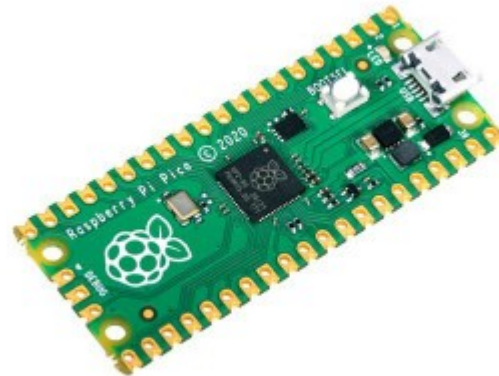


Micropython sur micro:bit – raspberry pi pico - ESP32

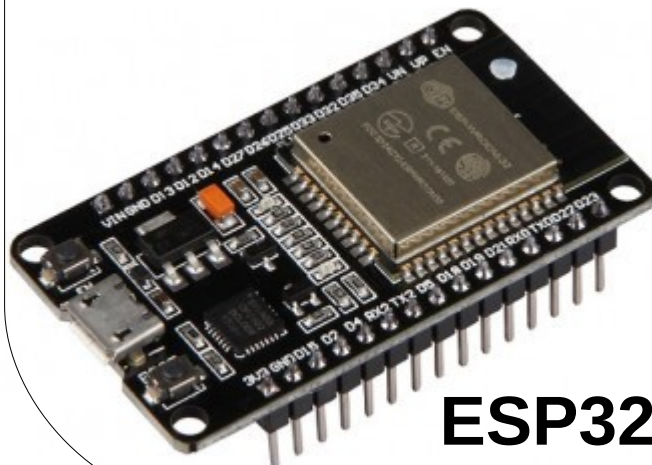
 **Micropython**



Micro:bit



Raspberry Pico



ESP32

VS



Arduino

Frédéric PLACIN
IREM info Bordeaux
14 avril 2022

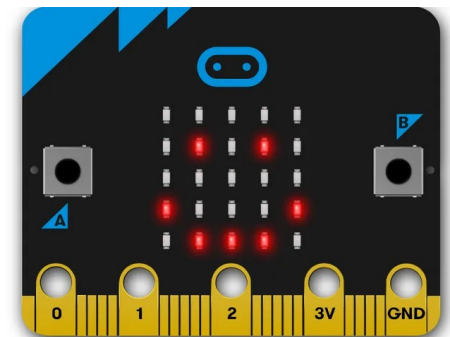
Micropython au lycée

Objectif : traiter la partie « Informatique embarquée et objets connectés » en SNT au lycée avec une **continuité pédagogique**

Problématique : Les lycées sont équipés d'arduino Uno (pour la physique) mais **la programmation arduino se fait en C++, très différente de python...**



Solution : travailler avec des cartes sous python : utiliser **micropython** avec des **micro:bit**



Un peu d'histoire : Retour sur l'Arduino

Présentation: Les premiers arduino sortent en 2005, avec un environnement de programmation IDE proche de Processing.
(Processing est en java, Arduino en C++ mais très peu de différences pour les élèves)

A screenshot of the Arduino IDE (version 1.0) window. The title bar reads "clignote | Arduino 1.0". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". The toolbar contains icons for opening, saving, compiling, and uploading. The main text area shows a C++ sketch named "clignote" with the following code:

```
void setup() {  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(13, LOW);  
  delay(1000);  
  digitalWrite(13, HIGH);  
  delay(1000);  
}
```

A status bar at the bottom indicates "Done Saving." and "Arduino Uno on COM17".A screenshot of the Processing IDE (version 3.5.3) window. The title bar reads "sketch_220406b | Processing 3.5.3". The menu bar includes "Fichier", "Modifier", "Sketch", "Dépanner", "Outils", and "Aide". The toolbar contains icons for running, stopping, and saving. The main text area shows a Java sketch named "sketch_220406b" with the following code:

```
void setup()  
{  
  size(200,200);  
}  
  
void draw()  
{  
  ellipse(mouseX,mouseY,10,10);  
}
```

A preview window on the right shows a sketch of a spiral made of small circles. The bottom status bar includes "Console", "Erreurs", and "Updates".

Le problème est que maintenant on travaille en PYTHON !!

Un peu d'histoire : Retour sur l'Arduino

Arduino Uno est bien équipé pour l'époque :

- Microprocesseur ATmega328P 8bits cadencé à 16MHz
- 2KB SRAM, 32KB FLASH, 1KB EEPROM
- 14 I/O digitale
- 6 entrées analogique 10 bits / 6 PWM
- Communication : I2C, SPI, UART
- Alimentation par USB (5V) ou par alimentation extérieure (7-12V)

Un Arduino Uno coûte ~ 20€ et ses clones ~ 10€

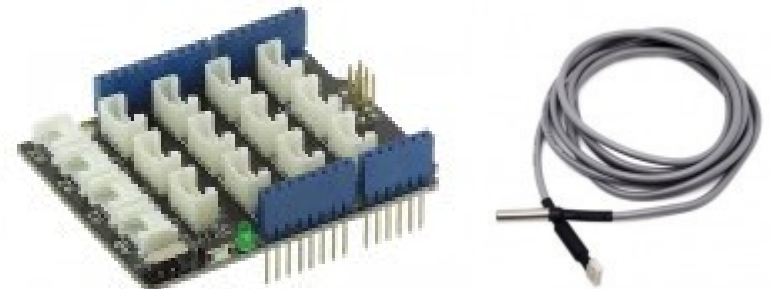
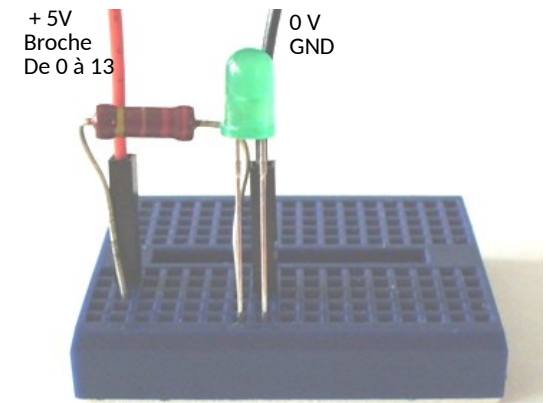
Un peu d'histoire : Retour sur l'Arduino

Arduino seul :

- On peut jouer avec une LED sur le port 13

Pour aller plus loin :

- Plaque d'essai, composants et broches...
mais c'est galère en classe
- Shield : exemple module grove + capteur...
mais c'est cher



La carte micro:bit

La carte micro:bit est un produit de la BBC (Anglaise), deux versions sont disponibles : la V1 et la V2

Microprocesseur ARM Cortex-M4 32bits cadencé à 64MHz (V2)

128KB RAM, 512KB FLASH

19 I/O digitale

6 entrées analogique 10 bits / 3 PWM

Communication : I2C, SPI, UART, **Bluetooth**

2 boutons poussoir + 3 touches sensibles

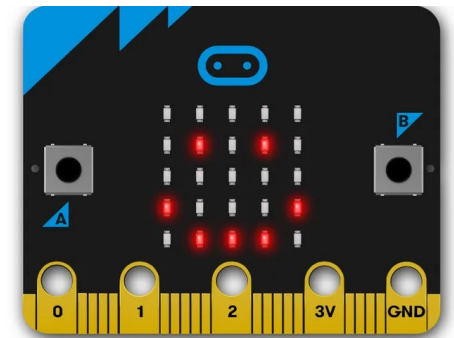
Un pad de 25 LED adressables

Microphone, haut-parleur buzzer (V2)

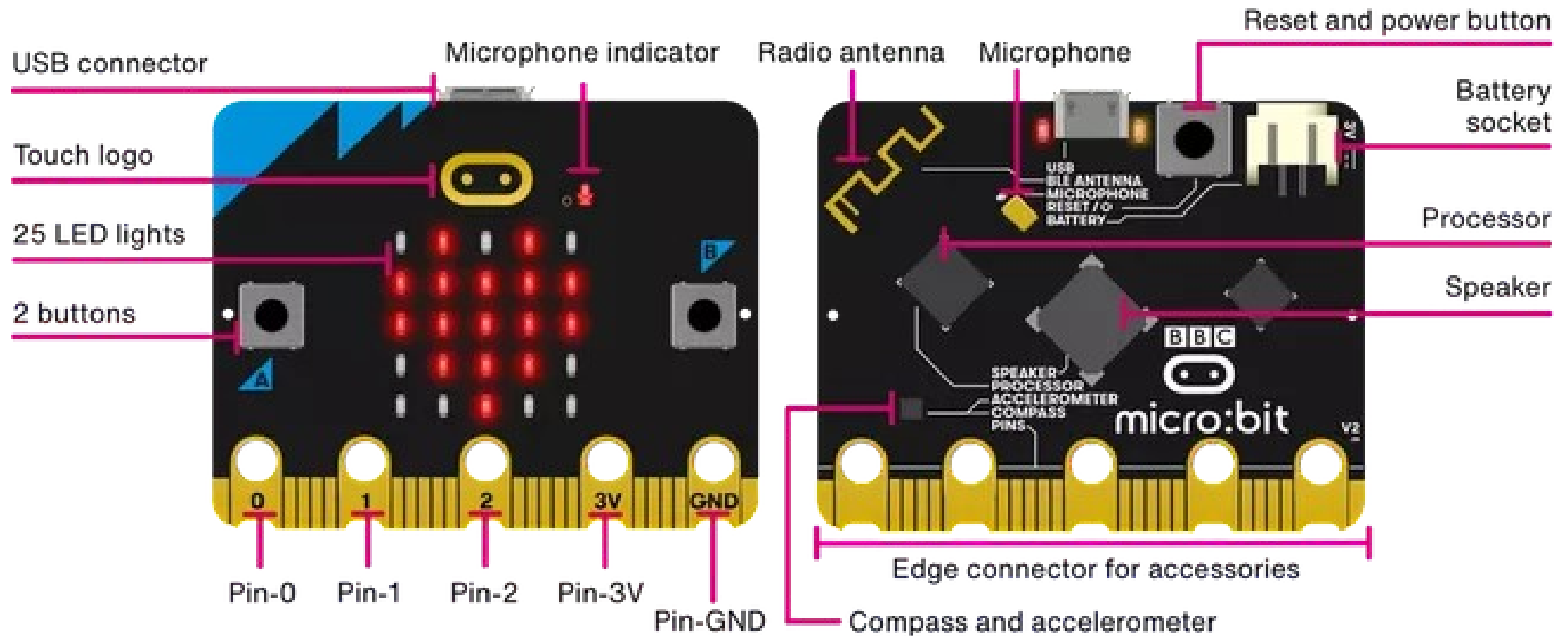
Accéléromètre, boussole, thermomètre

Alimentation par USB (5V) ou par alimentation extérieure (**1,8-3,6V**)

Une micro:bit V2 coûte ~ 20€



La carte micro:bit



Une micro:bit fonctionne en micropython

micropython

Micropython a été développé en 2013 par un physicien australien :
Damien P. George

Site officiel : <https://micropython.org/>

Version actuelle (janvier 2022) : v1.18

Minimum recommandé : Flash 256 ko – RAM 16 ko - 80MHz

Micropython reprend les standards de python

Micropython est fait pour fonctionner sur plusieurs microcontrôleurs
avec suffisamment de mémoire (arduino uno n'en a pas assez).

Il joue le rôle de système d'exploitation de la carte.

Attention, il y a un fork : **circuitpython** de la société Adafruit

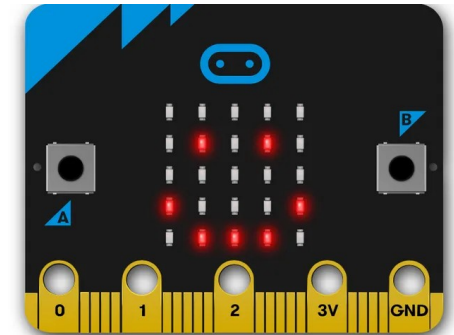
Le pb c'est que beaucoup de librairie de matériel sont en
circuitpython (pb de compatibilité...)

Programmation de la carte micro:bit

Il existe de nombreuses façon de programmer la micro:bit : microbit.org

Éditeur en ligne **MakeCode** de Microsoft

- Programmation à blocs, ou python, ou javascript
- Avantage : en ligne, multi-langages
- Inconvénient : Microsoft



Editeur en ligne **python de microbit**

- Avantage : en ligne, python
- Inconvénient : ça bugue parfois

Editeurs installés sur PC :

- **Mu-editor** : un peu lourd à installer (pour les ELIB) mais pratique.
- **Thonny** : attention, que à partir de la version 3.3.10

Ma préférence pour le moment va à Thonny : simple, efficace, et compatible avec d'autres cartes (Raspberry Pico, ESP32)

Programmation de la carte micro:bit avec Thonny

Attention : il faut Thonny 3.3.10 minimum

Configuration de Thonny :

Dans **Outils/option/interpréteur** choisir **Micropython (BBC micro:bit)**

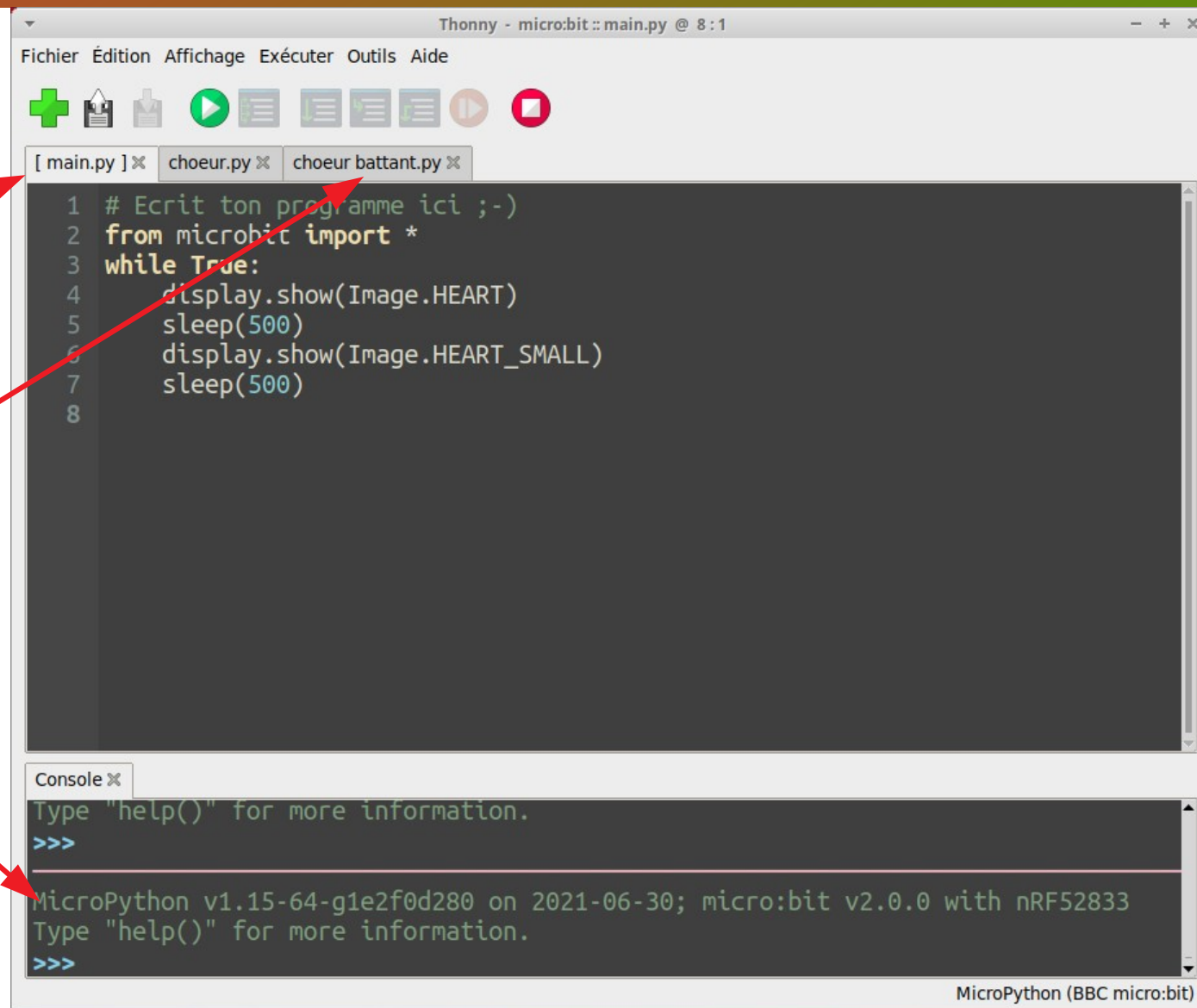
Choisir le port s'il ne l'a pas trouvé (nécessaire au lycée)

A partir de ce moment, les scripts python sont effectués sur le micropython de la micro:bit, même ceux de la console.

Utilisation sous Thonny :

- dans la console en ligne de commande
- avec un programme python en lançant le script (bouton vert)
- en enregistrant le programme sur la microbit sous le nom main.py pour un programme embarqué, avec les dépendances si besoin...

Programmation de la carte micro:bit avec Thonny



Programme
sur la microbit

Programme
sur l'ordinateur

Console du
micropython

Interpréteur en
cours

micro:bit en SNT

En SNT, on peut aborder python à l'aide de la micro:bit dès les premières séances de l'année :

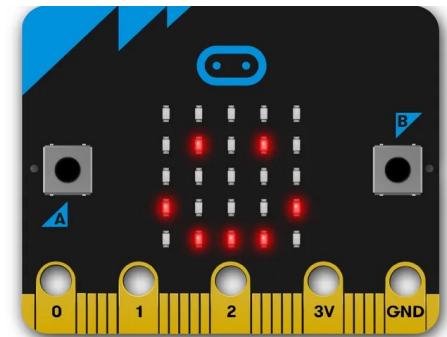
Au lycée Magendie, on a tester sur 4 classes avec 3/4 séances de TP d'une heure (ave Mu-editor et microbit V1):

1ère séance : prise en main, boucle while infinie, images

2ème séance : interactivité au bouton poussoir, structure conditionnelle

3ème séance : variables, boucle for, nombres aléatoires

4ème séance : IOT (objet connectés) – radio(Bluetooth)

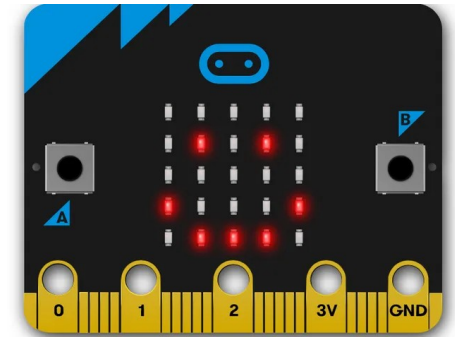


Après 4 séances : les élèves adorent et en redemandent mais au-delà de 3 mois il n'en reste pas grand-chose sur python...

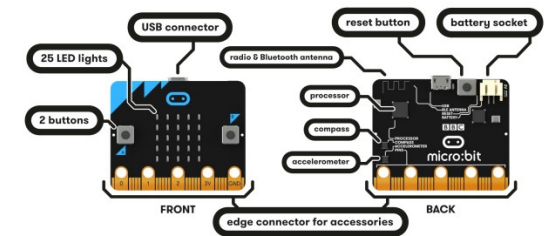
micro:bit en SNT – 1ère séance

La première séance : prise en main, boucle while infinie, images (durée 1h)

- Distribution papier d'un récapitulatif des fonctions microbit (à ramener à chaque séance)
- Présentation de **mu-editor**
- Parler de bibliothèque : **from microbit import ***
- Aide au premier programme : **display.show** afficher un smiley (ou autre chose)
- Faire essayer la fonction **scroll** - introduire la **boucle infinie**
- Introduction du **sleep** – alternance d'image : exercice coeur battant
- Exercice création d'image en expliquant le principe de la **matrice de pixel**



Fiche des fonctions micro:bit en python



- ligne d'importation obligatoire : **from microbit import ***
- afficher une image : **display.show(Image.HAPPY)**
d'autres images préenregistrées :
HEART; HEART_SMALL; HAPPY; SMILE; SAD; CONFUSED;
ANGRY; ASLEEP; SURPRISED; SILLY; FABULOUS; MEH; YES;
NO; CLOCK12; CLOCK11; CLOCK10; CLOCK9; CLOCK8;
CLOCK7; CLOCK6; CLOCK5; CLOCK4; CLOCK3; CLOCK2;
CLOCK1; ARROW_N; ARROW_NE; ARROW_E; ARROW_SE;
ARROW_S; ARROW_SW; ARROW_W; ARROW_W; TRIANGLE;
TRIANGLE_LEFT; CHESSBOARD; DIAMOND;
DIAMOND_SMALL; SQUARE; SQUARE_SMALL; RABBIT;
COW; MUSIC_CROTCHET; MUSIC_QUAVER;
MUSIC_QUAVERS; PITCHFORK; XMAS; PACMAN; TARGET;
TSHIRT; ROLLERSKATE; DUCK; HOUSE; TORTOISE;
BUTTERFLY; STICKFIGURE; GHOST; SWORD; GIRAFFE;
SKULL; UMBRELLA; SNAKE
- créer une image - chaque LED a une intensité lumineuse comprise entre 0 et 9 :

```
mon_image = Image("05050:"  
                  "05050:"  
                  "05050:"  
                  "05050:"  
                  "05050:")  
display.show(mon_image)
```
- faire défiler un texte : **display.scroll("Bonjour")**
- allumer une led spécifique : **display.set_pixel(x, y, val)**
- pose en millisecondes : **sleep(1000)** pose d'une seconde
- utilisation des boutons A et B : **button_a_is_pressed()** renvoie 0 ou 1
- utilisation des pins 0, 1 et 2 : **pin0.is_touched()** renvoie True ou False
idem pour les touches 1 et 2, en touchant GND aussi
- accéléromètre : **accelerometer.get_x()** donne une valeur de **accelerometer.get_y()** a, a, a, et a, **accelerometer.get_z()** entre -2048 et +2048
accelerometer.current_gesture() prend les valeurs : up, down, left, right, face up, face down, freefall, 3g, 6g, 8g, shake
- boussole : **compass.calibrate()** calibre la boussole
compass.heading() donne la direction 0-360
- bluetooth : **import radio** bibliothèque indispensable
radio.on() allume le bluetooth
radio.off() éteint le bluetooth
radio.config(channel=4, power=7)
radio.send("bonjour") envoie un message
radio.receive() réceptionne un message
- température : **microbit.temperature()** renvoie la température en °C
- codes python : **for**, **while**, **if-else** fonctionnent comme en python
while True :
 display.scroll("boucle infinie")
- aléatoire : **random.random()** rend un nombre entre 0 et 1
random.randint(debut, fin) rend un entier

micro:bit en SNT – 1ère séance

Premier programme :

```
from microbit import *  
display.show(Image.HAPPY)
```

Introduction de la boucle et indentation :

```
from microbit import *  
while True :  
    display.show(Image.HAPPY)
```

Introduction de sleep :

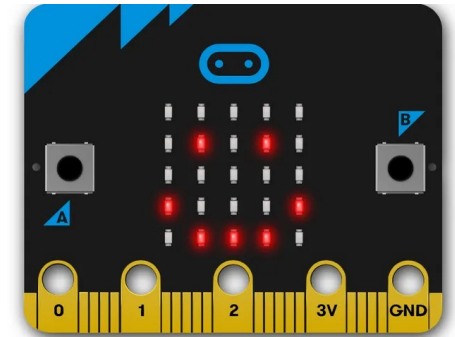
```
from microbit import *  
while True :  
    display.show(Image.HEART)  
    sleep(500)  
    display.show(Image.HEART_SMALL)  
    sleep(500)
```

Création d'image :

```
from microbit import *  
mon_image = Image("05050:"  
                  "05050:"  
                  "05050:"  
                  "99999:"  
                  "09990")  
display.show(mon_image)
```

micro:bit en SNT – 2ème séance

La deuxième séance : interactivité au bouton poussoir, structure conditionnelle (durée 1h)

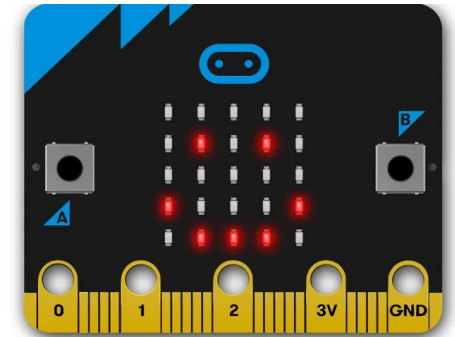


- Présenter la fonction **button_a.is_pressed**
- Présenter la structure **if else** avec un affichage différent selon l'appui bouton
- Exercice : utiliser le bouton B pour un affichage à trois ou quatres conditions
- Pour les rapides : utiliser les touches PIN 0,1 ou 2 associé au GND pour d'autres conditions.
- Pour les doués, utiliser l'accéléromètre en mode « shake »
- Pour ceux qui ont du temps : Scroller la température

micro:bit en SNT – 3ème séance

La troisième séance : variables, boucle for, nombres aléatoires
(durée 1h)

- introduire les nombres aléatoires et les variables
- Présenter la boucle **for**
- Exercice : faire un compte à rebour (9 → 0)
- Exercice : pour reprendre la leçon précédente faire un dè : quand on appui sur le bouton cela change le nombre aléatoirement



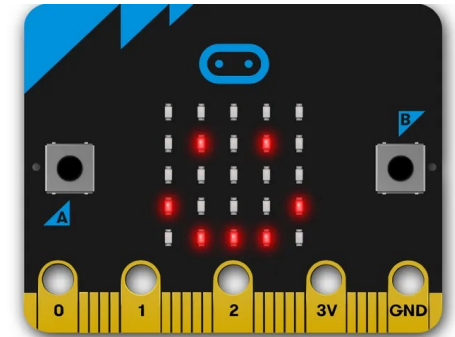
Attention, des programmes 'tout fait' traînent sur le WEB...

- Laisser la fin de séance pour que les élèves reviennent sur les précédentes fonctionnalités (ou alors faire la 4ème séance).

micro:bit en SNT – 4ème séance

La quatrième séance (optionnelle):
communications via bluetooth
(durée 1h)

- Présenter la bibliothèque radio.
- Laisser les élèves pour qu'ils arrivent à capter un signal émis par une micro:bit au bureau prof.
- Les laisser s'envoyer des messages en changeant de canal



micro:bit en NSI

Il n'y a pas vraiment de point du programme pour travailler sur de tel système en **terminale NSI**.

En première NSI : « *Périphérique d'entrée et de sortie – interface homme machine (IMH) – capteur/actionneur : Des activités peuvent être développées sur des objets connectés, des systèmes embarqués ou robots... »*

Plusieurs possibilités :

- refaire de la micro:bit
- concours de robotique ([RSK](#)) programmation en python
- concours [YES WE CODE](#) sur micro:bit ou STM32
- ...

Robot Soccer Kit : le système entier est fourni (sauf les ordi)

Yes We Code : fournissent une grosse mallette de composants.

Pour aller plus loin avec micro:bit

On peut étendre les possibilités de capteur/actionneur avec des shield spéciaux micro:bit.

- shield avec plaque d'essai
- shield I2C ou SPI
- shield GROVE
- shield PCA9685 pour les servomoteur et led

Mais certains systèmes demandent une bibliothèque qu'il faut télécharger sur la micro:bit avant de l'utiliser...



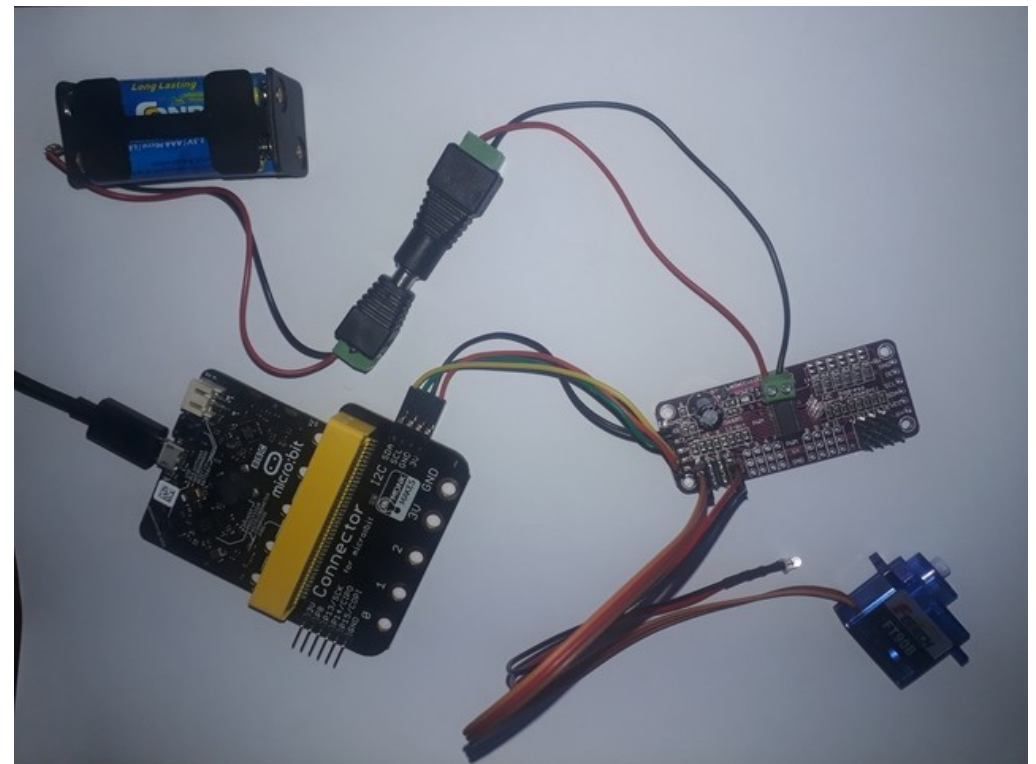
module auto-alimenté PCA9685 avec micro:bit

Un module autoalimenté PCA9685 peut être utile si on utilise plus de 2 moteurs car la micro:bit est limitée à 90mA en sortie (un petit servo-moteur est déjà trop)

Attention :le système demande une bibliothèque qu'il faut télécharger sur la micro:bit avant de l'utiliser...

Ce système gère jusqu'à 16 LED ou servo-moteur

Attention, il faut une bibliothèque à télécharger sur la microbit :
PCA9685.py



module auto-alimenté PCA9685 avec micro:bit

Programme
sur l'ordinateur

librairie sur la
microbit

Programme
sur la microbit

```
Thonny - micro:bit::main.py @ 16:1
Fichier Édition Affichage Exécuter Outils Aide

essai pca9685.py x [ PCA9685.py ] x [ main.py ] x

1 from microbit import sleep, i2c
2 import PCA9685
3
4 pwm = PCA9685.PCA9685(i2c)
5
6 servo_min = 150
7 servo_max = 600
8
9 pwm.set_pwm_freq(60)
10 while True:
11     pwm.set_pwm(0, 0, servo_min)
12     sleep(1000)
13     pwm.set_pwm(0, 0, servo_max)
14     sleep(1000)
15
16

Console x
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.15-64-g1e2f0d280 on 2021-06-30; micro:bit v2.0.0 with nRF52833
Type "help()" for more information.
>>>
```

MicroPython (BBC micro:bit)

Raspberry Pi Pico (RP2)

La carte Raspberry Pi Pico possède un soc RP2040 :

Processeur **dual-core** Arm Cortex-M0+ à **133 MHz**

Gravure : 40 nm

SRAM : 264 Ko

Flash : 2 Mo

26 I/O digitale

3 entrées analogiques 12 bits

16 PWM

Communication : I2C × 2, SPI × 2, UART × 2

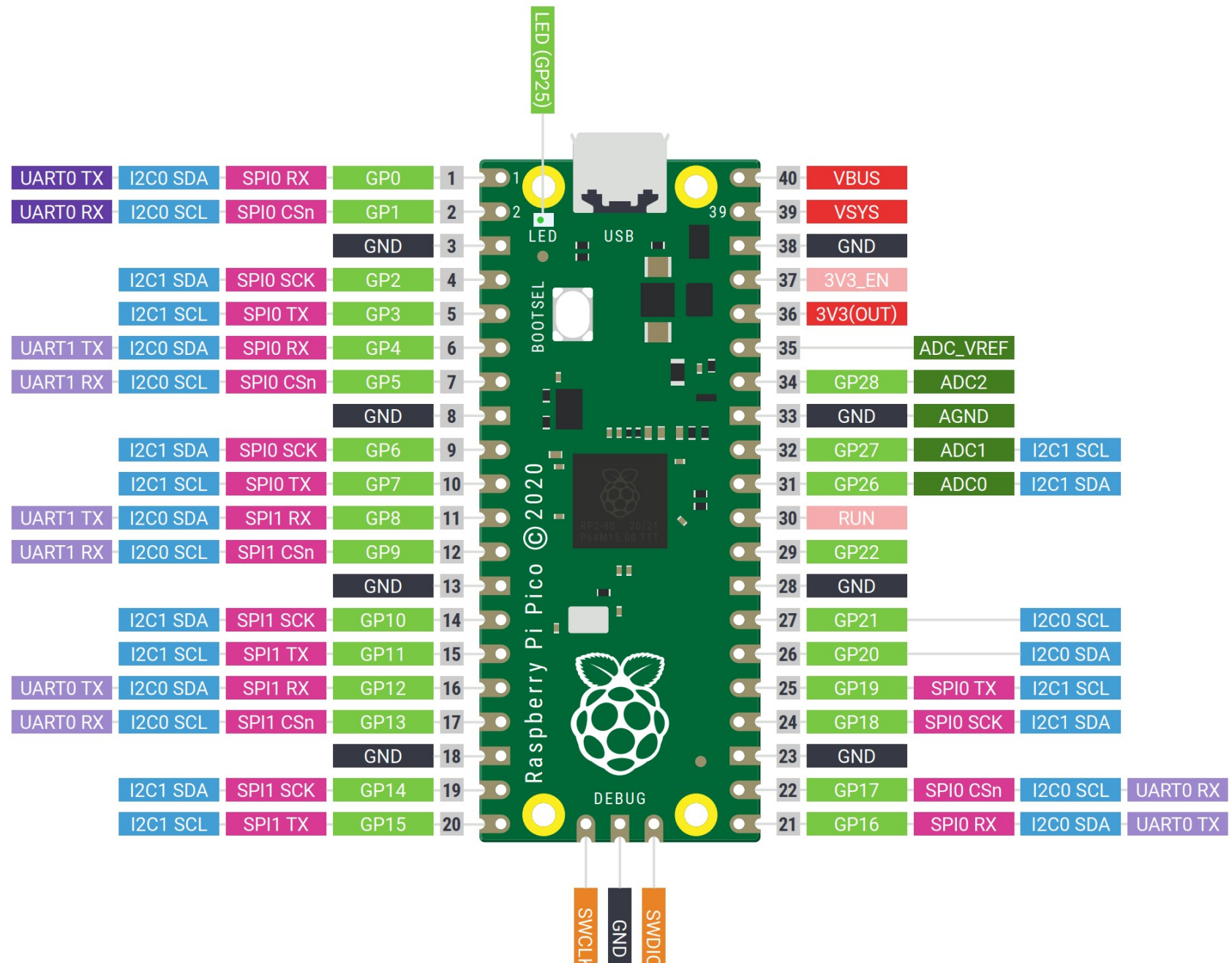


Alimentation par USB (5V) ou par alimentation extérieure (**1,8-5,5 V**)

Une RP2 coûte ~ 5 € !!!

Il manque un bluetooth ou un wifi pour être complet... mais vu le prix

Raspberry Pi Pico (RP2)

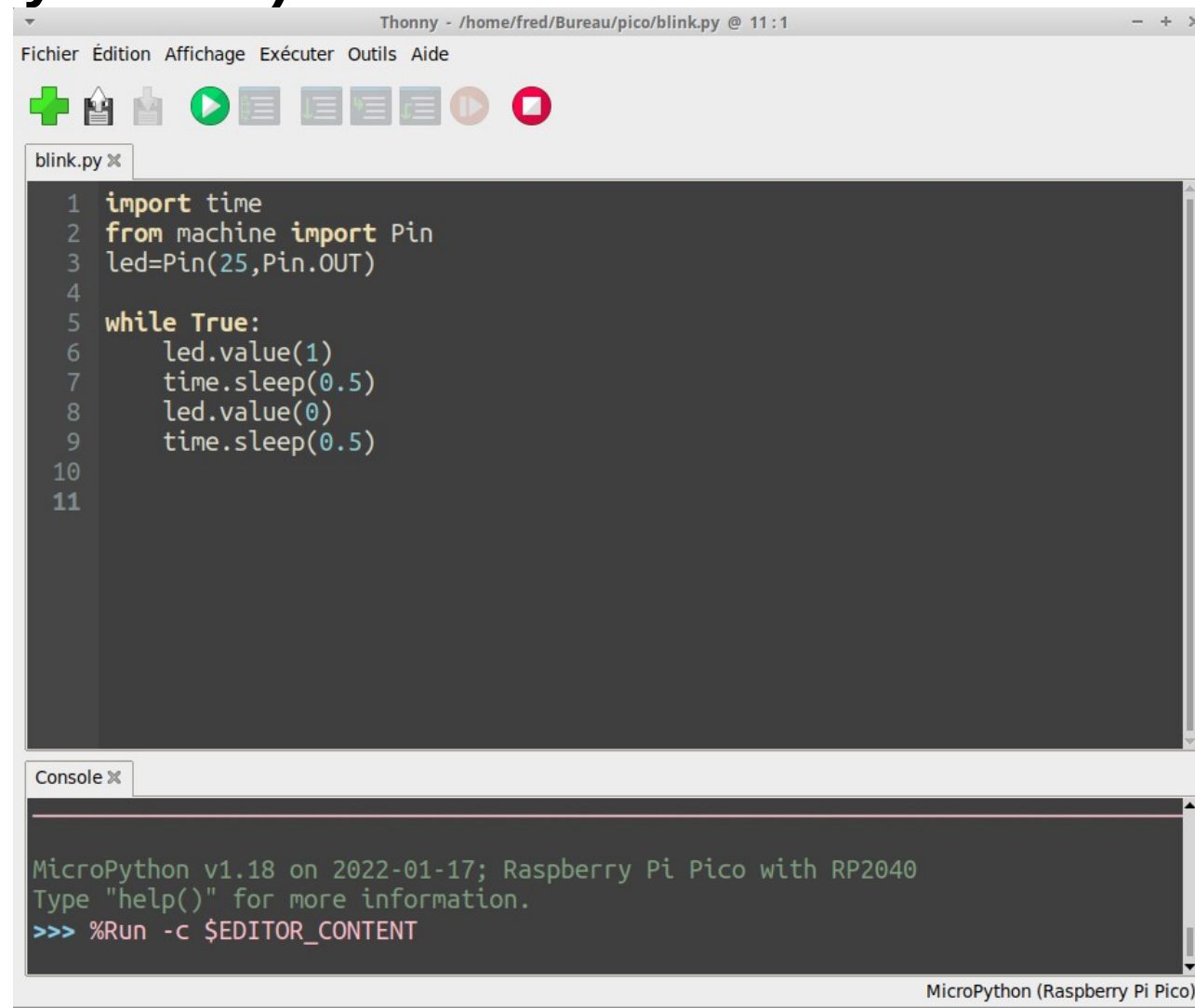


Pico RP2 et Thonny

Thonny doit être configuré pour le Pico :
Dans **Outils/option/interpréteur**
choisir **Micropython (Raspberry Pi Pico)**

**A la première utilisation,
il faut charger micropython.**

- On peut travailler en lignes de commande
- On peut charger le code avec le bouton vert
- On peut enregistrer le code dans un main.py pour des applications embarquées



The screenshot shows the Thonny IDE window titled "Thonny - /home/fred/Bureau/pico/blink.py @ 11:1". The menu bar includes "Fichier", "Édition", "Affichage", "Exécuter", "Outils", and "Aide". The toolbar contains icons for opening a file, saving, running (a green play button), and other functions. The editor window shows a file named "blink.py" with the following Python code:

```
1 import time
2 from machine import Pin
3 led=Pin(25,Pin.OUT)
4
5 while True:
6     led.value(1)
7     time.sleep(0.5)
8     led.value(0)
9     time.sleep(0.5)
10
11
```

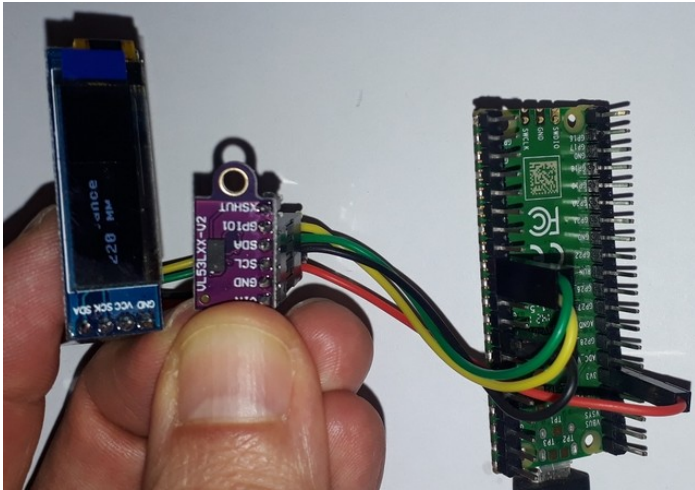
Below the editor is a "Console" window showing the MicroPython version and the command to run the script:

```
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico)".

Un télémètre en Pico

Exemple d'application : télémètre TOF avec écran OLED en I2C



Télémètre TOF : VL53L0X
OLED : SSD1306 (128*32)

Branchement sur le bus I2C
(PIN 8 et 9 du pico)

Nécessite les bibliothèques :
SSD1306_I2C
vl53l0x

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import time
import vl53l0x
```

```
i2c = I2C(0) # Init I2C SCL=Pin(GP9), SDA=Pin(GP8)
oled = SSD1306_I2C(128, 32, i2c) # Init oled
tof = vl53l0x.VL53L0X(i2c) # Init TOF
tof.set_measurement_timing_budget(40000)
tof.set_Vcsel_pulse_period(tof.vcsel_period_type[0], 12)
tof.set_Vcsel_pulse_period(tof.vcsel_period_type[1], 8)
while True:
    d=str(tof.ping()-50)
    oled.fill(0)
    oled.text("distance",5,5)
    oled.text(d+" mm",5,15)
    oled.show()
    time.sleep(0.1)
```

Un datalogueur en Pico

Exemple d'application : datalogueur à base de DS1820 et d'un lecteur de carte SD + alim indépendante

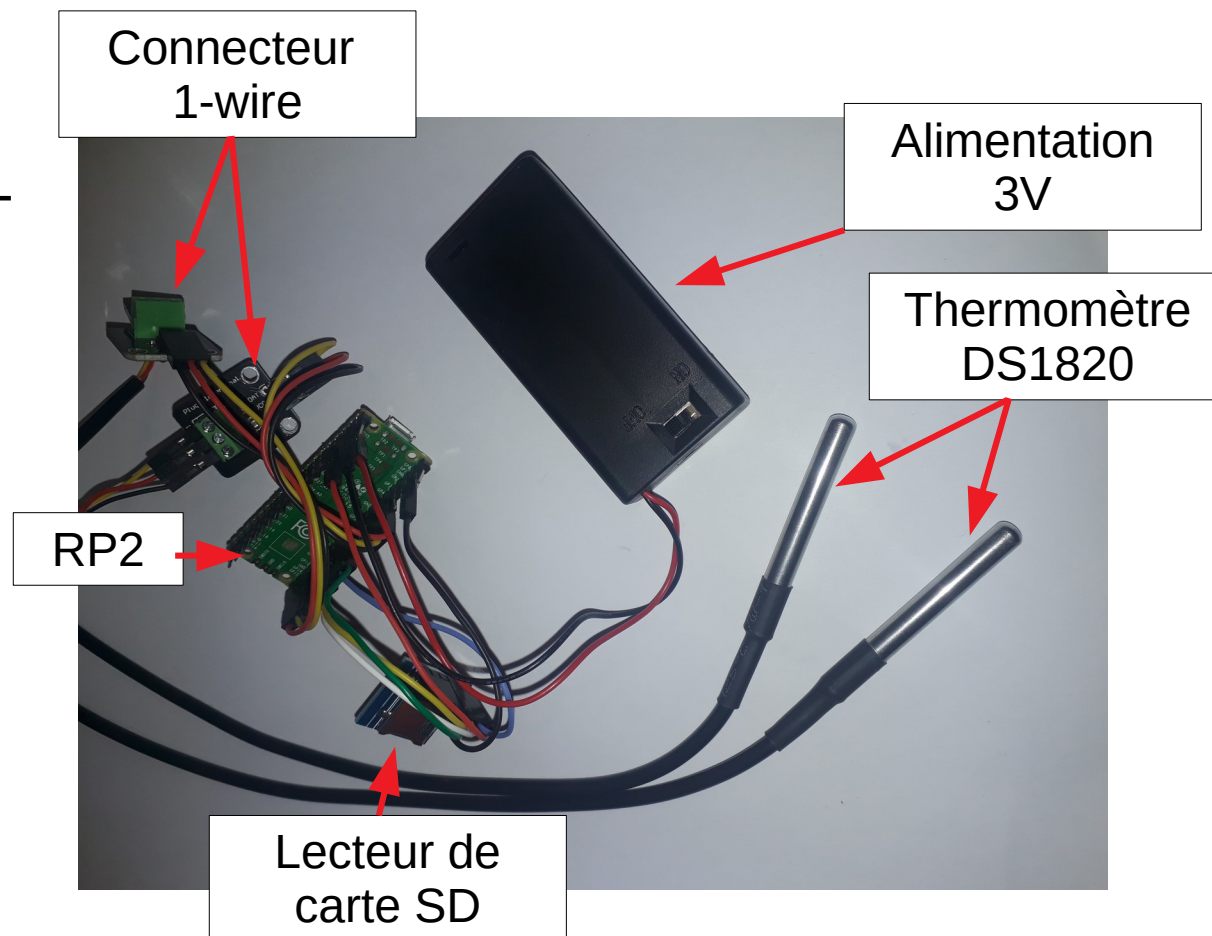
Lecteur de SD sur bus SPI

DS1820 sur 3 broches : GND + GPIO data +GPIO alim

Mesures toutes les 10 minutes

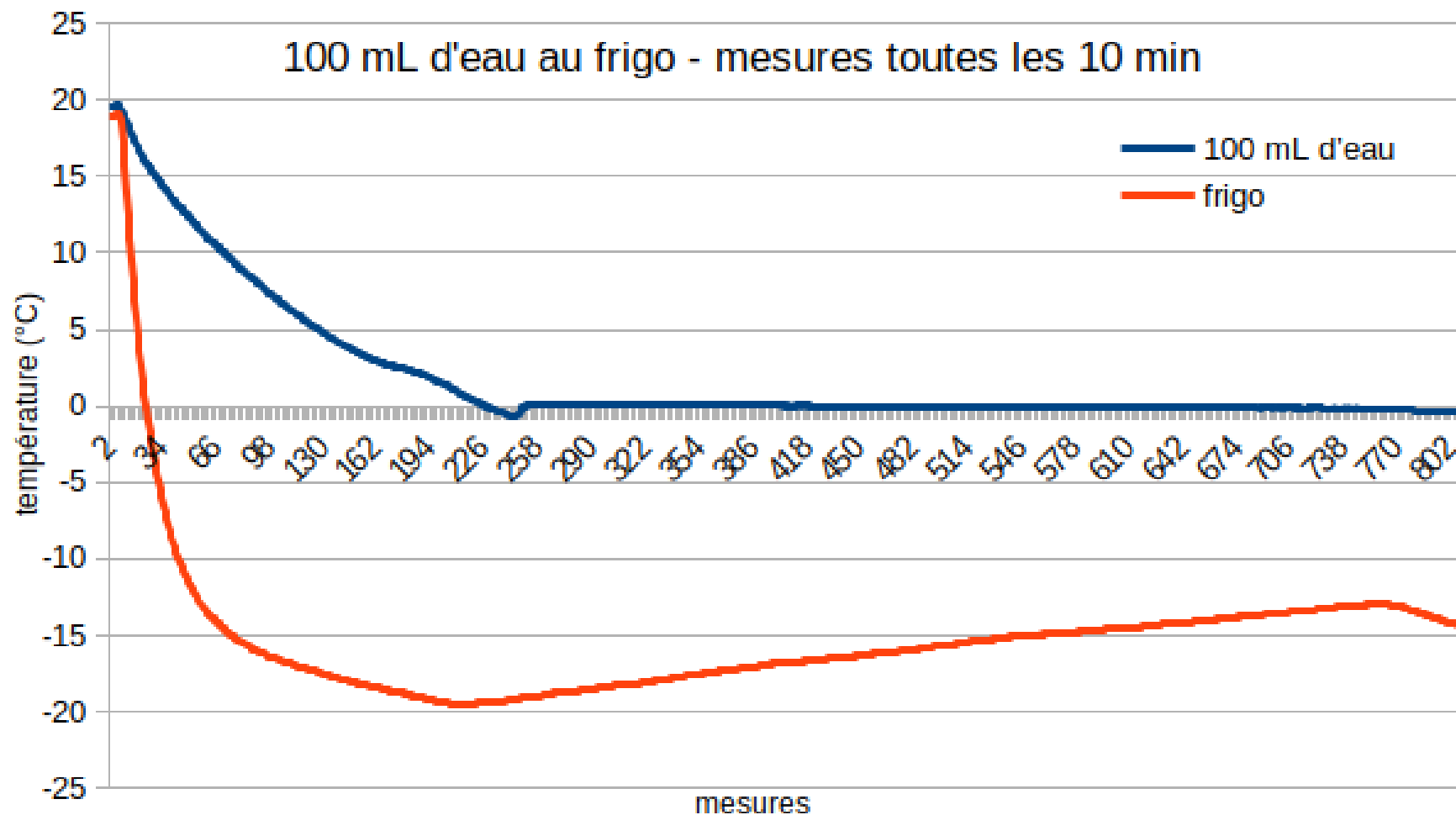
Nouveau fichier au démarrage de l'alimentation

(code trop long)



Un datalogueur en Pico

Objectif : mesurer la descente en température de 100 mL d'eau mise au congélateur (pour analyse par des terminales PC)



ESP32

Le SoC ESP32 (Espressif Systems) possède les caractéristiques suivantes :

Processeur **dual-core** Xtensa 32-bit à **240 MHz**

Gravure : 40 nm

SRAM : 528 Ko

Flash : 4 Mo

34 I/O digitale

18 entrées analogiques 12 bits

16 PWM

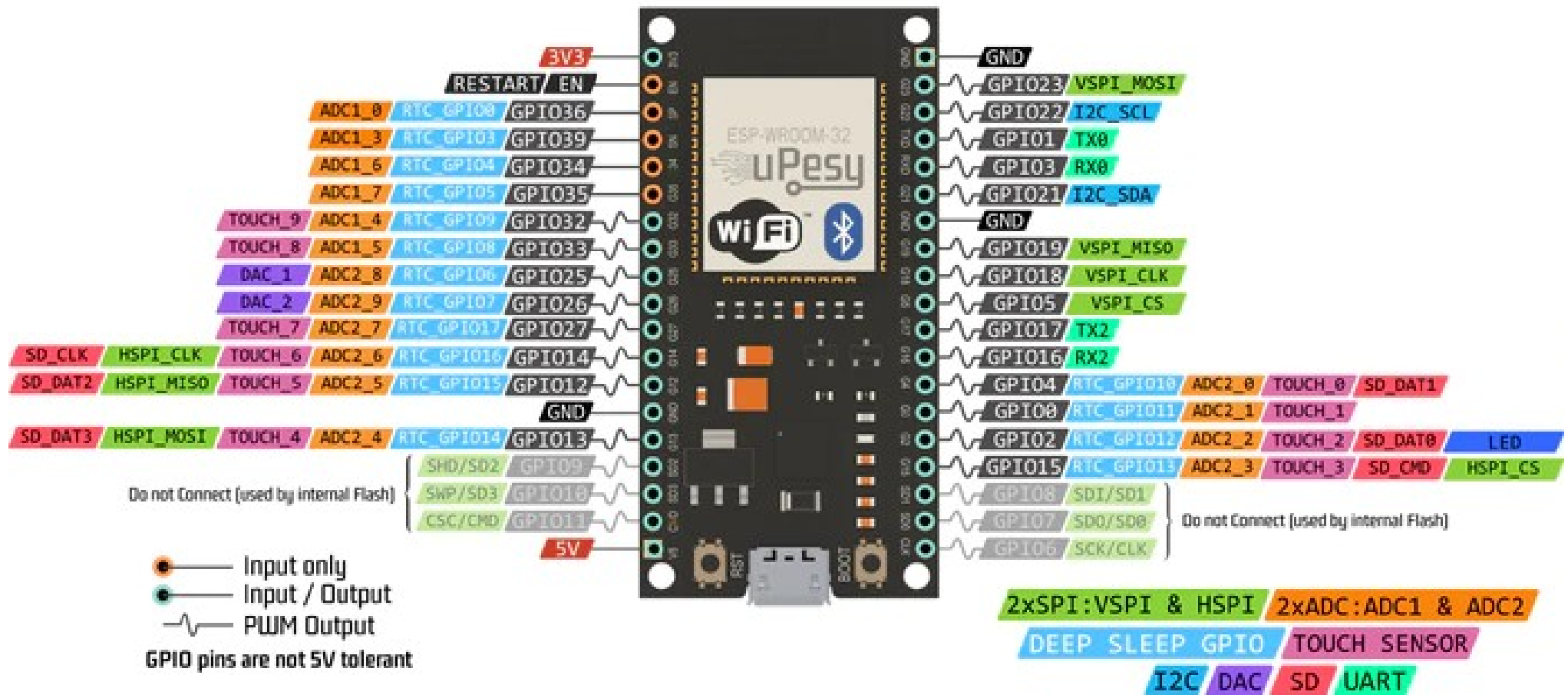
Communication : I2C × 2, SPI × 4, UART × 3

Wifi :802.11 b/g/n ; bluetooth

Alimentation par USB (5V) ou par alimentation extérieure (**1,8-5,5 V**)

Une ESP32 coûte ~ 10 € !!!

ESP32



Son grand atout : le wifi et le bluetooth

De nombreux exemples traînent sur le web en python mais plus via l'IDE Arduino en C

Un essai en ESP32

La programmation de l'ESP32 peut se faire avec Thonny de la même manière que le RP2

L'ESP32 peut se connecter à un réseau wifi existant : il faut fournir id et codes du réseau

Par défaut, l'ESP32 est en mode point d'accès.

Un exemple de connexion sur un réseau et modification d'un paramètre tel que la LED de l'ESP32 :

<http://gilles.thebault.free.fr/spip.php?article65>

Ce code permet de modifier par méthode GET la LED de l'ESP32.

Conclusion

La programmation en python sur microcontrôleur est accessible au lycée, et très appréciée des élèves.

Micro:bit est bien adapté pour le niveau SNT

Pour des projets évolués , micro:bit est également envisageable, mais peut être substitué par RP Pico (pas cher) ou l'ESP32 avec son wifi.

La programmation en python via Mu ou Thonny est accessible mais manque de facilité pour importer les librairies dédiées aux composants (c'est plus simple sur arduino)

Tout ceci évolue très vite...