

MicroPython sur carte micro:bit

Rendre les objets intelligents grâce à Python



Olivier Lécluse

Table des matières



Introduction	3
I - Utiliser l'éditeur Mu	4
II - kit de survie de la carte microbit sous MicroPython	9
1. La matrice LED	11
1.1. Allumer, éteindre l'écran	11
1.2. Afficher des messages	11
1.3. Afficher / Lire des pixels	11
1.4. Afficher des images	11
2. Utiliser les boutons	13
3. Utiliser les entrées-sorties	14
4. Utiliser la communication radio	17
5. la gestion du son	20
6. Utiliser l'accéléromètre	21
7. Utilisation de la boussole	24

Introduction



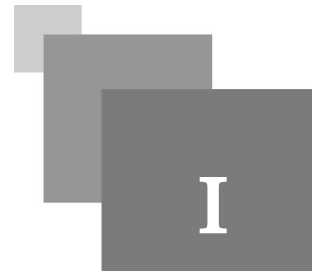
La carte micro:bit est un ordinateur à carte unique à processeur ARM conçu par la BBC pour l'éducation informatique au Royaume-Uni. En 2015, ce projet prévoit d'offrir gratuitement un exemplaire à chaque écolier britannique de douze ans. Plus récemment

la carte micro:bit peut se programmer en utilisant plusieurs langages : javascript (blocs ou texte), scratch3, LISP, C++ avec l'environnement Arduino mais aussi MicroPython. Nous nous intéresserons ici uniquement à la programmation de la carte sous MicroPython.

Pour la programmation sous MicroPython, nous utiliserons l'environnement *mu* que nous avons déjà rencontré pour les cartes Adafruit ou ESP8266/32 précédemment dans ce tutoriel. Cet environnement est simple à utiliser et s'adapte à plusieurs types de cartes différents. Il est libre, gratuit et multi-plateforme.



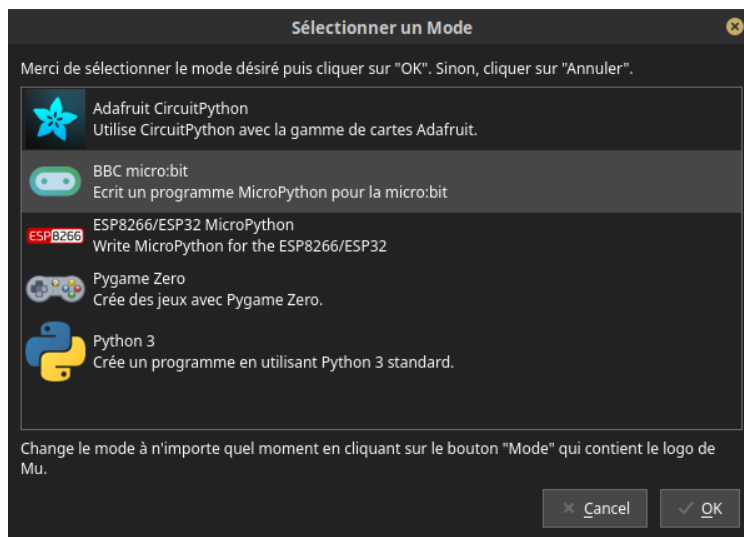
Utiliser l'éditeur Mu








Les modes

L'éditeur Mu peut s'adapter à plusieurs contextes en fonction des cartes qu'on lui connecte. Il peut même être utilisé pour coder en Python directement sur l'ordinateur, même si on utilise pas de carte microbit. Les fonctionnalités d'une carte à l'autre peuvent varier donc l'éditeur se met dans un **mode** spécifique à la carte qui est connectée.

Ce mode peut se changer en appuyant sur l'icône



Dans ce tutoriel, nous supposons que nous sommes en mode **BBC microbit**. L'environnement Mu va nous permettre de

- taper nos programmes et les envoyer sur la carte pour s'exécuter 
- transférer des fichiers vers et depuis la carte 
- accéder à la console interactive MicroPython (REPL) 
- tracer des graphiques 
- vérifier la syntaxe de nos programmes 


Où sont les fichiers

Mu stocke ses fichiers dans le dossier nommé **mu_code** situé dans votre dossier personnel.

La racine de ce dossier apparaît quand vous cliquez sur l'icône



Créer son premier programme


Cliquer sur  puis commencer à taper votre programme. Celui ci-dessous sera très bien :)

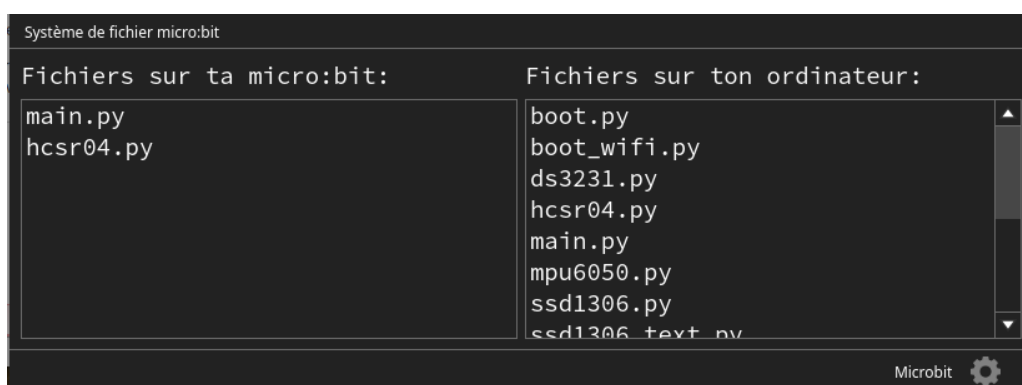
```
1 from microbit import *
2 display.show("Hello World")
```

Enregistrer le fichier à l'emplacement de votre choix en choisissant un nom, par exemple **testHW.py**. Il n'est pas nécessaire de l'enregistrer dans le dossier **mu_editor** mais cela peut être une bonne idée.


Flasher le programme sur la carte à l'aide de l'icône . Cela signifie que le programme sera transféré sur la carte - en remplaçant éventuellement celui qui était présent - et sera lancé automatiquement au démarrage.

Remarque : Fichier main.py

Si vous cliquez sur l'icône  pour voir les fichiers sur la carte, vous pouvez vous attendre à voir un fichier nommé **testHW.py** si c'est le nom que vous avez choisi. Or il n'en est rien. A la place, vous n'avez qu'un fichier nommé **main.py**, ce qui peut être déroutant au début.



Il faut savoir que le bouton flasher va transférer le programme en cours d'édition sous le nom **main.py** car c'est ce nom que MicroPython s'attend à trouver au démarrage de la carte et lance automatiquement.

Il est bien sûr possible de transférer sur la carte d'autres programmes python (par exemple des bibliothèques) tout en conservant leur nom original. La capture ci-dessus montre que la bibliothèque **hcsr04.py** (qui pilote un capteur de distance à ultrasons) a été copiée sur la carte. Il faut pour cela utiliser l'outil . On pourra inclure ce fichier à notre programme en cours à l'aide de la ligne

```
import hcsr04
```

La console interactive REPL

Le **REPL** permet un contact direct avec MicroPython installé sur la carte. C'est ce qui correspond à la console Python sur ordinateur ou la calculatrice Numworks. Pour lancer **REPL**, cliquez sur l'icône



```
BBC micro:bit REPL
Type "help()" for more information.
>>>
MicroPython v1.9.2-34-gd64154c73 on 2017-09-01; micro:bit v1.0.1 with
nRF51822
Type "help()" for more information.
>>>
>>> display.show("Hello world!!")
>>>
```

En cliquant sur l'icône **REPL**, le programme en cours d'exécution sur la carte (**main.py**) est stoppé et vous pouvez taper des commandes qui seront immédiatement exécutées par MicroPython. La capture ci-dessus vous affichera instantanément le message sur la matrice LED. C'est très commode de pouvoir interagir directement avec MicroPython en cours d'élaboration de programme. Cela permet par exemple :

- de tester une ligne de code comme `display.show("Hello world!!")`
- de se renseigner sur l'usage d'une commande. Essayez `help(display.show)`
- de connaître le contenu d'une librairie. Essayez `dir(Image)`
- de connaître les cause d'un plantage du programme
- etc....

Attention :


Vous ne pouvez pas avoir **REPL** ouvert en même temps que **Fichiers**.

Vous ne pouvez pas **Flasher** quand **REPL** est ouvert.

Déceler une erreur

Tapez le programme suivant. Il contient une erreur ! Le message d'erreur va défiler sur la matrice LED mais il n'est pas du tout évident à lire !

```
1 from microbit import *
2 a=[3,2,1]
3 display.show(a)
```

Les erreurs de syntaxe peuvent être décelées avant même le flashage grâce au bouton . L'erreur ici apparaît à l'exécution et n'a pas été décelée par **Vérifier**. Pour visualiser les retours d'erreur, nous allons utiliser le **REPL** :

Lancer **REPL**. L'invite de commande `>>>` apparaît. Nous allons provoquer un redémarrage de la carte. Il y a 2 manières de le faire

- A l'aide du bouton **reset** au dos de la carte
- en tapant **CTRL-D** dans le **REPL**.

Avec l'expérience, la seconde méthode (*soft reboot*) est plus rapide. Le programme se lance à nouveau mais cette fois-ci, les erreurs sont affichées dans la console. Nous pouvons lire

Traceback (most recent call last):
 File "__main__", line 3, in <module>
 TypeError: not an image
 et nous pouvons corriger notre programme.

Voir le résultat de la commande print

Vous pouvez afficher des informations à l'aide de l'instruction python **print**, mais la carte ne possède pas d'écran. Elles n'apparaissent pas non plus sur la matrice LED car celle-ci se pilote via la commande **display**. Essayez le programme ci-dessous qui simule le lancer de 10 dés.

```
1 from random import randint
2 for i in range(10):
3     print("lancer ", i, " : ", randint(1, 6))
```

Rien ne s'affiche sur la carte. Pour voir le résultat, ouvrez **REPL** puis **redémarrez le programme** par un *soft reset* (**CTRL-D**).

Utiliser l'outil Graphique

L'outil graphique permet de visualiser sous forme de courbe les valeurs affichées via la commande **print**. Celles-ci doivent être présentées dans un **tuple** (similaire à une liste mais délimitée par des parenthèses). A chaque élément du tuple correspondra une courbe. On peut ainsi tracer une ou plusieurs courbe.

Si on ne veut qu'une seule courbe, il faut quand même afficher un print à un seul élément ce qui ressemble à cela (**21,**). Ne pas oublier la virgule à la fin. Voici un exemple :

```
1 from microbit import *
2
3 while True:
4     temp = temperature()
5     print((temp,))
6     sleep(1000)
```

```
1 from microbit import *
2
3 while True:
4     temp = temperature()
5     print((temp,))
6     sleep(1000)
```

Grapher BBC micro:bit

BBC micro:bit REPL

```
(21,)
(21,)
(21,)
(21,)
(21,)
(21,)
(21,)
(21,)
(21,)
(21,)
(21,)
```


kit de survie de la carte microbit sous MicroPython


II

Dans cette partie, nous allons balayer l'ensemble des fonctionnalités de la carte microbit sous MicroPython en les illustrant par des exemples très simple.

On supposera ici que les bases de la programmation sous Python sont connues. L'objet n'est pas d'apprendre Python mais de voir comment ce dernier s'adapte sur la carte microbit. Pour vos premier pas en Python, si vous connaissez le système de programmation par blocs, vous pouvez consulter cette page qui vous donnera les clés pour commencer :

<https://arduiblog.com/2019/03/06/debuter-en-micropython/>

Le site de la DANE de Caen :

- *Découvrir la carte Micro:bit*
- *Programmer une carte micro:bit en Python*

Guide MicroPython sur le site micro:bit

La plupart des exemples donnés ci-dessous pourront fonctionner dans le simulateur python se trouvant à l'adresse :

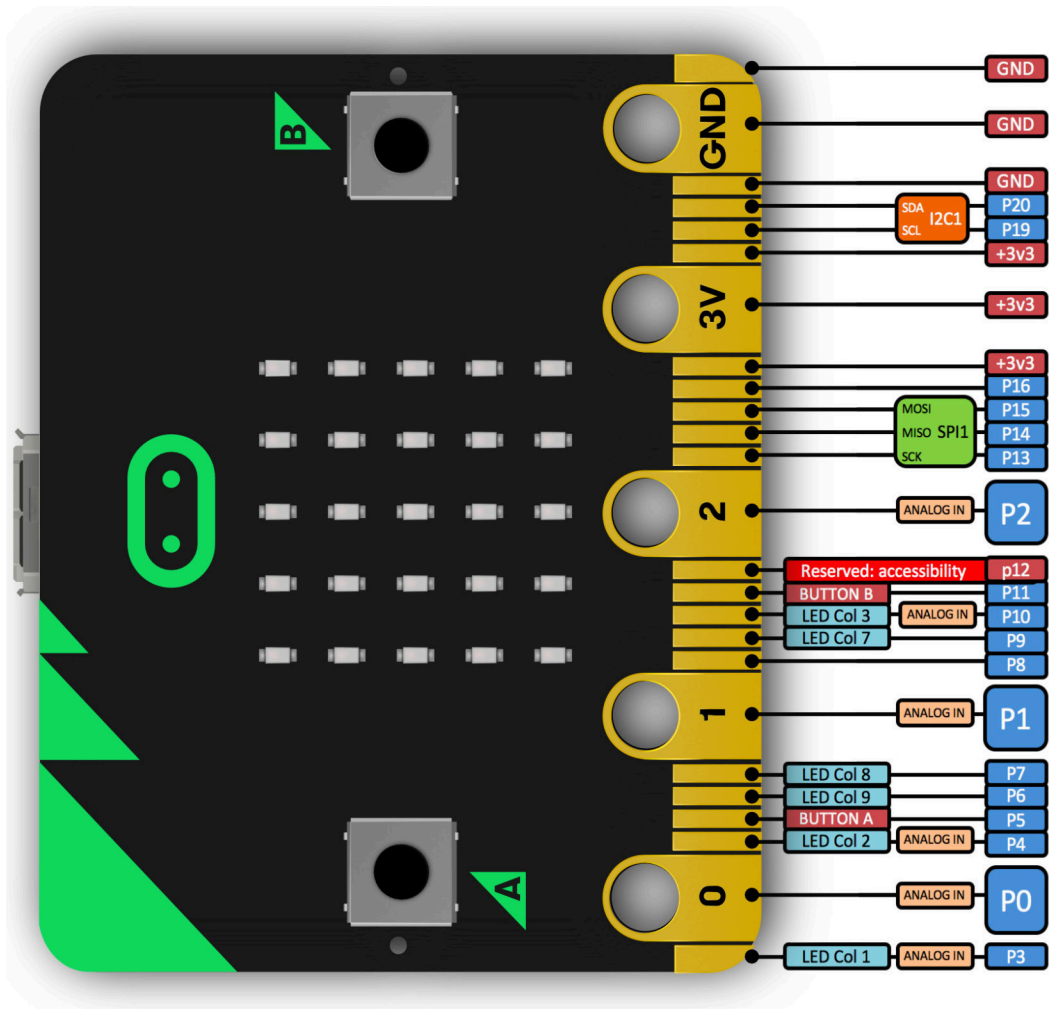
<https://create.withcode.uk/>

Tapez votre code puis validez par **CTRL-ENTREE**. Si votre code commence par

```
from microbit import *
```

l'exécution du code provoquera l'affichage d'une carte microbit virtuelle qui exécutera votre programme comme sur une vraie carte. Ce simulateur vous permet d'accéder également aux capteurs internes de la carte (accéléromètre, boussole, température...)

Pour finir avec cette introduction, voici un schéma qu'il est toujours utile d'avoir sous la main car il présente le brochage de la carte et l'affectation des entrées-sorties.



1. La matrice LED

1.1. Allumer, éteindre l'écran

- les commandes `display.on()` et `display.off()` permettent d'allumer et d'éteindre l'écran. Éteindre l'écran peut être utile lorsqu'on veut récupérer les broches d'entrée sortie associées à la matrice de LED (broches 3,4,5,7,9,10).
- les commandes `display.is_on()` et `display.is_off()` permettent de tester l'état de l'écran.
- ne pas confondre ces commandes avec `display.clear()` qui se contente d'éteindre les pixels de l'écran.

Exemple

Tapez les lignes suivantes dans le **REPL** pour constater l'effet. Le rallumage de l'écran remet ce dernier dans l'état où il était. Il n'a donc pas été effacé.

```
1 >>> display.show(1)
2 >>> display.off()
3 >>> display.on()
```

1.2. Afficher des messages

display.show() et *display.scroll()*

Affiche un texte ou un nombre sur l'écran. La méthode **show()** montre caractère par caractère alors que la méthode **scroll()** fait défiler plus progressivement.

Testez dans le **REPL** différentes commandes afin de voir le rôle des paramètres optionnels *wait* et *loop*

- `display.show(23)`
- `display.scroll('Hello World!', loop=True)`
- `display.show('Hello World!', wait=False, loop=True)`

1.3. Afficher / Lire des pixels

display.get_pixel() et *display.set_pixel()*

`display.get_pixel(x, y)` retourne l'illumination du pixel situé à la colonne *x* et la ligne *y* sous forme d'un entier de 0 (éteint) à 9 (complètement allumé)

`display.set_pixel(x, y, value)` allume le pixel situé à la colonne *x* et la ligne *y* avec une illumination *value* de 0 (éteint) à 9 (complètement allumé)

Exemple : Dégradé

testez le programme suivant

```
1 from microbit import *
2
3 for x in range(5):
4     for y in range(5):
5         display.set_pixel(x,y, (x+y)%9)
```

1.4. Afficher des images

Images prédéfinies

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

Il y a d'autres images prédéfinies dans MicroPython sur microbit que je vous laisse tester. On peut les obtenir par la commande `dir(Image)` :

```
'HEART', 'HEART_SMALL', 'HAPPY', 'SMILE', 'SAD', 'CONFUSED', 'ANGRY',  
'ASLEEP', 'SURPRISED', 'SILLY', 'FABULOUS', 'MEH', 'YES', 'NO',  
'CLOCK12', 'CLOCK1', 'CLOCK2', 'CLOCK3', 'CLOCK4', 'CLOCK5', 'CLOCK6',  
'CLOCK7', 'CLOCK8', 'CLOCK9', 'CLOCK10', 'CLOCK11', 'ARROW_N',  
'ARROW_NE', 'ARROW_E', 'ARROW_SE', 'ARROW_S', 'ARROW_SW', 'ARROW_W',  
'ARROW_NW', 'TRIANGLE', 'TRIANGLE_LEFT', 'CHESSBOARD', 'DIAMOND',  
'DIAMOND_SMALL', 'SQUARE', 'SQUARE_SMALL', 'RABBIT', 'COW',  
'MUSIC_CROTCHET', 'MUSIC_QUAVER', 'MUSIC_QUAVERS', 'PITCHFORK', 'XMAS',  
'PACMAN', 'TARGET', 'ALL_CLOCKS', 'ALL_ARROWS', 'TSHIRT', 'ROLLERSKATE',  
'DUCK', 'HOUSE', 'TORTOISE', 'BUTTERFLY', 'STICKFIGURE', 'GHOST',  
'SWORD', 'GIRAFFE', 'SKULL', 'UMBRELLA', 'SNAKE'
```

Créer sa propre image

Testez dans **REPL** l'exemple suivant :

```
>>> monImage = Image("90009:06060:00300:06060:90009")  
>>> display.show(monImage)
```

On peut aussi présenter différemment :

```
>>> monImage = Image("90009:"  
"06060:"  
"00300:"  
"06060:"  
"06060:")  
>>> display.show(monImage)
```

Chaque chiffre représente la valeur du pixel.

Afficher des animations

On peut définir une liste (ou un tuple) d'images pour les afficher sous forme d'une animation : essayez dans le **REPL**

```
>>> display.show(Image.ALL_CLOCKS)  
>>> display.show(Image.ALL_CLOCKS, delay=1000)  
>>> display.show(Image.ALL_CLOCKS, delay=100)
```

```
>>> display.show([Image.SAD, Image.HAPPY], delay=1000, loop=True,
wait=False)
```

Complément

Pour arrêter l'animation lorsqu'elle est jouée en tâche de fond, il suffit de faire un `display.clear()`

2. Utiliser les boutons

Comprendre le principe de base

Il y a deux boutons intégrés sur la carte microbit accessibles via les objets **button_a** et **button_b**. Ces objets proposent des méthodes très pratiques facilitant grandement l'écriture des programmes qui ont besoin d'interaction avec l'utilisateur.

Chaque pression sur le bouton est détectée par la carte et mémorisée. Il est donc impossible de rater l'événement, même si on interroge pas le bouton au moment précis ou celui-ci est pressé. Pour bien comprendre ce mécanisme, voici une expérience interactive à mener dans le **REPL** :

```
>>> button_a.get_presses()
```

renvoie 0, si le bouton A n'a pas été pressé.

A présent, appuyez plusieurs fois sur le bouton A puis relancez la commande dans le **REPL** :

```
>>> button_a.get_presses()
```

Vous voyez que le nombre de pressions est à présent affiché. Maintenant que le bouton a été interrogé, retapez

```
>>> button_a.get_presses()
```

le compteur est remis à 0 à chaque invocation de la commande. Il convient donc de stocker cette information dans une variable pour son usage ultérieur si besoin.

Méthode is_pressed()

Testez le programme suivant :

```
1 from microbit import *
2
3 while True:
4     if button_a.is_pressed():
5         display.show(Image.HAPPY)
6     elif button_b.is_pressed():
7         break
8     else:
9         display.show(Image.SAD)
10
11 display.clear()
```

Vous voyez ici un exemple d'utilisation de **is_pressed()** qui renvoie **True** si le bouton est présentement pressé au moment où la méthode est invoquée.

L'exemple montre aussi l'utilisation d'une boucle infinie **while True** et l'utilisation du bouton B pour mettre fin au programme, ce qui peut souvent s'avérer utile.

méthode `was_pressed()`

Pour finir, il est fréquent d'avoir recours à la méthode `was_pressed()` pour savoir si un bouton a été actionné pendant que le programme était occupé à une autre tâche. En voici un exemple.

```
1 from microbit import *
2
3 display.scroll("Appuyez sur le bouton A", delay=60, wait=True)
4
5 if button_a.was_pressed():
6     display.show(Image.HAPPY)
7 else:
8     display.show(Image.SAD)
9
```

On utilise ici la commande `scroll` en mode bloquant (avec `wait=True`). De ce fait, on ne peut pas détecter l'appui du bouton pendant l'affichage du message avec `is_pressed()`. `was_pressed()` permet de savoir si la consigne a bien été appliquée.

3. Utiliser les entrées-sorties

Attention

Une mauvaise utilisation des entrées-sorties peut endommager votre carte. Attention à ne pas dépasser la tension ou le courant prévu pour chaque broche d'E/S.

La carte microbit fonctionne sur 3,3V, même si elle est alimentée par les 5V de l'USB. Il faut donc veiller à ne pas appliquer de tension supérieure à 3,3V sur les entrées sortie. En particulier il faudra être vigilant en cas d'utilisation de capteur nécessitant une tension de 5V.

Complément : *shield gator:bit*

Afin de protéger la carte de mauvaise manipulation, il est possible d'accéder aux entrées-sorties via un shield (carte additionnelle sur laquelle la microbit vient se brancher). Personnellement, j'utilise le *shield gator:bit V2* de sparkfun qui me satisfait sur le plan de la protection de la carte : une tension de 5V appliquée sur une E/S sera rabaissée et n'endommagera pas la microbit. En plus, ce shield offre 5 led RVB *neopixel*, un haut-parleur et un convertisseur de tension permettant d'alimenter la carte depuis une alimentation entre 2,7V et 9V.

Soyez vigilant de bien choisir la version 2 du shield, référence sparkfun DEV-15162.

Fondamental : *Information complète*

Pour connaître le détail des spécificités de chaque broche de la microbit, je vous invite à consulter cette page :

<https://microbit.org/guide/hardware/pins/>

boutons tactiles

Les broches 0, 1 et 2 sont - en théorie - tactiles. On peut détecter lorsqu'on les touche. Pour ma part je ne trouve pas que cette détection soit très sensible et n'est pas vraiment utilisable. D'autre part, l'utilisation du shield *gator:bit* rend cette fonctionnalité inopérante.

Voici tout de même un exemple de code mettant en oeuvre la détection de toucher sur la broche pin0. Vous pourrez ainsi vous faire votre propre idée par vous-même.

```
1 from microbit import *
2
3 while True:
4     if pin0.is_touched():
5         display.show(Image.HAPPY)
6     else:
7         display.show(Image.SAD)
```

Faire clignoter une LED

Comment échapper à cet incontournable *blink* ! Le code se passe de commentaire et illustre la méthode `write_digital()` qui s'applique sur les broches pinXX

```
1 from microbit import *
2
3 while True:
4     pin0.write_digital(1)
5     sleep(500)
6     pin0.write_digital(0)
7     sleep(500)
8
```

Entrée numérique

Pour configurer la broche 1 en entrée numérique avec résistance de PULL_UP, on utilise les commandes suivantes :

```
>>> pin1.set_pull(pin1.PULL_UP)
```

les autres options de PULL sont **PULL_DOWN** et **NO_PULL**

On peut ensuite lire la broche en faisant

```
>>> pin1.read_digital()
```

qui renvoie 1 si la broche est à l'état haut, ou 0 à l'état bas.



Complément : Connaître la configuration d'une broche

Pour savoir si une broche est configurée en entrée (INPUT) ou en sortie (OUTPUT), on peut taper

```
>>> pin1.get_mode()
```

renvoie par exemple **'read_digital'** ou **'write_digital'**

La configuration des broches en *INPUT* ou *OUTPUT* se fait automatiquement selon qu'on fait un **read_digital()** ou un **write_digital()**. Par défaut les broches sont configurées en sortie. Essayez les commandes suivantes dans le **REPL** :

```
>>> pin1.get_mode()
```

```
'write_digital'
```

```
>>> pin1.read_digital()
```

```
>>> pin1.get_mode()
'read_digital'
>>> pin1.write_digital(0)
>>> pin1.get_mode()
'write_digital'
```

Pulse-Width-Modulation (PWM) : Modulation en largeur d'impulsion

Les broches ne savent délivrer que 0 ou 3,3V. Si on souhaite diminuer la luminosité d'une LED ou ralentir un moteur commandé par une broche de la microbit il est utile d'avoir recours au PWM afin de définir quel pourcentage de temps la broche sera à l'état haut.

La méthode permettant le PWM est `write_analog` et s'utilise ainsi :

```
>>> pin1.write_analog(512)
>>> pin1.get_mode()
'write_analog'
```

Les valeurs acceptées sont sur 10 bits donc de 0 (toujours à 0) à 1023 (toujours à 1). L'exemple ci-dessus montre un rapport cyclique de 50% dont haut la moitié du temps sur une période donnée.

Entrée analogique

Certaines broches (0, 1, 2, 3, 4, 10) peuvent recevoir une tension entre 0 et 3,3V. Elles renvoient alors une valeur lue entre 0 et 1023. La méthode à invoquer est **`read_analog()`** :

```
>>> pin1.read_analog()
49
```

Complément : Connaître les fonctionnalités acceptées par une broche

La commande **`dir`** permet de lister les méthodes associées à une broche :

```
>>> dir(pin1)
['write_digital', 'read_digital', 'write_analog', 'read_analog',
'set_analog_period', 'set_analog_period_microseconds', 'is_touched',
'PULL_UP', 'PULL_DOWN', 'NO_PULL', 'get_pull', 'set_pull', 'get_mode']
>>> dir(pin9)
['write_digital', 'read_digital', 'write_analog', 'set_analog_period',
'set_analog_period_microseconds', 'get_analog_period_microseconds',
'PULL_UP', 'PULL_DOWN', 'NO_PULL', 'get_pull', 'set_pull', 'get_mode']
```

La broche 1 possède la fonctionnalité **`read_analog()`** mais pas la broche 9.

Remarque

La microbit utilise des résistances externes de pullup faibles (10 Mohms) sur les broches 0, 1 et 2 afin de faire fonctionner la détection de toucher

Il y a aussi des résistances de pull-up externes de 10Kohms sur les broches 5 et 11 pour les boutons A et B

Les broches 1, 4, 6, 7, 9 et 10 sont utilisées pour la matrice de LEDs. Pour utiliser ces broches à d'autres usages, il faut éteindre l'écran (**display.off()**).

4. Utiliser la communication radio

Le module radio permet d'envoyer ou de recevoir des message vers ou depuis une ou plusieurs cartes microbit.

La communication se fait sans fil sur une fréquence entre 2,4GHz et 2,5Ghz selon le **canal** (numéroté entre 0 et 83) choisi.

Les messages ont une taille maxi de 251 caractères. Par défaut, la taille est fixée à 32 caractères.

Pour utiliser la radio, vous devez avoir 2 cartes microbit. La communication radio est très simple à mettre en oeuvre comme vous pourrez le voir. Attention cependant, les communications ne sont pas chiffrées, évitez de transmettre des informations sensibles !!

Initialiser la communication

commencer par importer le module radio

```
import radio
```

puis activez la radio qui par défaut est désactivée pour des raisons d'économie d'énergie

```
radio.on()
```

La transmission peut alors s'établir entre plusieurs cartes

Envoyer et recevoir

Pour envoyer un message sous forme d'une chaîne de caractères, utiliser

```
radio.send(message)
```

pour recevoir, utiliser

```
radio.receive()
```

Difficile de faire plus simple !!

Exemple : Envoyer le message "Hello World" d'une carte à l'autre

Sur la carte émettrice, tapez le programme suivant :

```
1 from microbit import *
2 import radio
3
4 radio.on()
5 while True:
6     if button_a.was_pressed():
7         radio.send("Hello World !!")
```

Sur la carte réceptrice, tapez le message suivant :

```
1 from microbit import *
2 import radio
```

```

3
4 radio.on()
5 while True:
6     incoming = radio.receive()
7     if incoming:
8         display.scroll(incoming)

```

Appuyez sur le bouton A de la carte émettrice et le message défile sur l'autre carte.

Envoyer un couple de 2 entiers

Imaginons que l'on souhaite faire un jeu de bataille navale en réseau, nous allons avoir besoin de transmettre un couple de coordonnées x et y par radio à l'autre carte. Cela se fait selon le même principe que celui décrit ci-dessus. On ajoute seulement une partie de code pour encapsuler et désencapsuler les informations qui nous intéressent dans une chaîne de caractère.

Dans l'exemple qui suit, on fait bouger un pixel sur une carte en utilisant les boutons et on observe sur l'autre carte le pixel bouger de la même manière.

Exemple : Bouger un pixel sur 2 écrans

Sur la carte émettrice :

```

1 from microbit import *
2 import radio
3
4 radio.on()
5
6 def envoi():
7     display.clear()
8     display.set_pixel(x,y,9)
9     message = "{},{}".format(x, y)
10    radio.send(message)
11
12 x, y=0,0
13 envoi()
14 while True:
15     if button_b.was_pressed():
16         x = (x+1)%5
17         envoi()
18     if button_a.was_pressed():
19         y = (y+1)%5
20         envoi()

```

Sur la carte réceptrice :

```

1 from microbit import *
2 import radio
3
4 radio.on()
5
6 while True:
7     incoming = radio.receive()
8     if incoming:
9         try:
10            target_x, target_y = incoming.split(',')
11        except:

```

```

12         continue
13     x, y = int(target_x), int(target_y)
14     display.clear()
15     display.set_pixel(x, y, 9)

```

Complément : Configuration avancée de la radio

Tout ce que l'on vient de voir a l'avantage de marcher immédiatement mais si on fait cela dans une même salle de classe, gare aux interférences ! Il va falloir que chaque groupe d'élèves configure sa radio de manière unique pour éviter les problèmes.

Configuration de la fréquence

```
radio.config(channel=XX)
```

où XX est un numéro entre 0 et 83. Ce paramètre détermine la fréquence d'émission.

Configuration de l'adresse

```
radio.config(address=0x75626974)
```

L'adresse, codée sur 4 octet, permet de filtrer les messages qui nous sont destinés. Seuls ceux provenant d'une carte présentant la même adresse seront pris en compte. L'adresse donnée en exemple peut être modifiée, c'est l'adresse par défaut utilisée.

Configuration du groupe

Au sein d'une même adresse, 256 groupes numérotés de 0 à 255 peuvent cohabiter :

```
radio.config(group=7)
```

seules les cartes possédant la même adresse et le même numéro de groupe communiqueront.

Comme vous le voyez, il est tout à fait possible de faire communiquer plusieurs groupes de personnes sans risquer d'interférences.

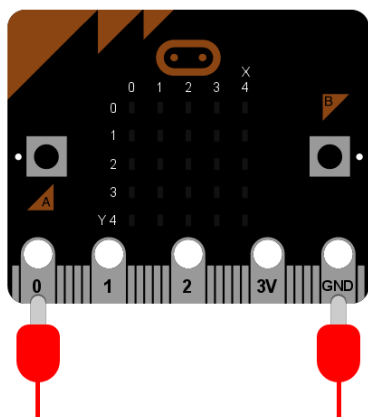
Autres paramètres configurables

- `radio.config(queue=3)` : nombre de messages dans la file d'attente. Au delà de 3 messages en attente, ils seront supprimés.
- `radio.config(length=32)` : longueur maximum du message. Celle-ci peut aller jusqu'à 251.
- `radio.config(power=6)` : puissance d'émission du signal (allant de 0 à 7)
- `radio.config(data_rate=radio.RATE_1MBIT)` : vitesse de transmission. Les vitesses admissibles sont **RATE_250KBIT**, **RATE_1MBIT** ou **RATE_2MBIT**

Remarque : Réinitialiser les paramètres

`radio.reset()` permet de revenir aux réglages par défaut.

5. la gestion du son



L'ajout de bruitages rend souvent les applications plus attractives - sauf peut-être en salle de classe ! En connectant un haut-parleur entre les broches 0 et GND de la carte microbit, il est possible de jouer des sons.

Complément

Il existe des *shields* sur laquelle la carte microbit s'enfiche qui intègrent un haut-parleur. La carte peut alors émettre des sons sans branchements supplémentaires. C'est le cas du *gator:bit V2 de sparkfun* que nous avons déjà rencontré dans la partie sur les entrées-sorties.

Jouer une musique préenregistrée

La carte intègre un certain nombre de musiques réenregistrées pour différentes ambiances. Essayez le code suivant :

```
1 import music
2
3 music.play(music.BIRTHDAY)
```

Difficile de faire plus simple ! voici la liste des musiques disponibles. Pour les retrouver depuis le **REPL**, faites `dir(music)`

```
'DADADADUM', 'ENTERTAINER', 'PRELUDE', 'ODE', 'NYAN', 'RINGTONE', 'FUNK',
'BLUES', 'BIRTHDAY', 'WEDDING', 'FUNERAL', 'PUNCHLINE', 'PYTHON',
'BADDY', 'CHASE', 'BA_DING', 'WAWAWAWAA', 'JUMP_UP', 'JUMP_DOWN',
'POWER_UP', 'POWER_DOWN'
```

Jouer une musique personnalisée

Vous pouvez composer votre propre musique comme vous pouvez le voir sur l'exemple suivant :

```
1 import music
2
3 tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
4         "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
5 music.play(tune)
```

Le format de chaque note est le nom de la note ("A" pour La, "B" pour Si etc...) suivi du numéro de l'octave suivi de ":" et enfin la longueur de la note.

Complément : Simplification d'écriture

Pour simplifier, MicroPython se rappelle les paramètres de la note précédente si ceux-ci sont omis pour la note suivante. La musique précédente peut ainsi se simplifier en

```
tune = ["C4:4", "D", "E", "C", "C", "D", "E", "C", "E", "F", "G:8", "E:4", "F", "G:8"]
```

Effets sonore

On peut enfin utiliser **music.pitch()** pour spécifier une fréquence et une durée précise. L'exemple ci-dessous reproduit une sirène de police et illustre cette méthode.

```
1 import music
2
3 while True:
4     for freq in range(880, 1760, 16):
5         music.pitch(freq, 6)
6     for freq in range(1760, 880, -16):
7         music.pitch(freq, 6)
```

6. Utiliser l'accéléromètre

La carte microbit est livrée avec un accéléromètre. Comme sur les smartphones, elle est capable de détecter son orientation ce qui peut donner naissance à des projets originaux.

L'utilisation est très simple

```
1 from microbit import *
2
3 while True:
4     accX = accelerometer.get_x()
5     if accX > 40:
6         display.show(">")
7     elif accY < -40:
8         display.show("<")
9     else:
10        display.show("-")
```

Complément

Il existe bien sûr **accelerometer.get_x()** et **accelerometer.get_y()**

Détecter des gestes

L'accéléromètre sait aussi interpréter les données d'accélération en gestes prédéfinis. Observez le code ci-dessous :

```
1 from microbit import *
2
3 while True:
4     gesture = accelerometer.current_gesture()
5     if gesture == "face up":
6         display.show(Image.HAPPY)
```

```

7     else:
8         display.show(Image.ANGRY)

```

Complément : Gestes prédéfinis

Les gestes reconnus sont : up, down, left, right, face up, face down, freefall, 3g, 6g, 8g, shake

Comme pour les boutons :

- les gestes sont accumulés dans une pile que l'on peut interroger par `accelerometer.get_gestures()`
- on peut détecter si un geste (par exemple une secousse) a eu lieu par `accelerometer.was_gestured("shake")`
- on peut détecter si un geste est en cours par `accelerometer.is_gesture("up")`

Exemple : Mini projet : le jeu du slalom

```

1 # Simple Slalom by Larry Hastings, September 2015
2 #
3 # This program has been placed into the public domain.
4
5 import microbit as m
6 import random
7
8 p = m.display.show
9
10 min_x = -1024
11 max_x = 1024
12 range_x = max_x - min_x
13
14 wall_min_speed = 400
15 player_min_speed = 200
16
17 wall_max_speed = 100
18 player_max_speed = 50
19
20 speed_max = 12
21
22
23 while True:
24
25     i = m.Image('00000:'*5)
26     s = i.set_pixel
27
28     player_x = 2
29
30     wall_y = -1
31     hole = 0
32
33     score = 0
34     handled_this_wall = False
35
36     wall_speed = wall_min_speed
37     player_speed = player_min_speed
38
39     wall_next = 0

```

```

40     player_next = 0
41
42     while True:
43         t = m.running_time()
44         player_update = t >= player_next
45         wall_update = t >= wall_next
46         if not (player_update or wall_update):
47             next_event = min(wall_next, player_next)
48             delta = next_event - t
49             m.sleep(delta)
50             continue
51
52         if wall_update:
53             # calculate new speeds
54             speed = min(score, speed_max)
55             wall_speed = wall_min_speed + int((wall_max_speed - wall_min_speed) *
speed / speed_max)
56             player_speed = player_min_speed + int((player_max_speed -
player_min_speed) * speed / speed_max)
57
58             wall_next = t + wall_speed
59             if wall_y < 5:
60                 # erase old wall
61                 use_wall_y = max(wall_y, 0)
62                 for wall_x in range(5):
63                     if wall_x != hole:
64                         s(wall_x, use_wall_y, 0)
65
66             wall_reached_player = (wall_y == 4)
67             if player_update:
68                 player_next = t + player_speed
69                 # find new x coord
70                 x = m.accelerometer.get_x()
71                 x = min(max(min_x, x), max_x)
72                 # print("x accel", x)
73                 s(player_x, 4, 0) # turn off old pixel
74                 x = ((x - min_x) / range_x) * 5
75                 x = min(max(0, x), 4)
76                 x = int(x + 0.5)
77                 # print("have", position, "want", x)
78
79                 if not handled_this_wall:
80                     if player_x < x:
81                         player_x += 1
82                     elif player_x > x:
83                         player_x -= 1
84                     # print("new", position)
85                     # print()
86
87             if wall_update:
88                 # update wall position
89                 wall_y += 1
90                 if wall_y == 7:
91                     wall_y = -1
92                     hole = random.randrange(5)
93                     handled_this_wall = False
94
95                 if wall_y < 5:
96                     # draw new wall

```

```

97         use_wall_y = max(wall_y, 0)
98         for wall_x in range(5):
99             if wall_x != hole:
100                 s(wall_x, use_wall_y, 6)
101
102         if wall_reached_player and not handled_this_wall:
103             handled_this_wall = True
104             if (player_x != hole):
105                 # collision! game over!
106                 break
107             score += 1
108
109         if player_update:
110             s(player_x, 4, 9) # turn on new pixel
111
112         p(i)
113
114     p(i.SAD)
115     m.sleep(1000)
116     m.display.scroll("Score:" + str(score))
117
118     while True:
119         if (m.button_a.is_pressed() and m.button_a.is_pressed()):
120             break
121         m.sleep(100)

```

7. Utilisation de la boussole

La carte microbit est équipée d'un magnétomètre pouvant servir de boussole. Son utilisation comme pour le reste de ses capteurs est très simple : on calibre le compas

par **compass.calibrate()** puis on interroge le compas par **compass.heading()**. Essayez ces deux commandes dans le **REPL**.

Pour savoir si le compas est déjà calibré, utilisez la méthode **compass.is_calibrated()**.

Exemple : Réalisation d'une boussole

Voici comment en quelques lignes réaliser une boussole !

```

1 from microbit import *
2
3 if not compass.is_calibrated():
4     compass.calibrate()
5
6 while True:
7     needle = ((15 - compass.heading()) // 30) % 12
8     display.show(Image.ALL_CLOCKS[needle])

```

Autres fonctions du magnétomètre

- La méthode **compass.get_field_strength()** renvoie la force du champ magnétique. Cela permet de détecter la présence d'un aimant.
- Cette force peut être décomposée selon les axes x, y et z : **compass.get_x()** **compass.get_y()** et **compass.get_z()**.