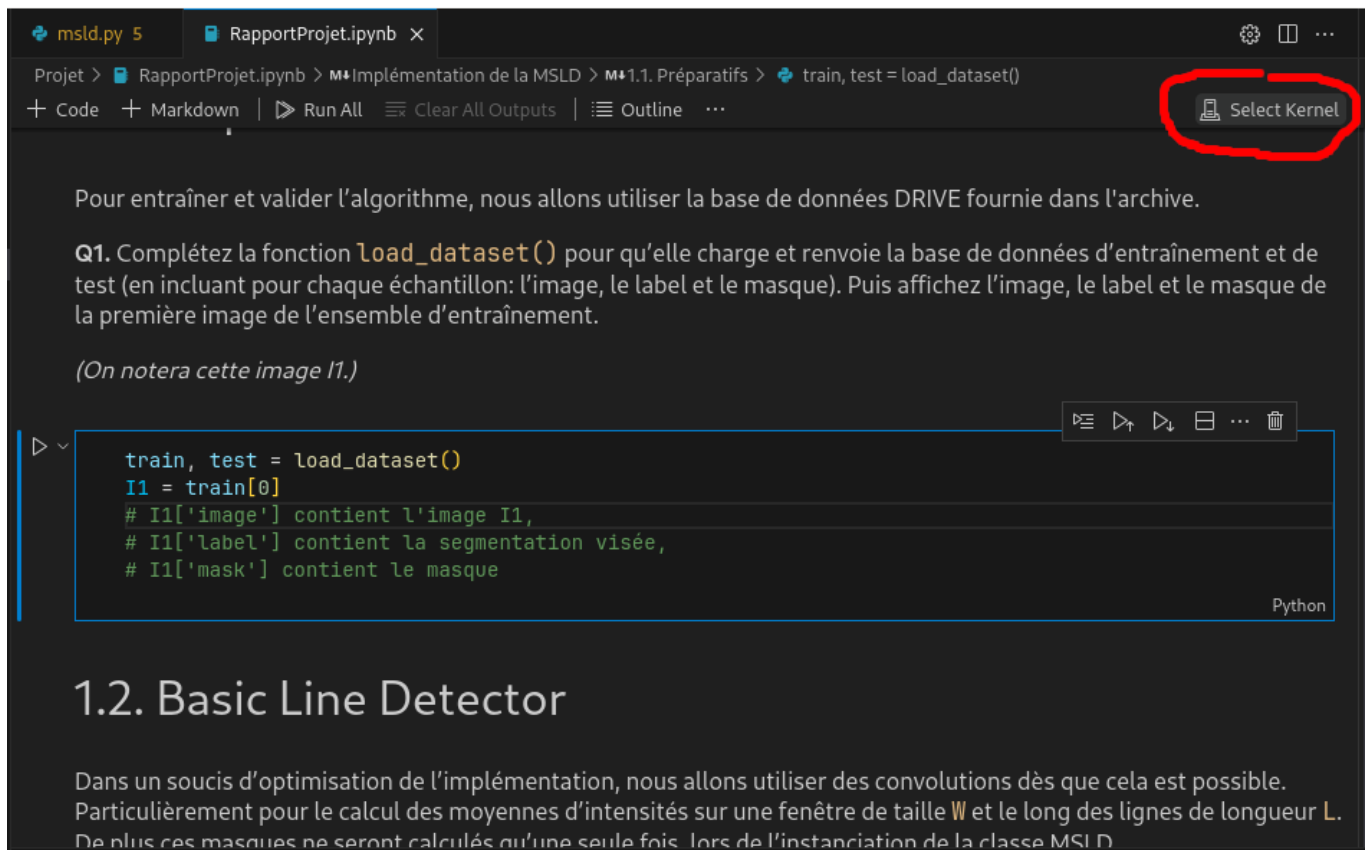
1. Ouvrir la palette de commande avec `Ctrl+Shift+P` (ou `Cmd+Shit+P` pour les utilisateurs de MacOS)
2. Sélectionner la commande "Python: Select Interpreter"



3. Sélectionner l'interpréteur `tpGBM` , ou naviguer vers l'exécutable Python s'il n'est pas affiché. Sous Windows, le chemin devrait ressembler à `C:\Users\emman\AppData\Local\miniconda3\envs\tpGBM\python.exe` ou `C:\Users\emman\miniconda3\envs\tpGBM\python.exe` . Sous MacOS ou Linux, ça devrait être `~/anaconda3/envs/tpGBM/bin/python` .



4. Ouvrir le notebook `RapportProjet.ipynb`
5. Cliquer sur "Select Kernel"



msld.py 5 RapportProjet.ipynb x

Projet > RapportProjet.ipynb > M•Implémentation de la MSLD > M•1.1. Préparatifs > train, test = load_dataset()

+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...

Select Kernel

Pour entraîner et valider l'algorithme, nous allons utiliser la base de données DRIVE fournie dans l'archive.

Q1. Complétez la fonction `load_dataset()` pour qu'elle charge et renvoie la base de données d'entraînement et de test (en incluant pour chaque échantillon: l'image, le label et le masque). Puis affichez l'image, le label et le masque de la première image de l'ensemble d'entraînement.

(On notera cette image `I1`.)

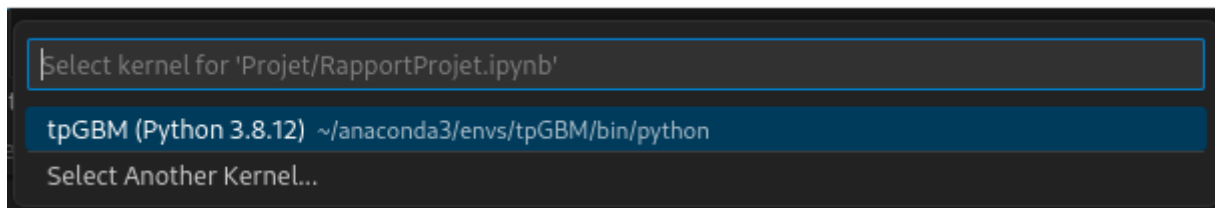
```
train, test = load_dataset()
I1 = train[0]
# I1['image'] contient l'image I1,
# I1['label'] contient la segmentation visée,
# I1['mask'] contient le masque
```

Python

1.2. Basic Line Detector

Dans un souci d'optimisation de l'implémentation, nous allons utiliser des convolutions dès que cela est possible. Particulièrement pour le calcul des moyennes d'intensités sur une fenêtre de taille `W` et le long des lignes de longueur `L`. De plus ces masques ne seront calculés qu'une seule fois lors de l'instanciation de la classe `MSI D`.

- Sélectionner l'interpréteur `tpGBM`. Si l'environnement `conda` n'est pas proposé, choisir "Python Environments..." puis sélectionner l'interpréteur `tpGBM` comme précédemment.



Select kernel for 'Projet/RapportProjet.ipynb'

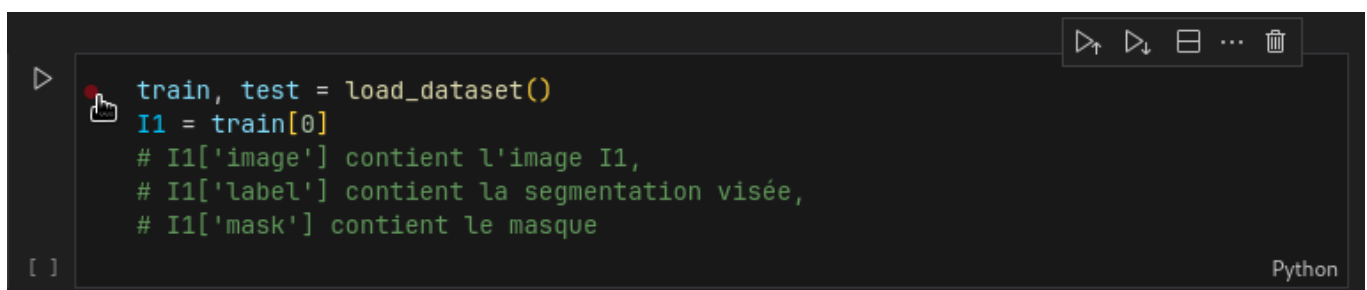
tpGBM (Python 3.8.12) ~/anaconda3/envs/tpGBM/bin/python

Select Another Kernel...

5. Débugger dans un Jupyter Notebook

Maintenant que les extensions sont installées et configurées, on peut commencer à travailler!

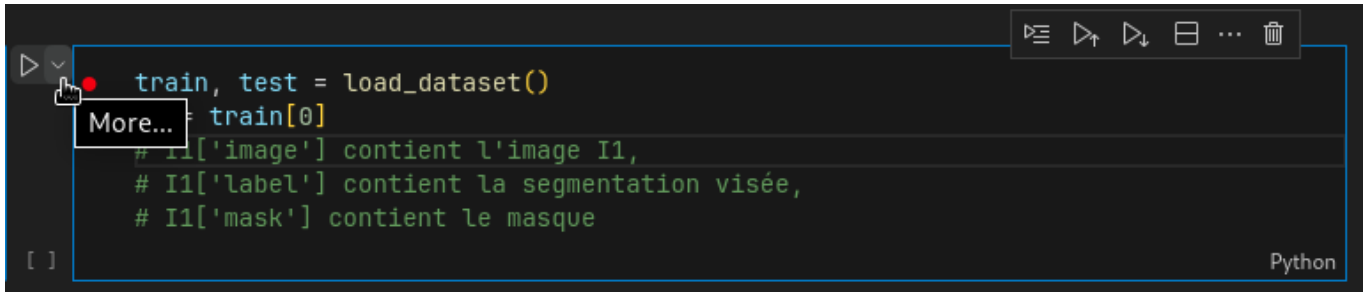
Dans le Notebook, on peut cliquer dans la marge à gauche des cellules de code pour placer un "breakpoint".



```
train, test = load_dataset()
I1 = train[0]
# I1['image'] contient l'image I1,
# I1['label'] contient la segmentation visée,
# I1['mask'] contient le masque
```

Python

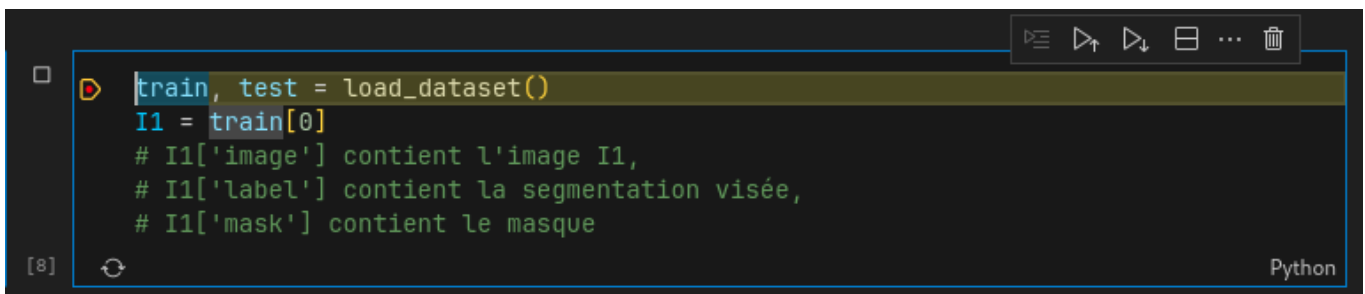
On peut ensuite cliquer sur la flèche à côté du bouton pour exécuter la cellule et sélectionner "Debug Cell" (ou bien utiliser le raccourci clavier `Alt+Ctrl+Shift+Enter`)



```
train, test = load_dataset()
I1 = train[0]
# I1['image'] contient l'image I1,
# I1['label'] contient la segmentation visée,
# I1['mask'] contient le masque
```

Python

Ceci lancera l'exécution de la cellule, mais en s'arrêtant **avant** d'exécuter la ligne où est placée le "breakpoint".



```
train, test = load_dataset()
I1 = train[0]
# I1['image'] contient l'image I1,
# I1['label'] contient la segmentation visée,
# I1['mask'] contient le masque
```

Python

Vous devriez voir apparaître en haut de votre écran les boutons pour contrôler le débogueur (que vous pouvez également contrôler avec les raccourcis clavier).



De gauche à droite, vous avez...

- "Continue" (`F5`): poursuit l'exécution jusqu'au prochain "breakpoint", ou jusqu'à la fin de la cellule s'il n'y en a pas d'autre.
- "Step Over" (`F10`): exécute la ligne actuelle, et attend à la ligne suivante.
- "Step Into" (`F11`): exécute la ligne actuelle, mais en entrant dans le "scope" la fonction appelée sur cette ligne.
- "Step Out" (`Shift+F11`): termine l'exécution du "scope" actuel, et attend avant de poursuivre l'exécution dans le "scope" extérieur.
- "Restart" (`Ctrl+Shift+F5`): relance l'exécution du débogueur depuis le début.
- "Disconnect" (`Shift+F5`): poursuit l'exécution et déconnecte le débogueur.

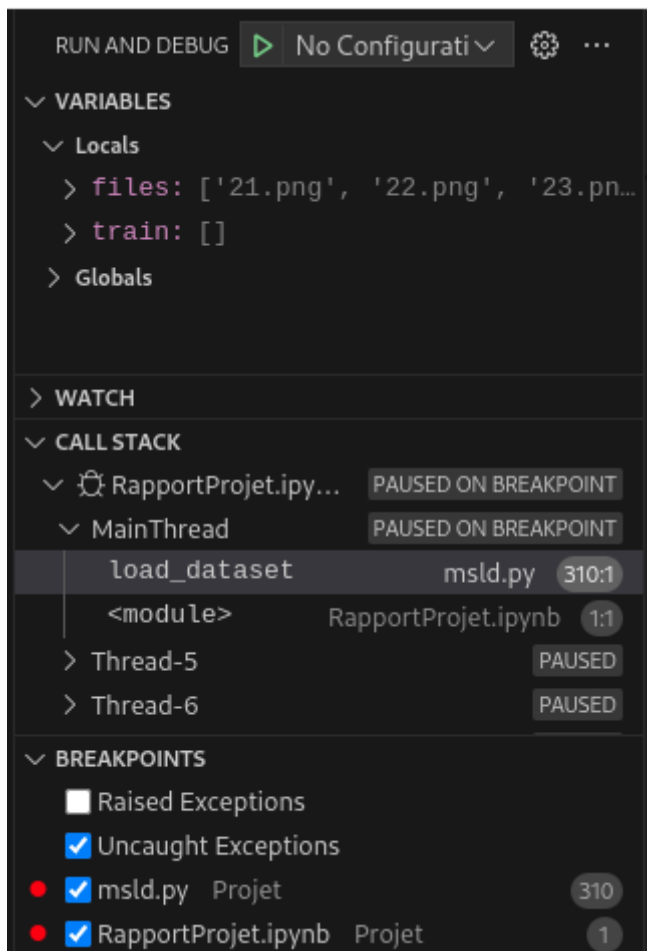
Vous pouvez également placer des "breakpoints" dans le fichier `msld.py`, où le débogueur s'arrêtera.

```

301 def load_dataset() -> List[dict]:
302     """Charge les images des ensembles d'entraînement et de test dans 2 listes de dictionnaires. Pour chaque
303     échantillon, il faut créer un dictionnaire dans le dataset contenant les champs ["name", "image", "label", "mask"].
304     On pourra ainsi accéder à la première image du dataset d'entraînement avec train[0]["image"].
305     """
306
307     files = sorted(os.listdir("DRIVE/data/training/"))
308     train = []
309
310     for file in files:
311         sample = {}
312         sample["name"] = file
313
314         # TODO 1.1.Q1 Chargez les images image, label et mask:
315         sample["image"] = ... # Type float, intensité comprises entre 0 et 1
316         sample["label"] = ... # Type booléen
317         sample["mask"] = ... # Type booléen
318
319     test = []
320     # TODO 1.1.Q1 De la même manière, chargez les images de test.
321
322     return train, test

```

Dans le panneau de débog à gauche de la fenêtre, vous pouvez inspecter les variables, vous déplacer d'un "scope" à l'autre, et voir les breakpoints que vous avez placés.



Vous pouvez également lancer une console de débog où vous pouvez exécuter du code arbitraire:

1. Ouvrir la palette de commande avec `Ctrl+Shift+P` (ou `Cmd+Shit+P` pour les utilisateurs de MacOS)

2. Sélectionner "Debug: Focus on Debug Console View" ou "Debug Console: Focus on Debug Console View" (les deux font la même chose).

