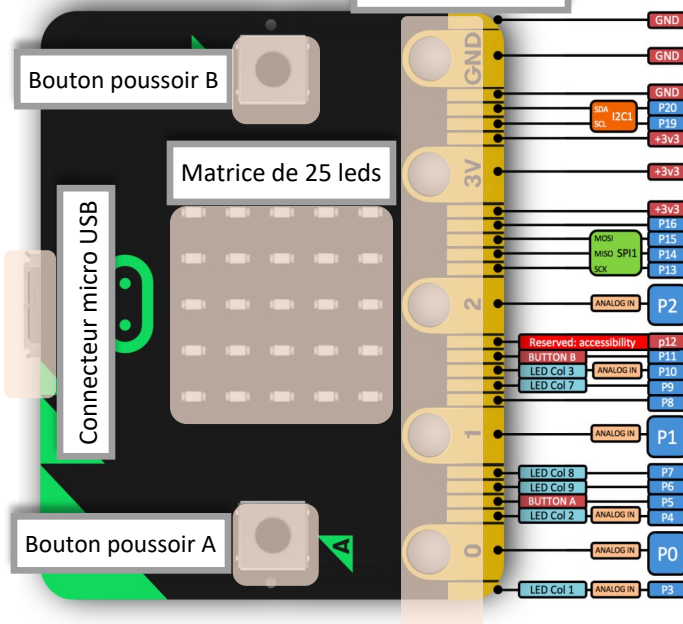


Introduction

La carte micro:bit peut être programmée de différentes manières, en langages graphiques mais aussi en lignes de codes avec **Python** pour les cartes à microcontrôleurs : **MicroPython**. Nous utiliserons lors des TP l'éditeur de texte **Mu**, pour coder et téléverser sur la micro:bit.

La carte micro:bit

Pins Entrées/Sorties

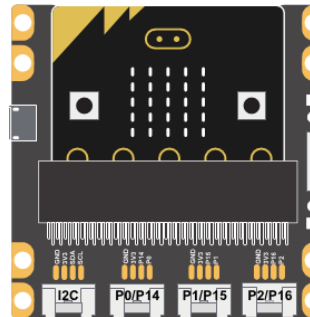


>> Caractéristiques <<

- Un processeur ARM pour exécuter votre code
- 25 leds programmables individuellement
- 2 boutons programmables
- 20 pins E/S numériques
- 6 pins d'entrée analogiques (Convertisseur Num. Analog.)
- Capteurs : Température, Accéléromètre, Boussole
- Communication : Radio, Bluetooth

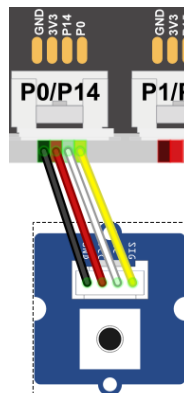
>> Utilisation des modules et du shield Grove <<

Le shield Grove pour micro:bit :



La carte micro:bit doit être positionnée avec la matrice de leds face à vous !

Connexion des modules Grove :

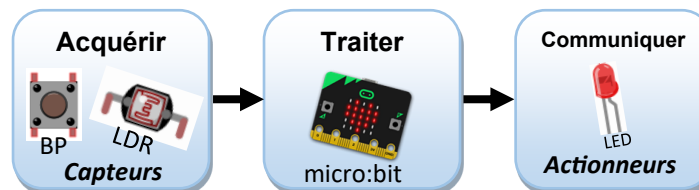


Exemple de connexion d'un module bouton poussoir avec les fils Grove :

- SIG (Signal) du BP → P0 du Shield
- NC (Not Connected) du BP → P14 du Shield
- Vcc (+ Alim) du BP → 3V3 du Shield
- GND (- Alim) du BP → GND du Shield

Dans ce cas **le signal du bouton poussoir sera à observer sur la pin 0 du micro:bit**

>> Chaîne d'information <<

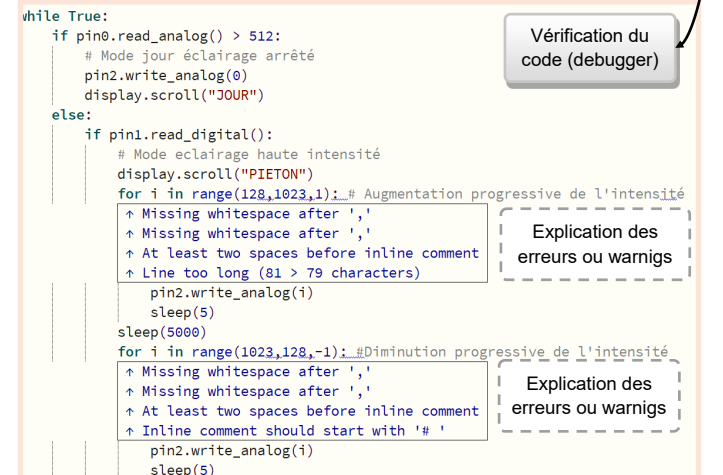
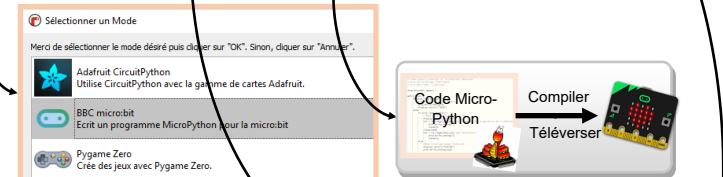
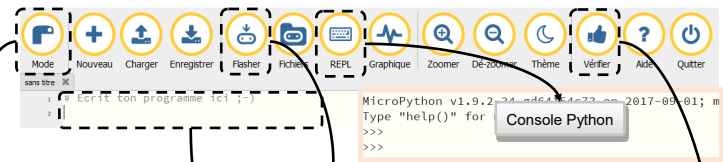


Utiliser Mu



Mu est le logiciel utilisé en TP pour coder en Python avec la micro:bit.

>> Barre d'outils <<



Structure d'un code Python

Un code **MicroPython** pour la carte **micro:bit** comporte plusieurs parties :

```
1 # SNT 2019
2 # Nom du programme
3 # Rôle du programme
4 # Nom Prénom du codeur
5 # V1.0 Date
6
7 from microbit import *
8
9 foo=22
10 pi=3.14
11 chaine="Vive le SNT !"
12
13 def Ma_fonction(param1,param2):
14     ...
15     #Bloc d'instructions
16     ...
17     return resultat
18
19 while True:
20     #Mon code débute ici
```

Cartouche du code

Import de la library du micro:bit

Déclaration de variables globales

Ajout de macros

Boucle principale

>> Bloc d'instructions <<

Un code Python est divisé en blocs, les instructions de contrôles : **if**, **while**, **for**, etc, nécessite une **indentation (tab)**, pour que Python comprenne qu'on souhaite coder dedans :

```
while True:
    #Je suis dans le while
    if pi==3.14:
        #Je suis dans le if
```

Les variables

Python s'occupe automatiquement du typage de la variable, lors de la première affectation :

```
foo=22 #foo est un entier
pi=3.14 #pi est un flottant
```

>> Types de variables <<

Principaux types de variables :

- `int` #Entier
- `float` #Flottant contient un nombre décimal

- `str` #String contient une chaîne de caractères (phrase)
- `bool` #Booléen contient True ou False (Vrai ou Faux)
- `file` #Contient un fichier

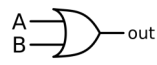

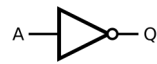
>> Transtypage <<

Python donne la possibilité d'imposer un type à une variable, exemple avec la console Python :

```
>> pi=3.14 #pi est un flottant
>> pi=int(pi) #On transforme pi en entier
>> pi
>> 3 #On a tronqué pi
```

Opérateurs

Dans une condition il faut parfois de comparer une variable avec une valeur ou encore deux variables entre-elles pour cela Python accepte les opérateurs suivants :

| Opérateurs logiques | |
|---------------------|---|
| or | OU logique  |
| and | ET logique  |
| not | NON logique  |

| Opérateurs de comparaison | |
|---------------------------|-------------------|
| == | Egalité |
| != | Différence |
| < | Infériorité |
| > | Supériorité |
| <= | Inférieur ou égal |
| >= | Supérieur ou égal |

| Opérateurs mathématiques | |
|--------------------------|--------------------------|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| / | Division |
| % | Modulo (Reste de la div) |
| ** | Puissance |

Structures conditionnelles

>> Si - if <<

```
if pi==3.14:
    #A faire si condition vraie
```

>> Si Sinon - if else <<

```
if pi==3.14:
    #A faire si condition vraie
else :
    #A faire si condition fausse
```

Imbrication des if else :

```
if pi==3.14:
    #A faire si pi=3.14
elif pi==0 :
    #A faire si pi=0
elif pi==1 :
    #A faire si pi=1
else :
    #A faire si toutes les
    #conditions précédentes
    #sont fausses
```

Structures répétitives

>> Tant que - while <<

```
while foobar>3:
    #A faire tant que foobar>3
```

>> Pour - for <<

```
for cpt in range(0,10,1):
    #cpt -> variable de comptage
    #0 -> initialisation de cpt
    #10 -> compte de 0 à 9
    #1 -> Compte de 1 en 1
```

Instructions MicroPython

>> Lire sur une pin numérique <<

```
valeur=pin0.read_digital()
#Stocke dans "valeur" l'état de la
#pin0 -> 0 ou 1
```

>> Ecrire sur une pin numérique <<

```
pin0.write_digital(1)
# Met la pin 0 à l'état haut
pin0.write_digital(0)
# Met la pin 0 à l'état bas
```

>> Utiliser le CAN <<

Le convertisseur analogique numérique permet de convertir une information provenant d'un capteur analogique entre 0 et 3.3V en un entier N compris entre 0 et 1023.

Veuillez bien regarder que la pin choisie soit une entrée analogique (voir page 1) → analog in

```
N=pin0.read_analog()
#Stocke dans N la valeur convertie
#par le CAN [0;1023]
```

>> Utiliser le PWM <<

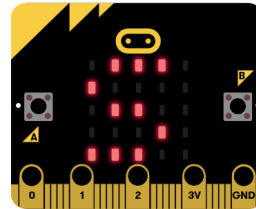
Le PWM (Pulse Width Modulation) ou Modulation de Largeur d'Impulsions en français permet de faire varier le rapport cyclique d'un signal carré. En sortie le composant connecté à la micro:bit verra une **variation de la tension entre 0 et 3.3V** pour un **nombre alpha codé entre 0 et 1023**.

```
pin0.write_analog(512)
# Met une tension de 3.3/2=1.65V sur la pin 0
pin0.write_analog(1023)
# Met une tension de 3.3V sur la pin 0
pin0.write_analog(0)
# Met une tension de 0V sur la pin 0
```

Utiliser le matériel de la micro:bit

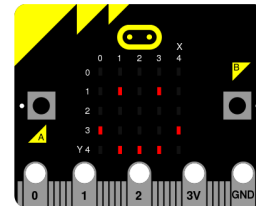
>> Afficher un texte sur la matrice de leds <<

```
display.scroll("SNT")
```



>> Afficher une image sur la matrice de leds <<

```
display.show(Image.HAPPY)
```



Une liste exhaustive des images est disponible sur le site : microbit-micropython.readthedocs.io

>> Utiliser les 2 boutons poussoirs <<

```
etat_A=microbit.button_a.is_pressed()
#etat_A=1 si bp A appuyé sinon 0

etat_A=microbit.button_a.was_pressed()
#Renvoie 1 si bp A appuyé depuis le début
#ou depuis le dernier appel de cette fonction

nb_appuis_A=microbit.button_a.get_presses()
#Renvoie le nb d'appuis sur bp A depuis le début
#ou depuis le dernier appel de cette fonction
```

>> Utiliser l'accéléromètre <<

```
mvt=accelerometer.current_gesture()
#Renvoie le type de mouvement effectué
#dans la variable "mvt"
```

L'accéléromètre intégré à la carte est capable de détecter certains types de mouvements préenregistrés :

| | | |
|----------------------------------|--|--|
| up (de bas en haut) | right (de gauche à droite) | freefall (en chute libre) |
| down (de haut en bas) | face up (tourné matrice en haut) | 3g, 6g, 8g (soumis aux accélérations du même nom) |
| left (de droite à gauche) | face down (tourné matrice en bas) | shake (secoué de haut en bas) |

>> Utiliser la boussole <<

```
microbit.compass.calibrate()
# Lance le programme de calibration

angle=microbit.compass.heading()
#Renvoie l'angle par rapport au nord
#0° -> N
#180° -> S
#90° -> E
#270° -> O
```

Pour un maximum de fiabilité **il est vraiment nécessaire de calibrer la boussole**