OBJETS CONNECTÉS

Carte micro :bit

de débutant à expert

Nom : ..... Prénom : ..... Classe : .....

Objectif. Tester des programmes courts pour comprendre le fonctionnement de la carte micro :bit (afficheur, boutons, capteurs) puis piloter le lampadaire LAMP :bit (lampe et capteur de lumière).
Matériel. Une carte micro :bit et son câble USB, un lampadaire LAMP :bit, un ordinateur avec le logiciel Mu.
Méthode. Pour chaque savoir-faire (SF), suivre les cinq étapes :

1. Lire le programme → **2. Prédire** ce qu'il va faire → **3. Tester** sur la carte → **4. Répondre** aux questions → **5. Relever** le défi

À la fin, compléter la **grille d'auto-évaluation** (page 12).

1. Présentation de la carte micro :bit

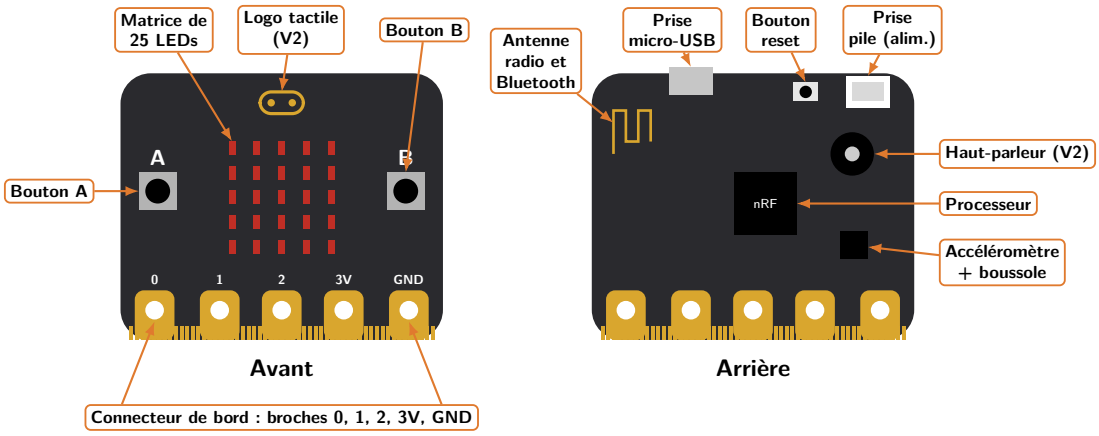
Ressources sur la carte micro :bit

— en français : `https://microbit.org/fr/`
— documentation MicroPython (anglais) : `https://microbit-micropython.readthedocs.io`
— le langage MicroPython : `http://micropython.org/`

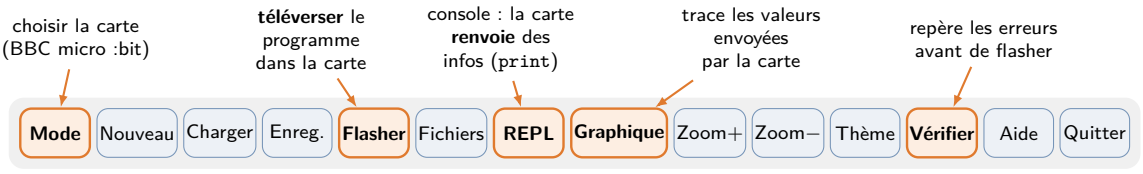
Logiciels

— Mu : `https://codewith.mu/en/download`
— sans installation : éditeur en ligne `https://python.microbit.org`

La carte micro :bit est un petit ordinateur (un *microcontrôleur*) équipé de capteurs et d'actionneurs.



Le logiciel Mu (choisir le mode « BBC micro :bit » au premier lancement) :



2. Savoir-faire sur la carte micro :bit

SF 1 Afficher un caractère sur la matrice de LEDs

1. Démarrer Mu et brancher la carte micro :bit avec le câble USB.
2. Recopier le programme ci-contre dans l'éditeur de Mu.
3. Cliquer sur **Flasher** pour téléverser le programme dans la carte.

Pendant le téléversement, la LED orange au dos de la carte clignote rapidement, puis reste fixe : le programme est chargé.

Programme

```
1 from microbit import *
2
3 display.show("A")
```

⚠ Si le téléversement échoue (message d'erreur, ou LED qui clignote très longtemps), cliquer à nouveau sur **Flasher**.

À retenir — Structure d'un programme

— `from microbit import *` doit **toujours** figurer au début du programme. Elle *importe* la bibliothèque `microbit`, qui contient toutes les instructions de la carte (`display`, `sleep`, `button_a`…).

— `display.show("A")` affiche le caractère `A` sur la matrice. On peut afficher une lettre, un nombre ou une image.

— `display.scroll("Bonjour")` fait **défiler** un texte plus long.

Savoir-faire : *afficher un message sur la matrice de LEDs.*

Questions de compréhension

1) Que signifie la ligne `from microbit import *` ?

.....

.....

2) Que va afficher l'instruction `display.show("B")` ?

.....

3) Supprimer la première ligne, flasher et observer. Que se passe-t-il ?

.....

.....

4) Tester `display.show("Bonjour")` puis `display.scroll("Bonjour")`. Quelle différence ?

.....

.....

SF 2 Répéter avec une boucle « pour » (`for`)

Afficher les entiers de 1 à 5 sur la matrice de LEDs. Recopier le programme ci-contre dans l'éditeur de Mu, puis le tester.

⚠ Les lignes 4 à 6 sont **décalées** (indentées) de 4 espaces : utiliser la touche Tab.

Programme

```
1 from microbit import *
2
3 for i in range(1, 6):
4     display.show(i)
5     sleep(1000)
6     display.clear()
```

À retenir — La boucle `for` (pour)

Une boucle `for` permet de **répéter** des instructions un **nombre de fois connu à l'avance**.

— `i` est une **variable** qui prend successivement les valeurs 1, 2, 3, 4, 5.

— `range(1, 6)` donne les entiers de 1 **jusqu'à 6 exclu** : 1, 2, 3, 4, 5. (`range(5)` donne 0, 1, 2, 3, 4.)

— Les instructions **indentées** après les deux-points « : » sont répétées à chaque tour de boucle.

— `sleep(1000)` met le programme en pause pendant 1000 ms, soit 1 seconde.

— `display.clear()` éteint toutes les LEDs.

Déroulement pas à pas — compléter le tableau :

Tour	Valeur de `i`	Ce qui se passe
1	1	affiche 1, attend 1 s, efface
2	2	affiche 2, attend 1 s, efface
3		
4		
5		

Savoir-faire : *utiliser une boucle `for` pour répéter des instructions un nombre défini de fois.*

Questions de compréhension

1) Combien de fois la boucle s'exécute-t-elle dans le programme ci-dessus ?

.....

2) Que signifie `range(1, 6)` ? Pourquoi le nombre 6 n'est-il pas affiché ?

.....

.....

3) Que faut-il écrire pour afficher les nombres de 0 à 9 ?

.....

4) Que faut-il modifier pour que chaque nombre reste affiché 2 secondes ?

.....

5) Que se passe-t-il si on supprime la ligne `display.clear()` ? Tester.

.....

.....

SF 3 Allumer une LED précise de la matrice

La matrice comporte 5 colonnes numérotées de 0 à 4 et 5 lignes numérotées de 0 à 4.
Le programme ci-contre allume la LED située à l'intersection de la **3e colonne** et de la **4e ligne**. Le nombre 9 est l'intensité lumineuse : on peut la choisir entre 0 et 9.

Programme

```
1 from microbit import *
2
3 display.set_pixel(2, 3, 9)
```

À retenir — Allumer une LED précise

`display.set_pixel(x, y, intensité)`

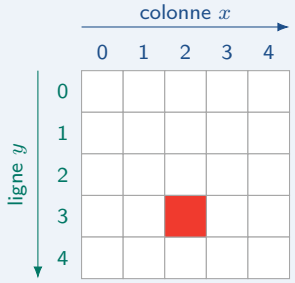
— `x` : numéro de **colonne**, de 0 à 4 (de gauche à droite) ;

— `y` : numéro de **ligne**, de 0 à 4 (de haut en bas) ;

— `intensité` : de 0 (éteinte) à 9 (très lumineuse).

⚠ En informatique, on compte souvent à partir de **0** et non de 1 !

Exemple : `display.set_pixel(2, 3, 9)` allume la LED de la colonne 2 (3e colonne), ligne 3 (4e ligne), avec l'intensité maximale.



Savoir-faire : *allumer une LED individuelle de la matrice.*

Questions de compréhension

1) Quel est le numéro de la première colonne ? Et celui de la dernière ?

.....

2) Que faut-il écrire pour allumer la LED en haut à gauche ?

.....

3) Que faut-il écrire pour allumer la LED au centre de la matrice ?

.....

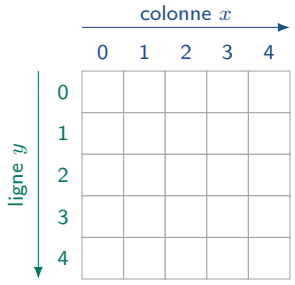
4) Quelle différence entre `display.set_pixel(0, 0, 9)` et `display.set_pixel(0, 0, 3)` ?

.....

5) **Prédire** : colorier sur la grille les LEDs allumées par le programme ci-dessous, puis vérifier sur la carte.

Programme

```
1 from microbit import *
2
3 display.set_pixel(0, 4, 9)
4 display.set_pixel(4, 0, 9)
5 display.set_pixel(1, 2, 4)
```



★ Défi n° 1

Si vous maîtrisez tout ce qui précède : en utilisant une boucle `for`, allumer entièrement la **première ligne** de la matrice.

SF 4 Afficher une image

Recopier le programme ci-contre dans l'éditeur de Mu, puis le tester. Remplacer ensuite `HEART` par d'autres noms de la liste ci-dessous.

Programme

```
1 from microbit import *
2
3 display.show(Image.HEART)
```

Images prédéfinies (toutes s'écrivent `Image.`*NOM*) :

`HEART`, `HEART_SMALL`, `HAPPY`, `SMILE`, `SAD`, `CONFUSED`, `ANGRY`, `ASLEEP`, `SURPRISED`, `SILLY`, `FABULOUS`, `MEH`, `YES`, `NO`, `CLOCK1` à `CLOCK12`, `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`, `TSHIRT`, `ROLLERSKATE`, `DUCK`, `HOUSE`, `TORTOISE`, `BUTTERFLY`, `STICKFIGURE`, `GHOST`, `SWORD`, `GIRAFFE`, `SKULL`, `UMBRELLA`, `SNAKE`.

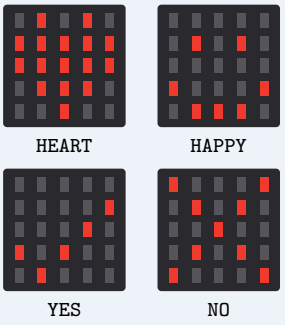
À retenir — Afficher des images

`display.show(Image.NOM_IMAGE)`

— MicroPython propose des images prédéfinies.

— Toujours écrire `Image.` (avec un I majuscule) suivi du nom en **MAJUSCULES**.

— On peut aussi **créer son image** : chaque chiffre est l'intensité d'une LED, ligne par ligne, les lignes étant séparées par « : ».
`Image("09090:99999:99999:09990:00900")` donne le cœur.



Savoir-faire : *afficher une image prédéfinie ou personnalisée.*

Questions de compréhension

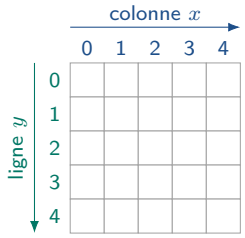
1) Que faut-il écrire pour afficher un visage triste ?

.....

2) Quelle est la différence entre `Image.HEART` et `Image.HEART_SMALL` ?

.....

3) **Prédire** : dessiner l'image affichée par l'instruction
`display.show(Image("00900:09990:99999:00900:00900"))`



SF 5 Mesurer la luminosité — boucle « tant que » (`while`)

1. Recopier le programme ci-contre puis cliquer sur **Flasher**.
2. Cliquer sur **REPL**, puis appuyer sur le bouton **reset** au dos de la carte.
3. La luminosité reçue par la matrice (entre 0 et 255) s'affiche dans la console toutes les secondes. Cacher la carte avec la main pour faire varier la valeur.

Programme

```
from microbit import *

while True:
    a = display.read_light_level()
    print(a)
    sleep(1000)
```

⚠ Lorsque la console REPL est ouverte, le bouton **Flasher** n'est plus disponible : fermer la console (bouton REPL) avant de téléverser un nouveau programme.

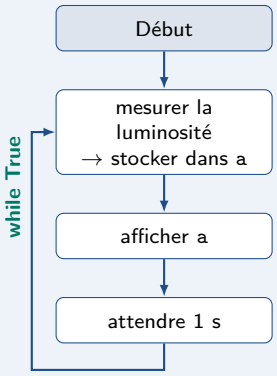
À retenir — La boucle `while` (tant que)

Une boucle `while` répète des instructions **tant qu'une condition est vraie**.

— `for` : on sait **combien de fois** on répète (par exemple 5 fois) ;

— `while` : on répète **tant que** quelque chose est vrai.

— `True` signifie « vrai » : `while True:` ne s'arrête donc **jamais** (boucle infinie). C'est le cas de la plupart des objets connectés, qui surveillent leurs capteurs en permanence.



Vocabulaire

— `display.read_light_level()` mesure la luminosité ambiante : un nombre entre 0 (obscurité) et 255 (forte lumière).

— `a = ...` : la **variable** `a` stocke la valeur mesurée.

— `print(a)` affiche la valeur de `a` dans la console (REPL).

Savoir-faire : *utiliser une boucle `while` pour répéter des instructions sans fin.*

Questions de compréhension

1) Pourquoi utilise-t-on `while True` et non `for` dans ce programme ?

.....

.....

2) Que signifie la valeur 0 renvoyée par `display.read_light_level()` ? Et la valeur 255 ?

.....

3) À quoi sert la variable `a` dans ce programme ?

.....

4) Que faut-il modifier pour afficher la luminosité toutes les 2 secondes ?

.....

Pour aller plus loin : remplacer `print(a)` par `print((a,))` puis cliquer sur **Graphique** : Mu trace la courbe de la luminosité en direct.

SF 6 Réagir aux boutons — test « si... alors... sinon » (`if`)

Recopier le programme ci-contre dans l'éditeur de Mu, puis le tester en appuyant sur le bouton A.

Programme

```
1 from microbit import *
2
3 while True:
4     if button_a.is_pressed():
5         display.show("A")
```

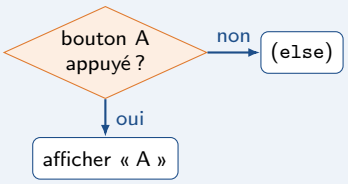
À retenir — Le test `if` (si) — `elif` (sinon si) — `else` (sinon)

Un test `if` exécute des instructions **seulement si** une condition est vraie.

- `button_a.is_pressed()` vaut `True` si le bouton A est enfoncé, `False` sinon.
- Si la condition est vraie, on exécute le bloc indenté ; sinon on ne fait rien…
- … ou on exécute le bloc `else` s'il existe.
- `elif` (contraction de *else if*) permet d'enchaîner plusieurs tests.

Indentation : les instructions qui dépendent de `if`, `elif` ou `else` doivent être décalées. Sans le décalage, Python ne sait pas qu'elles appartiennent au test.

```
if condition1:
    # si condition1 est vraie
elif condition2:
    # sinon, si condition2 est vraie
else:
    # dans tous les autres cas
```



Savoir-faire : *utiliser un test `if` pour exécuter des instructions selon une condition.*

Questions de compréhension

1) Que signifie `if` en français ?

2) Dans le programme, que se passe-t-il si on n'appuie pas sur le bouton A ?

3) Que faut-il ajouter pour afficher « B » quand on appuie sur le bouton B ?

4) Pourquoi faut-il placer le test `if` dans une boucle `while True` ?

5) À quoi sert l'indentation (le décalage des lignes de code) ?

★ Défi n° 2

Afficher un petit cœur (`Image.HEART_SMALL`) lorsque la luminosité est inférieure à 30, un gros cœur (`Image.HEART`) sinon.

★ Défi n° 3

Afficher un petit cœur lorsqu'on appuie sur le bouton A, un gros cœur lorsqu'on appuie sur le bouton B, et la lettre « X » sinon. *Indice : utiliser `elif`.*

SF 7 Détecter un mouvement avec l'accéléromètre

Recopier le programme ci-contre dans l'éditeur de Mu, puis le tester en secouant la carte.

Programme

```
from microbit import *

while True:
    if accelerometer.current_gesture() == "shake":
        display.show(Image.HEART)
    else:
        display.show(Image.HEART_SMALL)
```

À retenir — Détecter un geste

`accelerometer.current_gesture()` renvoie le geste détecté *en ce moment* par l'accéléromètre.

- On compare le geste détecté avec `==` (« est égal à »). ⚠ Ne pas confondre avec `=`, qui sert à donner une valeur à une variable.
- Gestes possibles : `"shake"` (secouer), `"face up"` (écran vers le haut), `"face down"` (écran vers le bas), `"up"`, `"down"`, `"left"` (penchée à gauche), `"right"` (penchée à droite), `"freefall"` (chute libre).
- `accelerometer.was_gesture("shake")` vaut `True` si la carte **a été** secouée depuis la dernière vérification.

Savoir-faire : *détecter un geste avec l'accéléromètre.*

Questions de compréhension

1) Que fait l'instruction `accelerometer.current_gesture()` ?

.....

2) Que signifie `==` dans le programme ?

.....

3) Que se passe-t-il quand on secoue la carte ? Et quand on ne la secoue pas ?

.....

4) Modifier le programme pour afficher une flèche vers la gauche lorsque la carte est penchée à gauche.

.....

.....

★ Défi n° 4

Le dé électronique. Compléter le programme ci-contre pour afficher un entier entre 1 et 6 lorsqu'on secoue la carte.

⚠ `randint(1, 6)` tire au hasard un entier entre 1 et 6, **6 inclus** (contrairement à `range`).

Programme

```
from microbit import *
from random import *

while True:
    entier = randint(1, 6)
    if accelerometer.was_gesture("shake"):
        ......................
```

3. Savoir-faire sur le lampadaire LAMP :bit

Le lampadaire LAMP :bit s'enfiche sur le **connecteur de bord** de la micro :bit. Il comporte une lampe (LED « LAMP ») et un capteur de lumière (« DETECT »).

SF 8 Identifier les broches utilisées

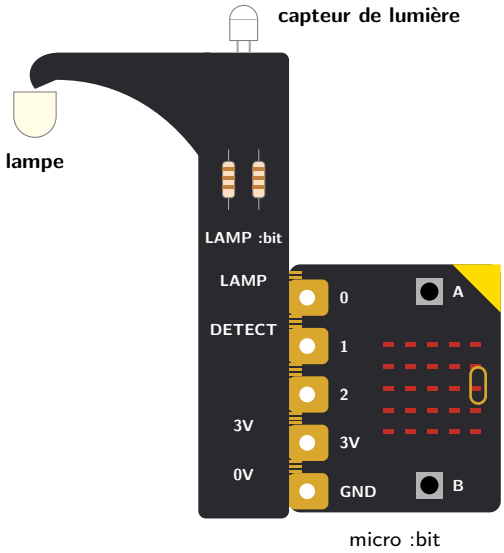
En observant la sérigraphie et les pistes du lampadaire, compléter avec le numéro de broche :

— la LED « LAMP » est reliée à la broche ..... ;

— le capteur « DETECT » est relié à la broche ..... ;

— l'alimentation « 3V » est reliée à la broche ..... ;

— la masse « 0V » est reliée à la broche ..... .



À retenir — Les broches (*pins*)

Les broches sont les connecteurs du bas de la carte. Elles permettent de la relier à d'autres composants (LED, capteurs, moteurs...). Chaque broche a un numéro (`pin0`, `pin1`, `pin2`...) et peut servir :

— de **sortie** : la carte **envoie** un signal (pour allumer une LED, par exemple) ;

— d'**entrée** : la carte **lit** un signal (pour mesurer la valeur d'un capteur).

Savoir-faire : *identifier les connexions d'un composant.*

SF 9 Commander une sortie numérique : allumer la lampe

Allumer la lampe pendant deux secondes, puis l'éteindre.
Recopier le programme ci-contre dans l'éditeur de Mu, puis le tester.

Programme

```
1 from microbit import *
2
3 pin0.write_digital(1)
4 sleep(2000)
5 pin0.write_digital(0)
```

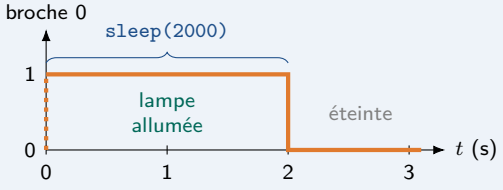
À retenir — Sortie numérique (*digital*)

`pin0.write_digital(valeur)` envoie un signal **numérique** sur la broche 0 :

— `1` : signal **haut** → la lampe s'allume ;

— `0` : signal **bas** → la lampe s'éteint.

Numérique (*digital* en anglais) : seulement **deux états** possibles, 0 ou 1, comme un interrupteur (ouvert ou fermé).



Savoir-faire : *commander une sortie numérique (allumer, éteindre une LED).*

Questions de compréhension

1) Que signifie « numérique » (*digital*) ?

.....

2) Quelles sont les deux valeurs possibles pour `write_digital` ?

.....

3) Que fait l'instruction `pin0.write_digital(1)` ?

.....

4) Combien de temps la lampe reste-t-elle allumée ? Que faut-il modifier pour qu'elle reste allumée 5 secondes ?

.....

★ Défi n° 5

Faire clignoter 10 fois la lampe du lampadaire en utilisant une boucle (une demi-seconde allumée, une demi-seconde éteinte).

★ Défi n° 6

Allumer le lampadaire lorsqu'on appuie sur le bouton A et l'éteindre lorsqu'on appuie sur le bouton B.

SF 10 Lire une entrée analogique : le capteur de lumière

1. Recopier le programme ci-contre puis le flasher.
2. Cliquer sur **REPL** et appuyer sur le bouton **reset** de la carte.
3. Cacher le petit capteur du lampadaire avec le doigt et observer les valeurs dans la console.

Programme

```
1 from microbit import *
2
3 while True:
4     nomquejeveux = pin1.read_analog()
5     print(nomquejeveux)
6     sleep(1000)
```

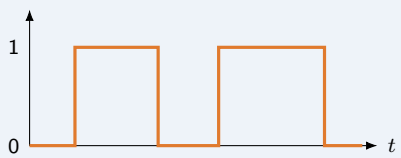
À retenir — Entrée analogique

`pin1.read_analog()` **lit** le signal présent sur la broche 1 et renvoie un nombre entier **entre 0 et 1023**.

— valeur faible (proche de 0) : peu de lumière ; valeur élevée (proche de 1023) : beaucoup de lumière.

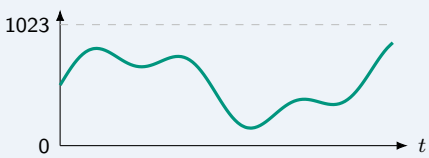
— Dans le programme, la valeur lue est stockée dans la variable `nomquejeveux`, affichée dans la console, et la mesure est répétée chaque seconde.

Numérique : 2 valeurs



ex. : interrupteur, lampe

Analogique : de 0 à 1023



ex. : luminosité, température

Savoir-faire : *lire la valeur d'un capteur sur une entrée analogique.*

Questions de compréhension

1) Quelle est la différence entre un signal numérique et un signal analogique ?

.....

.....

2) Pourquoi utilise-t-on `read_analog` et non `read_digital` pour le capteur de lumière ?

.....

.....

3) Quelle est la valeur minimale que peut renvoyer `read_analog` ? Et la valeur maximale ?

.....

4) Peut-on remplacer `nomquejeveux` par `luminosite` ? Pourquoi ?

.....

.....

★ Défi n° 7

Le lampadaire automatique. Allumer la lampe du lampadaire lorsque la luminosité mesurée est inférieure à 200, l'éteindre sinon.

★ Défi n° 8 (expert)

Le lampadaire économe. `pin0.write_analog(v)`, avec `v` entre 0 et 1023, règle la puissance de la lampe (0 : éteinte, 1023 : puissance maximale). Écrire un programme pour lequel **plus il fait sombre, plus la lampe éclaire**.
Si la lampe se met à clignoter, c'est peut-être qu'elle éclaire son propre capteur... Comment l'éviter ?

4. Bilan

Aide-mémoire MicroPython pour la micro :bit

Instruction	Rôle
`from microbit import *`	importer les instructions de la carte (toujours en 1re ligne)
`from random import *`	importer `randint` (tirage au hasard)
`display.show(x)`	afficher un caractère, un nombre ou une image
`display.scroll("texte")`	faire défiler un texte
`display.clear()`	éteindre toute la matrice
`display.set_pixel(x, y, v)`	allumer la LED colonne `x`, ligne `y`, intensité `v` (0 à 9)
`display.show(Image.HEART)`	afficher une image prédéfinie
`sleep(ms)`	faire une pause (en millisecondes : 1000 ms = 1 s)
`print(a)`	afficher la valeur de `a` dans la console (REPL)
`for i in range(a, b):`	répéter pour `i` = `a`, `a`+1, …, `b`−1
`while True:`	répéter indéfiniment
`if ...: / elif ...: / else:`	si… / sinon si… / sinon
`==  !=  <  >`	égal, différent, inférieur, supérieur
`button_a.is_pressed()`	`True` si le bouton A est enfoncé
`display.read_light_level()`	luminosité mesurée par la matrice (0 à 255)
`accelerometer.current_gesture()`	geste en cours : `"shake"`, `"left"`, `"face up"`…
`accelerometer.was_gesture("shake")`	`True` si la carte a été secouée depuis la dernière vérification
`randint(a, b)`	entier au hasard entre `a` et `b` (`b` inclus)
`pin0.write_digital(1)`	sortie numérique : 1 = allumé, 0 = éteint
`pin0.write_analog(v)`	sortie réglable : `v` de 0 à 1023
`pin1.read_analog()`	entrée analogique : valeur de 0 à 1023

⚠ Erreurs fréquentes

- oubli des **deux-points** « : » à la fin des lignes `for`, `while`, `if`, `elif`, `else` ;
- **indentation** absente ou irrégulière (mélange d'espaces et de tabulations) ;
- **majuscules** : `True` et `Image` prennent une majuscule, `display` n'en prend pas ;
- confusion entre `=` (donner une valeur) et `==` (comparer) ;
- console REPL encore ouverte au moment de flasher.

Grille d'auto-évaluation

SF	Je sais…	oui – en cours – non	Défi réussi
1	afficher un message sur la matrice de LEDs	❑ ❑ ❑	—
2	répéter des instructions avec une boucle `for`	❑ ❑ ❑	—
3	allumer une LED précise avec `set_pixel`	❑ ❑ ❑	n° 1 ❑
4	afficher une image prédéfinie ou personnalisée	❑ ❑ ❑	—
5	répéter sans fin avec `while True` et lire la luminosité	❑ ❑ ❑	—
6	utiliser un test `if` / `elif` / `else` avec les boutons	❑ ❑ ❑	n° 2 ❑ n° 3 ❑
7	détecter un geste avec l'accéléromètre	❑ ❑ ❑	n° 4 ❑
8	identifier les broches reliées à un composant	❑ ❑ ❑	—
9	commander une sortie numérique (`write_digital`)	❑ ❑ ❑	n° 5 ❑ n° 6 ❑
10	lire une entrée analogique (`read_analog`)	❑ ❑ ❑	n° 7 ❑ n° 8 ❑