Introduction à MicroPython

Programmation embarquée avec Python

Houssem-eddine LAHMER

Plan

Qu'est-ce que MicroPython ?

- MicroPython est une implémentation légère de Python 3 optimisée pour fonctionner sur des microcontrôleurs et des environnements contraints
- Créé par Damien George en 2013 (financé via Kickstarter)
- Écrit en C pour être compact et efficace
- Permet de programmer des systèmes embarqués avec la simplicité et l'expressivité de Python
- Open source (licence MIT)

Historique de MicroPython

- **2013** : Lancement du projet sur Kickstarter
- **2014** : Première version stable et création de la PyBoard
- **2016** : Support pour ESP8266
- **2017** : Support pour ESP32
- **2019** : CircuitPython (fork par Adafruit) gagne en popularité
- **2020+** : Adoption large dans l'IoT et l'éducation

MicroPython vs C/C++ embarqué

Avantages de MicroPython

- Développement rapide
- Syntaxe simple et intuitive
- REPL (Read-Eval-Print Loop)
- Pas de compilation
- Prototypage accéléré
- Idéal pour l'apprentissage

Avantages de C/C++

- Performances optimales
- Contrôle précis des ressources
- Empreinte mémoire réduite
- Accès direct au hardware
- Idéal pour les projets critiques
- Meilleure gestion de l'énergie

Cas d'usage de MicroPython

- **Prototypage rapide** : développement itératif et tests
- **Éducation** : apprentissage de l'électronique et programmation
- **IoT** : capteurs, collecte de données, contrôleurs
- **Domotique** : contrôle d'appareils domestiques
- **Robotique** : contrôle de moteurs, servos, capteurs
- **Projets créatifs** : art interactif, wearables
- **Maintenance** : remplacement de firmware industriel

Plateformes supportées

- **PyBoard** : La plateforme officielle de MicroPython
- **ESP8266** : Microcontrôleur Wi-Fi économique
- **ESP32** : Successeur de l'ESP8266 avec Bluetooth et plus de puissance
- **STM32** : Microcontrôleurs basés sur ARM Cortex-M
- **BBC micro :bit** : Plateforme éducative
- **Raspberry Pi Pico** : Microcontrôleur RP2040 à double cœur
- **WiPy, LoPy** : Plateformes IoT
- **LEGO Mindstorms** : Robotique éducative

Focus sur ESP32 et ESP8266

ESP8266

- Microcontrôleur Wi-Fi économique
- CPU Tensilica L106 80MHz
- 32/64 Ko RAM pour instructions
- 80 Ko RAM pour données
- GPIO limités
- Prix très abordable

ESP32

- Dual-core Tensilica Xtensa LX6 240MHz
- 520 Ko RAM
- Wi-Fi + Bluetooth
- RTC, ADC, DAC, I2C, SPI, etc.
- Plus de GPIO
- Cryptographie matérielle

PyBoard

Caractéristiques

- Plateforme de référence MicroPython
- Microcontrôleur STM32F405RG
- ARM Cortex-M4 168MHz
- 1MB Flash, 192KB RAM
- Accéléromètre 3 axes
- Lecteur de carte microSD
- Connectivité USB
- Interface complète : UART, I2C, SPI, ADC, PWM

[Emplacement pour image de PyBoard]

Firmware MicroPython

- Le firmware MicroPython est le programme qui s'exécute sur le microcontrôleur
- Il contient :
  - L'interpréteur Python
  - Les bibliothèques de base
  - Le système de fichiers
  - Les drivers spécifiques à la plateforme
- Différentes versions pour différentes plateformes
- Téléchargeable depuis `micropython.org/download`

Flashage du firmware

Méthodes de flashage selon la plateforme :

- **ESP8266/ESP32** : Utilisation d'esptool.py
- **PyBoard** : Glisser-déposer le fichier .dfu
- **Raspberry Pi Pico** : Glisser-déposer le fichier .uf2
- **micro :bit** : Glisser-déposer le fichier .hex

Exemple avec esptool pour ESP32 :

```
pip install esptool
esptool.py --port /dev/ttyUSB0 erase_flash
esptool.py --port /dev/ttyUSB0 --baud 460800
    write_flash --flash_size=detect 0 esp32-20210902-v1
    .17.bin
```

Accès au REPL

REPL (Read-Eval-Print Loop)

- Interface interactive pour exécuter du code Python
- Accessible via connexion série ou WebREPL
- Idéal pour tester et débugger

Connexion série :

```
# Linux/macOS
screen /dev/ttyUSB0 115200

# Windows avec PuTTY
# COM port: COM3, Baud rate: 115200
```

Commandes utiles du REPL

- **Ctrl+A** : Début de ligne
- **Ctrl+E** : Fin de ligne
- **Ctrl+B** : Retour (gauche)
- **Ctrl+C** : Interrompre (Keyboard Interrupt)
- **Ctrl+D** : Soft reset (en mode REPL)
- **Ctrl+F** : Avancer (droite)
- **Flèches haut/bas** : Historique des commandes

REPL spécial :

- **Ctrl+A** : Entrée dans mode RAW REPL
- **Ctrl+E** : Entrée dans mode PASTE (pour coller du code multi-lignes)

Outils de développement

- **Thonny IDE** : Interface graphique avec support MicroPython intégré
- **Visual Studio Code + Extensions** :
  - Extension "Pymakr"
  - Extension "MicroPython"
- **uPyCraft** : IDE dédié à MicroPython
- **rshell** : Outil CLI pour gérer les fichiers et le REPL
- **ampy** : Outil Adafruit pour transférer des fichiers
- **mpfshell** : Shell de fichiers pour MicroPython

Thonny IDE

Avantages :

- Interface simple et intuitive
- Support natif de MicroPython
- Explorateur de fichiers intégré
- REPL interactif
- Gestionnaire de flashage du firmware
- Debugger
- Multi-plateforme (Windows, macOS, Linux)

Installation : `thonny.org`

Utilisation de Thonny avec MicroPython

1. Installer Thonny
2. Connecter votre carte MicroPython via USB
3. Dans Thonny, sélectionner "Outils" ¿ "Options"
4. Sous "Interpréteur", choisir "MicroPython (ESP32)" ou autre selon votre carte
5. Sélectionner le port série correct
6. Cliquer sur "OK"
7. Le REPL MicroPython apparaît dans la fenêtre du shell
8. Vous pouvez maintenant écrire et exécuter du code MicroPython

Structure d'un projet MicroPython

```
/mon_projet
  |-- main.py          # Ex cut   apr s boot.py
  |-- boot.py          # Ex cut   au d marrage
  |-- lib/             # Biblioth ques
  |    |-- sensors.py
  |    |-- display.py
  |    |-- wifi_manager.py
  |-- config.py        # Configuration
  |-- data/            # Fichiers de donn es
```

- **boot.py** : Configuration initiale (Wi-Fi, système de fichiers)
- **main.py** : Code principal exécuté automatiquement
- Les autres fichiers sont importés selon besoin

Rappels Python

- MicroPython est un sous-ensemble de Python 3
- Syntaxe identique, mais bibliothèque standard réduite
- Types de base supportés :
  - `int`, `float`, `str`, `bool`
  - `list`, `tuple`, `dict`, `set`
- Structures de contrôle :
  - `if`, `elif`, `else`
  - `for`, `while`
  - `try`, `except`, `finally`
- Fonctions, classes et modules supportés

Exemples de code Python

```
# Variables et types de base
nombre = 42
pi = 3.14159
texte = "Bonjour MicroPython!"
active = True

# Listes et dictionnaires
mesures = [23.5, 24.1, 22.8, 25.2]
config = {"ssid": "MonWiFi", "password": "secret"}

# Structures de contr le
if nombre > 40:
    print("Nombre   lev  ")
else:
    print("Nombre bas")

for valeur in mesures:
    print(f"Temp rature: {valeur} C ")
```

Fonctions en Python

```
# D finition de fonction
def moyenne(liste):
    return sum(liste) / len(liste)
# Utilisation
temp_moyenne = moyenne(mesures)
print(f"Temp rature moyenne: {temp_moyenne:.1f} C ")
```

Classes en Python

```
# Classe
class Capteur:
    def __init__(self, pin, type="temp rature"):
        self.pin = pin
        self.type = type
        self.valeurs = []

    def mesurer(self):
        # Code pour lire la valeur du capteur
        valeur = 25.0  # Exemple
        self.valeurs.append(valeur)
        return valeur
```

Particularités de MicroPython

- **Optimisé pour les systèmes embarqués** :
  - Empreinte mémoire réduite
  - Exécution efficace
- **Bibliothèque standard réduite** :
  - Sous-ensemble de Python 3
  - Modules essentiels uniquement
- **Modules spécifiques pour l'accès au hardware** :
  - `machine` : GPIO, I2C, SPI, ADC, etc.
  - `network` : Wi-Fi, Bluetooth
  - `esp` : Fonctions spécifiques ESP
- **Préfixe "u" pour les modules** : `uos`, `utime`, `usocket`

Modules spécifiques à MicroPython

- **machine** : Contrôle du hardware
  - `Pin` : Contrôle des GPIO
  - `I2C`, `SPI` : Protocoles de communication
  - `ADC`, `DAC` : Conversion analogique-numérique
  - `PWM` : Modulation de largeur d'impulsion
  - `Timer` : Temporisateurs
- **network** : Connectivité réseau
- **esp32/esp8266** : Fonctions spécifiques aux plateformes
- **utime** : Fonctions temporelles
- **uos** : Système de fichiers

Exemple : Configuration GPIO et LED

```
from machine import Pin
import time
# Configuration d'une LED sur la broche 2
led = Pin(2, Pin.OUT)
# Clignotement simple
for _ in range(10):
    led.value(1)  # Allumer
    time.sleep(0.5)
    led.value(0)  #  teindre
    time.sleep(0.5)
```

Exemple : Gestion d'un bouton

```
from machine import Pin
import time
# Configuration d'un bouton sur la broche 0
bouton = Pin(0, Pin.IN, Pin.PULL_UP)
# Configuration d'une LED sur la broche 2 (r f rence
    )
led = Pin(2, Pin.OUT)
# Lecture de l' tat  du bouton
while True:
    if bouton.value() == 0:  # Bouton press  (logique
     invers e)
        print("Bouton press !")
        led.value(1)
    else:
        led.value(0)
    time.sleep(0.1)
```

Exemple : PWM pour contrôle de LED

```
from machine import Pin, PWM
import time

# Cr ation d'un objet PWM sur la broche 2
pwm = PWM(Pin(2))
pwm.freq(1000)  # Fr quence de 1kHz

# Variation de l'intensit  lumineuse
while True:
    # Augmentation progressive
    for i in range(0, 1024, 10):
        pwm.duty(i)  # 0-1023
        time.sleep(0.05)

    # Diminution progressive
    for i in range(1023, -1, -10):
        pwm.duty(i)
        time.sleep(0.05)
```

Exemple : Lecture analogique

```
from machine import ADC, Pin
import time

# Configuration d'un ADC sur la broche 36 (ESP32)
adc = ADC(Pin(36))
adc.atten(ADC.ATTN_11DB)  # Plage 0-3.3V

# Lecture p riodique
while True:
    val = adc.read()  # 0-4095
    tension = val * 3.3 / 4095
    print(f"Valeur: {val}, Tension: {tension:.2f}V")
    time.sleep(1)
```

Exemple : Configuration I2C

```
from machine import I2C, Pin
import time
# Configuration de l'I2C
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
# Recherche des p riph riques connect s
devices = i2c.scan()
print(f"P riph riques I2C trouv s: {[hex(d) for d
    in devices]}")
```

Exemple : Communication avec BME280

```
from machine import I2C, Pin
# Exemple avec un capteur BME280
BME280_ADDR = 0x76
# Registres du BME280
REG_ID = 0xD0
REG_CTRL_MEAS = 0xF4
# Configuration de l'I2C (r f rence)
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
# Lecture de l'ID du capteur
chip_id = i2c.readfrom_mem(BME280_ADDR, REG_ID, 1)
print(f"ID du capteur: {hex(chip_id[0])}")
```

diviser en deux slides

Fonctions et classes

```
# D finition de fonction
def moyenne(liste):
    return sum(liste) / len(liste)
# Utilisation
temp_moyenne = moyenne(mesures)
print(f"Temp rature moyenne: {temp_moyenne:.1f} C ")
# Classe
class Capteur:
    def init(self, pin, type="temp rature"):
        self.pin = pin
        self.type = type
        self.valeurs = []

    def mesurer(self):
        # Code pour lire la valeur du capteur
        valeur = 25.0  # Exemple
        self.valeurs.append(valeur)
        return valeur
```

Exemple : Contrôle des GPIO

```
from machine import Pin
import time
# Configuration d'une LED sur la broche 2
led = Pin(2, Pin.OUT)
# Clignotement simple
for _ in range(10):
    led.value(1)  # Allumer
    time.sleep(0.5)
    led.value(0)  #  teindre
    time.sleep(0.5)
# Configuration d'un bouton sur la broche 0
bouton = Pin(0, Pin.IN, Pin.PULL_UP)
# Lecture de l' tat  du bouton
while True:
    if bouton.value() == 0:  # Bouton press  (logique
     invers e)
        print("Bouton press !")
        led.value(1)
    else:
```

Exemple : Communication I2C

```
from machine import I2C, Pin
import time
# Configuration de l'I2C
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
# Recherche des p riph riques connect s
devices = i2c.scan()
print(f"P riph riques I2C trouv s: {[hex(d) for d
    in devices]}")
# Exemple avec un capteur BME280
BME280_ADDR = 0x76
# Registres du BME280
REG_ID = 0xD0
REG_CTRL_MEAS = 0xF4
# Lecture de l'ID du capteur
chip_id = i2c.readfrom_mem(BME280_ADDR, REG_ID, 1)
print(f"ID du capteur: {hex(chip_id[0])}")
```

Exemple : Wi-Fi et requêtes HTTP

```
import network
import urequests
import time
# Configuration du Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect("SSID", "PASSWORD")
# Attente de la connexion
while not wifi.isconnected():
    print("Connexion Wi-Fi en cours...")
    time.sleep(1)
print("Connect !")
print(f"Adresse IP: {wifi.ifconfig()[0]}")
# Requ te HTTP
response = urequests.get("http://worldtimeapi.org/api/
    ip")
data = response.json()
print(f"Heure actuelle: {data['datetime']}")
response.close()
```

Exemple : Configuration Wi-Fi

```
import network
import time
# Configuration du Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect("SSID", "PASSWORD")
# Attente de la connexion
while not wifi.isconnected():
    print("Connexion Wi-Fi en cours...")
    time.sleep(1)
print("Connect !")
print(f"Adresse IP: {wifi.ifconfig()[0]}")
```

Exemple : Requêtes HTTP

```
import urequests
import network
# R f rence au module Wi-Fi (supposant d j
    connect )
wifi = network.WLAN(network.STA_IF)
if wifi.isconnected():
    print(f"Connect  au r seau. IP: {wifi.ifconfig()
    [0]}")
    # Requ te HTTP
    response = urequests.get("http://worldtimeapi.org/
    api/ip")
    data = response.json()
    print(f"Heure actuelle: {data['datetime']}")
    response.close()
else:
    print("Erreur: Wi-Fi non connect ")
```

Gestion des ressources

MicroPython fonctionne dans un environnement contraint :

- Mémoire limitée (RAM, Flash)
- Puissance de calcul réduite
- Alimentation souvent par batterie

Bonnes pratiques :

- Utiliser `gc.collect()` pour forcer le garbage collector
- Éviter les grandes structures de données
- Préférer les `bytearray` aux `list` pour les données binaires
- Utiliser le mode deep sleep pour économiser l'énergie
- Libérer les ressources après utilisation (fermer les fichiers)
- Éviter les récursions profondes

Gestion de la mémoire

```
import gc

# Afficher la m moire disponible
def mem_info():
    print(f"M moire libre: {gc.mem_free()} octets")
    print(f"M moire allou e: {gc.mem_alloc()} octets
    ")

mem_info()

# Exemple de consommation de m moire
grand_tableau = [0] * 10000
mem_info()

# Lib ration
del grand_tableau
gc.collect()
mem_info()
```

Mode Deep Sleep

```
import machine
import time

# Fonction pour aller en deep sleep
def deep_sleep(ms):
    # Configurer le r veil
    rtc = machine.RTC()
    rtc.irq(trigger=rtc.ALARM0, wake=machine.DEEPSLEEP
    )
    rtc.alarm(rtc.ALARM0, ms)

    # Entrer en deep sleep
    print(f"Entr e en deep sleep pour {ms} ms")
    machine.deepsleep()

# D tecter la cause du r veil
if machine.reset_cause() == machine.DEEPSLEEP_RESET:
    print("R veil apr s deep sleep")
```

Ressources et documentation

- **Documentation officielle** : `docs.micropython.org`
- **GitHub** : `github.com/micropython/micropython`
- **Forum** : `forum.micropython.org`
- **MicroPython Cookbook** (Packt Publishing)
- **Getting Started with MicroPython** (O'Reilly)
- **Communautés** : Reddit r/MicroPython, Stack Overflow

Conclusion

- MicroPython offre une approche simplifiée pour la programmation embarquée
- Idéal pour le prototypage rapide et les applications IoT
- Supporté sur de nombreuses plateformes matérielles
- Écosystème en croissance avec de nombreuses bibliothèques
- Complémentaire au C/C++ pour certains cas d'usage
- Excellente option pour l'apprentissage et l'enseignement

Questions ?