MicroPython entre dans le monde de l'Arduino

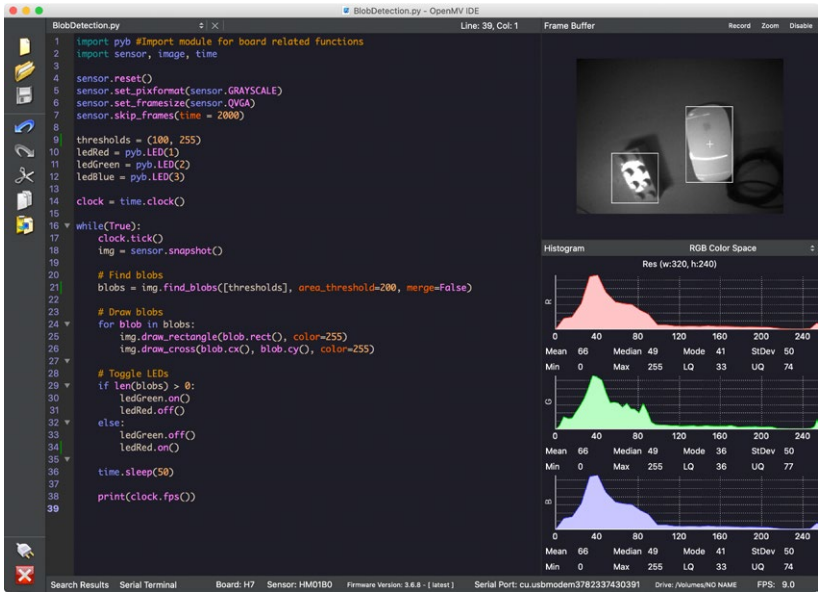
Stuart Cording (Elektor)

MicroPython vient d'arriver dans l'EDI Arduino, offrant la 1ère alternative significative à la programmation en C et C++. Pourquoi cet émoi ? Est-il si facile à utiliser ? À qui ce langage assez récent sur les µcontrôleurs profite-t-il ? Pour en savoir plus, Elektor s'est entretenu avec Sebastian Romero, le responsable des contenus chez Arduino.

À propos de Sebastian Romero (Responsable du contenu @Arduino)

Technologue créatif, Sebastian Romero est responsable du contenu chez Arduino. Il conçoit des processus interactifs et enseigne avec empathie. Lui et son équipe sont chargés de créer de passionnantes expériences d'apprentissage pour aider des millions d'ingénieurs, concepteurs, artistes, amateurs et étudiants à innover.

Dès la création d'Arduino, début des années 2000, le C et le C++ sont à la base de son développement logiciel. Grâce à une structure de programme prédéfinie par les fonctions `setup()` et `loop()`, les débutants en développement de logiciels embarqués furent guidés dans la configuration de leur carte et l'exécution de leur application dans une boucle. Voici qu'un nouveau langage est disponible. MicroPython est une version légèrement dépouillée de Python, un langage de programmation interprété polyvalent destiné aux µcontrôleurs. Mais, pourquoi utiliser un langage interprété sur du matériel destiné à des applications en temps réel ?



Détection de Blob sur Arduino Portenta H7 dans OpenMV.

« Par sa simplicité MicroPython est bien adapté aux débutants, les enseignants furent donc parmi les premiers à nous le demander », explique Sebastian Romero.

En C, on interagit avec les registres des µcontrôleurs bien plus facilement qu'en assembleur et la programmation orientée objet (POO) du C++ produit un code plus concis avec moins d'erreurs. Cependant en C, l'analyse syntaxique des chaînes de caractères est ardue et rien n'est prévu pour gérer les formats Web actuels tels que HTTP, JSON [1] ou RegEx [2] (expression régulière). L'enseignement actuel étant axé sur l'interaction avec Internet et les services Web, le C/C++ fut mis de côté au profit de langages comme Python qui simplifie le codage de telles applications.

« Il s'ensuit qu'un enseignant en Python, préférera s'y tenir pour aborder le sujet des µcontrôleurs », explique Sebastian.

Mais il n'y a pas que les enseignants. Les créateurs ont pu accéder à la vaste gamme de cartes compatibles avec MicroPython proposées par d'autres fabricants :

par ex. *ESP32, Raspberry Pi Pico, pyboard*. L'industrie se tourne aussi de plus en plus vers MicroPython. L'existence de bibliothèques Python explique en partie la croissance rapide de l'apprentissage machine (*ML*). Peu d'équipes compétentes en Python, souhaitent passer à C/C++ pour transférer leur modèle et leur application ML sur un µcontrôleur. Elles préfèrent s'en tenir à une seule pile de développement. L'autre problème est lié à la formation : trouver des programmeurs C/C++ devient plus difficile car les ingénieurs sont en majorité formés en Python.

MicroPython peut s'exécuter immédiatement après chaque modification, contrairement aux croquis C/C++ qu'il faut d'abord compiler et télécharger.

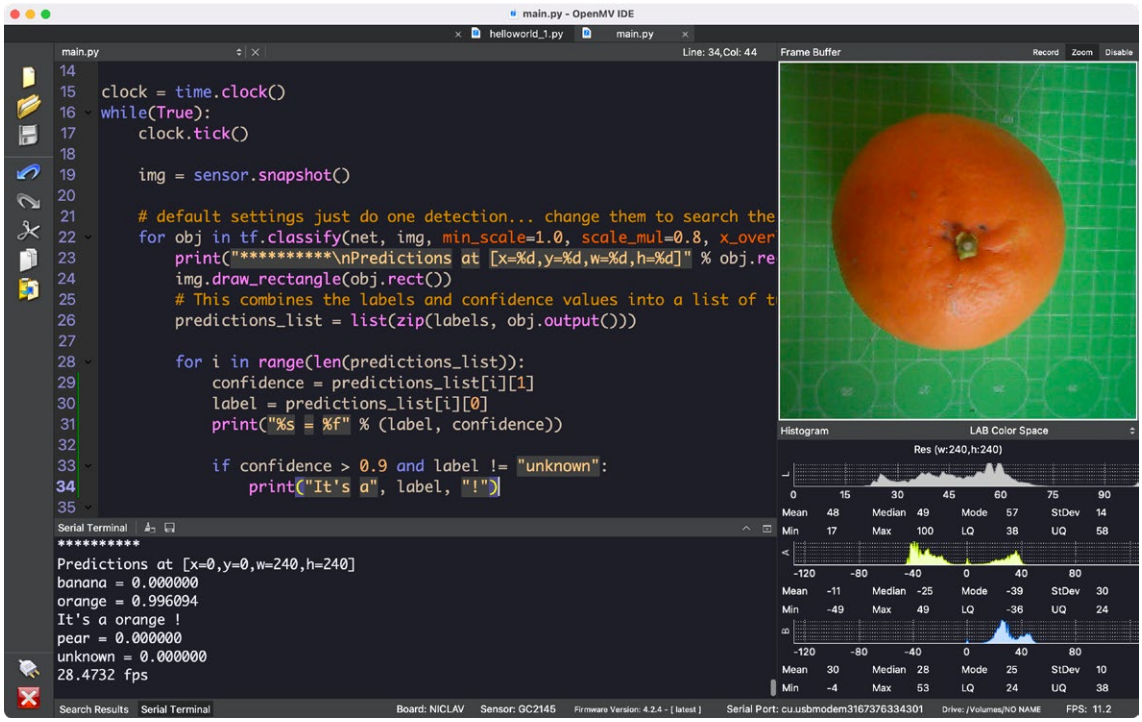
Quelles différences entre MicroPython et Python ?

Au départ, Python fut imaginé par *Guido van Rossum* en 1989 [3]. Il est conçu pour être ludique, explicite plutôt qu'implicite, simple et pour aboutir à un code lisible. En 2013, *Damien George* lança avec succès une campagne Kickstarter [4] pour livrer une version conçue de A à Z pour les µcontrôleurs ainsi que le matériel pyboard pour l'exécuter. *Micro Python*, ainsi nommé à l'époque, promettait un langage de script « permettant de faire clignoter des LED sans effort, lire des tensions, etc. ». Un PC reconnaît un µcontrôleur compatible USB comme clé USB et peut y télécharger du code. L'appareil peut aussi apparaître comme périphérique série, offrant une ligne de commande dite REPL (*Read-Evaluate-Print-Loop*).

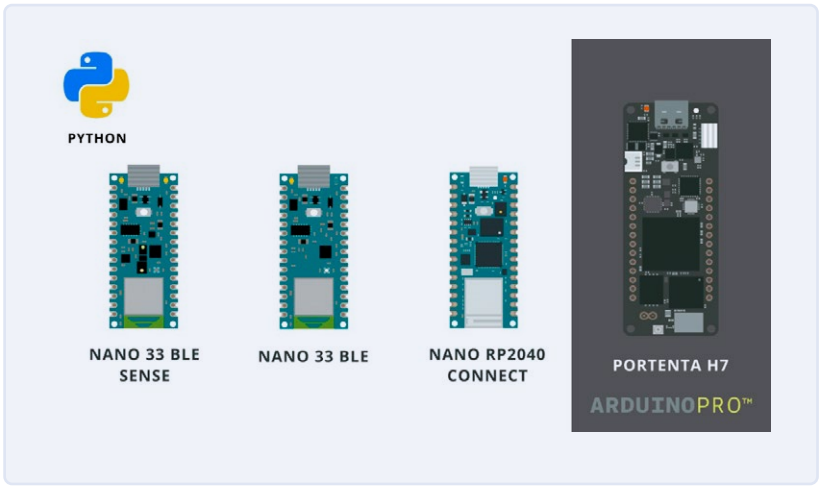
Comme prévisible, MicroPython nécessite une modeste quantité de mémoire tant pour s'exécuter que pour prendre en charge le code téléchargé et l'interpréter. Mais, même si 128 Ko de flash et 8 Ko de SRAM suffisent, cela n'autorise pas des fonctionnalités convenables. C'est pourquoi la plupart des cartes MicroPython ont un µcontrôleur doté d'au moins 256 Ko de flash et 16 Ko de SRAM. Corollairement, un dispositif doté d'un processeur assez puissant, cadencé à 50 MHz ou plus et offrant une gamme respectable de périphériques est préférable.

Sebastian nous confie en citant *Jim Mussared*, ingénieur embarqué de micropython.org, « Je reste toujours surpris de voir tout ce qui est réalisable avec seulement 16 Ko de SRAM ». « Pour les projets d'initiation, la plupart des étudiants n'utilisent que peu de mémoire, puis la taille du code augmente et les projets se complexifient. »

Souvent, ces projets MicroPython évaluent des données de capteurs et mettent en œuvre des machines à états.



Classement d'images sur Arduino Nicla Vision dans OpenMV.



Cela dit, la SRAM n'est pas utilisée que pour stocker des variables. Elle abrite aussi du *bytecode* (par ex. des modules importés) à exécuter par la machine virtuelle (VM) de MicroPython. La manipulation de données volumineuses, comme des chaînes de caractères, ou bien la création ou destruction de nombreux objets risquent de ne pas laisser assez de SRAM pour exécuter le compilateur. Mais, une fois le code mature, le bytecode peut être précompilé et stocké en système de fichiers flash ou implémenté en tant que bytecode gelé pour utiliser encore moins de SRAM [5].

MicroPython est plus délicat que Python lors de la saisie du code car l'espacement est rigoureux. Les utilisateurs apprennent vite que des fonctionnalités par défaut (comme l'implémentation complète de la bibliothèque standard) sont absentes à cause des limites du matériel.

Environnement de développement pour MicroPython Arduino

L'équipe d'Arduino a choisi le MicroPython maintenu par OpenMV, car les nouveaux Arduino équipés de caméra bénéficient ainsi des fonctions de vision industrielle et de la prise en charge intégrée de Tensor Flow Lite fournis par OpenMV [6]. L'utilisateur dispose donc d'un matériel et d'un EDI matures avec une prise en charge de qualité. OpenMV a été créé pour accéder à la vision industrielle sur les µcontrôleurs. Cela correspond bien au désir de nombreux utilisateurs de créer des applications ML basées sur les images.

OpenMV IDE (*Integrated Development Environment*, EDI en français) fournit l'EDI pour les codeurs MicroPython plutôt que l'EDI Arduino traditionnel. Il offre les fonctionnalités centrales attendues, comme la fenêtre de développement du code et le terminal série. Il prend aussi en charge les applications de vision industrielle, comme l'affichage d'un tampon d'images et des histogrammes des plages de couleur et de luminosité. En outre, un outil intégré permet de télécharger des images de caméra directement dans le studio Edge Impulse pour faciliter l'entraînement d'un modèle d'apprentissage automatique.

Mais la manière dont le code est développé et déployé constitue la plus grande innovation.

Sébastien poursuit : « MicroPython peut s'exécuter immédiatement après chaque modification, contrairement aux croquis C/C++ qu'il faut d'abord compiler et télécharger. Cela accélère considérablement le développement et rapproche la tâche de codage de celle de Python ».

Le REPL est un autre avantage qui permet d'exécuter de courts scripts ou de tester des fonctions individuelles directement sur le contrôleur cible.

Prise en charge matérielle Arduino pour MicroPython

En tout, cinq cartes Arduino prennent MicroPython en charge : Nano 33 BLE et Nano 33 BLE Sense ; Nano RP2040 Connect ; Portenta H7 et Nicla Vision. Sur la plupart, il faut débuter par une m. à j. du µlogiciel pour télécharger le runtime MicroPython en flash. Nous avons constaté que ce processus est simple, mais également bien documenté [7]. Les cartes Nano 33 ont une étape de préparation qui utilise l'EDI Arduino ; les autres sont immédiatement reconnues par OpenMV et programmées avec le µlogiciel nécessaire.
Dans OpenMV, l'application est écrite sous forme d'un script Python qui est téléchargé sur la carte cible. Entre la programmation et l'exécution du code, il n'y a qu'un clic sur le bouton *Play*.

Et ensuite ?

Quel futur pour Arduino maintenant que MicroPython est là ? Il est naturel de craindre que l'introduction de MicroPython relègue le classique croquis C/C++ au rang d'artefact historique. Mais ce n'est ni souhaité ni prévu. Si l'exécution en temps réel est une exigence, C/C++ sera toujours l'arme de prédilection.

« La demande de prise en charge de Python sur les µcontrôleurs croît, notamment dans l'industrie de pointe qui utilise déjà une pile Python et travaille au développement d'applications ML », explique Sebastian.

La prise en charge de MicroPython étend les options accessibles aux utilisateurs d'Arduino, sans les remplacer. Graduellement, les développeurs C/C++ trouve-

ront des cartes de taille habituelle prenant en charge MicroPython.

« Avec la plupart des cartes Arduino classiques, MicroPython aurait des fonctionnalités très limitées et ce ne serait donc pas une option raisonnable », ajoute Sebastian.

Avec MicroPython, les utilisateurs peuvent comme autrefois perpétuer le succès d'Arduino [8]. Le code C/C++ contribué depuis 15 ans est toujours maintenu et utilisé pour des pilotes en combinaison avec MicroPython. Des liens sont ensuite utilisés pour lier MicroPython à ce code de base. Pour ceux qui souhaitent contribuer au développement de MicroPython [10], GitHub héberge aussi OpenMV [9], un projet auquel l'équipe Arduino participe également.

On peut voir MicroPython comme un ajout à l'écosystème Arduino actuel, son adoption étant motivée par le succès de Python, le langage préféré pour les applications ML et l'interaction avec les services en nuage. Les µcontrôleurs étant de plus en plus souvent cadencés à des centaines de MHz et dotés d'une vaste mémoire, le passage à un langage interprété sera souvent considéré comme indolore. Bien sûr, il existe des exceptions où l'exactitude et la précision en temps réel sont cruciales, et le C/C++ sera toujours là en cas de besoin. Mais, pour l'instant, les enseignants et les étudiants ont l'avantage significatif de pouvoir transférer les connaissances de Python aux µcontrôleurs (avec en prime la syntaxe simplifiée et de la lisibilité du code). En outre, les développeurs industriels peuvent développer des applications avec un seul langage.

220415-04 — VF : Yves Georges

À propos de l'auteur

Stuart Cording est ingénieur et journaliste. Il a plus de 25 ans d'expérience dans l'industrie électronique. Un grand nombre de ses articles Elektor récents sont ici : www.elektormagazine.com/cording. Pour Elektor, il anime aussi le mensuel de diffusion en direct, Elektor Engineering Insights (www.elektormagazine.com/eei), et enseigne à Elektor Academy (www.elektormagazine.com/elektor-academy).

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (stuart.cording@elektor.com) ou contactez Elektor (redaction@elektor.fr).

LIENS

[1] JSON, Wikipédia : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
[2] Expression régulière, Wikipédia : https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re
[3] Python, Wikipédia : https://fr.wikipedia.org/wiki/Python_(langage)
[4] D. George, « Micro Python: Python for microcontrollers », Kickstarter, 2016 : www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers
[5] « MicroPython sur les µcontrôleurs » : https://docs.micropython.org/en/latest/reference/constrained.html
[6] OpenMV : https://openmv.io/
[7] K. Soderby, « Python with Arduino Boards », Arduino, 2022 : https://docs.arduino.cc/learn/programming/arduino-and-python
[8] Arduino, GitHub : https://github.com/arduino
[9] OpenMV, GitHub : https://github.com/openmv
[10] Repo officiel de µPython : https://github.com/micropython/micropython