Introduction à MicroPython

Syntaxe et Programmation pour Microcontrôleurs

Housem-eddine LAHMER

15 mai 2025

Plan du cours

Qu'est-ce que MicroPython ?

- MicroPython est une implémentation légère et efficace du langage Python 3
- Conçu pour fonctionner sur des microcontrôleurs et des systèmes embarqués
- Créé par Damien George en 2013 (projet Kickstarter)
- Compatible avec un sous-ensemble significatif de la bibliothèque standard Python

Pourquoi utiliser MicroPython ?

Avantages :

- Syntaxe Python simple et lisible
- Développement rapide et prototypage
- REPL interactif (Read-Eval-Print Loop)
- Grande communauté et documentation

Cas d'utilisation :

- IoT (Internet des Objets)
- Robotique
- Automatisation
- Projets éducatifs
- Prototypes industriels

Plateformes compatibles

- **Pyboard** - La carte officielle MicroPython
- **ESP32/ESP8266** - WiFi et Bluetooth
- **STM32** - Microcontrôleurs STM
- **Raspberry Pi Pico** - Basé sur RP2040
- **BBC micro :bit** - Orienté éducation
- **Teensy** - Haute performance
- **nRF52** - Bluetooth Low Energy

Mise en place de l'environnement

1. **Télécharger le firmware** adapté à votre carte
2. **Flasher le firmware** sur la carte :
   - ESP32/ESP8266 : esptool.py
   - Raspberry Pi Pico : mode bootloader (bouton BOOTSEL)
   - STM32 : DFU, ST-Link ou autres outils spécifiques
3. **Connecter** à la carte via série USB
4. **Accéder au REPL** (terminal série à 115200 bauds)

Outils de développement

- **Thonny IDE** - Interface simple avec support intégré pour MicroPython
- **Mu Editor** - Éditeur Python simplifié pour débutants
- **VS Code + extensions** - PyMakr, MicroPython, etc.
- **rshell, ampy** - Outils en ligne de commande
- **WebREPL** - Accès via navigateur (ESP8266/ESP32)

La fameuse LED clignotante – Blink

```
1  from machine import Pin
2  import time
3
4  led = Pin(2, Pin.OUT)  # LED sur la broche 2
5
6  while True:
7      led.value(1)           # Allumer la LED
8      time.sleep(0.5)        # Attendre 500 ms
9      led.value(0)           #  teindre la LED
10     time.sleep(0.5)        # Attendre 500 ms
```

Types de données – Numériques

```
1 # Types numériques
2 a = 42              # int
3 b = 3.14159         # float
```

Types de données – Texte et booléens

```
1 # Texte
2 text = "MicroPython"   # str
3
4 # Bool en
5 flag = True            # bool
```

Types de données – Collections

```
# Listes, tuples, dictionnaires
my_list  = [1, 2, 3, 4]
my_tuple = (1, 2, 3)
my_dict  = {"pin": 5, "mode": "OUT"}

# Vérification de type
print(type(a))  # <class 'int'>
```

Structures de contrôle – Conditions

```
1 sensor_value = 512
2 if sensor_value > 700:
3     print("Valeur    leve   ")
4 elif sensor_value > 300:
5     print("Valeur moyenne")
6 else:
7     print("Valeur faible")
```

Structures de contrôle – Boucles `for`

```
for i in range(5):   # 0, 1, 2, 3, 4
    print(i)
```

Structures de contrôle – Boucles `while`

```
counter = 0
while counter < 5:
    print(counter)
    counter += 1
```

Structures de contrôle – Gestion d'exceptions

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division par z ro impossible")
```

Fonction blink

```
# Fonction simple
def blink(pin_num, times, delay=0.5):
    led = Pin(pin_num, Pin.OUT)
    for _ in range(times):
        led.value(1)
        time.sleep(delay)
        led.value(0)
        time.sleep(delay)
    return True
```

Classe Button

```
class Button:
    def __init__(self, pin_num, pull=None):
        self.btn = Pin(pin_num, Pin.IN, pull)

    def is_pressed(self):
        return self.btn.value() == 0

    def wait_for_press(self):
        while not self.is_pressed():
            time.sleep_ms(10)
```

Module machine : GPIO et ADC

Le module machine permet d'accéder au matériel embarqué.

```
from machine import Pin, ADC

# GPIO num rique
button = Pin(5, Pin.IN, Pin.PULL_UP)
led    = Pin(2, Pin.OUT)

# Entr e analogique (ESP32)
adc = ADC(Pin(34))
analog_value = adc.read()  # 0-4095
```

Module machine : PWM, I²C, SPI, Timer

```
from machine import PWM, I2C, SPI, Timer

# PWM (sortie analogique)
pwm = PWM(Pin(13))
pwm.freq(1000)      # Fr quence en Hz
pwm.duty(512)       # Rapport cyclique 0-1023

# I2C
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
devices = i2c.scan()  # Liste des adresses d tect es

# SPI
spi = SPI(1, baudrate=1000000, polarity=0, phase=0,
          sck=Pin(18), mosi=Pin(23), miso=Pin(19))

# Timer (callbacks p riodiques)
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC,
         callback=lambda t: led.toggle())
```

Temporisation

```
1 import time
2
3 # Attentes bloquantes
4 time.sleep(1)          # 1 seconde
5 time.sleep_ms(50)      # 50 millisecondes
6 time.sleep_us(100)     # 100 microsecondes
```

Chronomètres

```
import time

# D marrage
start = time.ticks_ms()

# ... op rations ...

# Calcul du temps    coul    (en ms)
elapsed = time.ticks_diff(time.ticks_ms(), start)
```

Interruptions GPIO

```
1  from machine import Pin
2
3  # Handler appel  lors de l'appui
4  def button_pressed(pin):
5      print("Bouton appuy !")
6
7  # Configuration du bouton avec interruption
8  button = Pin(0, Pin.IN, Pin.PULL_UP)
9  button.irq(trigger=Pin.IRQ_FALLING,
10             handler=button_pressed)
```

Timer matériel

```
1  from machine import Timer
2
3  # Callback toggle LED
4  def timer_callback(t):
5      led.value(not led.value())
6
7  # Timer p riodique toutes les 1 s
8  timer = Timer(0)
9  timer.init(period=1000, mode=Timer.PERIODIC,
10             callback=timer_callback)
```

Informations système

```
1 import os
2 import sys
3
4 # Infos firmware et plateforme
5 print(os.uname())
6 print(sys.implementation)
```

Système de fichiers

```
1 import os
2
3 # Liste des fichiers sur la flash
4 print(os.listdir())
5
6 #   criture   dans un fichier
7 with open("data.txt", "w") as f:
8     f.write("Donn es de capteur\n")
```

Gestion de la mémoire

```
1  import gc
2
3  # Avant collecte
4  free_before = gc.mem_free()
5
6  # Forcer la collecte
7  gc.collect()
8
9  # Apr s collecte
10 free_after = gc.mem_free()
11
12 print(f"M moire lib r e : {free_after - free_before
       } octets")
```

Wi-Fi : Mode Station

```
import network, time

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('SSID', 'PASSWORD')

timeout = 10
while not wlan.isconnected() and timeout > 0:
    time.sleep(1)
    timeout -= 1

if wlan.isconnected():
    print('Connect , IP:', wlan.ifconfig()[0])
else:
    print(' chec  de connexion')
```

Wi-Fi : Point d'accès

```
1 import network
2
3 ap = network.WLAN(network.AP_IF)
4 ap.active(True)
5 ap.config(essid='MicroPython-AP',
6             password='micropythoN')
7 print('AP d marr , SSID:', ap.config('essid'))
```

Serveur Web

```
1  import socket
2
3  def web_page():
4      html = """<!DOCTYPE html>
5  <html><body>
6    <h1>MicroPython Web Server</h1>
7    <p>LED: {status}</p>
8    <a href="/?led=on">ON</a>
9    <a href="/?led=off">OFF</a>
10 </body></html>"""
11     return html.format(status="ON" if led.value() else
       "OFF")
12
13 s = socket.socket()
14 s.bind(('', 80))
15 s.listen(5)
16 #      boucle   d  acceptation   et  r ponse
```

MQTT pour IoT

```
from umqtt.simple import MQTTClient
import ubinascii, machine

client_id = ubinascii.hexlify(machine.unique_id())
server    = "broker.hivemq.com"
topic_sub = b"maison/salon/lumiere/commande"
topic_pub = b"maison/salon/lumiere/status"

def mqtt_callback(topic, msg):
    print(f"Re u {msg} sur {topic}")
    if msg == b"ON":  led.value(1)
    if msg == b"OFF": led.value(0)

client = MQTTClient(client_id, server)
client.set_callback(mqtt_callback)
client.connect()
client.subscribe(topic_sub)

def publish(value):
```

Capteur DS18B20 (OneWire)

```
import onewire, ds18x20
from machine import Pin
import time

# Initialisation
ds_pin    = Pin(4)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
roms      = ds_sensor.scan()  # Recherche des capteurs

# Lecture de temp rature
def read_temperature():
    ds_sensor.convert_temp()
    time.sleep_ms(750)          # Temps de conversion
    for rom in roms:
        temp = ds_sensor.read_temp(rom)
        print(f"Temp rature: {temp} C ")
    return temp
```

Capteur analogique (LDR)

```
from machine import ADC, Pin

# Configuration ADC
light_sensor = ADC(Pin(36))
light_sensor.width(ADC.WIDTH_12BIT)  # R solution
    0-4095
light_sensor.atten(ADC.ATTN_11DB)    # Plage 0  3  .3 V

# Lecture
light_value = light_sensor.read()
print(f"Luminosit : {light_value}")
```

Afficheur OLED (SSD1306)

```
from machine import Pin, I2C
import ssd1306

# Initialisation I2C et OLED
i2c = I2C(0, scl=Pin(22), sda=Pin(21))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

# Affichage de texte
oled.fill(0)
oled.text("MicroPython",    0,  0)
oled.text("Temp:",          0, 16)
oled.text(f"{25.5} C",      0, 32)
oled.show()
```

Formes et pixels sur OLED

```
# Dessiner des formes simples
oled.rect(10, 10, 20, 20, 1)          # Rectangle vide
oled.fill_rect(50, 10, 20, 20, 1)     # Rectangle plein
oled.line(0, 0, 127, 63, 1)           # Ligne diagonale
oled.pixel(64, 32, 1)                 # Pixel isol
oled.show()
```

Contrôle Servomoteur

```
1  from machine import Pin, PWM
2  import time
3
4  # PWM     50 Hz pour servo
5  servo = PWM(Pin(13), freq=50)
6
7  def set_servo_angle(angle):
8      # 0.5  2  .5 ms       duty 20  120    / 20 ms
9      duty = int(20 + (angle/180)*100)
10     servo.duty(duty)
11
12 # Balayage  d angles
13 for angle in range(0, 181, 10):
14     set_servo_angle(angle)
15     time.sleep(0.1)
```

Contrôle Moteur DC (L298N)

```
1  from machine import Pin, PWM
2
3  # Broches du pont en H
4  motor1A      = Pin(27, Pin.OUT)
5  motor1B      = Pin(26, Pin.OUT)
6  motor_enable = PWM(Pin(25), freq=1000)  # Contr le
       vitesse
7
8  def motor_forward(speed):
9      motor1A.value(1)
10     motor1B.value(0)
11     motor_enable.duty(speed)  # 0  1023
12
13 def motor_stop():
14     motor1A.value(0)
15     motor1B.value(0)
```

Modes d'économie d'énergie

```
1  import machine
2  import esp32  # Sp cifique    ESP32
3
4  # Deep sleep - Consommation minimale
5  def enter_deep_sleep(sleep_time_ms):
6      print("Entr e en deep sleep pour", sleep_time_ms,
       "ms")
7      machine.deepsleep(sleep_time_ms)
8
9  # Light sleep - R veil plus rapide
10 def enter_light_sleep(sleep_time_ms):
11     print("Entr e en light sleep pour", sleep_time_ms
       , "ms")
12     esp32.light_sleep(sleep_time_ms)
13
14 # V rifier la cause du r veil
15 wake_reason = machine.reset_cause()
16 if wake_reason == machine.DEEPSLEEP_RESET:
17     print("R veil depuis deep sleep")
```

Optimisation de la mémoire

```
import gc

# V rifier la m moire disponible
free_mem = gc.mem_free()
print(f"M moire libre: {free_mem} octets")

# Forcer le garbage collector
gc.collect()

# Conseils d'optimisation:
# 1.  viter les grandes listes et dictionnaires
# 2. Lib rer les variables inutilis es
big_list = [i for i in range(1000)]
big_list = None  # Permettre au GC de lib rer la
    m moire
gc.collect()

# 3. Utiliser des g n rateurs plut t que des listes
def generate_values():
```

Station météo

```
from machine import Pin, I2C, ADC
import time
import dht
import ssd1306

# Capteurs
dht_sensor = dht.DHT22(Pin(4))
light_sensor = ADC(Pin(36))
light_sensor.atten(ADC.ATTN_11DB)

#  cran  OLED
i2c = I2C(0, scl=Pin(22), sda=Pin(21))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

while True:
    try:
        # Lecture des capteurs
        dht_sensor.measure()
        temp = dht_sensor.temperature()
```

Système d'arrosage automatique

```
1  from machine import Pin, ADC, PWM
2  import time
3  import network
4  import urequests
5
6  # Capteur d'humidit  du sol
7  soil_sensor = ADC(Pin(32))
8  soil_sensor.atten(ADC.ATTN_11DB)
9
10 # Pompe    eau (via transistor/relais)
11 pump = Pin(26, Pin.OUT)
12
13 # LED d'indication
14 led_red = Pin(25, Pin.OUT)     # Sol sec
15 led_green = Pin(27, Pin.OUT)   # Sol humide
16
17 THRESHOLD_DRY = 2500    # Valeur     ajuster selon
       capteur
18 THRESHOLD_WET = 1500    # Valeur     ajuster selon
```

Techniques de débogage

- **Utiliser le REPL interactif** pour tester du code à la volée
- **print() stratégique** - Afficher variables et états
- **LED de débogage** - Indiquer visuellement l'état du programme
- **Gestion d'exceptions** - Capturer et enregistrer les erreurs
- **Exporter les journaux** via connexion série ou fichier
- **Mode verbeux** - Activer/désactiver selon besoin

```
1 # Exemple de fonction de journalisation
2 DEBUG = True
3
4 def log(message, level="INFO"):
5     if DEBUG or level == "ERROR":
6         timestamp = time.time()
7         print(f"[{timestamp}] {level}: {message}")
8
```

Bonnes pratiques de programmation

- **Structure modulaire** - Séparer le code en fichiers
- **Fonction principale** `main()` - Point d'entrée clair
- **Gestion propre des ressources** - Libérer les GPIO, fermer les connexions
- **Boucle principale avec watchdog** - Redémarrer en cas de blocage
- **Persistance paramètres** - Utiliser des fichiers JSON
- **Économiser l'énergie** - Utiliser les modes veille
- **Éviter le polling continu** - Préférer les interruptions

```
# Structure recommandée pour un projet MicroPython
# boot.py - Exécuté au démarrage
# main.py - Point d'entrée principal
# lib/    - Modules personnalisés et bibliothèques
# config/ - Fichiers de configuration

```

Structure d'un projet complet

```
# boot.py - Configuration initiale
import machine
import gc

# Configuration du mat riel
machine.freq(240000000)  # ESP32     240MHz
gc.enable()  # Activer le garbage collector

# Importer les param tres depuis config.json
try:
    import ujson as json
    with open('config.json') as f:
        config = json.load(f)
except:
    config = {"ssid": "default", "password": "default"
    }

# main.py - Point d'entr e principal
def main():
```

Ressources d'apprentissage

- **Documentation officielle :** `https://docs.micropython.org/`
- **Forums :**
  - Forum MicroPython : `https://forum.micropython.org/`
  - Reddit r/micropython : `https://www.reddit.com/r/micropython/`
- **Tutoriels :**
  - Tutoriels Random Nerd : `https://randomnerdtutorials.com/`
  - PyBoard Tutorials : `https://docs.micropython.org/en/latest/pyboard/tutorial/`
- **Livres :**
  - "Programming with MicroPython" - Nicholas Tollervey
  - "MicroPython for the Internet of Things" - Charles Bell
- **GitHub :** Exemples de projets et bibliothèques

Conclusion

- MicroPython est un excellent choix pour :
  - Prototypage rapide
  - Projets IoT
  - Applications éducatives
  - Systèmes embarqués avec contraintes modérées
- Points forts :
  - Syntaxe Python familière
  - Grande communauté
  - Compatibilité avec de nombreuses plateformes
  - Cycles de développement accélérés
- Questions à considérer :
  - Performance vs facilité d'utilisation
  - Contraintes mémoire pour projets complexes
  - Consommation d'énergie pour applications autonomes

Exercice 1 : LED RGB contrôlée par PWM

Objectif : Créer une LED RGB qui change progressivement de couleur.

```
1  from machine import Pin, PWM
2  import time
3
4  # Initialisation des broches PWM pour LED RGB
5  red = PWM(Pin(13), freq=1000, duty=0)
6  green = PWM(Pin(12), freq=1000, duty=0)
7  blue = PWM(Pin(14), freq=1000, duty=0)
8
9  def set_color(r, g, b):
10     # Valeurs entre 0-1023
11     red.duty(r)
12     green.duty(g)
13     blue.duty(b)
14
15 # Animation arc-en-ciel
16 def rainbow_cycle(cycles=3):
17     for _ in range(cycles):
18         # Rouge    jaune
```

Exercice 2 : Datalogger avec carte SD

Objectif : Enregistrer des données de capteurs sur une carte SD.

```
import machine
import os
import sdcard
import time

# Configuration SPI pour carte SD
spi = machine.SPI(1,
                  baudrate=10000000,
                  polarity=0,
                  phase=0,
                  sck=machine.Pin(18),
                  mosi=machine.Pin(23),
                  miso=machine.Pin(19))

cs = machine.Pin(5, machine.Pin.OUT)

# Monter la carte SD
sd = sdcard.SDCard(spi, cs)
```

Exercice 3 : Mini serveur API REST

Objectif : Créer une API REST pour contrôler vos périphériques.

```
import socket
import json
from machine import Pin

# GPIO     contr ler
pins = {
    "led": Pin(2, Pin.OUT),
    "relay": Pin(4, Pin.OUT),
    "fan": Pin(16, Pin.OUT)
}

# Cr er un socket serveur
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('0.0.0.0', 80))
s.listen(5)
s.settimeout(2)

def handle_request(request):
```

Comparaison avec Arduino

MicroPython

- + Développement rapide
- + REPL interactif
- + Collections de données avancées
- + Gestion d'exceptions
- - Performance brute inférieure
- - Empreinte mémoire plus grande
- - Moins de bibliothèques

Arduino

- + Performance maximale
- + Faible empreinte mémoire
- + Vaste écosystème de bibliothèques
- + Compatibilité matérielle étendue
- - Développement plus verbeux
- - Pas d'interpréteur interactif
- - Structures de données limitées

Comparaison avec CircuitPython et autres

CircuitPython

- Fork d'Adafruit de MicroPython
- Focalisé sur les débutants
- Pilotes intégrés pour matériel Adafruit
- Mode de stockage USB natif
- Documentation très accessible
- Moins de plateformes supportées

Autres alternatives

- **Espruino :** JavaScript pour microcontrôleurs
- **Lua :** RTOS et NodeMCU
- **Rust :** Embarqué pour performance critique
- **mJS :** Mongoose JavaScript
- **TinyGo :** Go pour microcontrôleurs

Cas 1 : Monitoring industriel

Scénario : Surveillance de machines industrielles via ESP32 et MicroPython

```
# Monitoring avec capteurs multiples et alertes

# Capteurs
temp_sensor = DS18X20(...)      # Temp rature
current_sensor = ACS712(...)    # Courant  lectrique
vibration = MPU6050(...)        # Vibrations

THRESHOLDS = {
    "temp_max": 85.0,           #  C
    "current_max": 15.0,        # A
    "vibration_max": 2.5        # g
}

class MachineMonitor:
    def __init__(self, machine_id):
        self.machine_id = machine_id
        self.alert_state = False
```

Cas 2 : Objets connectés (IoT)

Scénario : Réseau de capteurs connectés au cloud

```
import network
import urequests
import ujson
import time
from machine import Pin, ADC, deepsleep

# Configuration
API_ENDPOINT = "https://api.example.com/data"
API_KEY = "your_api_key"
WIFI_SSID = "YourNetwork"
WIFI_PASS = "YourPassword"
SLEEP_TIME = 15 * 60 * 1000  # 15 minutes en ms

# Capteurs
soil_moisture = ADC(Pin(32))
soil_moisture.atten(ADC.ATTN_11DB)
rain_sensor = Pin(27, Pin.IN)
battery_adc = ADC(Pin(33))  # Pour mesurer la tension
```

Cas 3 : Interface utilisateur avec écran tactile

Scénario : Thermostat intelligent avec écran tactile

```
from machine import Pin, I2C, Timer
import ili9341  # Biblioth que d' cran  LCD
import xpt2046  # Biblioth que d' cran  tactile
import time
import ujson

# Configuration mat rielle
spi = machine.SPI(2, baudrate=40000000)
display = ili9341.Display(spi, dc=Pin(4), cs=Pin(5),
    rst=Pin(32))
touch = xpt2046.Touch(spi, cs=Pin(15))

# Capteur de temp rature
i2c = I2C(0, scl=Pin(22), sda=Pin(21))
temp_sensor = ...  # Capteur sp cifique

# Configuration thermostat
config = {
```

TinyML avec MicroPython

- **TinyML** = Machine Learning pour microcontrôleurs
- Utilise des modèles très optimisés et quantifiés
- Cas d'utilisation :
  - Reconnaissance de mots-clés
  - Détection d'anomalies
  - Classification d'images simple
  - Analyse de séries temporelles
- Bibliothèques :
  - `ulab` - NumPy-like pour MicroPython
  - `tf-micro` - TensorFlow Lite pour microcontrôleurs
  - Modèles préentraînés et quantifiés

```
# Exemple avec ulab
import ulab as np

# Classifier simple
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def predict(features, weights, bias):
```

Edge Computing avec MicroPython

- **Edge Computing** = Traitement des données proche de la source
- Avantages :
  - Réduction de la bande passante
  - Latence réduite
  - Autonomie accrue
  - Fonctionnement hors-ligne
- Techniques :
  - Prétraitement des données
  - Compression
  - Analyse statistique locale
  - Prise de décision autonome

```
1 # Exemple de pr traitement pour   conomiser   la bande
    passante
2 WINDOW_SIZE = 30  # 30 secondes de donn es
3
4 def process_window(values):
5     """Calcule des statistiques sur une fen tre de
    donn es"""
6     if not values:
```

Futures tendances de MicroPython

- **Performance :** Optimisations continues du moteur d'exécution
- **Matériel :** Support pour davantage de microcontrôleurs
- **Bibliothèques :** Croissance de l'écosystème de packages
- **IoT :** Meilleure intégration avec les services cloud
- **Sécurité :** Améliorations pour l'IoT sécurisé
- **Interfaces :** Outils de développement plus sophistiqués
- **IA embarquée :** Plus d'outils pour TinyML

Merci pour votre attention !

Questions ?

Contact : votre.email@example.com
https://github.com/votrecompte