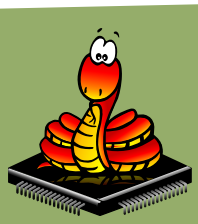

MicroPython
pour l'ESP32 et ses copains
Partie 1 : installation et premiers programmes

Günter Spanner (Allemagne)

Python remplace de plus en plus le C comme langage de programmation de prédilection, et cette tendance s'observe désormais aussi dans le monde des microcontrôleurs. En prenant l'ESP32 comme exemple, nous allons voir dans cet article comment programmer un microcontrôleur moderne en MicroPython.

Matériel nécessaire

1. Petite plaque d'expérimentation*
2. ESP32-PICO-KIT V4*
3. LED rouge
4. Résistance de 150 Ω
5. Module d'affichage compatible SSD1306*
6. Fils

* Disponible dans la boutique d'Elektor

Ces dernières années, Python a connu un énorme regain de popularité, notamment en raison de la disponibilité de systèmes monocartes tels que le Raspberry Pi. Mais Python a également trouvé des applications plus larges dans d'autres domaines tels que l'intelligence artificielle et l'apprentissage automatique. Il semble donc naturel d'examiner comment employer Python (ou la variante MicroPython en tout cas) dans des montages à microcontrôleurs. Cet article se penche sur les bases de MicroPython, et en particulier sur les instructions les plus importantes et les bibliothèques disponibles. Nous utiliserons un microcontrôleur ESP32 pour réaliser quelques petites applications : le microcontrôleur sera programmé avec un micrologiciel qui interprète les commandes Python. Le programme Python lui-même est écrit dans un environnement de développement sur PC, et les commandes Python peuvent être envoyées du PC au microcontrôleur, soit sous forme d'un programme entier, soit une par une. Cela peut se faire soit par USB, soit par un réseau sans fil. Examinons ces aspects un par un.

Environnements de programmation et de développement

Contrairement à l'écosystème Arduino, MicroPython peut fonctionner avec pas mal d'environnements de développement intégrés, ou EDI. Actuellement, les deux environnements les plus utilisés sont :

1. µPyCraft
2. Thonny

On peut aussi envisager d'installer Anaconda Navigator, surtout utilisé pour la programmation de microcontrôleurs dans le domaine de l'intelligence artificielle. Chaque approche présente des avantages et des inconvénients particuliers. Par exemple, l'EDI µPyCraft n'offre qu'une interface utilisateur relativement peu sophistiquée, avec des éléments graphiques simples qui rappellent les systèmes d'exploitation orientés texte.

Thonny, quant à lui, possède une interface utilisateur graphique complète dans le style de Windows (**fig. 1**). Cet EDI est très populaire dans la communauté des makers, notamment parce qu'il est disponible pour le Raspberry Pi avec le système d'exploitation Raspbian. De nombreux utilisateurs de Raspberry Pi sont donc déjà très familiers avec Thonny. Thonny tourne sur tous les principaux systèmes d'exploitation, notamment Windows, Mac OS X et Ubuntu Linux, et peut être téléchargé gratuitement sur l'internet [1].
Avant d'utiliser Thonny, il est nécessaire que Python 3 soit installé sur la machine qui sera utilisée pour la programmation. Si ce n'est pas encore le cas, vous pouvez trouver les instructions d'installation sur le site web correspondant [2].
Ensuite, installez Thonny lui-même depuis [1]. Sélectionnez d'abord le système d'exploitation approprié en haut à droite de la page. L'exécution du fichier téléchargé lancera le processus d'installation habituel. L'EDI est maintenant prêt pour la création de notre première application Python.

Installation de l'interpréteur

L'étape suivante consiste à installer le micrologiciel MicroPython lui-même. On peut l'obtenir sur le site officiel de MicroPython [3], avec toute une liste de microcontrôleurs disponibles. Dans cet article, nous nous intéresserons plus particulièrement à l'ESP32, et nous devons donc télécharger la dernière version compatible avec ce type de microcontrôleur. Sous *Espressif ESP-based boards*, cliquez sur l'image dont la légende est *Generic ESP32 module*. La page qui s'ouvre alors propose un certain nombre de variantes de micrologiciel. Notez que la version la plus récente du micrologiciel est souvent qualifiée « d'instable ». Cette option est plutôt destinée aux développeurs du micrologiciel de l'interpréteur lui-même et à d'autres personnes de nature aventureuse. Ceux d'entre nous qui préfèrent un système plus stable choisiront la version la plus récente qui ne soit pas qualifiée « d'instable », par exemple :

GENERIC: esp32-idf3-20200902-v1.13.bin

Cliquez sur le lien approprié et le micrologiciel sera téléchargé.

Maintenant connectez la carte à microcontrôleur (par ex. un ESP32-PICO-KIT, voir l'encadré **Matériel nécessaire**) au PC sur l'USB, puis lancez l'EDI Thonny. Ici, il faut choisir *Sélectionner l'interpréteur* dans le menu *Exécuter*, ce qui ouvre la fenêtre des *Options de Thonny* (**fig. 2**). Sous l'onglet *Interpréteur*, vous pouvez choisir parmi une série d'options, dont certaines sont spécifiques à l'ESP32.
Sous *Port*, sélectionnez le port USB auquel l'ESP32 est connecté. Vous pouvez également sélectionner l'option *Essayer de détecter le port automatiquement*. Toutefois, cette option ne fonctionne pas toujours de manière fiable.
En cliquant dans la case sous *Micrologiciel*, le programme d'installation démarre. Le port n'est pas configuré automatiquement et doit être saisi à nouveau. Ensuite, naviguez vers le micrologiciel que nous venons de charger. En cliquant sur *Installer*, le processus d'installation démarre. Lorsque le processus est terminé, la fenêtre reste ouverte : fermez-la et vous êtes prêt pour commencer à programmer votre ESP32 en MicroPython.

Bibliothèques

Travailler avec Python signifie recourir à des bibliothèques. Écrire chaque programme à partir de zéro est, pour le moins, très inefficace, et l'immense succès de Python est en grande partie dû à la quantité de bibliothèques disponibles pour ce langage.

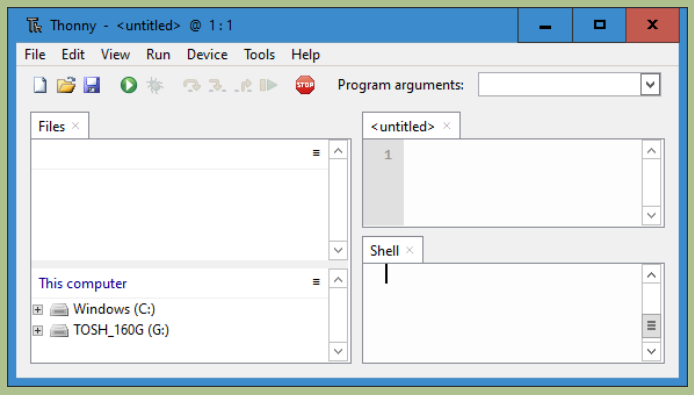

Figure 1. L'EDI Thonny.

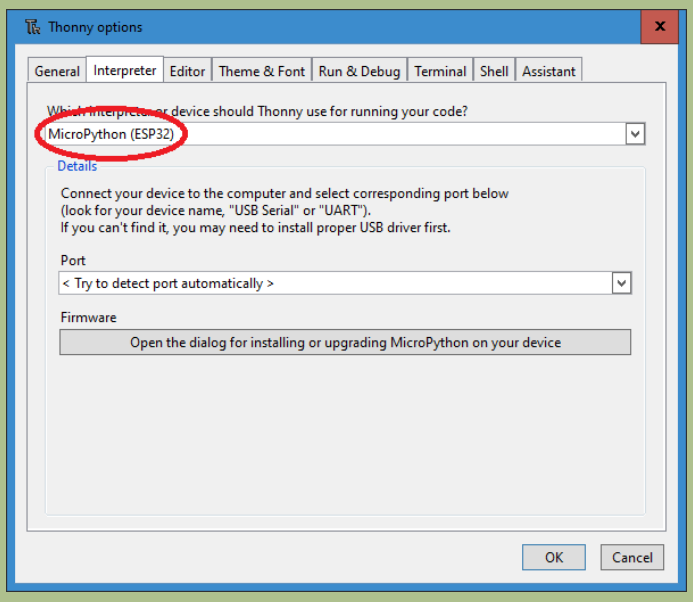

Figure 2. Options d'installation du microprogramme Thonny.

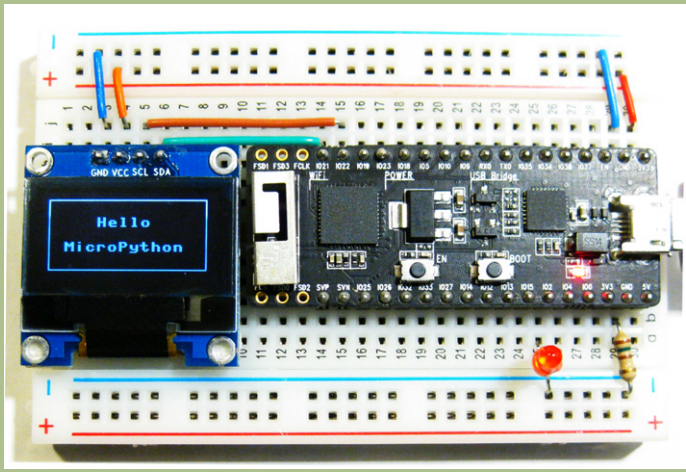

Figure 3. Microcontrôleur ESP32 avec un écran OLED et une LED connectée à la broche 2.

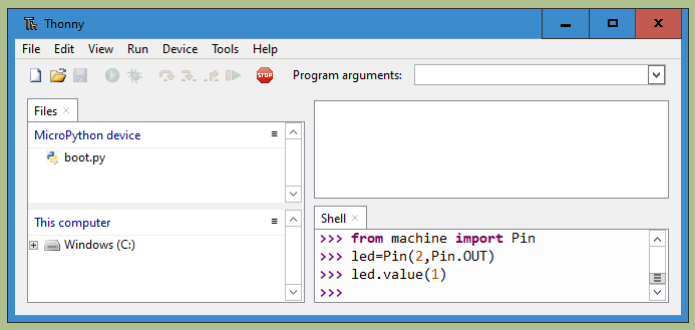

Figure 4. Commutation de l'état d'un port d'E/S à l'aide de la console REPL.

Malheureusement, toutes les bibliothèques ne peuvent pas être exploitées avec les ressources limitées d'un microcontrôleur, et une sélection spéciale de bibliothèques a donc été créée pour être utilisée avec MicroPython. Un grand nombre d'entre elles sont disponibles en standard dans le téléchargement de l'EDI Thonny. Les deux bibliothèques standard les plus importantes dans MicroPython sont *machine* et *time*. L'instruction `import` est utilisée pour rendre les bibliothèques disponibles pour une utilisation sur le microcontrôleur. Les lignes suivantes (entre autres) rendent une fonction de bibliothèque accessible :

```
> import module
> from module import name
```

Dans le premier cas, le module entier est importé ; dans le second, seules les fonctions spécifiées le sont.
Le module *machine* contient des fonctions spécifiques au matériel d'un microcontrôleur particulier. Ces fonctions permettent d'accéder directement et sans limites aux composants matériels et de les commander, notamment l'unité centrale, les temporisateurs, les bus et les broches d'entrée/sortie. Sachez qu'une mauvaise utilisation de ce module peut provoquer des erreurs, des plantages, voire au pire endommager le matériel.
La classe `Pin` est l'une des parties les plus importantes du module *machine*. Un objet `Pin` commande une broche d'entrée/sortie et, par convention, un objet `Pin` est affecté à une broche physique sur le microcontrôleur. L'objet permet de piloter les niveaux de sortie et de lire les niveaux d'entrée.
La classe `Pin` fournit des méthodes pour définir le mode de la broche. Par exemple, `IN` et `OUT` peuvent être utilisés pour configurer respectivement une broche en entrée ou sortie.
Le module *Time* fournit une série de fonctions liées au temps. La classe `sleep` de ce module met en pause l'exécution du programme en cours pendant le nombre de secondes spécifié. L'argument peut même être une valeur à virgule flottante, ce qui permet de définir des délais exacts d'une fraction de seconde. Les commandes

```
from machine import Pin
from time import sleep
```

rendent disponibles les fonctions `Pin` et `Sleep`. Avec la première, nous pouvons dialoguer individuellement avec les broches des ports du microcontrôleur, et avec la seconde nous pouvons implémenter une fonction basée sur le temps. En utilisant la commande

```
led = Pin(2, Pin.OUT)
```

nous pouvons créer un objet `led` associé à la broche d'E/S numéro 2, et configurer cette broche en sortie. Nous pouvons maintenant passer diverses valeurs à cet objet. Par exemple, si nous exécutons la commande

```
led.value(1)
```

alors la valeur 1 sera écrite dans l'objet. Cela signifie que la broche d'E/S associée, la broche 2, sera placée à un niveau de tension haut. Dans le cas de l'ESP32, il s'agit de 3,3 V.

La console REPL

Il est possible de surveiller l'état d'une broche de port en y connectant simplement une LED via une résistance de limitation de courant. (La **figure 3** montre comment faire, et comment aller plus loin en connectant un écran OLED : nous y reviendrons plus tard.) Le port, et donc la LED, peuvent être contrôlés très simplement

à l'aide de ce que l'on appelle la « console REPL ». REPL est l'abréviation de « Read Evaluate Print Loop » (lecture-évaluation-affichage), une méthode d'entrée interactive de MicroPython qui permet d'exécuter des commandes directement sur l'ESP32. La console REPL nous offre donc un moyen très simple de tester des commandes et d'exécuter des programmes.
La console REPL est appelée « *Console* » dans Thonny et se trouve en bas de la fenêtre principale. Vous pouvez saisir les commandes des extraits ci-dessus directement dans cette zone (**fig. 4**).
Après avoir entré la dernière des commandes ci-dessus, la LED doit s'allumer. La commande `led.value(0)` permet de l'éteindre.
La console REPL possède quelques fonctions intéressantes qui facilitent le travail avec MicroPython. Par exemple, les lignes de commande précédentes sont stockées, et les touches de déplacement vers le haut et le bas permettent de rappeler les commandes précédemment saisies si nécessaire.
Une autre fonction pratique est la complétion par tabulation. En appuyant sur la touche « Tab » du clavier, on tente automatiquement de compléter un mot partiellement saisi. Cette fonction permet même d'obtenir des informations sur les fonctions et les méthodes disponibles dans un module ou applicables à un objet.
Par exemple, en entrant « ma » et en appuyant sur la touche « Tab », on obtient automatiquement « machine » (en supposant que le module *machine* a déjà été importé comme décrit ci-dessus). En appuyant sur la touche point (« . ») puis sur la touche « Tab », vous obtiendrez une liste complète de toutes les fonctions disponibles dans le module *machine* (**fig. 5**). Dans de nombreux cas, cette fonction rend inutile la recherche de commandes, d'objets et de méthodes dans la documentation.

Accès sans fil à l'aide de WebREPL

L'un des principaux avantages de l'ESP32 est son excellente prise en charge de la connectivité sans fil. Quoi de plus naturel, donc, que de rendre la console REPL disponible via une telle connexion ? Pour ce faire, nous pouvons utiliser l'interface WebREPL.
La première étape pour rendre WebREPL opérationnel est de s'assurer qu'il est installé et activé sur l'ESP32. Par défaut, WebREPL n'est pas activé et doit être activé en envoyant une fois la commande suivante :

```
import webrepl_setup
```

sur le port série. Vous aurez la possibilité d'activer ou de désactiver la fonction, et de définir un mot de passe. Après la configuration, il faut redémarrer l'ESP32.
Pour utiliser WebREPL sur un réseau sans fil, l'ESP32 doit d'abord être connecté à ce réseau. Pour ce faire, envoyez les commandes suivantes à la console série REPL.

```
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('ssid', 'password')
```

Les champs `ssid` et `password` doivent bien sûr être remplacés par les informations d'identification de votre réseau sans fil local. La commande

```
wlan.ifconfig()
```

affichera alors les paramètres de l'adresse IP que l'ESP32 utilise pour communiquer avec le réseau (**fig. 6**). Les commandes

```
import webrepl
webrepl.start()
```

vont maintenant activer le client WebREPL. Dans un navigateur web, vous pouvez aller à l'adresse

```
http://micropython.org/webrepl/#xxx.xxx.xxx.xxx:8266
```

Vous pouvez ensuite utiliser l'onglet *Connecter* pour vous connecter à l'ESP32 en utilisant le mot de passe défini précédemment.
Lorsque WebREPL est lancé, vous pouvez constater que la console est en mode « raw REPL ». Ce mode permet la saisie directe de commandes par copier-coller. Dans ce contexte, vous pouvez appuyer sur CTRL-B pour repasser en mode normal et à la saisie ordinaire des commandes.
Nous sommes maintenant en mesure de commander l'ESP32

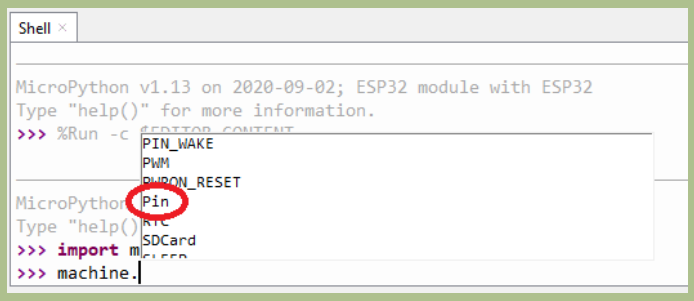

Figure 5. Autocomplétion.

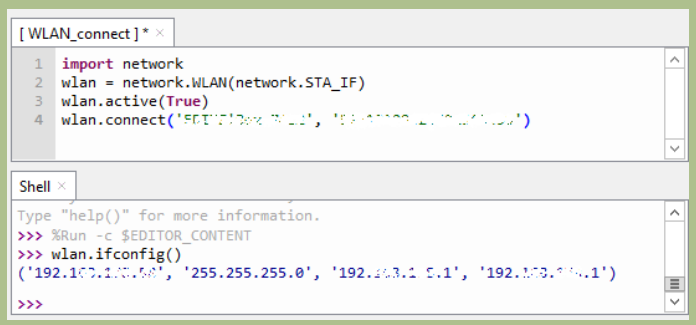

Figure 6. L'ESP32 est connecté au réseau sans fil local.

totalement sans fil. Les commandes simples pour commuter des sorties nous permettent déjà de mettre en œuvre des fonctions de base de domotique (**fig. 7**). Si un système de fichiers approprié est installé sur l'ESP32, on peut aussi effectuer des mises à jour logicielles sans fil, appelées mises à jour OTA (*over-the-air*). Lorsqu'on travaille avec WebREPL, il faut se rappeler qu'il s'agit d'une fonction expérimentale et qu'il ne faut pas s'attendre à ce qu'elle se comporte de manière absolument fiable dans toutes les situations.

Commande par programme

La console REPL ou WebREPL est la plus adaptée pour tester des idées. Pour l'écriture de programmes conventionnels, le panneau de l'éditeur (au-dessus de la console REPL dans la fenêtre Thonny) est plus approprié. Nous pouvons par ex. y préparer le programme décrit ci-dessous, conçu pour la commande automatique de l'éclairage nocturne et celui des cages d'escalier (**fig. 8**).
Une fois le code saisi et lancé à l'aide de l'icône de démarrage (flèche blanche dans un cercle vert), vous serez invité à enregistrer le programme. Choisissez ici l'option *MicroPython device* et entrez un nom de programme (par ex. *Automatic_LED*) et confirmez la sauvegarde. La LED va maintenant s'allumer pendant trois secondes, puis s'éteindre automatiquement. Si vous appuyez à nouveau sur le bouton de démarrage, le programme peut être relancé immédiatement.
Pour transformer ce programme en une démo classique de LED clignotante, il suffit d'ajouter une instruction `while`. Celle-ci fera en sorte qu'une commande ou un bloc de commandes données soient exécutées de manière répétée. Le cas particulier de `while True:` crée une boucle qui se répète sans fin : dans ce cas, cela signifie que la LED clignotera en continu.

```
from machine import Pin
from time import sleep

led = Pin(2, Pin.OUT)

while True:
  led.on()
  sleep(0.5)
  led.off()
  sleep(0.5)
```

Remarquez ici que l'instruction `while` doit se terminer par un deux-points. Le bloc de commandes suivant est, conformément à la convention Python standard, indenté par un nombre fixe d'espaces. Thonny fournit cette indentation automatiquement : dès que vous saisissez un deux-points suivi de la touche Entrée, le curseur apparaît sur la ligne suivante indentée d'un pas. Et bien sûr, l'éditeur de programmes offre également la fonction de complétion par tabulation. La combinaison CTRL-C permet d'arrêter un programme en cours d'exécution.

MicroPython en quelques mots

Bien que l'essence même de MicroPython soit sa collection de bibliothèques, une bonne maîtrise des commandes de base est nécessaire pour comprendre les programmes des autres et pour écrire les siens. Nous examinerons brièvement les commandes les plus puissantes de MicroPython.
Les commentaires simples sont introduits par le symbole '#'. Le commentaire s'étend du symbole '#' à la fin de la ligne.

```
>>> print("hello ESP32")          # this is a comment
  hello ESP32
```

Les commentaires sont ignorés pendant l'exécution du programme ; ils ne sont là que pour fournir des informations au programmeur. La commande `print()` utilisée ici permet d'afficher des informations

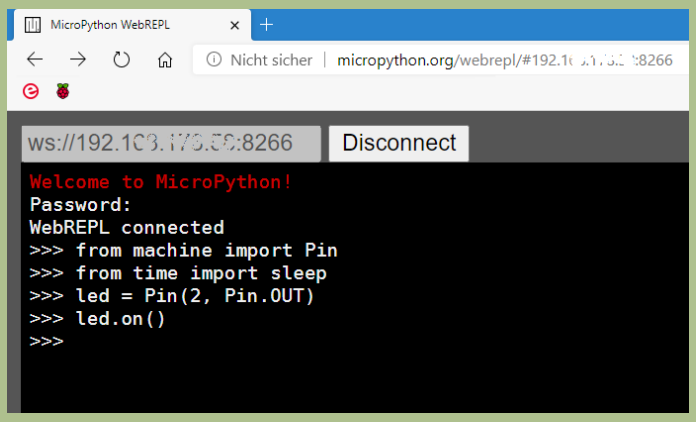

Figure 7. WebREPL dans le navigateur.

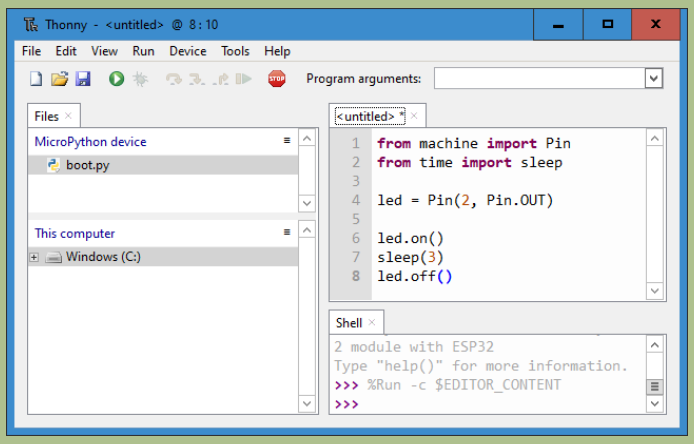

Figure 8. LED automatique.

Produits

- **Livre en anglais « MicroPython for microcontrolers »**
  www.elektor.fr/micropython-for-microcontrollers
- **ESP32-PICO-KIT V4**
  www.elektor.fr/esp32-pico-kit-v4
- **Plaque d'expérimentation**
  www.elektor.fr/breadboard-830-tie-points
- **Module d'affichage compatible SSD1306**
  www.elektor.fr/blue-0-96-oled-display-i2c-4-pin

dans la console, qu'elles soient textuelles ou numériques ; vous pouvez aussi appeler `print()` directement depuis la fenêtre du terminal.
Comme nous l'avons vu plus haut dans l'exemple de la LED clignotante, les instructions peuvent être regroupées en blocs identifiés par leur indentation. Cela signifie que les accolades (« { » et « } ») et autres mécanismes similaires ne sont pas nécessaires. L'avantage de ce principe est que l'on est plus ou moins forcé d'adopter un style de programmation structuré.
Il est très facile de créer une variable dans MicroPython, et en particulier il n'est pas nécessaire de spécifier son type. On peut aussi utiliser directement les variables dans la console comme suit :

```
>>> a=17
>>> b=12
>>> print(a*b)

          204
```

Dans MicroPython, les opérateurs arithmétiques ont leurs interprétations mathématiques conventionnelles. Outre l'addition, la soustraction, la multiplication et la division, nous disposons également de l'opérateur `//` pour la division d'entiers, de `%` pour le modulo (ou le reste de la division) et de `**` pour l'exponentiation.
Les instructions habituelles de branchement et de bouclage existent aussi dans MicroPython. Le branchement est pris en charge par le mot clé `if`, suivi d'une condition ; si la condition est vraie, les instructions suivantes sont exécutées. Un mot-clé `else` peut suivre, introduisant un ensemble d'instructions à exécuter à la place si la condition est fausse. Par exemple :

```
if True:
  # block 01
  print ("True")
else:
  # block 02
  print ("False")
```

Les boucles permettent d'exécuter des instructions de manière répétée. Un ensemble d'instructions est exécuté tant qu'une condition spécifiée est remplie. Deux variantes sont disponibles :

- Boucles « while »
- Boucles « for »

Ainsi, une façon d'imprimer les chiffres de 1 à 9 sur la console serait d'utiliser une boucle `while` comme suit :

```
number=1
while number<10:
  print(number)
  number=number+1
```

L'ensemble des instructions à répéter est indiqué par l'indentation. Cette tâche peut également être effectuée en utilisant une boucle `for` comme suit :

```
for number in range(1, 10):
  print(number)
```

Signal d'alarme automatique

Nous en savons maintenant assez pour écrire notre premier programme d'application pratique. Il s'agira d'une balise SOS automatique qui pourrait être utilisée pour signaler une urgence en voile ou en escalade.

```
from machine import Pin
from time import sleep
led=Pin(2,Pin.OUT)

while True:
  for n in range(3):
    led.value(1)
    sleep(.1)
    led.value(0)
    sleep(.5)
  sleep(1)
  for n in range(3):
    led.value(1)
    sleep(.4)
    led.value(0)
    sleep(.5)
  sleep(1)
  for n in range(3):
    led.value(1)
    sleep(.1)
    led.value(0)
    sleep(.5)
  sleep(2)
```

Panneau OLED : un petit écran d'affichage

Il est sans doute possible d'utiliser une seule LED pour communiquer des informations utiles, en utilisant par ex. le code morse comme dans l'application de balise SOS ci-dessus. Une approche bien plus moderne consiste bien sûr à afficher des données sur un panneau OLED. De nombreux panneaux de ce type utilisent le pilote d'affichage SSD1306. Ici, nous utilisons un écran d'une diagonale de seulement 0,96 pouces (environ 2,5 cm) et d'une résolution de 128×64 pixels.
MicroPython est livré en standard avec une bibliothèque pour les écrans à base de SSD1306. Elle permet d'afficher des données textuelles et numériques ainsi que de créer des graphiques simples. Les modules les plus simples à base de SSD1306 ont une interface

qui n'utilise que quatre broches, ce qui suffit pour piloter l'écran avec le protocole de bus I2C. La connexion à l'écran est illustrée à la **figure 3**, et les connexions nécessaires sont présentées dans le tableau ci-dessous.

Broche OLED	Broche ESP32
VDD	3V3
GND	GND
SCK	GPIO 22
SDA	GPIO 21

Le script présenté dans le listage 1 envoie un texte à l'écran et dessine quelques graphiques simples comme ce cadre autour du texte (fig. 3).
La bibliothèque correspondante est disponible sous forme de paquetage standard (*ssd1306.py* dans l'archive de téléchargement [5]) et peut être téléchargée séparément sur la carte. Les broches utilisées pour l'interface I2C sont déclarées comme suit :

```
i2c = I2C (-1, scl = Pin (22), sda = Pin (21))
```

Le premier paramètre, « -1 », indique que le module utilisé ne dispose pas d'une broche de réinitialisation ou d'interruption. Le nombre de pixels horizontaux et verticaux du module est spécifié à l'aide de la commande :

```
oled = SSD1306_I2C(128, 64, i2c)
```

L'écran est maintenant prêt. La fonction `text()` sert à écrire des informations dans le tampon d'affichage, et l'affichage lui-même est mis à jour avec la méthode `show()`. La fonction `text()` accepte les arguments suivants :

- Le message (une chaîne de caractères)
- Les coordonnées x et y du texte en pixels
- En option la couleur du texte : 0 pour le noir (non allumé) et 1 pour le blanc (allumé)

La méthode `show()` permet de rendre les modifications visibles à l'écran. La méthode `rect()` permet de dessiner un rectangle à l'écran. Elle accepte les arguments suivants :

- Coordonnées x et y du coin inférieur gauche du rectangle
- Coordonnées x et y du coin supérieur droit du rectangle
- Couleur de pixel : 0 pour le noir et 1 pour le blanc

La commande

```
oled.rect(5, 5, 116, 52, 1)
```

fait donc apparaître un cadre rectangulaire près du bord de l'écran. Avec ces simples commandes, il est possible pour une application d'afficher toutes sortes d'informations, du texte de base aux relevés complexes de capteurs.

Listage 1. Message sur l'écran OLED.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c=I2C(-1,scl=Pin(22),sda=Pin(21))
oled = SSD1306_I2C(128, 64, i2c)
lin_hight = 9
col_width = 8

oled.fill(0)
oled.text("Hello", 5*col_width, 2*lin_hight)
oled.text("MicroPython", 2*col_width,
4*lin_hight)

oled.rect(5, 5, 116, 52, 1)
oled.show()
```

Perspectives

MicroPython est un langage de programmation moderne et puissant. De plus, ses bibliothèques permettent de réaliser rapidement et facilement des projets complexes. Dans cet article, nous avons montré comment installer un EDI approprié et créer quelques applications simples. Dans la deuxième partie de cette série, nous examinerons d'autres aspects de MicroPython et, à titre de démonstration pratique, nous donnerons un exemple de pilotage d'un panneau d'affichage matriciel à LED de grand format. Vous trouverez de plus amples informations sur MicroPython, sur le microcontrôleur ESP32 et sur les exemples présentés ici dans le livre *MicroPython for Microcontrollers* [4].

(210179-04)

Contributeurs
Texte et illustrations : **Günter Spanner**
Rédaction : **Jens Nickel**
Mise en page : **Harmen Heida**
Traduction : **Denis Lafourcade**

Des questions, des commentaires ?
Contactez Elektor (redaction@elektor.fr).

LIENS

[1] **Thonny** : https://thonny.org/
[2] **Python** : www.python.org/downloads
[3] **MicroPython** : http://micropython.org/download
[4] **G. Spanner, *MicroPython for Microcontrollers*, Elektor 2020** : www.elektor.fr/micropython-for-microcontrollers
[5] **Logiciel** : www.elektormagazine.fr/210179-04