Aabu

Zeste de savoir

MicroPython : Python pour les microcontrôleurs

16 décembre 2022

Table des matières

Introduction

Python est un langage de programmation populaire, comme en témoigne sa présence durable dans le top 10 de l'index TIOBE . Dans le monde de l'embarqué, il est cependant éclipsé par des langages tels que le C, C++ et leurs variantes, en partie parce qu'il était jusqu'à récemment inadapté aux contraintes de ce domaine.

Cette lacune est comblée depuis 2014 par MicroPython, une implémentation de Python destinée aux microcontrôleurs, qui est accompagnée par un ensemble de cartes électroniques compatibles. Cette combinaison permet ainsi à tous de programmer dans leur langage favori pour la réalisation de leurs projets embarqués.

FIGURE 0.1. – Logo de MicroPython (source).

1. Le langage et ses bibliothèques

MicroPython a été publié en 2014 à l'issue d'une campagne de financement sur Kickstarter par le chercheur et ingénieur australien Damien P. George, qui est toujours le principal contributeur du projet.

1.1. Du véritable Python adapté pour les microcontrôleurs

MicroPython est une implémentation presque complète de Python 3. On retrouve quasiment toutes les constructions du langage, avec notamment les classes, les fermetures, les exceptions, les générateurs, les coroutines (y compris asynchrones) et j'en passe.

Certaines fonctionnalités avancées du langage ne sont pas implémentées, en général pour des raisons de performance sur les systèmes embarqués. Par exemple, MicroPython ne supporte pas les fonctionnalités les plus avancées concernant les classes et fonctions et offre des possibilités d'introspection limitées. Quelques autres différences mineures existent, mais MicroPython reste largement compatible avec CPython.

MicroPython propose aussi quelques améliorations pour les systèmes embarqués, notamment afin d'optimiser le langage pour une empreinte mémoire réduite. Par exemple, il est possible de déclarer certaines valeurs comme constantes afin d'économiser de la mémoire en évitant la création d'objets inutiles.[1] L'exemple ci-dessous extrait de la documentation montre un cas d'utilisation sur des entiers.

```
1 from micropython import const
2
3 CONST_X = const(123)
4 CONST_Y = const(2 * CONST_X + 1)
```

On peut également écrire de l'assembleur *inline* dans un script, pour accéder aux fonctionnalités de plus bas niveau. Voici un court exemple extrait de la documentation qui montre comment on allumerait une LED de cette manière.[2]

```
@micropython.asm_thumb
def led_on():
    movwt(r0, stm.GPIOA)
    movw(r1, 1 << 13)
    strh(r1, [r0, stm.GPIO_BSRRL])
```

1.2. Une bibliothèque standard allégée, modulaire et permettant l'accès au matériel

Comme MicroPython vise des plateformes aux ressources réduites, notamment en termes de mémoire vive et de stockage, seul un nombre limité de modules de la bibliothèque standard de Python sont implémentés, souvent avec des fonctionnalités réduites à l'essentiel.

Les développeurs ayant des besoins plus avancés ne sont pas pour autant oubliés, puisqu'il existe un projet pour fournir à MicroPython une bibliothèque standard plus fournie, avec des modules réécrits pour s'adapter aux limitations de MicroPython. Cette bibliothèque se veut modulaire et la liberté est laissée au développeur d'installer seuls les quelques modules nécessaires pour leur application.

Une spécificité majeure de MicroPython est de fournir des bibliothèques pour contrôler le matériel. Sans en faire une liste exhaustive, il existe des objets permettant de piloter des interfaces simples comme des diodes ou des convertisseurs analogique-numérique, mais aussi des périphériques plus complexes, comme des accéléromètres ou des contrôleurs CAN par exemple.

1.3. Quelques différences majeures avec CPython sous le capot

La libération de la mémoire est gérée différemment dans MicroPython que dans CPython. Ce dernier utilise principalement le comptage de référence pour suivre l'utilisation des objets et les supprimer quand ils ne sont plus accessibles; autrement dit, un objet vers lequel ne pointe plus aucune référence sera supprimé. MicroPython quant à lui utilise uniquement un système de ramasse-miette *marquant et nettoyant* (*mark and sweep*). On trouve plus d'information sur cette implémentation sur le dépôt GitHub et le visionnage de cette conférence vous présentera plus en détails les différences de gestion de la mémoire entre CPython et MicroPython.

La gestion des coroutines asynchrones est aussi simplifiée. L'implémentation ne fait de différence entre une coroutine asynchrone et un simple générateur. Il n'y a pas de différence fondamentale entre `await` et `yield from`. Cette simplification se traduit également par l'absence de `futures` (tout est coroutine) et la possibilité d'utiliser `await` sur des itérateurs et générateurs, ce qui est impossible avec CPython.

1. Plus de détail à ce sujet dans la documentation, ici et là .
2. D'autres exemples se trouvent dans le tutoriel sur l'assembleur *inline* et dans la documentation de référence sur cette fonctionnalité.

2. Le matériel compatible

La compatibilité d'une carte avec MicroPython est essentiellement conditionnée par l'existence d'un support pour son microcontrôleur. Les cartes les mieux supportées sont fournies avec des modules offrant une interface complète vers leur matériel. Les différentes cartes offrent toute une gamme de fonctionnalités et performances, s'adaptant à des besoins divers.

2.1. Cartes « officielles »

Lors du Kickstarter, MicroPython a été publié avec une carte, qui est devenue de fait une sorte de carte «officielle». Il s'agit de la pyboard v1.0 puis v1.1. Cette carte fournit une panoplie de périphériques autour d'un microcontrôleur ARM STM32F405RG. On dispose ainsi d'un lecteur de cartes SD, de DAC et d'ADC, de contrôleurs CAN, de LED, et même d'un petit accéléromètre.

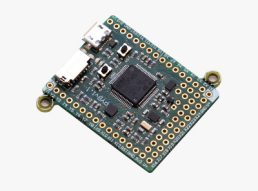

FIGURE 2.2. – pyboard v1.1

Une carte un peu plus costaude a été publié récemment, qui offre en particulier une connectivité WiFi et Bluetooth.

FIGURE 2.3. – pyboard-D, avec connectivité WiFi et Bluetooth.

D'autres cartes existent. On peut citer notamment des cartes ESP32 et ESP8266 ainsi que wipy, qui ont le privilège d'apparaître dans la documentation officielle. Il existe également des portages de MicroPython pour tout une gamme de cartes plus ou moins populaires. La liste complète des portages est visible sur le dépôt GitHub de MicroPython .

2.2. Portages Unix et Windows

MicroPython ne se cantonne pas aux microcontrôleurs: il existe un portage Unix et un portage Windows qui se base dessus.

Ce portage peut servir en remplacement de CPython sur un nano-ordinateur (comme un Raspberry Pi par exemple) ou tout simplement comme outil de développement pour tester les fonctionnalités indépendantes de la carte cible.

3. MicroPython sur un exemple

Comme pour Python, il est possible de travailler avec la boucle interactive, ce qui est pratique pour tester rapidement quelque chose.

```
Device path /dev/ttyACM0
Quit: Ctrl+] | Stop program: Ctrl+C | Reset: Ctrl+D
Type 'help()' (without the quotes) then press ENTER.

>>> a = 5
>>> b = 3
>>> a + b
8
```

FIGURE 3.4. – Boucle interactive (*read eval print loop*).

Ce n'est cependant pas la manière habituelle d'utiliser MicroPython, puisqu'on utilise plutôt des scripts. Voyons une démonstration rapide sur la *pyboard*.

3.1. Le système d'initialisation en bref

On peut transférer les fichiers à exécuter sur la *pyboard* comme sur n'importe quel périphérique de stockage USB. Le minimum vital consiste en deux fichiers:

— `boot.py`,
— `main.py`.

À l'initialisation de la carte, l'interpréteur se charge de lire `boot.py` pour savoir comment configurer la carte. Après la configuration du matériel, l'interpréteur se charge d'exécuter `main.py`.

En bref, il suffit de mettre son code dans `main.py` pour le voir exécuté par MicroPython!

3.2. Faisons clignoter des LED

Voici un petit exemple qui a pour effet de faire clignoter la LED rouge de la carte de plus en plus lentement puis de l'arrêter.

Comme je n'ai pas besoin de changer quoi que ce soit à la configuration par défaut, le fichier `boot.py` est vide. Le fichier `main.py` quant à lui contient le code suivant:

```
1 import pyb  # bibliothèque pour manipuler les fonctionnalités de
      la pyboard
```

```
import time  # gestion du temps
led = pyb.LED(1)  # créer un handler sur la LED n°1 (rouge)
delays = (i for i in range(100, 1000, 20))  # générer des durées
    croissantes par pas de 20 ms
for d in delays:  # pour chaque durée
    led.toggle()  # basculer l'état de la LED
    time.sleep_ms(d)  # attendre une durée d
```

Dans toute sa simplicité, ce code montre tout de même deux choses:

— l'interfaçage avec le matériel sous forme de modules et de classes (le module `pyb` et la classe `LED`)
— la disponibilité de Python véritable (classes, boucles et générateur dans ce cas-ci),

Vous pouvez tester ce programme en le copiant sur le simulateur en ligne . D'autres exemples sont aussi disponibles sur ce site.

4. Apprendre ou contribuer à MicroPython

La communauté MicroPython est petite en comparaison d'autres mastodontes tels qu'Arduino , mais est aussi bien active. Le point d'entrée principal vers l'écosystème de MicroPython est le site officiel .

4.1. Apprendre MicroPython

Le premier pas pour utiliser MicroPython est d'apprendre Python. Ensuite, il existe la possibilité de tester MicroPython en ligne sur un simulateur de *pyboard*!

Par la suite, la documentation en ligne centralise l'essentiel de l'information autour du langage MicroPython, de sa bibliothèque standard, ainsi que des informations sur les principaux portages, dont la *pyboard*. Il y a également des pages spécifiques pour l'aide à l'optimisation, des notes techniques sur les différences avec CPython, etc.

La communauté anglophone est présente sur le forum . Ce forum contient énormément de questions sur des problèmes pratiques qui sont d'une grande valeur quand on rencontre un problème similaire. Les sujets de ce forum ont tendance à faire partie des premiers résultats lors d'une recherche au sujet de MicroPython sur votre moteur de recherche favori.

D'autres ressources sur des sujets plus spécifiques existent. Une des plus remarquables à mes yeux est le recueil de tutoriels traitant de l'asynchrone écrits par Peter Hinch, un utilisateur avancé de MicroPython. Ces tutoriels sont des compléments utiles aux ressources généralistes sur le sujet[1].

Pour ceux qui préfèrent le papier, il existe également des livres, notamment chez O'Reilly.

1. Voir par exemple l'article Découvrons la programmation asynchrone en Python publié sur Zeste de Savoir.

Les ressources en français sont malheureusement plus rares. Le site micropython.fr en contient quelques-unes.

4.2. Contribuer à MicroPython

Le développement de MicroPython est très actif. Le dépôt GitHub de MicroPython centralise le développement du cœur de MicroPython, tandis que le projet de bibliothèque standard modulaire et légère a son propre dépôt .

Comme beaucoup de projet open source, MicroPython a également des forks, comme CircuitPython , qui se focalise sur les débutants et la facilité d'apprentissage.

Conclusion

Si vous pensiez que Python était lourd, j'espère que cet article vous aura montré que le langage sait aussi s'adapter aux environnements plus contraints de l'embarqué!

5. Liens

— site officiel de MicroPython,
— dépôt GitHub pour MicroPython ,
— dépôt GitHub pour la bibliothèque micropython-lib ,
— Documentation de MicroPython .

Merci à @artragis pour la validation de cet article.

Liste des abréviations

ADC Analog to Digital Converter. 4

CAN Controller Area Network. 4

DAC Digital to Analog Converter. 4

LED Light Emitting Diode. 4