# IDE VSCode - Premiers programmes en Python "étape par étape"

[Mise à jour le : 3/9/2022]

- **Sources**
  - Python.org : documentation.
  - Getting Started with Python in VS Code
  - **Généralités** sur le wiki VSCode pour l'installation et la présentation des extensions Python dans VSCode.
- **Lectures connexes**
  - **Real Python** - How to Use Python: Your First Steps
  - Introduction to Python
  - Python Keywords: An Introduction
  - Fonctions natives (built-in)[1)]

---

> **Objectif** : **créer**, **tester** et **déboguer** un programme écrit en langage **Python** contenu dans un fichier installé sur le serveur **NAS_SIN**.

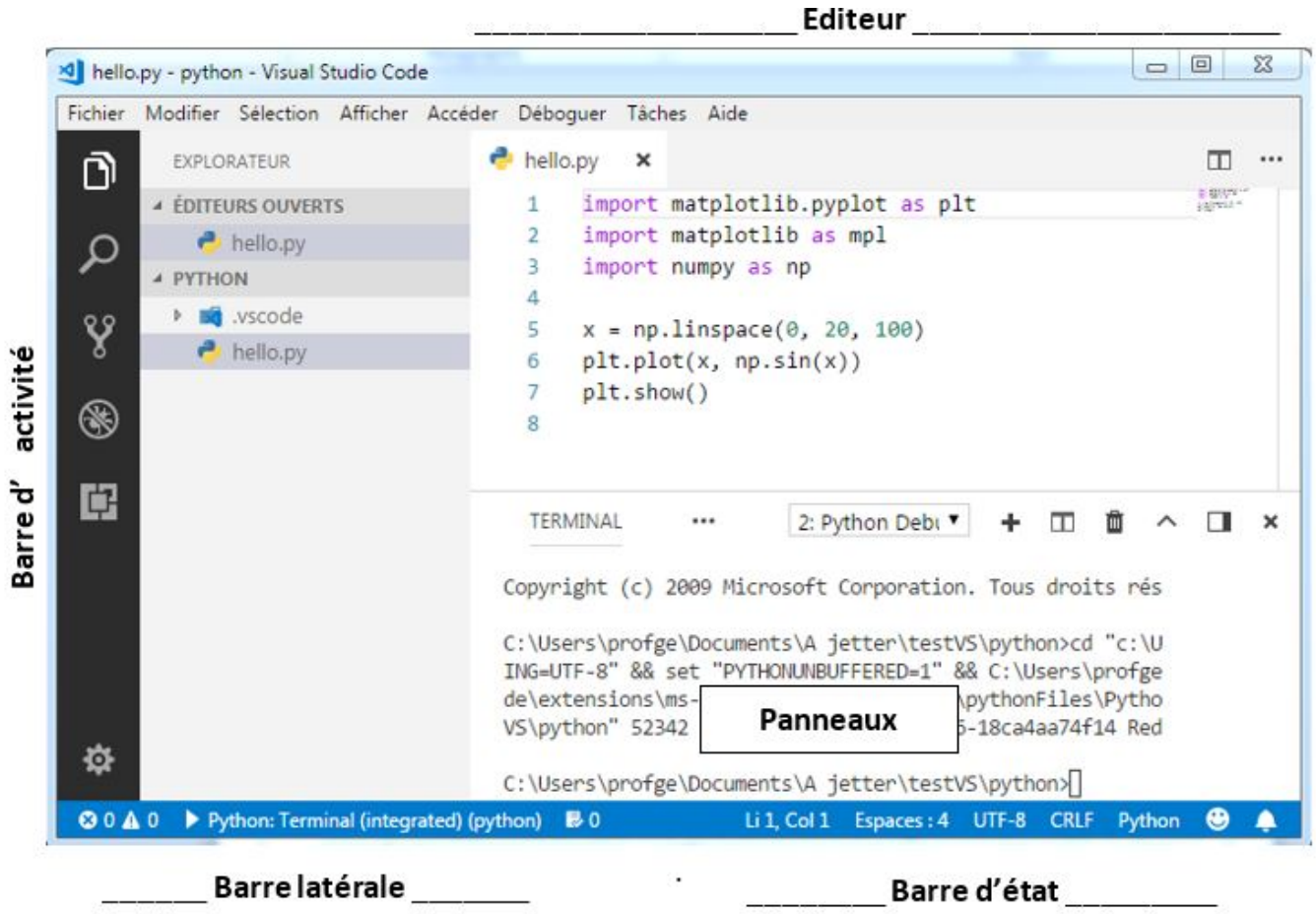
## 1. Généralités

- **Source** : Getting Started

**Visual Studio Code** est un **éditeur de code source** léger, mais puissant, disponible pour **Windows**, macOS et Linux. Il est livré avec un support intégré pour JavaScript, TypeScript et Node.js et possède des extensions pour d'autres langages (tels qu' Arduino, C++, C#, Java, Python, PHP, Go etc.) et des runtimes (.NET et Unity) .

### 1.1 L'interface utilisateur

> Comme beaucoup d'autres éditeurs, VSCode présente un **explorateur** sur la gauche de son interface montrant tous les fichiers et dossiers auxquels vous avez accès, un **éditeur** sur la droite, montrant le contenu des fichiers que vous avez ouverts.




Cette interface est divisée en cinq zones :

- **La barre d'activité** (Activity Bar) est située à l'extrême gauche. Cette barre vous permet de passer d'une vue à l'autre et vous fournit des indicateurs contextuels supplémentaires, tels que le nombre de changements sortants lorsque Git est activé.
- **La barre latérale** (Side Bar) contient des vues différentes comme l'explorateur pour vous aider tout en travaillant sur votre projet.
- **L'éditeur** (Editor Groups) – C'est la zone principale pour éditer vos fichiers. Vous pouvez ouvrir jusqu'à trois éditeurs côte à côte.
- **Les panneaux** (Pannels) permettent d'afficher différents panneaux sous la zone de l'éditeur pour des informations de sortie ou de débogage, des erreurs et des avertissements, ou un terminal intégré. Le panneau peut également être déplacé vers la droite pour plus d'espace vertical.
- **La barre d'état** (Status Bar) donne des informations sur le projet ouvert et les fichiers que vous modifiez.

> Chaque fois que vous chargez un projet dans VSCode, il apparaît dans le **même état** que lors de sa dernière fermeture. Le dossier, la disposition et les fichiers ouverts sont conservés.

## 2. Hello World

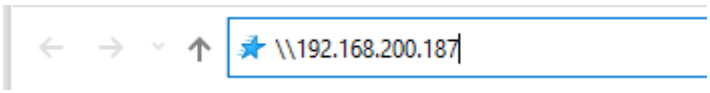
Le logiciel et ses extensions étant installés sur votre PC, vous allez découvrir son utilisation à travers

l'activité développée dans la suite du document. Nous nous fixons l'**objectif** suivant :

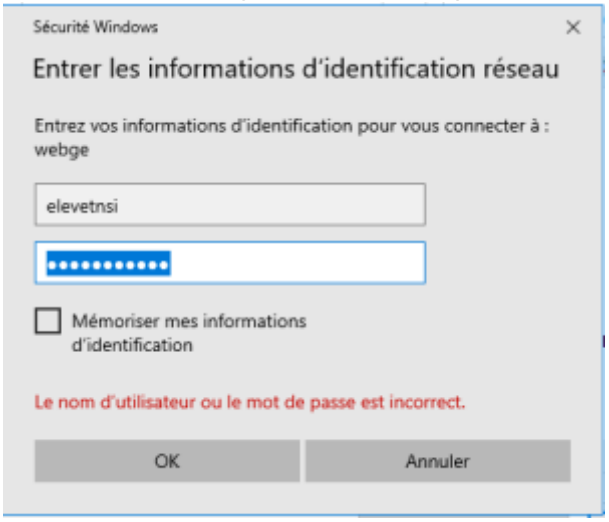
> Écrire le code d'un simple “**Hello World** !”

### Étape 1 - Créer un dossier dans le répertoire Python de votre compte sur le NAS SIN

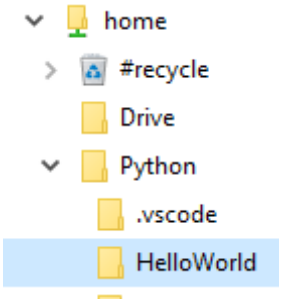
1. Ouvrir le gestionnaire de fichiers. Entrer l'adresse du serveur.




2. Se connecter à son compte en entrant le login et le mot de passe.



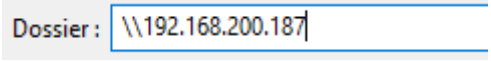

3. Créer un nouveau répertoire dans **home/Python**.
*Exemple* : HelloWorld




> Le **dossier** préalablement créé devient “l'**espace de travail**” de VSCode. VSCode stocke les paramètres spécifiques à cet espace de travail dans *.vscode / settings.json*. Ces paramètres sont distincts des paramètres utilisateur stockés globalement.

### Étape 2 - Placer le dossier dans Visual Studio Code

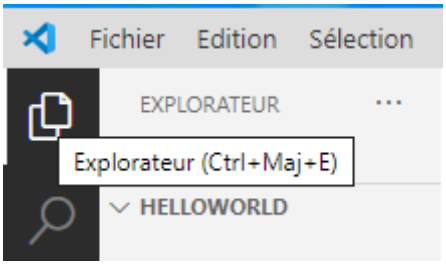
1. Menu Fichier → Ouvrir le dossier



> Sélectionner le dossier (ex:HelloWorld) et le faire glisser dans la **zone d'édition** de VSCode

### Étape 3 - Créer le fichier hello.py

- Dans la **barre d'activité**, cliquez sur l'icône **Explorateur** si celui-ci n'apparaît pas comme ci-dessous.




- Cliquer sur l'icône "*nouveau fichier*" représentée ci-dessous et le nommer *hello.py*.

- La fenêtre de l'**explorateur** doit ressembler à la copie d'écran ci-dessous.

Le mot Python doit apparaître dans la barre d'état et la version de l'interpréteur, par exemple : doit être présente dans la barre latérale. Si ce n'est pas le cas, voir l'étape 4.




### Étape 4 - Sélectionner un interpréteur Python

- **Source** : Using Python environments in VS Code

Lorsqu'un fichier .py a été créé dans le répertoire du projet, il est possible de sélectionner une version de l'interpréteur Python à partir de la barre d'état.
Python est un langage interprété. Pour exécuter du code Python et obtenir la complétion de code du langage, vous devez indiquer à VSCode l'interpréteur à utiliser en **cliquant sur "*Select Python Environment*" dans la barre latérale** (voir ci-dessous) ou par **F1 → python:Sélectionner l'interpréteur** ou .

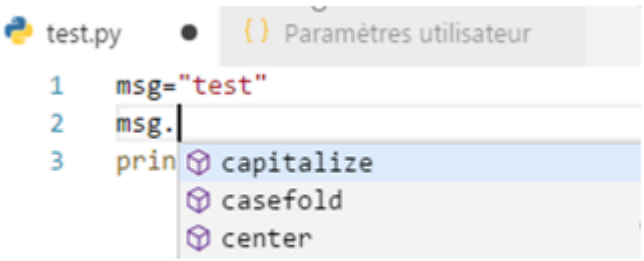
### Étape 5 - Éditer le code

- **Source** : Editing Python in Visual Studio Code

Saisir le code ci-dessous dans la zone d'édition.

Exemple1.py

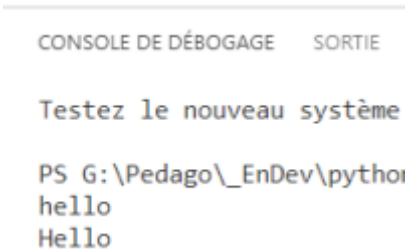
```
# %%
msg = "hello"
print(msg)
print (msg.capitalize())
```




> **#%%** définit une cellule de code de type **Jupyter** dans le code Python. La **complétion de code** permet de retrouver les méthodes associées à l'objet *msg* telles que *capitalize*.
> Le symbole identifie une **méthode** ou une **fonction**. Les autres symboles sont décrits **ici**.

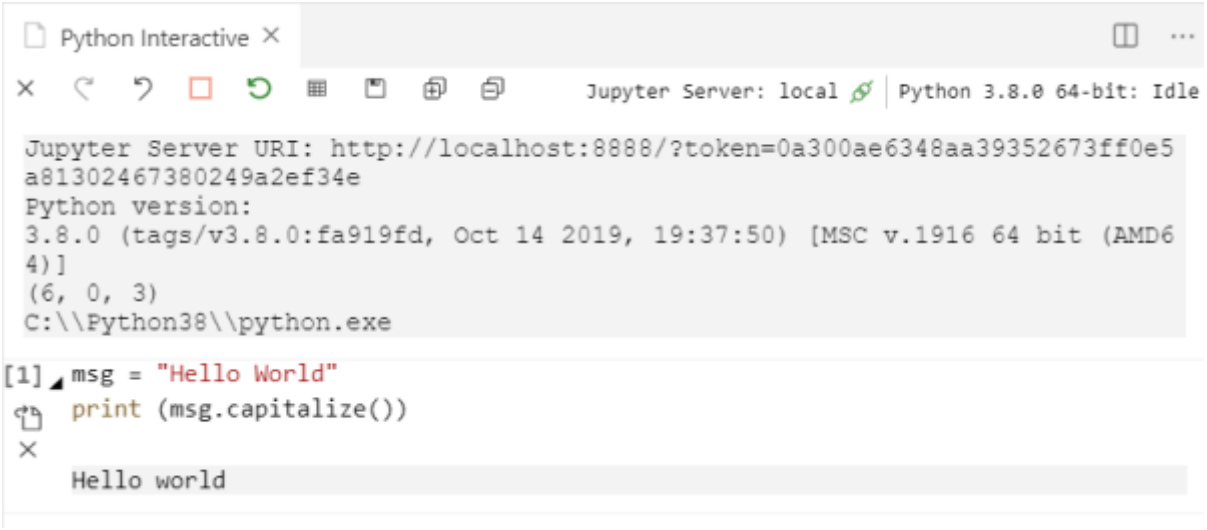
### Étape 6 - Exécuter le code

1. **Pour exécuter le code dans un terminal**
   - Cliquer sur
   *Résultat attendu*




2. **Pour exécuter le code dans interactive**

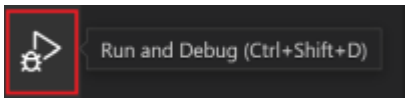
- Cliquer sur **Run Cell** dans la barre suivante Run Cell | Run Below | Debug Cell pour ouvrir la fenêtre ***Python interactive***.
  *Résultat attendu*



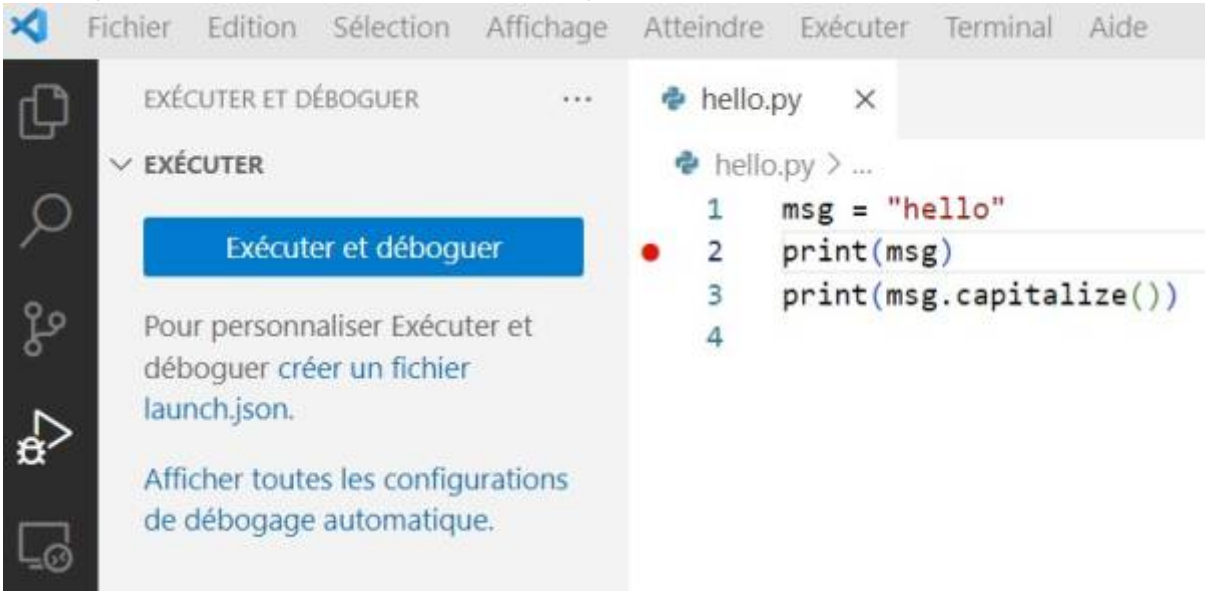

### Étape 7 - Déboguer un script Python

- **Source** : Python debugging in VS Code

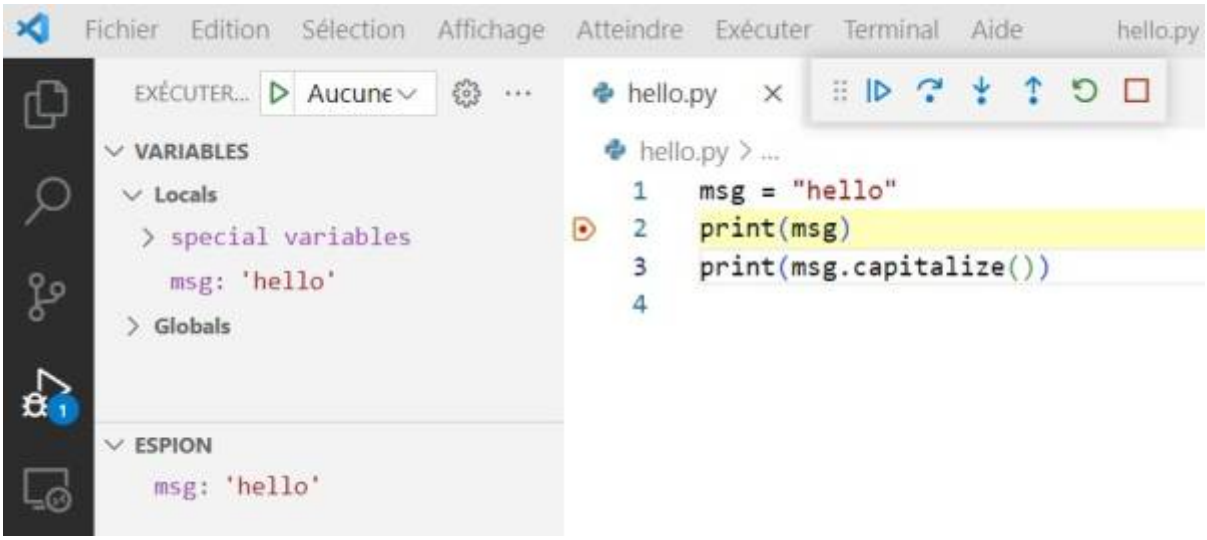
### Methode 1

- Cliquer sur 

- Placer un point d'arrêt ● avec un clic droit à gauche du texte comme ci-dessous




- Cliquer sur **Exécuter et Déboguer**.




- Description de la **barre d'outils**.

| | |
|---|---|
| | Continuer (**F5**) |
| | Pas-à-pas principal (**F10**) |
| | Pas-à-pas détaillé (**F11**) |
| | Pas-à-pas sortant (**Maj+F11**) |
| | Redémarrer (**Ctrl+Maj+F5**) |
| | Déconnecter (**Maj+F5**) |

> **Placer** la variable *msg* dans la cellule “espion” et exécuter le code en pas-à-pas pour la voir évoluer.

### Methode 2

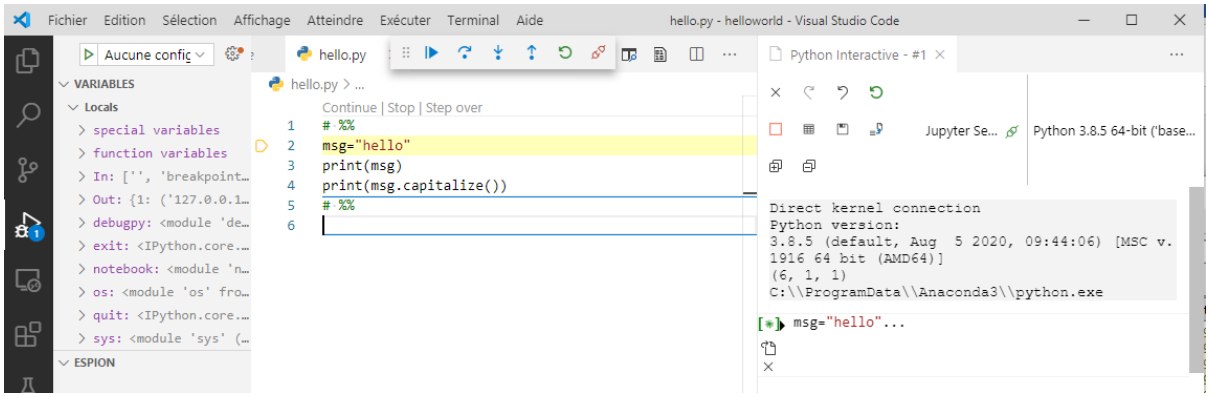
> La création d'une cellule Jupyter avec **#%%** simplifie la mise au point du programme.

- **Modifier le code**
  - Ajouter **#%%** sur la première ligne de code comme ci-dessous.

```
#%%
# msg = "hello"
print(msg)
print(msg.capitalize())
```

- **Lancer le debug**

- Cliquer sur **Debug Cell** dans Run Cell | Run Below | Debug Cell . L'éditeur apparaît comme ci-dessous.




- **Tester** comme précédemment

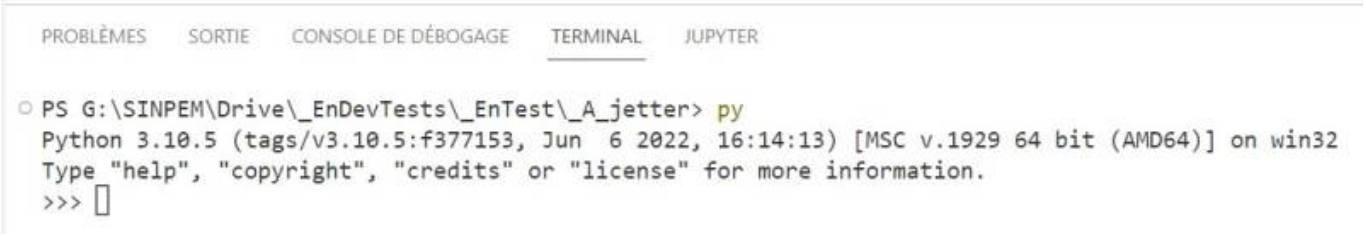
## 3. Entrée, sortie dans le terminale

- **Source** : Work with Python interpreters

Visual Studio Code comprend un terminal intégré. Il prend en charge des fonctionnalités telles que les liens et la détection d'erreurs.

> Utiliser le raccourci **Ctrl+ù** ou la commande de menu **Affichage > Terminal** pour ouvrir le terminal. Entrer la commande **py** dans le terminal pour lancer l'interpréteur Python.

*Exemple*



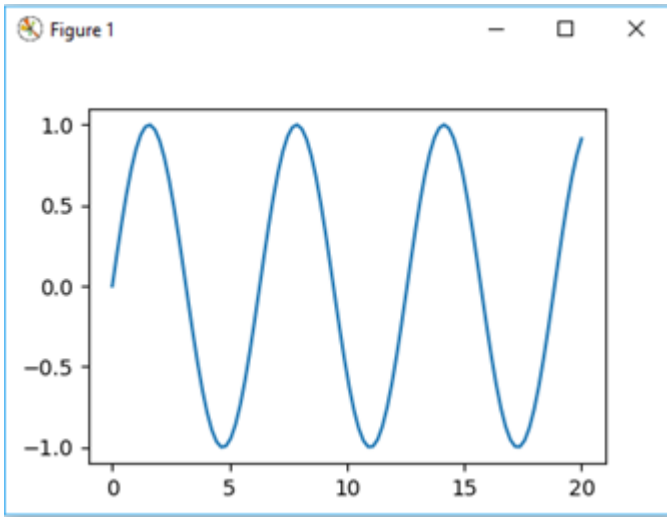

Reprendre la démarche utilisée avec “*Hello World*” pour tester le code ci-dessous dans le terminal.

exemple2.py

```
msg = "bonjour, quel est ton nom ?"
print(msg.capitalize())
nom = input()
print("Bonjour " + nom)
```

## 4. Utiliser des modules

Un module est une **liste de fonctions et de variables** contenues dans un **fichier**. Pour travailler avec les fonctionnalités du module, il suffit de l'**importer** avec le mot-clé **import**.




*Exemple* : tracé d'une courbe

sinus.py

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

x = np.linspace(0, 20, 100)
plt.plot(x, np.sin(x))
plt.show()
```

Cet exemple ne fonctionnera pas si les modules **matplotlib** et **numpy** n'ont pas été installés et un message d'erreur apparaîtra dans le terminal.
Dans VSCode, les modules s'installent en lançant **pip** dans le terminal.

*Exemple* : installation de numpy

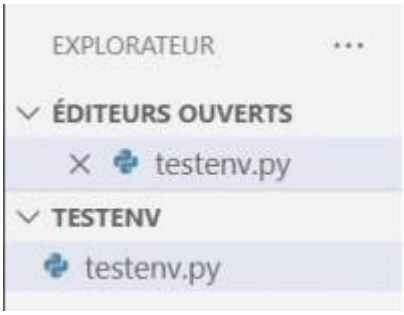
```
PROBLÈMES   SORTIE   CONSOLE DE DÉBOGAGE   TERMINAL   JUPYTER

PS G:\SINPEM\Drive\_EnDevTests\_EnTest\_A_jetter> pip install numpy
Requirement already satisfied: numpy in c:\users\phili\appdata\local
```

## Quiz

- How to Run Your Python Scripts

## Pour aller plus loin




### A. Travailler avec un environnement virtuel

- source : Python Virtual Environments: A Primer

> Le projet ci-contre a été préalablement créé dans VSCode pour illustrer les commandes **env**.

#### a) Création

Chaque fois que l'on travaille sur un projet Python qui utilise des dépendances externes installées avec **pip** , il est préférable de créer d'abord un environnement virtuel avec :

*.powershell

```
PS> python -m venv venv
```

*Exemple dans le terminal VSCode*

> L'environnement virtuel (**venv**) a été créé mais n'est pas encore activé !

**b) Activation**
Généralement, avant de commencer à l'utiliser, vous devez d'abord activer l'environnement en exécutant un script fourni avec l'installation.

*.powershell

```
PS> venv\Scripts\activate
```

*Exemple dans le terminal VSCode*

L'activation dans le projet est matérialisée par : (**venv) </callout>** c) Installation de packages **<code powershell *.powershell> (venv) PS> python -m pip install <package-name> </code>** ***Exemple dans le terminal VSCode*** d) désactivation**
Une fois que vous avez fini de travailler avec cet environnement virtuel, vous pouvez le désactiver.

*.powershell

```
(venv) PS> deactivate
PS>
```

*Exemple dans le terminal VSCode*

### B. Ressources

- **Python environments** - Contrôler quel interpréteur Python est utilisé pour l'édition et le débogage.
- **Editing code** - En savoir plus sur la saisie semi-automatique, IntelliSense, le formatage et le refactoring pour Python.
- **Linting**[2] - Activer, configurer et appliquer une variété de linters Python.
- **Debugging**[3] - Apprendre à déboguer Python à la fois localement et à distance.
- **Unit testing**[4] - Configurer des environnements de test unitaires et découvrir, exécuter et déboguer des tests.
- **Settings reference** - Explorer toute la gamme de paramètres liés à Python dans VSCode.

1)
Fonctions toujours disponibles.
2)
Le linting (“linter” son code) est une pratique qui vise à améliorer la qualité de votre code et de ce fait la reprise et la maintenabilité de celui-ci.
3)
Le débogueur vous permet d'analyser son état, de voir les valeurs des variables, d'afficher la

liste des fonctions appelées et ainsi de comprendre comment le programme en est arrivé là. [4)]

Le test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme.

From:
http://webge.fr/dokuwiki/ - **WEBGE Wikis**

Permanent link:
**http://webge.fr/dokuwiki/doku.php?id=outils:vscode:pythonpaspas&rev=1662977825**

Last update: **2022/09/12 12:17**