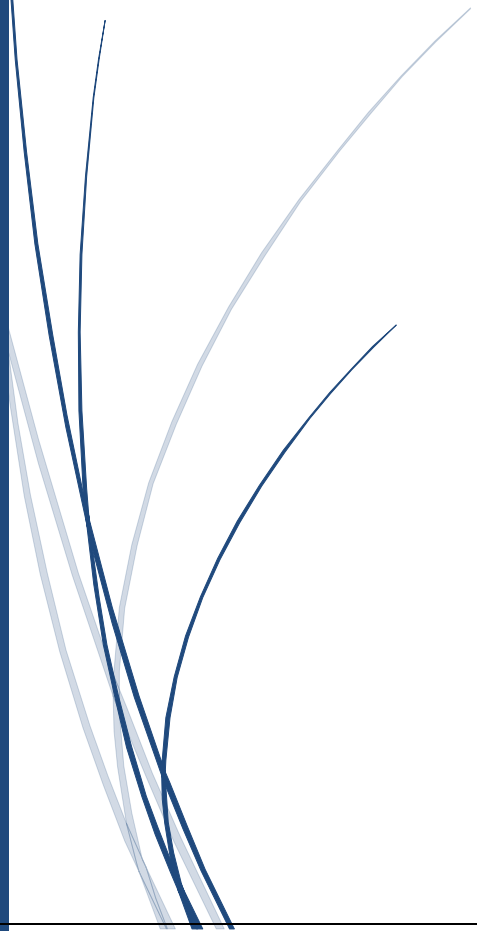
Visual Studio Code

Editeur de code cross-platform open source. Travail collaboratif, HTML, CSS, JS, PHP, Markdown, Arduino, Python, microPython etc.



P Mariano

Dernière mise à jour : le 2/3/2020

Table des matières

1 Généralités

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 que C, C++, C#, Java, Python, PHP, Go etc.) et des runtimes (.NET et Unity) .

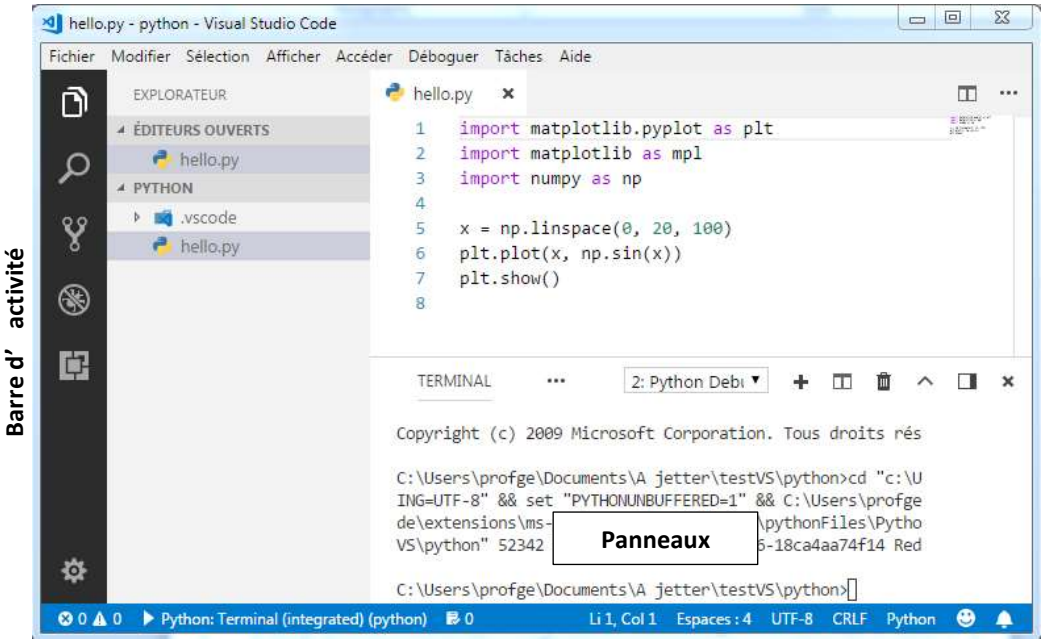
1.1 Caractéristiques

- **Complétion de code (IntelliSense**) pour CSS, HTML, JavaScript, JSON, Less, Sass, coloration syntaxique pour PHP.
- **Code action** : proposition de correction d'erreur.
- **Emmet abréviations** en HTML, Razor, CSS, Less, Sass, XML et Jade avec la touche de tabulation [Tab].
- **Débogage** dans Nodejs.

1.2 L'interface utilisateur

Comme beaucoup d'autres éditeurs de code, VS Code adopte une interface utilisateur commune et la disposition d'un explorateur sur la gauche, montrant tous les fichiers et dossiers auxquels vous avez accès, et un éditeur sur la droite, montrant le contenu des fichiers que vous avez ouvert.

_____ Éditeur _____



_____ Barre latérale _____ **_____ Barre d'état _____**

L'interface utilisateur 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 démarrez VS Code, il s'ouvre dans le même état que lors de sa dernière fermeture. Le dossier, la disposition et les fichiers ouverts sont conservés.

L'explorateur

L'explorateur est utilisé pour parcourir, ouvrir et gérer tous les fichiers et dossiers de votre projet. Après l'ouverture d'un dossier dans VS Code, le contenu du dossier est affiché dans l'Explorateur. Vous pouvez :

- **Créez, supprimez et renommez des fichiers et des dossiers**.
- **Déplacez les fichiers et dossiers par glisser-déposer**.
- **Utilisez le menu contextuel pour explorer toutes les option**s.

1.3 Réglage des paramètres de l'utilisateur et de l'espace de travail

Il est facile de configurer Visual Studio Code grâce à ses différents paramètres. Les différentes parties de l'éditeur et de l'interface utilisateur de VS Code comportent des options que vous pouvez modifier.

VS Code propose deux paramétrages :

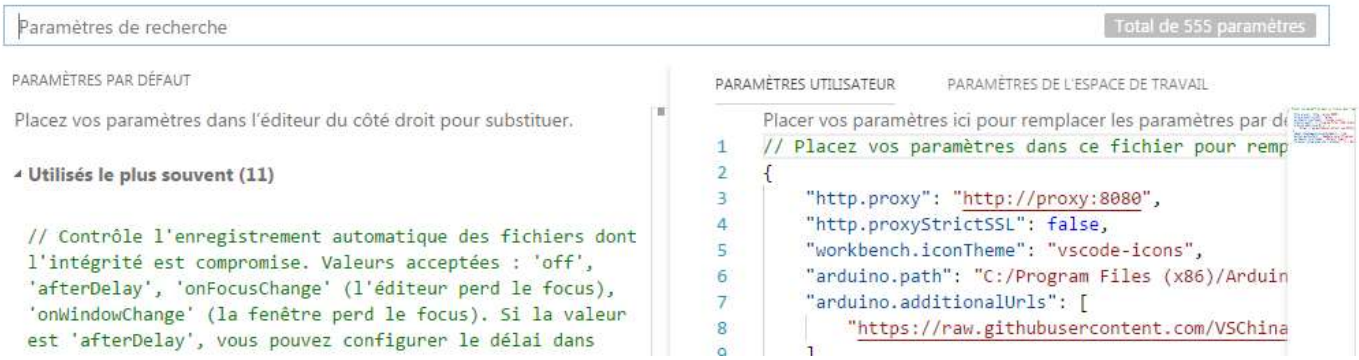
- **Paramètres utilisateur** (User Settings**)** – Les paramètres s'appliquent à toutes les instances de VS Code.
- **Paramètres de l'espace de travail** (Workspace Settings**)** – Les paramètres sont stockés dans votre espace de travail et ne s'appliquent que lorsque l'espace de travail est ouvert.

Les paramètres de l'espace de travail remplacent les paramètres utilisateur.

Pour paramétrer VS Code :
- Sélectionner : **Fichier -> Préférences -> Paramètres -> onglet** "**utilisateur** "
- Sélectionner : **Fichier -> Préférences -> Paramètres -> onglet** " **espace de travail** "

VS Code est fourni avec une liste de **paramètres par défaut**. Pour changer un paramètre, il faut le copier et le modifier dans le fichier settings.json.

Dans l'exemple ci-dessous, l'espace de travail contient des paramètres permettant de gérer un proxy, etc.



1.3.1 Modifier les paramètres

Si vous passez la souris sur un paramètre dans la zone "Paramètres par défaut" de l'éditeur de paramètres, vous verrez une action "Modifier" avec une petite icône en forme de crayon. Cliquez sur le crayon pour afficher une liste déroulante avec les valeurs de réglage disponibles. Sélectionnez une valeur et le paramètre sera ajouté au fichier de paramètres actuellement ouvert.

1.3.2 Groupes de paramètres

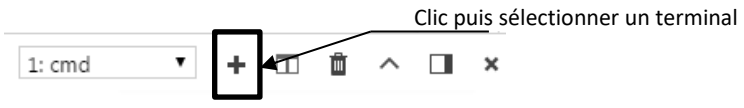
Les paramètres par défaut sont représentés dans des groupes afin que vous puissiez les parcourir facilement. Il y a un groupe couramment utilisé en haut de l'onglet.

1.3.3 Emplacements des fichiers de paramètres sur le PC

La position du fichier de paramètres dépend de la plateforme :

Windows	**% APPDATA% \ Code \ Utilisateur \ settings.json**
Linux	$HOME/.config/Code/User/settings.json

1.3.4 Ajouter un terminal

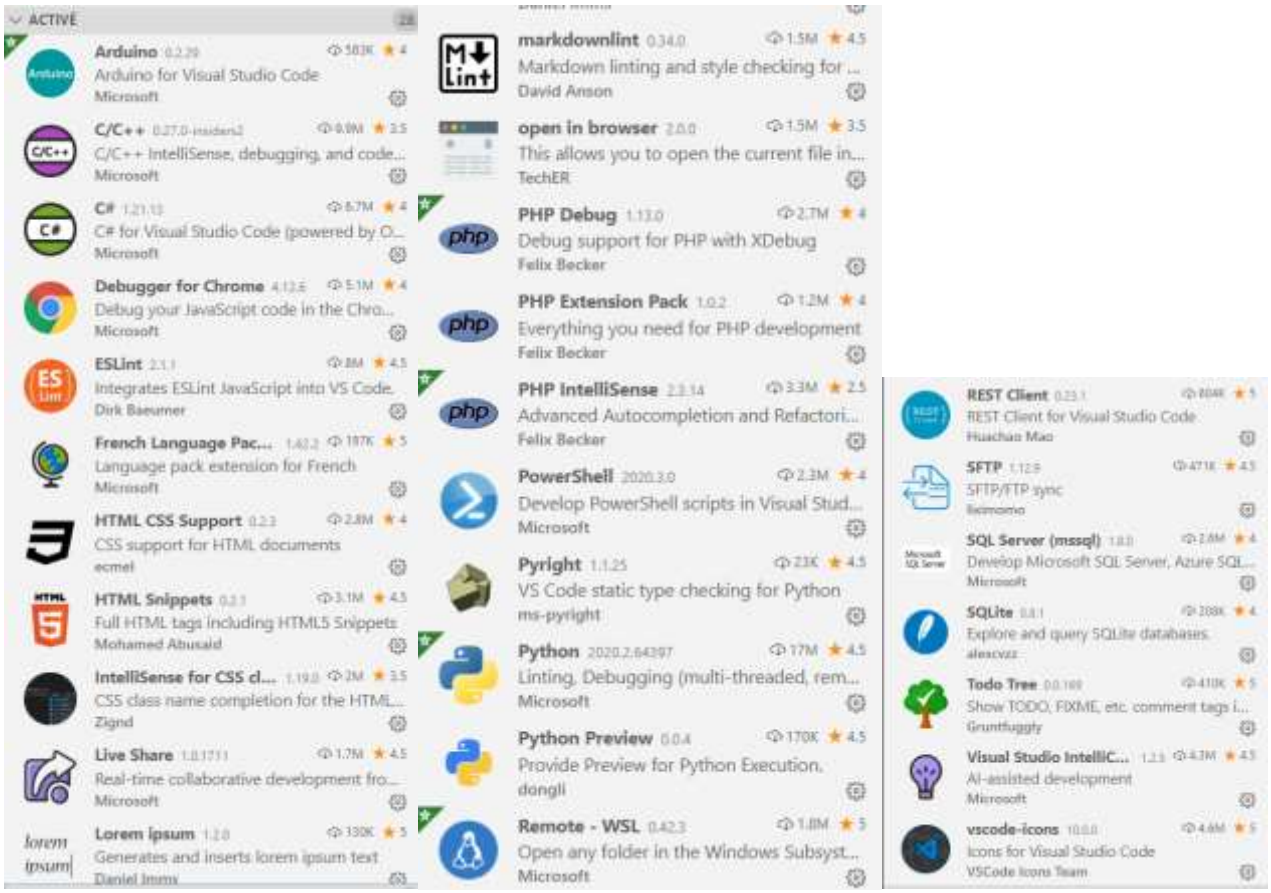


1.3.5 Paramètres de l'éditeur spécifique à la langue

[lien]

1.4 Installation des extensions

Exemples



1.4.1 Installation automatique (réglage du proxy)

➔ Fichier
  ➔ Préférence
    ➔ Paramètres utilisateur

Dans le fichier **Setting.json**, entrer :

```
// Paramètre de proxy à utiliser. S'il n'est pas défini, il est récupéré à partir des variables d'environnement http_proxy et https_proxy
  "http.proxy": "http://proxy:8080",

// Spécifie si le certificat de serveur proxy doit être vérifié par rapport à la liste des autorités de certification fournies.
  "http.proxyStrictSSL": false,
```

1.4.2 Installation manuelle si le gestionnaire n'accède pas aux ressources (pb de proxy)

- **Télécharger** l'extension à l'aide d'un navigateur après avoir configuré le code ci-dessous :

```
https://${publisher}.gallery.vsassets.io/_apis/public/gallery/publisher/${publisher}/extension/${extension name}/${version}/assetbyname/Microsoft.VisualStudio.Services.VSIXPackage
```

${publisher} : à remplacer par l'auteur de l'extension identifié dans la page de l'extension ou sur github.
${extension name} : à remplacer par le nom de l'extension.
${version} : à remplacer par la version de l'extension.

- **Renommer** le fichier téléchargé en .vsix

- **Ouvrir** le .vsix avec VSCode (-> fichier -> ouvrir le fichier)

Exemple : Téléchargement de l'extension "HTML Snippets"

La page de l'extension donne :
- son nom : ***HTML-snippets***
- sa version **: *0.0.14***
- **le nom du rédacteur :** *Mohamed Abusaid,* mais ce n'est pas le **${publisher}** qu'il faut utiliser ici. Pour le connaître, il faut se rendre sur la page github de l'extension. Le projet est dans le répertoire ***abusaidm.*** C'est ce que nous cherchons pour **${publisher}**

La ligne à entrer dans la barre d'un navigateur est :

```
https://abusaidm.gallery.vsassets.io/_apis/public/gallery/publisher/abusaidm/extension/html-snippets/0.0.14/assetbyname/Microsoft.VisualStudio.Services.VSIXPackage
```

Une fois chargé, le fichier compressé *Microsoft.VisualStudio.Services.VSIXPackage* pourra par exemple être renommé en *HTML.Snippets.**vsix*** (seule l'extension est importante) avant d'être ouvert avec VSCode.

Ceci aura pour effet d'installer l'extension. Elle sera alors visible dans le gestionnaire de paquets.  sous la forme :

2 Utilisation de l'éditeur

Quel que soit le langage utilisé, la création d'un projet suit les étapes suivantes : créer un répertoire pour le projet, placer ce répertoire dans **VS Code**, ajouter un ou plusieurs fichiers, ajouter un ou plusieurs sous-répertoires.

2.1 Créer un dossier pour le projet

Créer un dossier destiné à recevoir le projet **en dehors** de VS Code (par exemple sur le bureau). Intégrer ce dossier à VS code à partir de **Fichier -> Ouvrir le dossier** ou par un glisser-déposer.

Exemple : Projet HelloWorld

2.2 Ajouter un fichier dans le projet

Placer la souris sur le nom du projet. Les icônes suivantes apparaissent  . Cliquez sur  pour créer un fichier. Exemple : fichier hello.py

2.3 Ajouter un répertoire dans le projet

Placer la souris sur le nom du projet. Les icônes suivantes apparaissent  . Cliquez sur  pour créer un sous-répertoire. Exemple : répertoire lib.

2.4 Quelques raccourcis

- Curseur multiple (**Alt+Click**)
- Remplacement multiple (**Ctrl+F2**)
- Sélection du mot sous le curseur ou de sa prochaine occurrence (**Ctrl+D**)
- Complétion de code (**Ctrl+ESPACE**) ou "." pour la sélection, (**ENTER**) pour la validation
- Snippets utilisateur (à créer : Fichier -> préférences) et abréviations Emmet intégrés [lien] http://goo.gl/cgcijP
- Go to définition (**F12**) si supporté par le langage
- Go to symbol (**Ctrl+shift+O**)
- Survol (Pour les langages qui le supportent comme CSS, des informations complémentaires sont délivrées
- Affichage des erreurs et des avertissements (**Ctrl+Shift+M**)

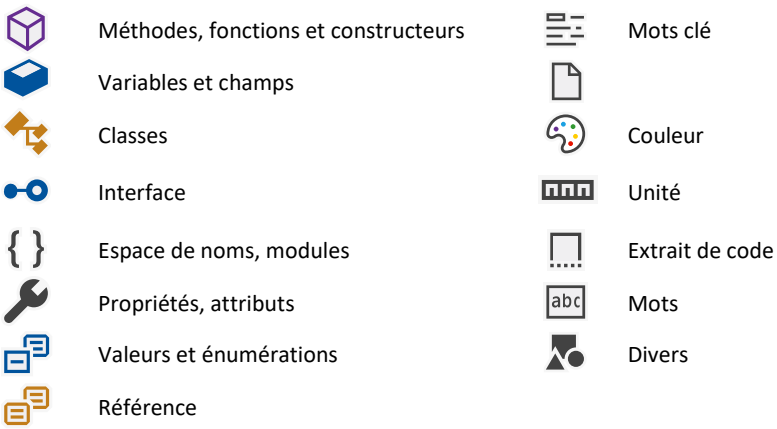
2.5 La complétion de code (Intellisense)

2.5.1 Généralités

IntelliSense est un terme général pour une variété de fonctionnalités d'édition de code, y compris : la complétion du code, les informations sur les paramètres, les brèves et les listes de membres. Les fonctionnalités d'IntelliSense sont parfois appelées "complétion de code", "aide au contenu" et "astuce de code".

Dans VSCode, IntelliSense est fourni pour JavaScript, TypeScript, JSON, HTML, CSS, Less et Sass. VS Code prend en charge les complétions basées sur des mots pour n'importe quel langage de programmation, mais peut également être enrichi en installant une extension de langue.

2.5.2 Type de complétion



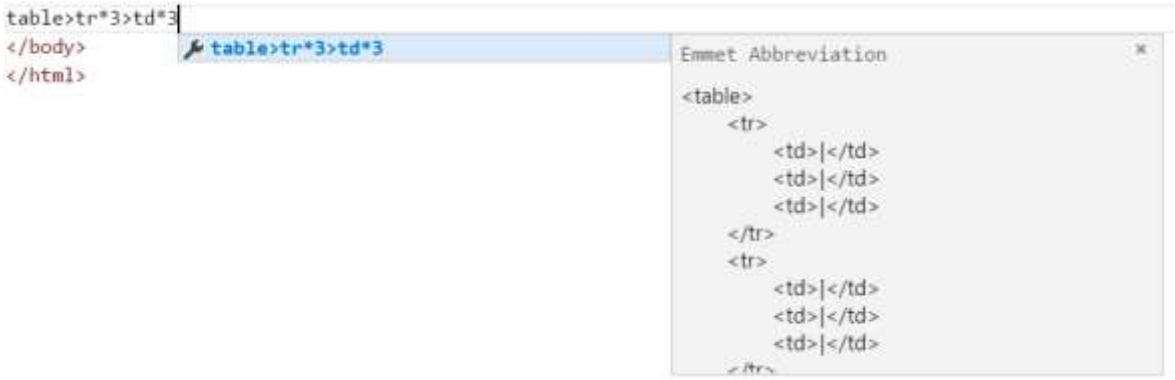
3 Emmet

Emmet est inclus dans VS Code. Aucune extension n'est nécessaire. Le résultat attendu apparaît dans une fenêtre.

3.1 Comment développer les abréviations et les extraits d'Emmet ?

Les abréviations Emmet et les extensions de snippets sont activées par défaut dans les fichiers HTML, haml, jade, slim, jsx, xml, xsl, css, scss, sass, less et stylus. Ainsi que tout langage qui en hérite comme PHP.

Exemple



Lorsque vous commencez à taper une abréviation Emmet, vous verrez l'abréviation affichée dans la liste des suggestions. Si vous avez ouvert la documentation de suggestion, vous verrez un aperçu de l'expansion au cours de la frappe. Si vous êtes dans un fichier de feuille de style, l'abréviation développée apparaît dans la liste de suggestions triée parmi les autres suggestions CSS.

3.2 Utilisation de Tab pour les extensions d'Emmet

Si vous avez désactivé le paramètre editor.quickSuggestions, vous ne verrez aucune suggestion lors de la frappe. Vous pouvez toujours déclencher des suggestions manuellement en appuyant sur Ctrl + Espace et voir l'aperçu.

Désactiver Emmet dans les suggestions #
Si vous ne souhaitez pas voir les abréviations Emmet dans les suggestions, utilisez le paramètre suivant :

```
"emmet.showExpandedAbbreviation": "never"
```

3.3 Commande de suggestion Emmet

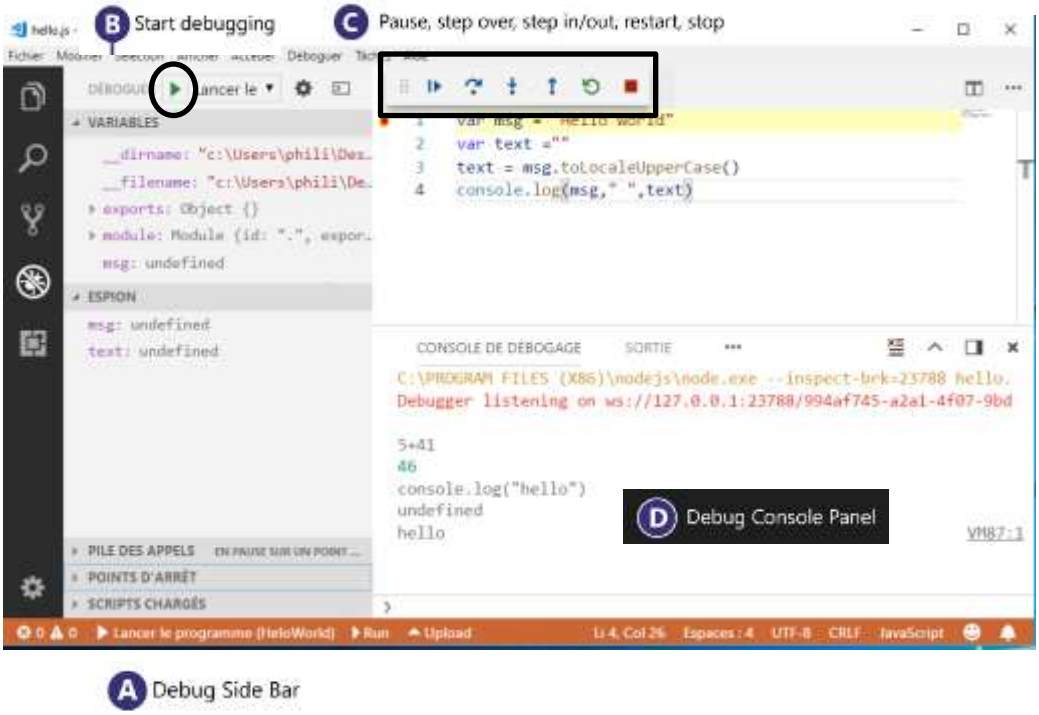
Pour vous assurer que les suggestions Emmet figurent toujours en haut de la liste des suggestions, ajoutez les paramètres suivants :

```
"emmet.showSuggestionsAsSnippets": true,
"editor.snippetSuggestions": "top"
```

Pour aller plus loin [lien]

4 Débogage

L'une des principales caractéristiques de Visual Studio Code est sa prise en charge du débogage.



VS Code dispose d'un support de débogage intégré pour l'exécution de Node.js et peut déboguer JavaScript, TypeScript et tout autre langage transféré vers JavaScript tels que PHP, Ruby, Go, C #, Python, C ++, Powershell etc.

4.1 Afficher le débogueur

Pour afficher le débogueur, cliquer sur 

4.2 Déboguer une application

1. Sélectionnez la configuration nommée "**Lancer le programme**" à l'aide de la liste déroulante. Si aucune configuration n'existe, cliquez sur  pour en créer une.
2. Appuyez sur **F5.** VS Code essayera de déboguer le fichier actif. Si le fichier n'est pas reconnu, voir le paragraphe ci-dessous.

Dès qu'une session de débogage commence, le panneau DEBUG CONSOLE est affiché et la barre d'état devient orange.

---

3. Une fois qu'une session de débogage démarre, la barre d'outils Déboguer apparaît en haut de l'éditeur.

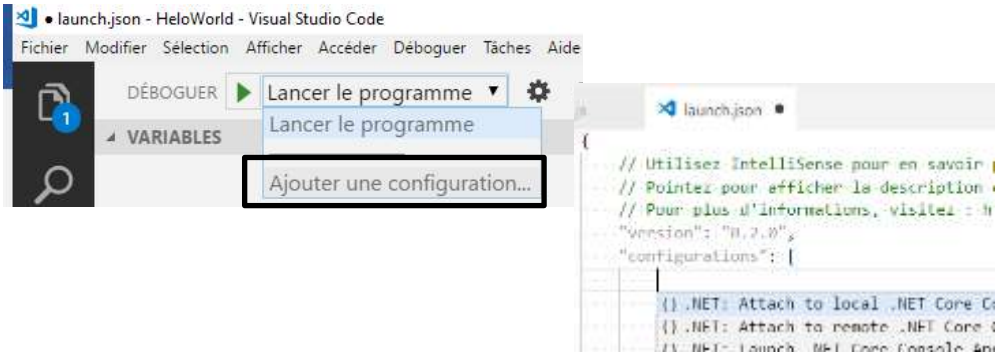
- Continue / Pause F5
- Pas à pas principal (Step Over) F10
- Pas à pas détaillé (Step Into) F11
- Pas à pas sortant (Step Out) Shift+F11
- Redémarrer (Restart) Ctrl+Shift+F5
- Arrêter (Stop) Shift+F5

4.3 Configurer le débogueur

Pour la plupart des scénarios de débogage, la création d'un fichier de configuration est bénéfique car elle vous permet de configurer et d'enregistrer les détails de configuration du débogage. VS Code conserve les informations de configuration de débogage dans un fichier **launch.json** situé dans un dossier **.vscode** dans votre espace de travail (dossier racine du projet) ou dans vos paramètres utilisateur ou paramètres d'espace de travail.

Pour créer un fichier **launch.json**, ouvrez votre dossier de projet dans VS Code (Fichier> Ouvrir un dossier), puis cliquez sur l'icône dans la barre supérieure du débogueur.

VS Code essayera de détecter automatiquement votre environnement de débogage mais s'il échoue, vous devrez le choisir manuellement :



Voici la configuration générée pour le débogage Node.js:


```
"version": "0.2.0",
    "configurations": [

        {
            "type": "node",
            "request": "launch",
            "name": "Lancer le programme",
            "program": "${workspaceFolder}/hello.js"
        },
    ]
```

Si vous revenez dans l'explorateur de fichiers (Ctrl + Maj + E), vous verrez que VS Code a créé un dossier **.vscode** et ajouté le fichier launch.json à l'espace de travail.
Les débogueurs VS Code prennent généralement en charge le lancement d'un programme en mode débogage ou la connexion à un programme déjà en cours en mode débogage.

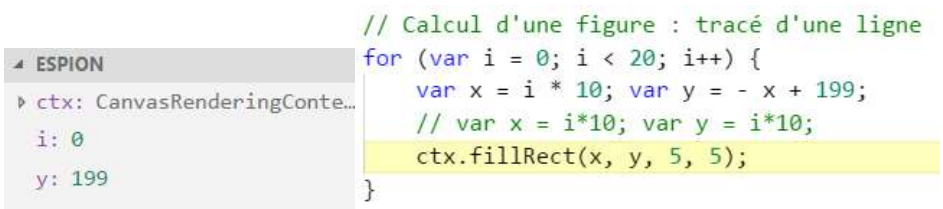
4.4 Points d'arrêt (breakpoints)

Placer un **point d'arrêt** avec un clic gauche de la souris comme dans l'exemple ci-dessous.

```
1  var canvas = document.getElementById("canvas");
2  var ctx = canvas.getContext("2d");
3  ctx.fillStyle = "Red";
4
5  // Tracé du tour du Canvas
6  ctx.strokeRect(0,0,200,200);
7
8  // Tracé d'un rectangle
9  ctx.fillStyle = "blue";
10 ctx.fillRect(10,10, 30, 10);
```

4.5 Surveiller les variables

Surveiller les variables en les ajoutant dans la zone 'ESPION'.



5 Travail collaboratif

L'extension Live Share pour VS Code permet de collaborer en temps réel !

5.1 Installation de VS Live Share

- Dans le gestionnaire d'extension, entrer *VS Live Share* et installer l'extension. Une fois l'extension installée, l'icône apparaît dans la barre d'activité et l'icône ci-dessous dans la barre latérale.

Live Share

5.2 Configurer un compte

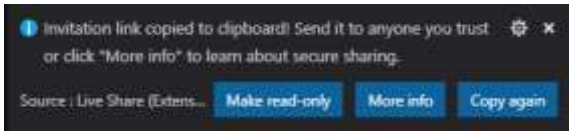
- Cliquer sur Live Share pour s'identifier. La page ci-dessous propose de se connecter.

- Après la connexion le compte de connexion apparait dans la barre latérale. Exemple : Philippe Live Share
- Ce compte peut être modifié en cliquant sur Live Share puis ***sign out***.

5.3 Démarrer une session de collaboration

- Cliquer sur
- Dans DÉTAILS DE LA SESSION cliquer sur pour démarrer la session de collaboration.

- Ok



- Un **lien de partage** a été mémorisé dans le presse-papier.

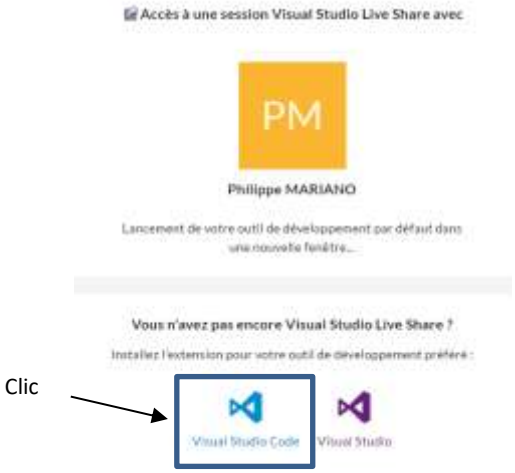
5.4 Envoyer le lien de partage

Ouvrir un **logiciel de messagerie** et transmettre le lien de partage à la personne avec laquelle on souhaite collaborer.

Exemple : https://prod.liveshare.vsengsaas.visualstudio.com/join?8DCC019FF7CCB2…

5.5 Se connecter à une session

- Ouvrir le lien reçu par mail avec un navigateur. Une page comme sur la copie d'écran ci-dessous s'ouvre. Sélectionner VS Code pour ouvrir la session de collaboration.



Clic

5.6 Fermer une session

- Cliquer sur 
- Dans 

, cliquer sur 

5.7 Partager un terminal

- Cliquer sur 
- Dans 

, cliquer sur 

6 L'extension Arduino

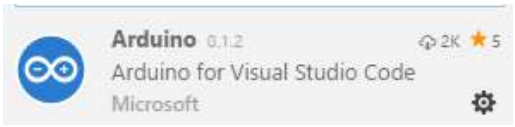
L'extension Arduino facilite le codage, la construction, le déploiement et le débogage des croquis Arduino dans Visual Studio Code. Elle dispose des fonctionnalités suivantes :

- **IntelliSense** et **coloration syntaxique**
- **Vérification** et téléchargement des croquis
- **Gestion** des bibliothèques
- **Liste d'exemples** intégrée
- **Moniteur série** intégré
- **Extraits** de croquis
- **Conception** de projet Arduino automatique
- **Intégration** de la palette de commandes (**F1**) des commandes fréquemment utilisées (par exemple, vérification, téléchargement ...)
- **Débogage** pour certaines cartes

6.1 Installer l'extension Arduino dans Visual Studio Code

Conditions préalables : Arduino IDE est requis. Les versions de l'IDE Arduino prises en charge sont la 1.6.x et ultérieures.

1. Sélectionner 

2. Installer 

et ajouter éventuellement


3. Régler le chemin vers Arduino

Fichier -> Préférences -> Paramètres

```
"arduino.path": "C:/Program Files (x86)/Arduino",
```

4. Régler le thème pour avoir une coloration syntaxique significative (Exemple : Dark+ ou Light+)

Fichier -> Préférences ->Thème de couleur ou [**Ctrl-k, Ctrl-T**]

Remarque : Paramétrage d'un éventuel proxy

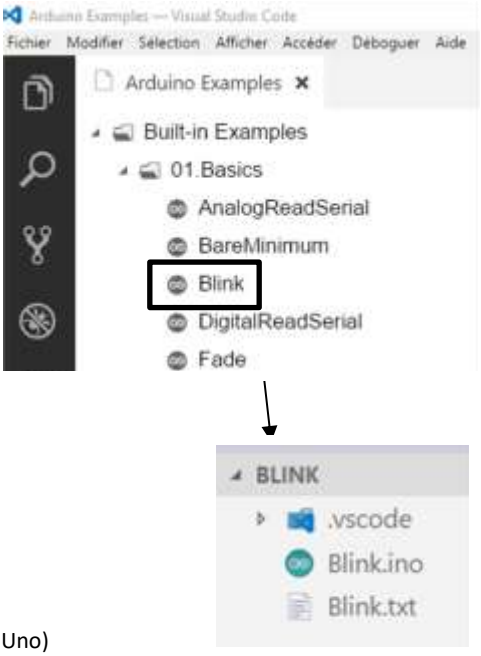
```
"http.proxy": "http://proxy:8080",
"http.proxyStrictSSL": false,
```

6.2 Liste des commandes

Cette extension fournit plusieurs commandes dans la **Palette de commandes** (**F1** ou **Ctrl + Maj + P**) pour travailler avec les fichiers * .ino :

- **Arduino: Board Manager :** Gérer les paquets. Vous pouvez ajouter une carte Arduino tierce en configurant des URL de gestionnaire de cartes supplémentaires.
- **Arduino: Change Baud Rate** : Changer le débit en baud du port série sélectionné.
- **Arduino: Change Board Type** : Changer le type de carte.
- **Arduino: Close Serial Monitor** : Fermer le port série.
- **Arduino: Examples** : Voir la liste des exemples de code.
- **Arduino: Initialize** : Construire un projet VS Code à partir d'un sketch Arduino.
- **Arduino: Library Manager** : Explorer et gérer les bibliothèques.
- **Arduino: Open Serial Monitor** : Ouvrir le moniteur série dans la fenêtre de sortie.
- **Arduino: Select Serial Port** : Changer le port série.
- **Arduino: Send Text to Serial Port** : Envoyer une ligne de texte via le port série actuel.
- **Arduino: Upload** : Télécharger un sketch sur une carte Arduino.
- **Arduino: Verify** : Construire le sketch.
- **Arduino: Board Config** : Sélection du type de carte.
- **Arduino: Select Programmer** : Sélectionner un programmateur externe.
- **Arduino: Upload Using Programmer** : Télécharger un sketch en utilisant un programmateur externe (

6.3 Utiliser un exemple



1. Lancer Visual Studio Code

Cliquer sur l'icône 

2. **Afficher** la fenêtre de sortie :

Afficher -> Sortie ou (**Ctrl-Shift-U**)

3. Choisir un **exemple** : **F1** -> **Arduino : Examples**

Exemples (**Blink**)

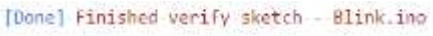
4. Configurer le matériel (choix de la carte et du port Comx)

Avant : 

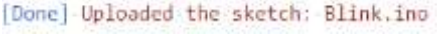
- Cliquez sur <Select Board Type> pour choisir la carte (Exemple : Arduino/Genuino Uno)
- Cliquez sur <Select Serial Port> pour changer le port COM

Après : 

5. **Vérifier** le programme : F1 -> Arduino : Verify ou (**Ctrl - Alt - R**)

Code dans la fenêtre de sortie : 

6. **Transférer** le programme : F1 -> Arduino : Upload ou (**Ctrl - Alt - U**)

Code dans la fenêtre de sortie : 

<u>Remarque</u> : Le projet généré à partir d'un exemple est automatiquement sauvegardé dans Documents/Arduino/generated_examples

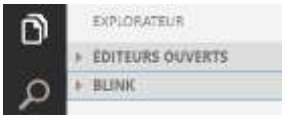
---

6.4 Créer un projet

Un sketch Arduino doit être contenu dans un projet Visual Studio Code. Un projet doit être contenu dans un répertoire.

1. Créer un répertoire

<u>Exemple</u> : Blink



2. **Lancer** Visual Studio Code puis **ouvrir** le répertoire avec Visual Studio Code

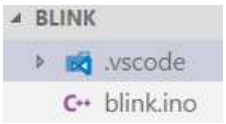
Fichier -> Ouvrir le dossier ou (**Ctrl-k, Ctrl-O**) ou par un glisser-déplacer.

<u>Exemple</u> : 

3. **Créer** un fichier .ino (ceci entraînera la création d'un sous-répertoire .vscode) .

Clic sur 

<u>Exemple</u> : blink.ino



4. Configurer le matériel (choix de la carte et du port Comx)

Ceci entraînera la création d'un répertoire avec l'extension *.vscode*

Avant : 

Après : 

6.5 Faire référence à des bibliothèques dans le code du projet

Voir l'annexe " **Inclure des bibliothèques à un projet**" puis utiliser #include suivi du nom de fichier avec l'extension .h

#include <*nombiblio.h*> (chemin connu de l'éditeur)
#include "*nombiblio.h*" (local au projet)

6.6 Écrire le code du programme avec des extraits de code (snippets)

Les extraits de code permettent de créer les champs **setup** et **loop** Arduino et des **gabarits de code C++**. On peut les obtenir de la façon suivante :
- F1 -> Insert Snippet et sélectionner l'extrait de code dans la liste ou
- F1 -> Insérer un extrait de code ou
- En entrant le début du **mot-clé** et en sélectionnant l'icône puis touche **<TAB>**

Exemple : génération des champs setup / loop arduino
Entrer le mot-clé **asl** dans l'éditeur.

Résultat :

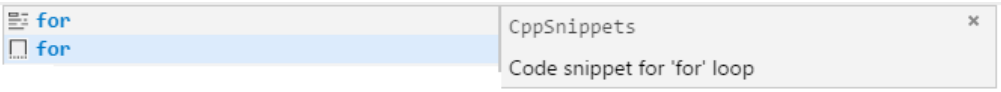
```
void setup()
{

}

void loop()
{

}
```

Exemple : génération d'un gabarit pour la boucle for
Entrer le mot-clé **asl** dans l'éditeur.



Résultat :

```
for (size_t i = 0; i < length; i++)
{

}
```

6.7 Tester le programme

F1 -> Arduino : Verify ou **Ctrl-Alt-R**

6.8 Télécharger le programme dans la carte

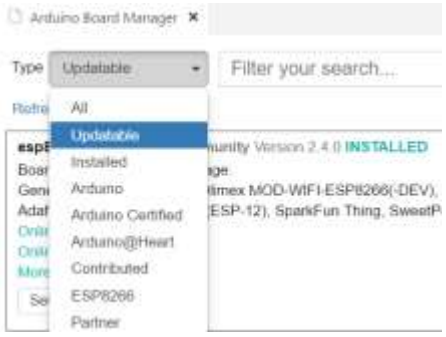
F1 -> Arduino : Upload ou **Ctrl-Alt-U**

Pour plus d'informations, voir le tutoriel "**Prise en main de l'IDE VSCode**" sur le Wiki Arduino http://bit.ly/2MJpXFk

6.9 Annexes Arduino

6.9.1 Gestionnaire de cartes Arduino (Arduino Board Manager)

Ce gestionnaire permet d'ajouter une carte Arduino tierce en configurant des URL de gestionnaire de cartes supplémentaires. L'installation d'une nouvelle carte installe un lien dans les paramètres utilisateurs comme expliqué dans le paragraphe suivant.



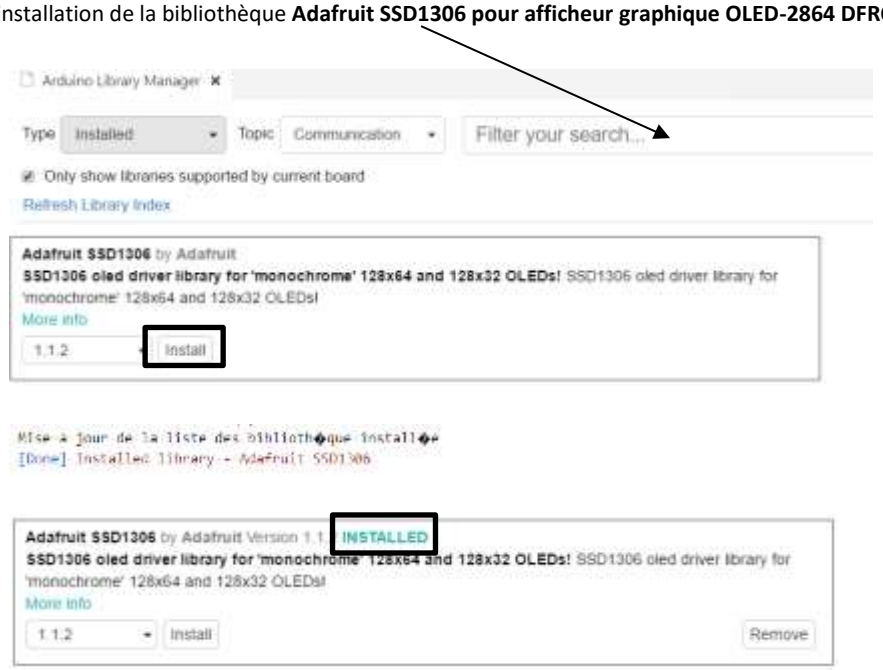
- F1 -> Arduino -> Board Manager
  - Sélectionner "Updatable" pour mettre les paquets installés à jour
  - Sélectionner "Installed" pour lister les cartes installées
  - Sélectionner Arduino etc. pour lister les cartes pouvant être installées.

Remarque : L'installation d'un paquet avec le gestionnaire ajoute un chemin à la propriété arduino.additionalUrls dans les paramètres utilisateur. Cette opération peut également être réalisée manuellement comme expliquée ci-dessous.
Fichier -> Préférences -> Paramètres puis ajouter "arduino.additionalUrls": "", à la liste des paramètres utilisateur.
Exemple : `"arduino.additionalUrls":"https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json,http://arduino.esp8266.com/stable/package_esp8266com_index.json",`

6.9.2 Installer des bibliothèques sur le PC (Library Manager)

F1 -> Arduino : Library Manager

Exemple : installation de la bibliothèque **Adafruit SSD1306 pour afficheur graphique OLED-2864 DFROBOT**



6.9.3 Inclure des bibliothèques à un projet

F1 -> Arduino : Library Manager
Après avoir été installée, la bibliothèque peut être incluse dans un projet

Exemple :

Le chemin vers la bibliothèque est automatiquement ajouté dans le fichier c_cpp_properties.json

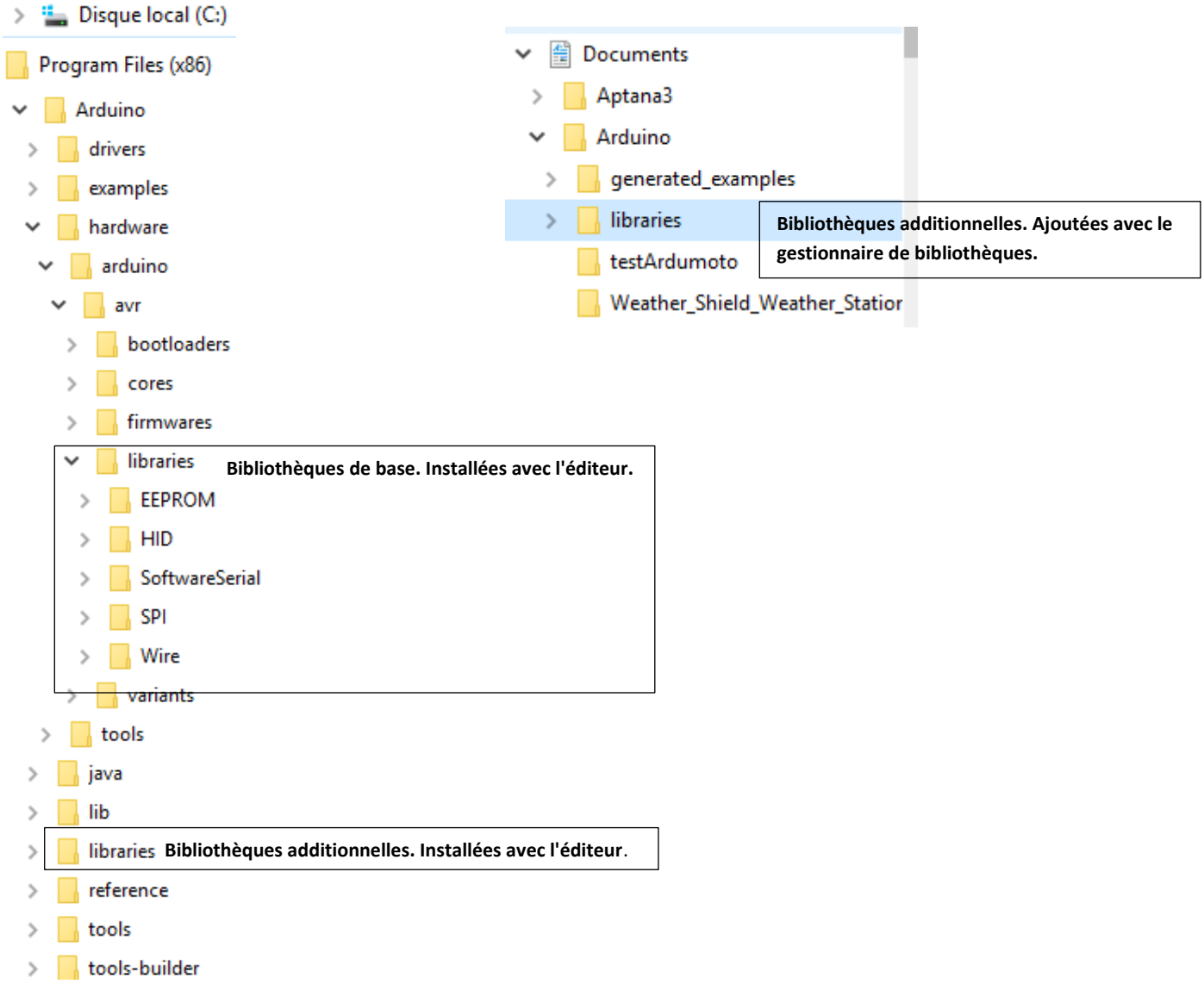
6.9.4 Le fichier de configuration du projet c_cpp_properties.json


```
{
  "configurations": [
    {
      "name": "Win32",
      "includePath": [
        "C:\\Program Files (x86)\\Arduino\\hardware\\arduino\\avr\\cores\\arduino",
        "C:\\Users\\phili\\Documents\\Arduino\\libraries\\Adafruit_SSD1306",
        "C:\\Program Files (x86)\\Arduino\\hardware\\arduino\\avr\\libraries"
      ],
      "browse": {
        "limitSymbolsToIncludedHeaders": false,
        "path": [
        "C:\\Program Files (x86)\\Arduino\\hardware\\arduino\\avr\\cores\\arduino",
        "C:\\Users\\phili\\Documents\\Arduino\\libraries\\Adafruit_SSD1306",
        "C:\\Program Files (x86)\\Arduino\\hardware\\arduino\\avr\\libraries"
        ]
      },
      "intelliSenseMode": "msvc-x64"
    }
  ],
  "version": 3
}
```

Chemin vers la bibliothèque Adafruit ajoutée avec le **gestionnaire de bibliothèques**

Chemin vers la bibliothèque **Wire.h ajouté manuellement !**

6.10 Organisation des répertoires Arduino



Bibliothèques de base. Installées avec l'éditeur.

Bibliothèques additionnelles. Ajoutées avec le gestionnaire de bibliothèques.

Bibliothèques additionnelles. Installées avec l'éditeur.

6.11 Problèmes rencontrés

- Bibliothèque non reconnue dans le code

  Exemple : `#include <Wire.h>`

  Solution : Entrer le chemin vers la bibliothèque manuellement dans le fichier c_cpp_properties.json du projet comme expliqué dans le paragraphe " Le fichier de configuration du projet".

7 L'extension Pymakr

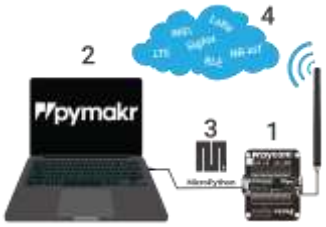
Pymakr est un environnement de développement, s'intégrant à VS Code, qui assure la communication avec une carte **Pycom**. Vous pourrez exécuter un fichier sur la carte, synchroniser l'ensemble du projet ou taper et exécutez directement des commandes en utilisant la ligne de commande **REPL**.

Pymakr fonctionne avec Mac OSX, Linux et **Windows**. Il se connecte à n'importe quelle carte Pycom : WiPy, WiPy2.0, LoPy et toute carte plus récente. Il fonctionne mieux avec **un firmware 1.6.11.b1** ou supérieur. Un microprogramme antérieur peut avoir un comportement inattendu lors de la synchronisation des fichiers.

7.1 Les cartes Pycom pour l'internet des objets (IoT)

7.1.1 Généralités

Source : https://docs.pycom.io/chapter/gettingstarted/



Les modules Pycom actuels (2018) sont organisés autour d'un ESP32, offrant :
- **512 Kb** available for the user as internal storage, (external SD card support available)
- Up to **96 Kb of RAM** available **for python code**.
- Hardware floating point unit
- Up to **24 GPIO** Pins

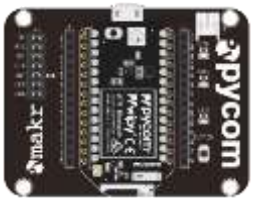
- **2x UARTs**
- **SPIs**
- **Timers**
- **RTC**
- **PWM**
- **ADC**
- **DAC**
- **I2C**
- **SD**

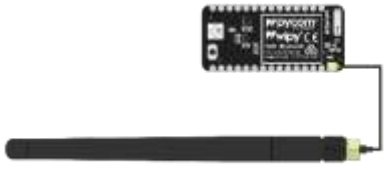
- **Bluetooth**
- LoRa (only available in the LoPy)
- **hashlib** MD5, SHA1, SHA256, SHA384 and SHA512 hash algorithms
- **AES** encryption
- **SSL/TLS** support

7.1.2 Connexion du module à une carte d'extension

Source : https://docs.pycom.io/chapter/gettingstarted/connection/wipy.html

- Recherchez **le bouton de réinitialisation** sur le module (situé dans un coin de la carte, à côté de la **LED**).
- Localisez le connecteur USB sur la carte d'extension.
- Insérez le module WiPy sur la carte d'extension avec le bouton de réinitialisation pointant vers le connecteur USB. Il doit s'enclencher fermement et les broches ne doivent plus être visibles.





Antennes WiFi / Bluetooth (en option)
Tous les modules pycom, y compris le WiPy, sont équipés d'une antenne WiFi intégrée ainsi que d'un connecteur **U.FL** pour une antenne externe. L'antenne externe est facultative et n'est nécessaire que si vous avez besoin de meilleures performances ou si vous installez le WiPy de manière à bloquer le signal WiFi. La commutation entre les antennes se fait via un logiciel, les instructions pour cela peuvent être trouvées ici.

7.1.3 Configuration de l'ordinateur

- **Drivers**
  Les modules sont automatiquement reconnus sous Windows 8/10/+. Voir le lien ci-dessous pour Windows 7
  https://docs.pycom.io/chapter/gettingstarted/installation/drivers.html

- **Utilitaire de mise à jour du firmware (exemple pour la carte WiPy)**
  Cet outil automatise le processus de mise à niveau du microprogramme de votre périphérique Pycom. Il est important que vous l'utilisiez avant de mettre en œuvre le module. Non seulement pour vous assurer d'avoir le micrologiciel le plus stable et le plus complet, mais aussi pour vous assurer que toutes les fonctionnalités de votre appareil sont activées.
  Par exemple : cet outil active également votre connectivité sigfox gratuite pendant deux ans.

"Nous vous recommandons vivement de mettre à jour votre microprogramme vers la dernière version, car nous améliorons constamment et ajoutons de nouvelles fonctionnalités aux périphériques." Source Pycom.
https://docs.pycom.io/chapter/gettingstarted/installation/firmwaretool.html

---

AVERTISSEMENT

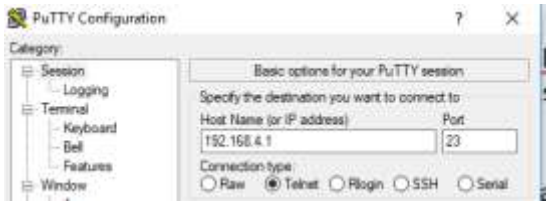
Assurez-vous que le cavalier **TX** est présent sur la carte d'extension. Sans ce cavalier, la mise à jour échouera.

- **Test de connexion de la carte au PC en Wifi avec Telnet (utiliser PuTTY sous Windows)**

*"Une fois que le microprogramme a été mis à jour, faire un reset (appuyez sur le bouton à côté de la LED) pour que la carte se positionne en mode diffusion WiFi (**point d'accès**). Cela vous permet de vous connecter au serveur interne du périphérique pour télécharger des fichiers / scripts et modifier les paramètres de configuration. **Une alimentation peut être nécessaire** !*

*Ouvrez les paramètres réseau sur votre ordinateur. Vous verrez apparaître un SSID avec un nom semblable à **lopy-wlan-xxxx**, **wipy-wlan-xxxx**, etc.*

*Connectez-vous au réseau et entrez le mot de passe **www.pycom.io** .*
Configurez Putty comme ci-dessous



*Puis cliquez sur le bouton **Open.** Une **console** s'ouvre comme ci-dessous.*

Le nom d'utilisateur (login) et le mot de passe (password) par défaut sont respectivement **micro** et **python**.

Note

Lorsque le module est connecté au réseau de la carte et à moins que vous ne disposiez d'une deuxième carte réseau, vous ne pourrez pas accéder à Internet depuis votre ordinateur !

7.2 Programmer les cartes Pycom en MicroPython avec Pymakr

7.2.1 Installer l'extension Pymakr dans VSCode

1. Sélectionner 

2. Installer 

Le terminal s'ouvrira après l'installation du paquet.

7.2.2 Première connexion au module

Le module peut être connecté au PC avec un câble USB ou en Wifi.

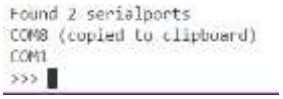
- **Connexion par un port série USB**
  Après l'installation de Pymakr, vous devez prendre quelques secondes pour le configurer lors de la première utilisation.

1. Connectez le module Pycom à l'ordinateur avec un câble USB. Si vous utilisez une carte d'extension et que vous venez d'effectuer une mise à niveau du microprogramme, veillez à retirer le câble entre GND et G23 et réinitialisez votre appareil en appuyant sur le bouton RESET.

2. Ouvrez Visual Studio Code et assurez-vous que le plug-in Pymakr est correctement installé.

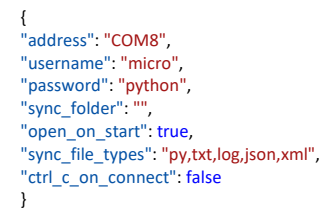
3. Cliquez sur ***All Commands***

4. Dans la liste, sélectionnez **Pymakr > Extra > List Serial Ports**



5. Cela liste les ports série disponibles. Si Pymakr est capable de détecter automatiquement ce qui doit être utilisé, il sera copié dans votre presse-papier. Si ce n'est pas le cas, copiez manuellement le bon port série.

6. Cliquez à nouveau sur ***All Commands***, puis sur **Pymakr > Global Settings**. Cela va ouvrir un fichier JSON. Collez l'adresse du port série que vous avez copiée précédemment dans le champ "adresse" et enregistrez le fichier.



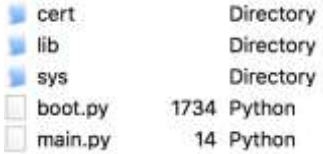
7. Pour finir, fermez le fichier JSON, cliquez sur ***All Commands***, puis sur Pymakr > Connect pour connecter votre appareil. La console Pymakr devrait montrer trois flèches '>>>', indiquant que vous êtes connecté !

- **Connexion en wifi via Telnet**
  - Assurez-vous que le périphérique Pycom est activé.
  - Connectez l'ordinateur hôte au point d'accès WiFi nommé d'après votre carte (le **SSID** sera comme suit, par exemple **lopy-wlan-xxxx, wipy-wlan-xxxx**, etc.). Le mot de passe est **www.pycom.io.**
  - Suivez les étapes décrites ci-dessus dans la section "Connexion via USB série", mais entrez **192.168.4.1** comme adresse.
  - Le nom d'utilisateur et le mot de passe par défaut sont respectivement **micro** et **python**.
  - Enfin fermez le fichier JSON, cliquez sur All commands, puis Pymakr> Connect, Pymakr se connectera maintenant via Telnet.

7.2.3 Introduction à MicroPython

Les modules fonctionnent avec **MicroPython**; une implémentation de **Python 3.5** optimisée pour fonctionner sur des microcontrôleurs.

Lors du démarrage, deux fichiers sont exécutés automatiquement : d'abord **boot.py** puis **main.py**. Ils sont placés dans le dossier **/flash** de la carte. Tous les autres fichiers ou bibliothèques peuvent également être placés ici, et peuvent être inclus ou utilisés depuis boot.py ou main.py.
La structure des dossiers dans /flash ressemble à l'image ci-dessous. Les fichiers peuvent être gérés soit en utilisant **FTP** ou en utilisant le plug-in **Pymakr**.



7.2.4 Exemples de code écrits en MicroPython

- **Affectation d'une variable**

```
variable = "Hello World"
print(variable)
```

- **Expressions conditionnelles**

```
temperature = 15
target = 10
if temperature > target:
    print("Too High!")
elif temperature < target:
    print("Too Low!")
else:
    print("Just right!")
```

- **Boucles (for et while)**

Les boucles sont une autre caractéristique importante de tout langage de programmation. Cela vous permet de faire défiler votre code et de répéter les fonctions / affectations / etc.

Les boucles **for** permettent de contrôler le nombre de fois qu'un bloc de code s'exécute dans une plage.

```
x = 0
for y in range(0,9):
    x += 1
print(x)
```

Les boucles **while** sont similaires aux boucles for, elles vous permettent cependant d'exécuter une boucle jusqu'à ce qu'une condition soit vrai / faux. Dans le cas ci-dessous, la boucle se répète si x est inférieur à 9.

```
x = 0
while x < 9:
    x += 1
print(x)
```

- **Les fonctions**

Les fonctions sont des blocs de code référencés par leur nom. Les données peuvent y être passées pour être exploitées (c'est-à-dire, les paramètres) et peuvent éventuellement retourner des données (la valeur de retour). Toutes les données transmises à une fonction sont explicitement transmises.

La fonction ci-dessous prend deux nombres et les ajoute ensemble, produisant le résultat.

```
def add(number1, number2):
  return number1 + number2 # Déclaration

add(1,2) # Appel
```

La fonction suivante prend un nom en entrée et renvoie une chaîne contenant une phrase de bienvenue.

```
def welcome(name):
  welcome_phrase = "Hello, " + name + "!"
  print(welcome_phrase)

welcome("Alex") # expect "Hello, Alex!"
```

- **Structures de données**
  - **Listes**

    Structure de données contenant une collection ordonnée (séquence) d'éléments.

    ```
    networks = ['lora', 'sigfox', 'wifi', 'bluetooth', 'lte-m']
    print(networks[2]) # expect 'wifi'
    ```

  - **Dictionnaires**

    Un dictionnaire est comme un carnet d'adresses où vous pouvez trouver l'adresse ou les coordonnées d'une personne en ne connaissant que son nom, c'est-à-dire que les **clés** (noms) sont associées aux **valeurs** (détails).

    ```
    address_book = {'Alex':'2604 Crosswind Drive','Joe':'1301 Hillview Drive','Chris':'3236 Goldleaf Lane'}
    print(address_book['Alex']) # expect '2604 Crosswind Drive'
    ```

  - **Tuple**

    Similaire aux listes, mais immuable, c'est-à-dire que vous ne pouvez pas modifier les tuples après l'instanciation.

    ```
    pycom_devices = ('wipy', 'lopy', 'sipy', 'gpy', 'fipy')
    print(pycom_devices[0]) # expect 'wipy'
    ```

7.2.5 Premier projet avec Pymakr

- **Créer un projet dans Pymakr**

1. Créez un nouveau répertoire vide sur votre ordinateur. Pour cet exemple, nous allons en créer un appelé **RGB-Blink**.

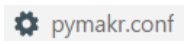
2. Ouvrir **Visual Studio Code** et charger le répertoire en cliquant sur **Fichier> Ouvrir** ou par un glisser-déplacer.

3. Maintenant que vous avez créé un projet, nous devons y ajouter des fichiers. Un projet MicroPython standard a la structure suivante :

```
RGB-Blink
|-lib
    |- some_library.py
|-boot.py
|-main.py
```

   - **boot.py** est le premier script exécuté sur le module. Il est souvent utilisé pour le connecter à un réseau WiFi afin que l'on puisse utiliser **Telnet** et **FTP** sans se connecter au point d'accès Wifi (WiFi AP) ni encombrer le fichier main.py. Si vous débutez, vous n'avez pas besoin d'utiliser un fichier boot.py.
   - **main.py** s'exécute directement après boot.py et devrait contenir le code principal que vous souhaitez exécuter sur votre appareil.
   - Le répertoire **lib.** C'est souvent une bonne idée de séparer le code réutilisable en bibliothèques. Si vous voulez créer ou utiliser des bibliothèques créées par d'autres, vous devrez créer un **répertoire lib** pour les placer. Il est important de placer les fichiers .py directement dans la bibliothèque plutôt que de créer une arborescence de répertoires. **Par défaut, MicroPython ne détectera aucune bibliothèque dans les sous-répertoires**.

Maintenant que la structure du projet est configurée, vous pouvez définir les paramètres spécifiques au projet dans Pymakr, par exemple le port série à utiliser.
Cliquez sur le bouton ***All commands*** dans **Visual Studio Code**, puis sélectionner **F1** suivi de **Pymakr> Project Settings.**

Cela crée un fichier appelé 

dans votre projet et le remplit avec les paramètres par défaut copiés à partir de vos paramètres globaux. Une explication détaillée de ces paramètres peut être trouvée ici.

- **Exemple : contrôler la LED embarquée**

Maintenant que vous avez configuré votre projet, nous pouvons passer à la programmation du module. La première chose à faire est d'importer quelques bibliothèques afin d'interagir avec la LED embarquée. Le firmware Pycom est livré avec une grande quantité de bibliothèques pour les fonctionnalités standard intégrées. Vous pouvez en savoir plus à ce sujet dans la documentation de l'API. Pour cet exemple, vous devrez ouvrir le fichier main.py et ajouter le code suivant :

```
import pycom
import time
```

Cela va importer deux bibliothèques, **Pycom** qui est responsable des fonctions spécifiques de Pycom, telles que la LED embarquée et **time** qui est une bibliothèque standard utilisée pour la gestion du temps.

Vous avez peut-être remarqué que lorsque vous allumez votre module, la LED clignote régulièrement en bleu. Cette "pulsation" permet de savoir que le module a démarré correctement. Avant de pouvoir changer la couleur de cette LED, nous devons désactiver ce battement. Ajoutez le code suivant après les bibliothèques :

```
pycom.heartbeat(False)
```

Il est maintenant temps de tester votre code. Sur le panneau Pymakr en bas de la fenêtre, vous verrez un bouton d'exécution 

. (Si vous n'êtes pas encore connecté à votre appareil, vous devrez d'abord le faire). Lorsque vous cliquez sur le bouton Run, le code du fichier actuellement ouvert est exécuté sur l'appareil. Le bouton *Run* est brièvement remplacé par un bouton *Stop*. Après avoir exécuté ce code, vous devriez voir que la LED embarquée arrête de clignoter en bleu.

Maintenant que nous pouvons confirmer que le périphérique est connecté et que Pymakr peut exécuter du code, nous pouvons compléter notre script pour faire clignoter la LED comme ceci :

```
import pycom
import time

pycom.heartbeat(False)

while True:
    pycom.rgbled(0xFF0000)  # Red
    time.sleep(1)
    pycom.rgbled(0x00FF00)  # Green
    time.sleep(1)
    pycom.rgbled(0x0000FF)  # Blue
    time.sleep(1)
```

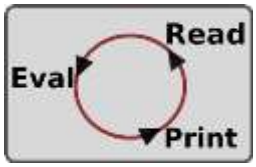
Remarque : comme le code du script exécute une boucle while, une fois lancé, il s'exécute indéfiniment. Vous remarquerez que cela vous empêche d'accéder au REPL interactif sur le module (vous ne pouvez pas voir l'invite >>>). Pour arrêter le script, cliquez sur le terminal Pymakr et appuyez sur ctrl-c sur votre clavier ou cliquez sur le bouton ■ Stop qui a remplacé le bouton RUN.

- **Téléchargement du code dans le module**
  Dans le paragraphe précédent, nous avons fait fonctionner le code sur le module Pycom en utilisant la fonction d'exécution de Pymakr. Ceci est utile pour un test rapide, mais a quelques inconvénients. Premièrement, le code ne reste pas sur l'appareil de façon permanente, si vous le redémarrez, il ne l'exécutera plus. Deuxièmement, cela ne fonctionnera que si vous utilisez des bibliothèques intégrées au firmware. Si vous avez besoin de bibliothèques supplémentaires, celles-ci doivent d'abord être copiées sur l'appareil. C'est là qu'intervient la fonction de téléchargement. Si au lieu de lancer l'envoi, vous cliquez sur télécharger ▼ Download, Pymakr téléchargera tous les fichiers dans le projet (à condition que leur type soit dans la liste du paramètre **sync_file_types** du fichier **pymakr.conf**). Ceux-ci persistent alors sur votre appareil, même entre les redémarrages, et vous permet d'utiliser les bibliothèques du dossier lib dans votre projet.

  Si vous avez besoin de supprimer des fichiers de votre appareil, vous avez deux options, soit se connecter via FTP et gérer vos fichiers de cette façon ou formater le ***flash interal*** de l'appareil comme suit :

  ```
  import os
  os.mkfs('/flash')
  ```

7.2.6 Interagir avec le module via REPL (Read Evaluate Print Loop)



REPL signifie **Read Evaluate Print Loop. C'est** le nom donné à **l'invite interactive** MicroPython avec laquelle vous pouvez accéder à un module xPy. L'utilisation de REPL est de loin la manière la plus simple de **tester votre code** python et d'exécuter des commandes.
Vous pouvez utiliser REPL en plus d'écrire des scripts dans main.py.

REPL inclut les fonctionnalités suivantes :
- **Historique des entrées** : utilisez les flèches vers le haut et vers le bas pour faire défiler l'historique
- **Tab completion** : appuyez sur la touche Tab pour compléter automatiquement les variables ou les noms de module
- Arrêtez du code en cours d'exécution avec **Ctrl-C**
- Copier / coller le code : **Ctrl-C** et **Ctrl-V**

Il existe un certain nombre de raccourcis utiles pour interagir avec MicroPython REPL.

- **Ctrl-A** sur une ligne vide entrera en mode REPL brut. Ceci est similaire au mode collage permanent, sauf que les caractères ne sont pas renvoyés.
- **Ctrl-B** sur un espace passe au mode REPL normal.
- **Ctrl-C** annule toute entrée ou interrompt le code en cours d'exécution.
- **Ctrl-D** sur une ligne vide fera une réinitialisation logicielle.
- **Ctrl-E** entre en 'mode collage' qui vous permet de copier et coller des blocs de texte. Quittez ce mode en utilisant Ctrl-D.
- **Ctrl-F** effectue un "démarrage sécurisé" de l'appareil qui empêche boot.py et main.py de s'exécuter

Pour utiliser REPL, vous devez vous connecter au xPy dans le **terminal de VSCode**, via Telnet (sinon avec PuTTY sous Windows), ou avec un convertisseur USB vers série câblé sur l'un des deux UART. Lorsque vous utilisez le REPL sur Telnet, une authentification est nécessaire. Les informations d'identification par défaut sont les suivantes :

user: micro
password: python

pour en savoir plus : https://docs.pycom.io/chapter/gettingstarted/programming/repl/serial.html

7.2.7 FTP

Il y a un petit système de fichiers interne accessible sur chaque périphérique Pycom, appelé **/flash**. Celui-ci est stocké dans la mémoire flash série externe. Si une **carte micro SD** est connectée et montée, elle sera également disponible. Lorsque l'appareil démarre, il le fait toujours à partir du fichier **boot.py** situé dans le système de fichiers /flash.

Le système de fichiers est accessible via le **serveur FTP natif** qui s'exécute sur chaque périphérique Pycom. Ouvrez un client FTP et connectez-vous à :

url : ftp://192.168.4.1
username : micro
password : python

Voir **network.server** pour plus d'informations sur la façon de changer les valeurs par défaut. Les clients recommandés sont :

- MacOS / Linux: client FTP par défaut
- Windows: Filezilla et FireFTP.

Par exemple, à partir d'un terminal MacOS / Linux:

```
$ FTP 192.168.4.1
```

Le serveur FTP ne supporte pas le mode actif, seul le mode passif. Par conséquent, si vous utilisez le client FTP Unix natif, immédiatement après vous être connecté, exécutez la commande suivante :

```
FTP> passive
```

Le serveur FTP ne prend en charge qu'une connexion à la fois. Si vous utilisez d'autres clients FTP, consultez leur documentation pour savoir comment limiter les connexions maximales autorisées à une à la fois.

- **FileZilla**

  Si vous utilisez FileZilla, il est important de bien configurer les paramètres.
  N'utilisez pas le bouton de connexion rapide. Au lieu de cela, ouvrez le gestionnaire de site et créez une nouvelle configuration. Dans l'onglet Général, assurez-vous que le cryptage est défini sur :

  Utilisez uniquement le protocole FTP (non sécurisé). Dans l'onglet Paramètres de transfert, limitez le nombre maximal de connexions à un. Les autres clients FTP peuvent se comporter de manière similaire ; voir leur documentation pour plus d'informations spécifiques

7.2.8 Démarrage sécurisé (Safe Boot)

À la mise sous tension ou en appuyant sur le bouton de réinitialisation, un module Pycom démarrera en mode standard ; le fichier **boot.py** sera exécuté en premier, suivi de **main.py**. Il est possible de modifier la procédure de démarrage du module en fixant certaines broches à l'état logique haut ou bas au démarrage du module.

- **bootloader**

  Si vous avez mis à jour votre appareil avant de l'utiliser, vous avez déjà mis l'appareil en mode bootloader. Ceci est réalisé en connectant P2 à GND pendant que l'appareil démarre. Si vous avez utilisé un Pysense / Pytrack pour faire la mise à jour, il l'a fait automatiquement pour vous. Vous avez seulement besoin de mettre votre module Pycom en mode bootloader si vous mettez à jour son firmware, ou si vous programmez votre propre code de bas niveau. Ce n'est pas nécessaire si vous mettez à jour votre code micropython.

- **Safe Boot**

  Parfois, le code que vous avez écrit vous empêchera d'accéder au REPL ou vous empêchera de mettre à jour votre code. Par exemple :
  - Vous avez désactivé le WiFi / UART
  - Votre code est bloqué et vous ne pouvez plus faire fonctionner le REPL

- Vous définissez un socket comme bloquant, mais ne recevez jamais de données

Afin de résoudre ce problème, vous pouvez démarrer votre module en toute sécurité. Cela empêchera l'exécution de boot.py et main.py et vous redirigera directement dans le REPL interactif. Après la réinitialisation, si la broche P12 est maintenue haute (c'est-à-dire si elle est connectée au 3V3), la DEL de pulsation commence à clignoter lentement en orange.

Si au bout de 3 secondes la broche est toujours maintenue haute, la LED commencera à clignoter plus rapidement. Dans ce mode, le module fera comme expliqué précédemment, mais il sélectionnera également l'image OTA précédente si vous avez mis à jour le module via la procédure de mise à jour OTA (les mises à jour effectuées via l'outil de mise à jour du firmware ne comptent pas). Ceci est utile si vous avez flashé une mise à jour OTA qui bloque l'appareil.

Pin P12 relâché pendant :

1re fenêtre de 3 secondes	2e fenêtre de 3 secondes
Désactive boot.py et main.py	Comme précédemment, mais en utilisant le précédent firmware OTA.

La sélection effectuée lors du démarrage sécurisé n'est pas persistante. Par conséquent, après la réinitialisation normale suivante, le dernier microprogramme sera exécuté à nouveau.
Si des problèmes surviennent dans le système de fichiers ou si vous souhaitez réinitialiser votre module pour supprimer votre code, exécutez le code suivant dans REPL :

```
>>> import os
>>> os.mkfs('/flash')
```

Sachez que la réinitialisation du système de fichiers flash supprimera tous les fichiers se trouvant dans le stockage interne du périphérique (et non la carte SD) et ne pourra pas être récupérée.

- **RESET**
  Les périphériques Pycom prennent en charge les réinitialisations logicielles et matérielles. Une réinitialisation logicielle efface l'état de la machine virtuelle MicroPython, mais laisse les périphériques matériels non affectés. Pour effectuer une réinitialisation logicielle, appuyez sur **Ctrl + D** sur le REPL ou depuis un script, lancez :

```
>>> import sys
>>> sys.exit()
```

Une réinitialisation matérielle équivaut à exécuter un cycle d'alimentation sur l'appareil. Pour réinitialiser l'appareil, appuyez sur l'interrupteur de réinitialisation ou lancez :

```
>>> import machine
>>> machine.reset()
```

7.2.9 Les outils de la barre latérale (All commands)

<u>Les commandes</u> (accessibles à partir de All commands)
- **Pymakr > Connect (ctrl-shift-c)** : se connecte à la carte.
- **Pymakr > Disconnect** : Déconnexion de la carte.
- **Pymakr > Run current file** (ctrl-shift-r) : Exécute le fichier actuellement ouvert sur la carte.
- **Pymakr > Upload project** : Charge le projet de la carte vers le PC.
- **Pymakr > Download project** : Télécharge le projet du PC dans la carte.
- **Pymakr > Project settings** : ouvre des paramètres spécifiques au projet. Ils écrasent les paramètres globaux.
- **Pymakr > Global settings** : (ctrl-shift-g) : ouvre le fichier de paramètres globaux à l'installation.
- **Pymakr > Extra >Get board version : a**ffiche la version du firmware de la carte connectée.
- **Pymakr > Extra > Get WIFI AP SSID :** : obtient le SSID du point d'accès wifi des cartes.
- **Pymakr > Extra > List serial ports** : liste tous les ports séries disponibles et copie le premier dans le presse-papiers.
- **Pymakr > Help** : Imprime cette liste de commandes et de paramètres.

Raccourcis utiles:
- ctrl-shift-c: (Re) connecter
- ctrl-shift-g: Paramètres globaux
- ctrl-shift-s: synchroniser le projet
- ctrl-shiftt-r: Exécute le fichier courant

7.2.10 Les paramètres du fichier pymakr.json

Utilisez **All commands-> 'Global settings**' pour régler les paramètres de la carte. Complétez l'adresse IP. Si vous avez changé le nom d'utilisateur et le mot de passe avec autre chose que « **micro** » et « **python** », faites la mise à jour (pas utile avec un port com). Si vous voulez synchroniser un sous-dossier de votre projet au lieu du projet entier, entrez le nom du sous-dossier dans le champ 'dossier de synchronisation'.

Tous les paramètres possibles (nom: *valeur par défaut* - description):
- **adresse** : *192.168.4.1* - adresse IP ou port com de la carte
- **nom d'utilisateur** : *micro* - nom d'utilisateur du forum, uniquement pour Telnet
- **password** : python - mot de passe du forum, uniquement pour Telnet
- **sync_folder** : *<vide>* - Dossier à synchroniser. Vide pour synchroniser le dossier principal des projets.
- **sync_file_types** : *py, txt, log, json, xml* - type de fichiers à synchroniser
- **ctrl_c_on_connect** : *false* - si vrai, exécute un ctrl-c sur **connect** pour arrêter l'exécution des programmes
- **open_on_start** : *true* - pour ouvrir le terminal et se connecter au tableau lors du démarrage de vsc
- **statusbar_buttons** : *[ ]* - Les boutons d'accès rapide à afficher dans la barre d'état.

Les options sont : ['status', 'run', 'upload', 'télécharger', 'déconnecter', 'listserial', 'settings', 'projectsettings', 'getversion', 'getssid']
N'importe lequel d'entre eux peut être utilisé dans le projet config pour remplacer la configuration globale.

Par défaut, seuls les types de fichiers suivants sont synchronisés : py, txt, log, json et xml. Cela peut être modifié en utilisant le champ 'Sync file types' dans les paramètres.

La limite de synchronisation est définie sur **350 Ko**. Si votre dossier de synchronisation contient plus que cela, le plugin refusera de se synchroniser.

8 L'extension Python

Installer **Python 3.5 ou supérieur** (lien) puis installer l'extension **Python pour VS Code**.
(voir le §1.4 "Installation des extensions")

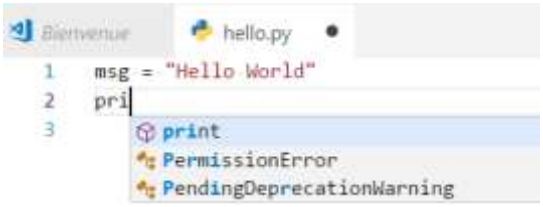
8.1 Un premier projet : Hello world

1. Créer un dossier *HelloWorld* et l'ouvrir dans VS Code. (voir le §2 "Utilisation de l'éditeur").

2. Créer un fichier hello.py (voir le §2 "Utilisation de l'éditeur"). Le nom du langage Python doit apparaître dans la **barre d'état** et sa version dans la **barre latérale** Python 3.6 (32 bit).
   Si ce n'est pas le cas voir le §6.2 ci-dessous.

3. Écrire le code suivant dans hello.py
   msg = "Hello World"
   print(msg)

4. L'autocomplétion permet de retrouver les méthodes associées à l'objet msg telles que capitalize. (voir le §2 "Utilisation de l'éditeur" pour la signification des symboles)

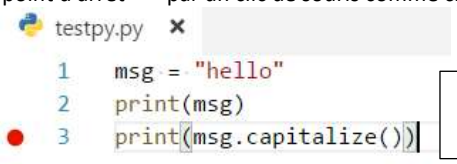


8.2 Sélectionner la version de Python

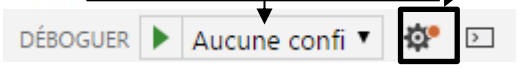
Lorsqu'un fichier .py a été créé dans le répertoire du projet, il est possible de sélectionner une version de Python à partir de la barre d'état.

8.3 Déboguer un script Python

- **Exemple 1 : Sortie d'informations**
  - **Placer** un point d'arrêt par un clic de souris comme ci-dessous :



  - **Sélectionner** puis le débogueur Python



  - **Clic** sur pour démarrer le débogueur. La barre d'outils ci-dessous apparait.

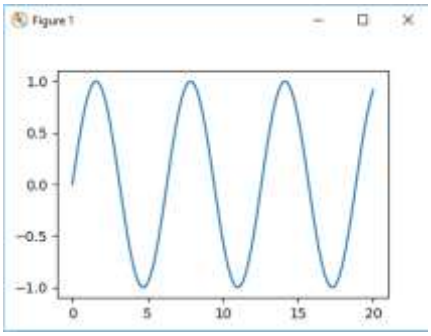
  - Le programme s'exécute dans **le terminal.**

- **Exemple 2 : Entrée, sortie d'informations**

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

  - Le programme s'exécute dans **le terminal.**

8.4 Utiliser des bibliothèques



- **Exemple 3 : Tracé d'une courbe**

```
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 bibliothèques **matplotlib** et **numpy** n'ont pas été installées, un **message d'erreur** apparaitra dans le terminal :

File "c:\Users\phili\Desktop\testPython\testpy.py", line 1, in <module>
import matplotlib.pyplot as plt
ModuleNotFoundError: No module named 'matplotlib'

Pour importer les bibliothèques, entrer le code ci-dessous dans le terminal :

pip3 install matplotlib

Important : Le gestionnaire de paquet **pip3** doit être accessible dans la console **cmd**. Pour cela, ajouter le chemin vers **pip3.exe** dans les **variables d'environnement**. Ne pas oublier de redémarrer VSCode.

- **Exemple 4 : Interface graphique avec Pygame**

```
# Bibliothèques
import pygame # https://www.pygame.org/docs/
from pygame.locals import *
```

Pour la suite du code, voir le cours "**Interface graphique Pygame pour Python** " sur OpenClassrooms : http://bit.ly/2KPlBaB

Pour importer les bibliothèques, entrer le code ci-dessous dans le terminal :

pip3 install pygame

8.5 Pour aller plus loin

- 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 - Activer, configurer et appliquer une variété de linters Python.
- Debugging - Apprenez à déboguer Python à la fois localement et à distance.
- Unit testing - Configurer des environnements de test unitaires et découvrir, exécuter et déboguer des tests.
- Settings reference - Explorez toute la gamme de paramètres liés à Python dans VS Code.

9 L'extension Markdown

- **Qu'est-ce que Markdown ?**
  Markdown est un système d'édition et de formatage de texte ; c'est à la fois une syntaxe, un script de conversion texte → HTML et un format de fichier. Il est couramment utilisé pour les fichiers de documentation d'un projet ou d'un jeu de données souvent nommé ***readme.md***. Il est stocké au format texte classique et est plus léger que sa version interprétée puisqu'il ne contient pas les balises HTML.

  La philosophie du système veut que le texte écrit soit lisible sans interpréteur particulier en mode texte. Il est léger et épuré de l'essentiel de la verbosité d'un langage balisé. Les éléments de syntaxe sont des caractères de ponctuation qui font sens visuellement même non convertis. Une fois converti, le navigateur web (qui joue alors le rôle d'interpréteur) en rendra la lecture plus claire.

  Les fichiers sont généralement enregistrés avec l'extension **.md** (ou **.markdown**) pour indiquer aux interpréteurs la nature du texte qu'il vont lire ; mais ça n'a rien d'obligatoire.
  Comme le résultat sera exporté en HMTL, **vous pouvez tout à fait introduire directement des balises HTML** dans votre texte ; mais celui-ci deviendra moins lisible et ne pourra plus être édité par quelqu'un ne maîtrisant pas le HTML. Attention, le formatage markdown ne sera pas appliqué à l'intérieur de ces balises.

- **Outils**
  - En ligne : Markable
  - Hors ligne : Write Monkey
  - Extension navigateur : Markdown Here

- **Syntaxe (extrait)**

Markdown		**HTML**	**Commentaires**
#titre1	⇔	<h1>	: titre1
##titre2	⇔	<h2>	: titre2
texte	⇔	text	: emphase
texte	⇔	text	: gras
--texte--	⇔	<u>text</u>	: souligné
~~ texte ~~	⇔		: barré
Sauter une ligne	⇔		: nouveau paragraphe
Retour en fin de ligne	⇔	 	: un espace en fin de ligne
> Une citation	⇔	<blockquote></blockquote>	
Saut de ligne puis indentation			: bloc de code
[google] (http://www.google.com "link to google")	⇔	google	: lien

La syntaxe complète est disponible **ici :** https://goo.gl/h4Rq21

- **Utilisation dans VSCode**
  Lors de l'édition d'un fichier Markdown dans VCode avec **markdownlint** installé, toute ligne qui viole une des règles de markdownlint (voir doc dans VSCode) déclenchera un avertissement dans l'éditeur.

  Les avertissements sont indiqués par un trait vert ondulé et peuvent également être vus en appuyant sur **Ctrl + Maj + M** pour ouvrir la boîte de dialogue Erreurs et avertissements. Passez le pointeur de la souris sur une ligne verte pour voir l'avertissement ou appuyez sur **F8 et Maj + F8** pour parcourir tous les avertissements (les avertissements markdownlint commencent tous par **MD ###**). Pour plus d'informations sur un avertissement markdownlint, placez le curseur sur une ligne et cliquez sur l'icône de l'ampoule ou appuyez sur Ctrl +. pour ouvrir la boîte de dialogue d'action de code. Si vous cliquez sur l'un des avertissements dans la boîte de dialogue, l'entrée d'aide de cette règle s'affiche dans le navigateur Web par défaut.

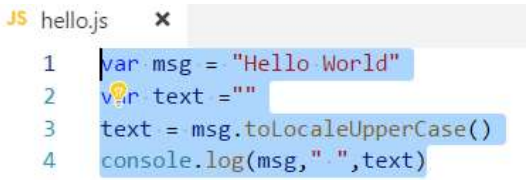
10 Javascript

Visual Studio Code est écrit en TypeScript / JavaScript et alimenté par un serveur Node.

https://code.visualstudio.com/Docs/nodejs

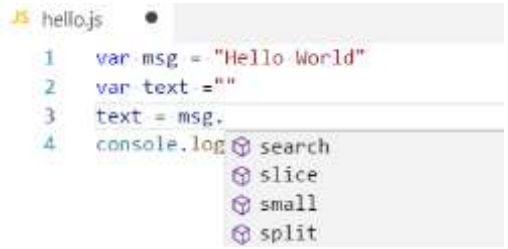
10.1 Un premier projet : Hello World

1. Créer un dossier *HelloWorld* et l'ouvrir dans VS Code. (voir le §2 "Utilisation de l'éditeur").
2. Créer un fichier hello.js (voir le §2 "Utilisation de l'éditeur"). Le nom du langage JavaScript doit apparaître dans la **barre d'état**.



3. Écrire le code suivant dans hello.js

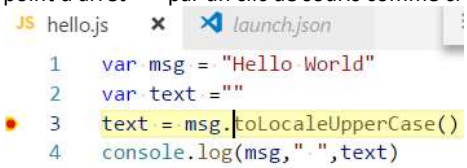
```
var msg = "Hello World"
var text =""
text = msg.toLocaleUpperCase()
console.log(msg," ",text)
```



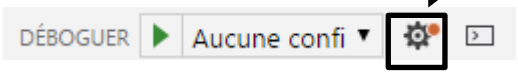
4. L'autocomplétion permet de retrouver les méthodes associées à l'objet msg telles que toLovaleUpperCase. (voir le §2 "Utilisation de l'éditeur" pour la signification des symboles)

10.2 Déboguer un code JavaScript

- **Exemple 1 : Sortie d'informations**
  - **Placer** un point d'arrêt par un clic de souris comme ci-dessous :



  - **Sélectionner** puis le débogueur Python



  - **Clic** sur pour démarrer le débogueur. La barre d'outils ci-dessous apparait.

  - Le programme s'exécute dans **la console.**

11 Raccourcis clavier

F1 ou **ctrl-shift-P**	Editeur de commande
Ctrl+P	pour naviguer dans un fichier ou un symbole en entrant son nom
?	pour obtenir la liste des commandes disponibles
Ctrl+Tab	Sélection d'un fichier pour l'afficher dans la fenêtre active
Ctrl+Shift+O	Définition des symboles Remarque Faire suivre @ de : pour les grouper par catégories
Ctrl+G	pour atteindre une ligne dans un fichier
Ctrl+Shift+F	recherche d'un texte dans tous les fichiers du projet
Ctrl+Shift+M	pour voir le récapitulatif des erreurs et avertissements
Ctrl+Click	pour atteindre une définition
Shift+F12	recherche de référence
Alt+F12	définition
F2	renommer
F8, shift-F8	Recherche d'erreur dans la page

12 Installation sous Linux

Chrome
Visual Studio Code
.deb
Ouvrir avec
Software center