Bibliothèque des classes du module ESP32

Ecole Robots
Apprendre à coder avec les robots éducatifs

ArTeC®

Sommaire

1. Démarrer

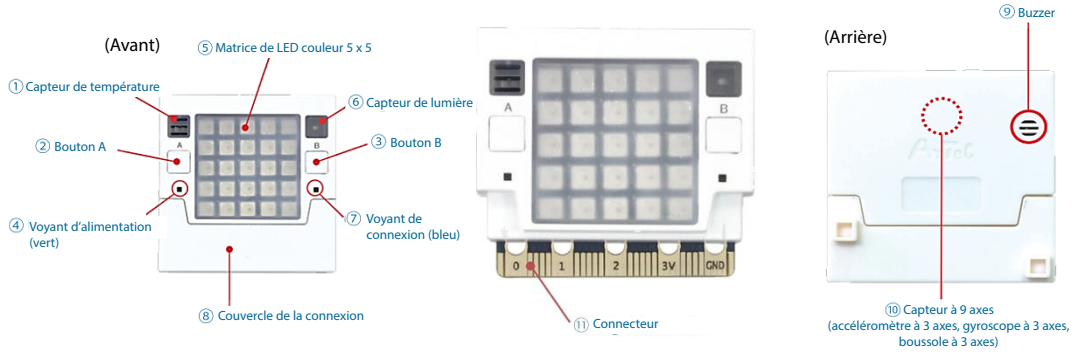
Ce manuel référence les classes de la bibliothèque de l'unité centrale ArtecRobot 2.0. Ce manuel requiert la maîtrise de quelques bases en langage Python.

2. Studuino:bit

Studuino:bit est un appareil informatique équipé d'un microcontrôleur, de capteurs, de LED et d'autres éléments. MicroPython est l'un des environnements de programmation compatible avec Studuino:bit.

2.1. Configuration du Studuino:bit

La configuration du Studuino:bit est la suivante :



Les éléments utilisables avec la bibliothèque des classes Studuino:bit sont le capteur de température (1), le bouton A (2), le bouton B (3), la matrice de LED couleur 5 x 5 (5), le capteur de lumière (6), le buzzer (9), le capteur à 9 axes (10) (accéléromètre à 3 axes, gyroscope à 3 axes et boussole à 3 axes) et le connecteur (11).

Désormais, la matrice de LED couleur 5x5 sera nommée panneau LED. Les 3 éléments du capteur à 9 axes (accéléromètre à 3 axes, gyroscope à 3 axes et boussole à 3 axes) seront nommés respectivement accéléromètre, gyrosocope et boussole. Le connecteur sera nommé par ses ports.

2.2. MicroPython

MicroPython est une version de Python développée spécifiquement pour être utilisée avec des microcontrôleurs. La syntaxe de programmation est la même que celle de Python 3.0. Cependant certaines bibliothèques Python ne sont pas utilisables en MicroPython parce qu'elles n'ont pas été conçues pour fonctionner avec la mémoire limitée et le CPU d'un microcontrôleur. Toutefois, quelques bibliothèques MicroPython spécifiques (comme la bibliothèque des microcontrôleurs GPIO) sont incluses comme standard.

MicroPython a été largement diffusé et la bibliothèque Studuino:bit utilise une version spécifique de Studuino:bit.

3. Environnement de développement

uPyCraft est un des environnements de programmation utilisés pour MicroPython. Voir la partie **5.2** pour apprendre à l'utiliser.

4. Bibliothèque Studuino:bit

La bibliothèque des classes de Studuino:bit est structurée ainsi :

Paquet	Module	Classe	Élément
pystubit	dsply	StuduinoBitDisplay	panneau LED
	image	StuduinoBitImage	
	bzr	StuduinoBitBuzzer	Buzzer
	button	StuduinoBitButton	Boutons A/B
	sensor	StuduinoBitLightSensor	Capteur de lumière
		StuduinoBitTemperature	Capteur de température
		StuduinoBitAccelerometer	Accéléromètre
		StuduinoBitGyro	Gyroscope
		StuduinoBitCompass	Boussole
	terminal	StuduinoBitTerminal	Ports

4.1. La classe StuduinoBitDisplay

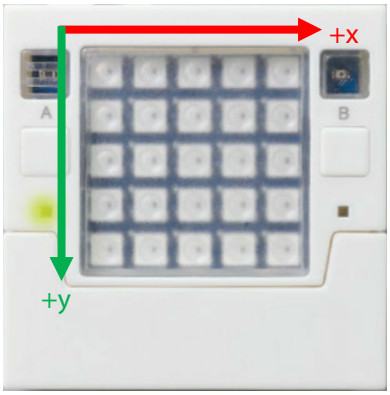
La classe StuduinoBitDisplay sert à contrôler le panneau LED (matrice LED de couleur 5x5) situé sur la face avant de la carte. Cette classe peut être utilisée pour que les LED affichent des images, des animations et du texte.

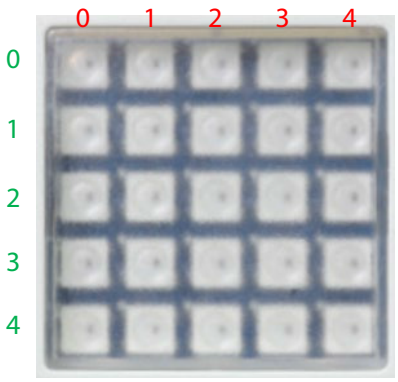
4.1.1. Constructeurs

Utilisez-les pour créer un objet qui contrôle l'affichage des LED.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
```

4.1.2. Paramètres des LED

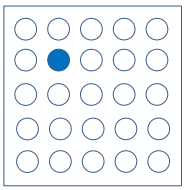




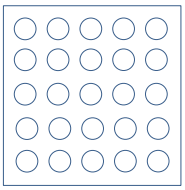
Utilisez la méthode **set_pixel(x, y, color)** pour changer les paramètres d'une LED spécifique. Chaque LED du panneau est désignée par des coordonnées x-y spécifiques (voir image ci-dessus). Le paramètre **x** dans la méthode **set_pixel** sélectionne la coordonnée x, le paramètre **y** sélectionne la coordonnée y et **color** règle la couleur. Le paramètre **color** peut être spécifié avec une liste, un tuple ou un nombre entier. Dans le cas d'un tuple ou d'une liste, utilisez (R, G, B) ou [R, G, B] pour régler les niveaux de lumière rouge, vert et bleue. Dans le cas d'un nombre entier, réglez les couleurs avec un code hex comme montré ci-dessous. Vous pouvez aussi utiliser la méthode **clear()** pour régler la luminosité de toutes les LED sur le panneau à **0** (éteintes).


```
from pystubit.dsply import StduinoBitDisplay
display = StduinoBitDisplay()
display.set_pixel(1, 1, (0, 0, 10))
display.clear()
```

Ces méthodes font toutes réagir le panneau LED comme ci-dessous.



Exécution de la méthode set_pixel



Exécution de la méthode clear

4.1.3. Récupérer des données des LED

Utilisez la méthode **get_pixel(x, y)** pour obtenir des informations sur des LED spécifiques. Réglez les paramètres x et y qui correspondent aux coordonnées x-y de la LED en question sur le panneau pour trouver la couleur de la LED en format tuple.

```
from pystubit.dsply import StduinoBitDisplay
display = StduinoBitDisplay()
display.set_pixel(1, 1, (0, 0, 10))
c1 = display.get_pixel(0, 0)
c2 = display.get_pixel(1, 1)
```

La variable **c1** obtient les données (0, 0, 0) de la LED se trouvant aux coordonnées (0, 0) du panneau, tandis que la variable **c2** obtient les données (0, 0, 10) des coordonnées (1, 1).

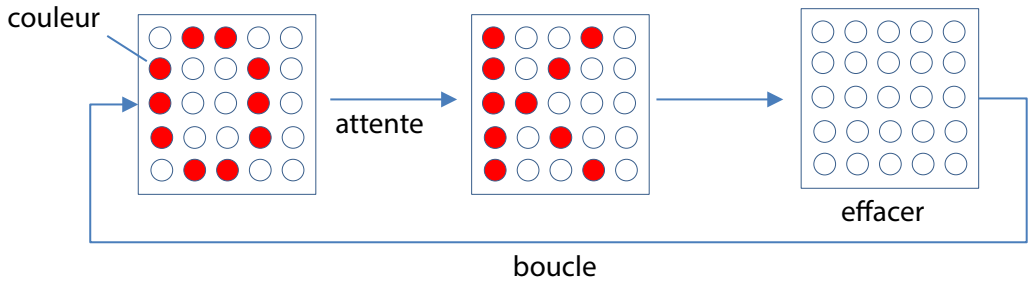
4.1.4. Animation

La méthode **show(value, delay, wait=True, loop=False, clear=False, color=None)** peut être utilisée pour afficher des lettres et des images avec des LED.

N'importe quelle chaîne de nombres et de lettres réglée dans le paramètre **value** est affichée dans l'ordre par les LED. Si vous définissez des ensembles pour des objets image (voir partie **4.2**) dans le paramètre **value**, les LED afficheront chaque image dans l'ordre. Le paramètre **delay** peut être utilisé pour régler la vitesse à laquelle l'affichage passe d'une image à l'autre en millisecondes. Si le paramètre **wait** est réglé sur **True**, les autres méthodes seront bloquées jusqu'à ce que l'animation soit finie. S'il est réglé sur **False**, les autres méthodes s'exécuteront en arrière-plan. Si le paramètre **loop** est réglé sur **True**, l'animation se lancera en boucle. Si le paramètre **clear** est réglé sur **True**, le panneau LED s'effacera une fois l'animation finie. Réglez la couleur de l'image affichée avec le paramètre **color**. Le paramètre **color** peut être spécifié avec une liste, un tuple ou un nombre entier.

```
from pystubit.dsply import StduinoBitDisplay
display = StduinoBitDisplay()
display.show('OK', 100, wait=True, loop=True, clear=True, color=(10, 0, 0))
```

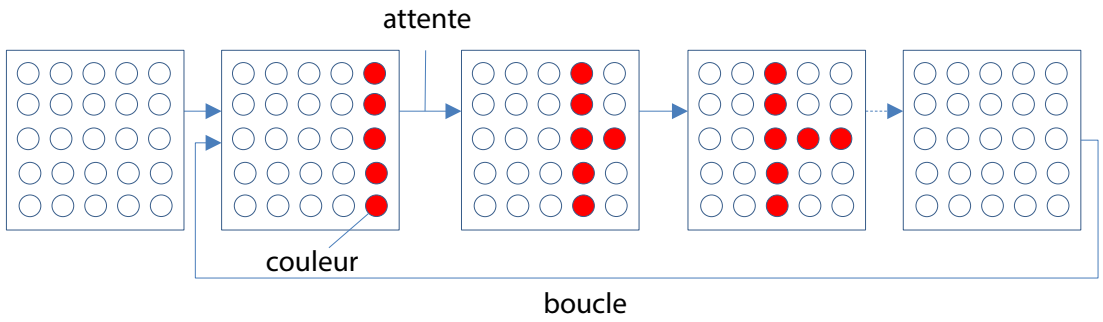
Cette méthode **show** fait afficher la lettre O en rouge pendant 100 millisecondes, la lettre K pendant 100 millisecondes, puis efface l'affichage. Cette animation tourne en boucle.



La méthode **display.scroll(string,delay,wait=True,loop=False,color=None)** peut être utilisée pour faire défiler une série de caractères sur le panneau. Il est possible de régler la chaîne de lettres et de nombres lors de son défilement avec le paramètre **string**. Utilisez le paramètre **delay** pour régler la vitesse de défilement. Si le paramètre **wait** est réglé sur **True**, les autres méthodes seront bloquées jusqu'à ce que l'animation ait fini de s'afficher. S'il est réglé sur **False**, les autres méthodes s'exécuteront en arrière-plan. Si le paramètre **loop** est réglé sur **True**, l'animation tournera en boucle. Réglez la couleur de l'image affichée avec le paramètre **color**. Le paramètre **color** peut être spécifié avec une liste, un tuple ou un nombre entier.

```
from pystubit.dsply import StduinoBitDisplay
display = StduinoBitDisplay()
display.scroll( 'Hello' , 100, wait=True, loop=True, color=(10, 0, 0))
```

Cette méthode **scroll** fait défiler la chaîne Hello en rouge en boucle, comme montré ci-dessous.

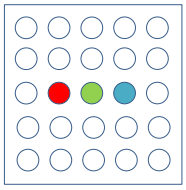


4.1.5. Les paramètres d'alimentation du panneau d'affichage

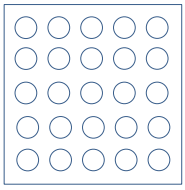
Même si la luminosité de toutes les LED du panneau est réglée à 0, le panneau continuera de consommer de l'énergie. Vous pouvez utiliser la méthode **off()** pour éteindre le panneau de LED. De la même manière, la méthode **on()** peut être utilisée pour allumer le panneau de LED. Vérifiez l'état de l'alimentation du panneau avec la méthode **is_on()**.

```
from pystubit.dsply import StduinoBitDisplay
display = StduinoBitDisplay()
display.set_pixel(1, 2, (10, 0, 0))
display.set_pixel(2, 2, (0, 10, 0))
display.set_pixel(3, 2, (0, 0, 10))
display.off()
ds1 = display.is_on()
display.on()
ds2 = display.is_on()
```

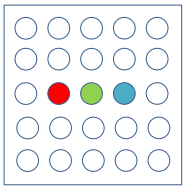
Dans l'exemple ci-dessus, le panneau LED allume la LED aux coordonnées (1,2) en rouge, (2,2) en vert et (3,2) en bleu. La méthode **off** éteint les LED et la méthode **on** les rallume.



Avant l'exécution de la méthode off



Exécution de la méthode off

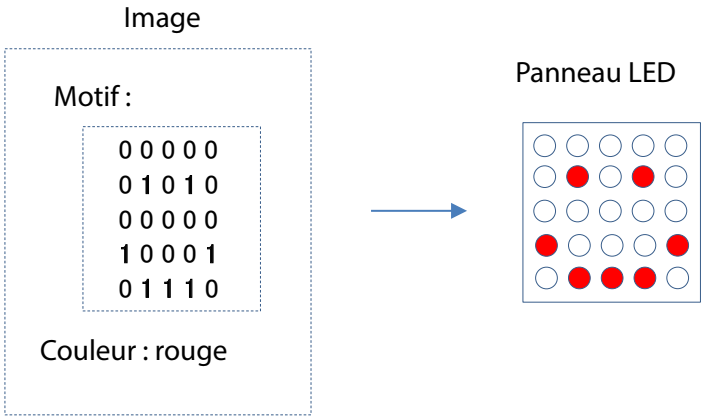


Exécution de la méthode on

La variable **ds1** sera réglée sur False, tandis que la variabel **ds2** sera réglée sur **True**.

4.2. La classe StuduinoBitImage

La classe StuduinoBitImage facilite la création d'images à afficher avec les LED. Les images sont constituées d'un motif et de données de couleur que les LEDs doivent afficher. Les motifs sont écrits avec une série de 0 (qui signifient OFF) et de 1 (qui signifient ON).
Ci-dessous, un exemple de conversion d'une image en affichage LED.



4.2.1. Constructeurs

Il existe 4 façons de créer une image.

StuduinoBitImage(*,color)	Crée une image avec les 5x5 espaces (tous 0).
StuduinoBit Image(string,*,color)	Crée un motif avec une chaîne de 0 et de 1 dans le paramètre **string**.
StuduinoBitImage(width, height,*, color)	Crée une image en utilisant des espaces d'une largeur (paramètre **width**) et d'une hauteur (paramètre **height**) spécifiques.
StuduinoBit Image(width, height, buffer,*,color)	Crée une image en utilisant des espaces d'une largeur (paramètre **width**) et d'une hauteur (paramètre **height**) spécifiques en utilisant un motif pécifique (paramètre **buffer**).

Le paramètre **color** peut être utilisé pour régler la couleur de base dans tous les constructeurs.

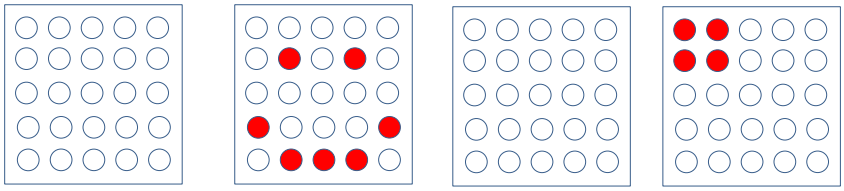
Le paramètre peut être spécifié avec une liste, un tuple ou un nombre entier. Si le paramètre **color** n'a pas été paramétré, la couleur de base sera rouge (31,0,0).

```
from pystubit.image import StuduinoBitImage
img1 = StuduinoBitImage()
img2 = StuduinoBitImage( '00000:'
                         '01010:'
                         '00000:'
                         '10001:'
                         '01110:' )
img3 = StuduinoBitImage(2,2)
img4 = StuduinoBitImage(2,2, bytearray([1,1,1,1]))
```

Les images que vous avez créées peuvent être affichées avec des objets display, comme ci-dessous.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
display.show(img1)
```

Les variables img1, img2, img3 et img4 apparaîtront comme ceci sur le panneau LED.



img1 img2 img3 img4

Les constructeurs mis sur le paramètre **string** peuvent être affichés sur une seule ligne, comme ci-dessous.

```
img2 = StuduinoBitImage( '00000:01010:00000:10001:01110:' )
```

4.2.2. Trouver la taille d'une image

Trouvez la taille d'une image avec les méthodes **width()** et **height()**.

```
from pystubit.image import StuduinoBitImage
img = StuduinoBitImage(2,3)
w = img.width()
h = img.height()
```

La variable **w** se réglera sur **2**, tandis que h se réglera sur **3**.

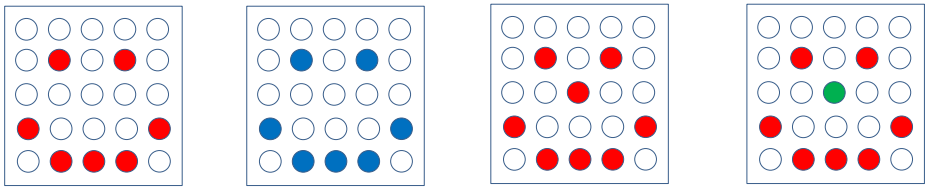
4.2.3. Réglages de l'image

Utilisez la méthode **set_pixel(x, y, value)** pour modifier le motif d'une image. Spécifiez les positions du pixel avec les coordonnées x-y comme les LED dans display, puis réglez le paramètre **value** sur **0** ou **1**. Vous pouvez aussi utiliser la méthode **set_pixel_color(x, y, color)** pour changer la couleur d'un seul pixel. Spécifiez les coordonnées du pixel (paramètres x et y) et réglez la couleur dans le paramètre **color**.

La méthode **set_base_color(color)** peut être utilisée pour régler la couleur de base.

```
from pystubit.image import StuduinoBitImage
original = StuduinoBitImage( '00000:01010:00000:10001:01110:' )
img1.set_base_color((0,0,10))
img2.set_pixel(2,2,1)
img3.set_pixel_color(2,2,(0,10,0))
```

Les variables **original**, **img1**, **img2** et **img3** apparaîtront sur le panneau LED comme ci-dessous. Dans **img1**, la couleur de base s'est changée en bleu (0, 0, 10) avec **set_base_color**. Dans **img2**, le pixel (2, 2) a été réglé sur 1 avec **set_pixel** (le faisant apparaître avec la couleur de base). Dans **img3**, le pixel (2, 2) a été changé en vert (0, 10, 0) avec **set_pixel_color**.



4.2.4. Trouver les données d'une image

Utilisez la méthode **get_pixel(x, y)** pour obtenir des informations sur le motif d'une image. En spécifiant les coordonnées du pixel (paramètre x et y), vous obtenez les données du motif de ce pixel (0 ou 1).
Utilisez la méthode **get_pixel_color(x, y)** pour trouver les données d'un pixel sur une image. Spécifiez les coordonnées du pixel (paramètre x et y) pour obtenir sa couleur en format tuple.

```
from pystubit.image import StuduinoBitImage
original = StuduinoBitImage( '00000:01010:00000:10001:01110:' )
v1 = img.get_pixel(0, 0)
c1 = img.get_pixel_color(0, 0)
v2 = img.get_pixel(1, 1)
c2 = img.get_pixel_color(1, 1)
```

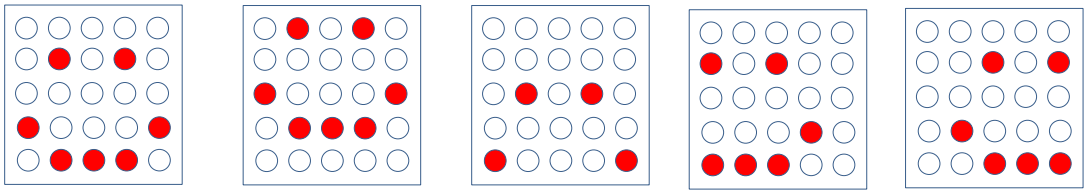
Les variables **v1** et **c1** obtiennent les données du motif (0) et les données de couleur (0, 0, 0) de la LED aux coordonnées (0, 0) de l'image, tandis que les variables **v2** et **c2** obtiennent les données du motif (1) et les données de couleur (31, 0, 0) des coordonnées (1, 1).

4.2.5. Manipuler des images

Les méthodes **shift_up(n)**, **shift_down(n)**, **shift_left(n)** et **shift_right(n)** peuvent être utilisées pour décaler une image vers le haut, le bas, à gauche ou à droite de la quantité spécifiée dans le paramètre **n**.

```
from pystubit.dsply import StduinoBitImage
original = StduinoBitImage( '00000:01010:00000:10001:01110:' )
up = original.shift_up(1)
down = original.shift_down(1)
left = original.shift_left(1)
right = original.shift_right(1)
```

Les variables **original, up, down, left** et **right** apparaîtront sur le panneau LED comme ci-dessous. Toutes les images se déplacent d'une ligne vers le haut, le bas, la gauche ou la droite.

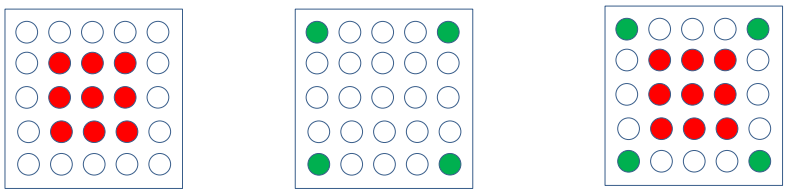


original　up　down　left　right

Un opérateur + vous permet de combiner plusieurs images en une.

```
from pystubit.dsply import StduinoBitImage
img1 = StduinoBitImage( '00000:01110:01110:01110:00000' ,color=(10,0,0))
img2 = StduinoBitImage( '10001:00000:00000:00000:10001' ,color=(0,10,0))
img3 = img 1 + img2
```

Les variables **img1, img2** et **img3** apparaîtront sur le panneau LED comme ci-dessous. L'**img1** et **img2** ont été combinées pour former l'**img3**.



img1　img2　img3

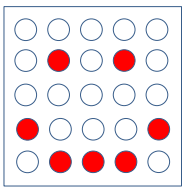
La méthode **copy()** sert à copier une image.

4.2.6. Images incluses

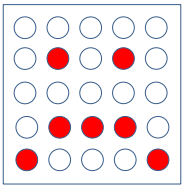
StuduinoBitImage permet d'utiliser une variété de motifs qui sont inclus dans le Studuino:bit (le module ESP32) par défaut.

```
from pystubit.dsply import StuduinoBitImage
img1 = StuduinoBitImage.HAPPY
img2 = StuduinoBitImage.SAD
img3 = StuduinoBitImage.ANGRY
```

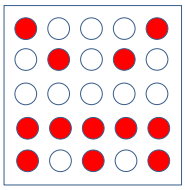
Les variables **img1**, **img2** et **img3** apparaîtront sur le panneau LED comme ci-dessous.



img1



img2



img3

Pour une liste de toutes les images incluses accessible avec StuduinoBitImage, voir la partie **5.4**.

4.3. La classe StuduinoBitBuzzer

La classe StuduinoBitBuzzer est utilisée pour contrôler le buzzer.

4.3.1. Les constructeurs

Utilisez ces lignes de code pour créer un objet qui contrôle un buzzer.

```
from pystubit.bzr import StuduinoBitBuzzer
buzzer = StuduinoBitBuzzer()
```

4.3.2. Ajuster le son

La méthode **on(sound, *, duration=None)** lance le buzzer. Le paramètre sound règle la fréquence du son avec un nombre de note MIDI ou le nom d'une note. Pour une liste des numéros et des noms de notes utilisables , voir la partie **5.3**. Le paramètre **duration** règle la longueur du son de sortie en millisecondes. Si le paramètre **duration** reste non spécifié, la méthode **off()** sert à arrêter le buzzer.

```
from pystubit.bzr import StuduinoBitBuzzer
buzzer = StuduinoBitBuzzer()
buzzer.on('C4', duration=1000)
```

Ces lignes feront jouer au buzzer la note C4 pendant 1 seconde.

4.3.3. Libérer des PWM

Le Studuino:bit utilise des PWM (*pulse-width modulation*, c'est-à-dire modulation de largeur d'impulsion) pour ses sons de sortie. Il peut utiliser jusqu'à 4 PMW simultanément. Si aucun PWM n'est disponible, vous ne pourrez utiliser aucun des objets buzzer que vous avez créé.

Vous pouvez résoudre ce problème avec la méthode **release()** pour libérer un PWM assigné à un autre objet buzzer.

4.4. La classe StuduinoBitButton

La classe StuduinoBitButton sert à contrôler les deux boutons présents sur la face avant du Studuino:bit (carte ESP32).

4.4.1. Les constructeurs

Ces lignes servent à créer un objet qui contrôle les boutons. Réglez **A** ou **B** dans le paramètre.

```
from pystubit.button import StuduinoBitButton
button_a = StuduinoBitButton( 'A' )
```

4.4.2. Trouver l'état des boutons

Utilisez la méthode **get_value()** pour trouver la valeur (0/1) d'un bouton (un nombre entier). La méthode **is_pressed()** retournera **True** ou **False** pour indiquer si un bouton est pressé. La méthode **was_pressed()** retournera **True** ou **False** pour indiquer si un bouton a été pressé par le passé. La méthode **get_presses()** trouve le nombre de fois qu'un bouton a été pressé.

```
from pystubit.button import StuduinoBitButton
button_a = StuduinoBitButton( 'A' )
button_b = StuduinoBitButton( 'B' )
print( 'Press A button' )
while not button_a.is_pressed:
    pass
print( 'A button is pressed' )
```

Presser le bouton A fera s'afficher les mots "A button is pressed" sur le terminal.

4.5. La classe StuduinoBitLightSensor

La classe StuduinoBitLightSensor sert à contrôler le capteur de lumière de la carte ESP32.

4.5.1. Les constructeurs

Ces lignes de code servent à créer un objet qui contrôle le capteur de lumière.

```
from pystubit.sensor import StuduinoBitLightSensor
light_sensor = StuduinoBitLightSensor()
```

4.5.2. Trouver la luminosité

Utilisez la méthode **get_value()** pour trouver la valeur analogique du capteur de lumière. Elle sera formatée en nombre entier entre 0 et 4095.

```
from pystubit.sensor import StduinoBitLightSensor
import time
light_sensor = StduinoBitLightSensor()
while True:
    print(light_sensor.get_value())
    time.sleep_ms(500)
```

Ces lignes de code afficheront la valeur du capteur de lumière sur le terminal toutes les 500 millisecondes.

4.6. La classe StduinoBitTemperature

La classe StduinoBitTemperature sert à contrôler le capteur de température de la carte.

4.6.1. Les constructeurs

Ces lignes servent à créer un objet qui contrôle le capteur de température.

```
from pystubit.sensor import StduinoBitTemperature
temperature = StduinoBitTemperature ()
```

4.6.2. Trouver la température

Utilisez la méthode **get_value()** pour trouver la valeur analogique du capteur de température. Elle sera formatée sous la forme d'un nombre entier entre 0 et 4095. Utilisez la méthode **get_celsius(ndigits=2)** pour obtenir du capteur la température en degrés celsius. Le paramètre **ndigits** permet de régler le nombre de décimales.

```
from pystubit.sensor import StduinoBitTemperature
import time
temperature = StduinoBitTemperature ()
while True:
    print(temperature.get_value())
    print(temperature.getcelsiusvalue())
    time.sleep_ms(500)
```

Ces lignes afficheront la valeur analogique et la température en celsius du capteur de température sur le terminal toutes les 500 millisecondes.

4.7. La classe StduinoBitAccelerometer

La classe StduinoBitAccelerometer sert à contrôler l'accéléromètre de la carte ESP32.

4.7.1. Les constructeurs

Ces lignes servent à créer un objet qui contrôle l'accéléromètre.

```
from pystubit.sensor import StuduinoBitAccelerometer
acc = StuduinoBitAccelerometer(fs='2g', sf='ms2')
```

Utilisez les paramètres **fs** (full scale, c'est-à-dire la fin maximale de l'échelle de mesure) et **sf** dans les constructeurs de StuduinoBitAccelerometer pour régler vos unités. Réglez **2g, 4g, 8g** ou **16g** dans le paramètre **fs** pour mesurer jusqu'à 2G, 4G, 8G ou 16G d'accélération. Il sera réglé à 2G par défaut. Mettez **ms2** ou **mg** dans le paramètre **sf** pour mesurer l'accélération soit en m/sec^2 ou en milli-Gs. Il sera réglé sur m/sec^2 par défaut.

4.7.2. Réglages de l'accéléromètre

Utilisez la méthode **set_fs(value)** pour régler la mesure maximale de l'accéléromètre. Vous pouvez régler le paramètre **value** sur **2g, 4g, 8g** ou **16g**. Utilisez la méthode **set_sf(value)** pour régler les unités de mesure. Elle peut être réglée sur **ms2** ou **mg**.

4.7.3. Trouver l'accélération

Utilisez la méthode **get_values(ndigits=2)** pour trouver les valeurs de votre accéléromètre. Les valeurs sont retournées en format tuple (x, y, z). Les méthodes **get_x(ndigits=2)**, **get_y(ndigits=2)** et **get_z(ndigits=2)** servent à trouver les accélérations des axes x, y et z séparément. Quelque soit la méthode, le paramètre **ndigits** sert à régler le nombre de décimales.

```
from pystubit.sensor import StuduinoBitAccelerometer
import time
acc = StuduinoBitAccelerometer()
while True:
    print(acc.get_values())
    time.sleep_ms(500)
```

Ces lignes servent à afficher les valeurs de l'accéléromètre sur le terminal toutes les 500 millisecondes.

4.8. La classe StuduinoBitGyro

La classe StuduinoBitGyro sert à contrôler le gyroscope de la carte.

4.8.1. Les constructeurs

Ces lignes servent à créer un objet qui contrôle le gyroscope.

```
from pystubit.sensor import StuduinoBitGyro
gyro = StuduinoBitGyro (fs='250dps', sf='dps'))
```

Utilisez les paramètres **fs** (la fin maximale de l'échelle de mesure) et **sf** dans les constructeurs de StuduinoBitGyro pour régler vos unités. Réglez **250dps, 500dps, 1000dps** ou **2000dps** dans le paramètre **fs** pour mesurer jusqu'à 250 deg/s, 500 deg/s, 1000 deg/s ou 2000 deg/s de vitesse angulaire. Il sera réglé sur 250 deg/s par défaut. Mettez **dps** ou **rps** dans le paramètre **sf**

pour mesurer en deg/s ou en rad/s. Il sera réglé par défaut sur deg/s.

4.8.2. Réglages du gyroscope

Utilisez la méthode **set_fs(value)** pour régler la mesure maximale de votre gyroscope. Réglez le paramètre **value** sur **250dps, 500dps, 1000dps** ou **2000dps**. Utilisez la méthode **set_sf(value)** pour régler les unités de mesure. Il peut être réglé sur **dps** ou **rps**.

4.8.3. Trouver la vitesse angulaire

Utilisez la méthode **get_values(ndigits=2)** pour trouver la valeur du gyroscope. Les valeurs seront retournées dans les unités spécifiées par le constructeur ou par la méthode **set_sf** dans un format tuple (x, y, z).

Les méthodes **get_x(ndigits=2)**, **get_y(ndigit=2)** et **get_z(ndigits=2)** servent à trouver la vitesse angulaire des axes x, y et z séparément. Quelque soit la méthode, le paramètre **ndigits** sert à régler le nombre de décimales.

```
from pystubit.sensor import StuduinoBitGyro
import time
gyro = StuduinoBitGyro ()
while True:
    print(gyro.get_values())
    time.sleep_ms(500)
```

Ces lignes servent à afficher les valeurs du gyroscope sur le terminal toutes les 500 millisecondes.

4.9. La classe StuduinoBitCompass

La classe StuduinoBitCompass sert à contrôler le capteur de géomagnétisme (ou boussole) de la carte.

4.9.1. Les constructeurs

Ces lignes servent à créer un objet pour contrôler la boussole.

```
from pystubit.sensor import StuduinoBitCompass
compass = StuduinoBitCompass ()
```

4.9.2. Calibration

Calibrez la boussole avec la méthode **calibrate()**. Le processus de calibration utilisera le panneau LED. Utilisez la méthode **is_calibrated()** pour déterminer si la boussole a été calibrée. Utilisez la méthode **clear_calibration()** pour effacer toutes les données de calibration existantes.

4.9.3. Trouver les directions

Utilisez la méthode **heading()** pour trouver la direction (en degrés, entre 0 et 360). Si votre boussole n'a pas été calibrée, la calibration commencera automatiquement quand vous lancez

la méthode **heading()**. Elle est définie sur 0° lorsque l'unité centrale est placée avec l'affichage LED vers le haut et le connecteur USB orienté plein sud. Les angles augmentent à partir de là dans le sens des aiguilles d'une montre.

4.9.4. Trouver les valeurs de la boussole

Utilisez la méthode **get_values()** pour trouver les valeurs de la boussole. Les valeurs seront retournées sous format tuple (x,y,z) en utilisant des unités µT (micro tesla).
Les méthodes **get_x(), get_y()** et **get_z()** servent à trouver la direction vers laquelle sont tournés les axes x, y et z séparément.

```
1 from pystubit.sensor import StuduinoBitCompass
2 import time
3 compass= StuduinoBitCompass ()
4 while True:
5     print(compass.get_values())
6     time.sleep_ms(500)
```

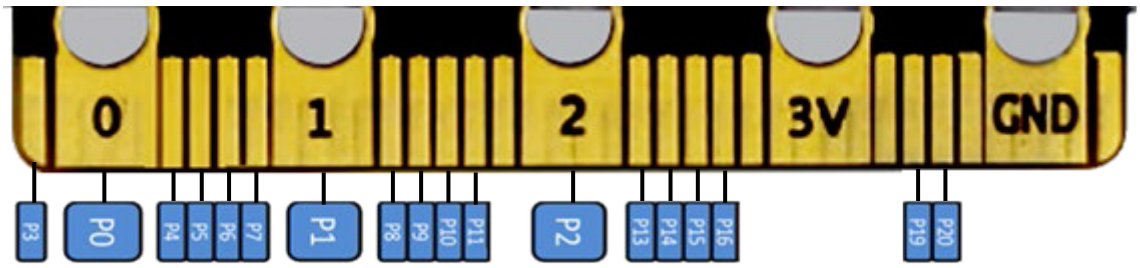
Ces lignes de code servent à afficher les valeurs de la boussole sur le terminal toutes les 500 millisecondes.

4.10. La classe StuduinoBitTerminal

La classe StuduinoBitTerminal sert à contrôler les ports de la carte.

4.10.1. Les constructeurs

Les constructeurs du StuduinoBitTerminal servent à créer des objets qui contrôlent tous les ports de la carte de P0 à P20, exceptés P17 et P18. Les ports disponibles pour cette classe StuduinoBitTerminal sont montrés ci-dessous, de P0 à P20.



Écrivez ces lignes pour créer un objet qui contrôle un port spécifique entre P0 et P20.

```
from pystubit.sensor import StuduinoBitTerminal
p0 = StuduinoBitTerminal( 'P0' )
```

Les entrées et sorties possibles des ports entre P0 et P20 sont listées ci-dessous.

Port	I/O Type	Port	I/O Type
P0	Entrée/sortie numérique et analogique	P10	Entrée/sortie numérique
P1	Entrée/sortie numérique et analogique	P11	Entrée/sortie numérique

P2	Entrée numérique et analogique	P12	Entrée/sortie numérique
P3	Entrée numérique et analogique	P13	Entrée/sortie numérique
P4	Entrée/sortie numérique	P14	Entrée/sortie numérique
P5	Entrée/sortie numérique et analogique	P15	Entrée/sortie numérique
P6	Entrée/sortie numérique	P16	Entrée/sortie numérique et analogique
P7	Entrée/sortie numérique	P19	Entrée/sortie numérique
P8	Entrée/sortie numérique et analogique	P20	Entrée/sortie numérique
P9	Entrée/sortie numérique et analogique		

4.10.2. Entrée/sortie numérique

Si l'objet port que vous avez créé est de type entrée/sortie numérique, utilisez la méthode **write_digital(value)** pour envoyer des signaux numériques avec. Le paramètre **value** peut être réglé sur 0 ou 1. Utilisez la méthode **read_digital()** pour lire les signaux numériques. Utilisez la méthode **set_analog_hz()** pour choisir une fréquence PWM de sortie à utiliser et utilisez la méthode **write_analog(value)** pour envoyer des signaux PWM.

4.10.3. Entrée/sortie numérique et analogique

Si l'objet port que vous avez créé est de type entrée/sortie numérique et analogique, utilisez les méthodes d'entrée/sortie numérique décrites au-dessus et la méthode **read_analog()** pour lire les signaux analogiques.

4.10.4. Entrée numérique et analogique

Si l'objet port que vous avez créé est de type entrée numérique et analogique, utilisez les méthodes **read_digital()** et **read_analog()**.

5. Annexes

5.1. Le module board

Le module board contient les définitions des objets utilisés pour faire fonctionner le matériel Studuino:bit. Créez une déclaration comme ci-dessous pour utiliser un objet.

```
from pystubit.board import *
```

Les objets définis par le module board sont listés ci-dessous.

Objet	Matériel

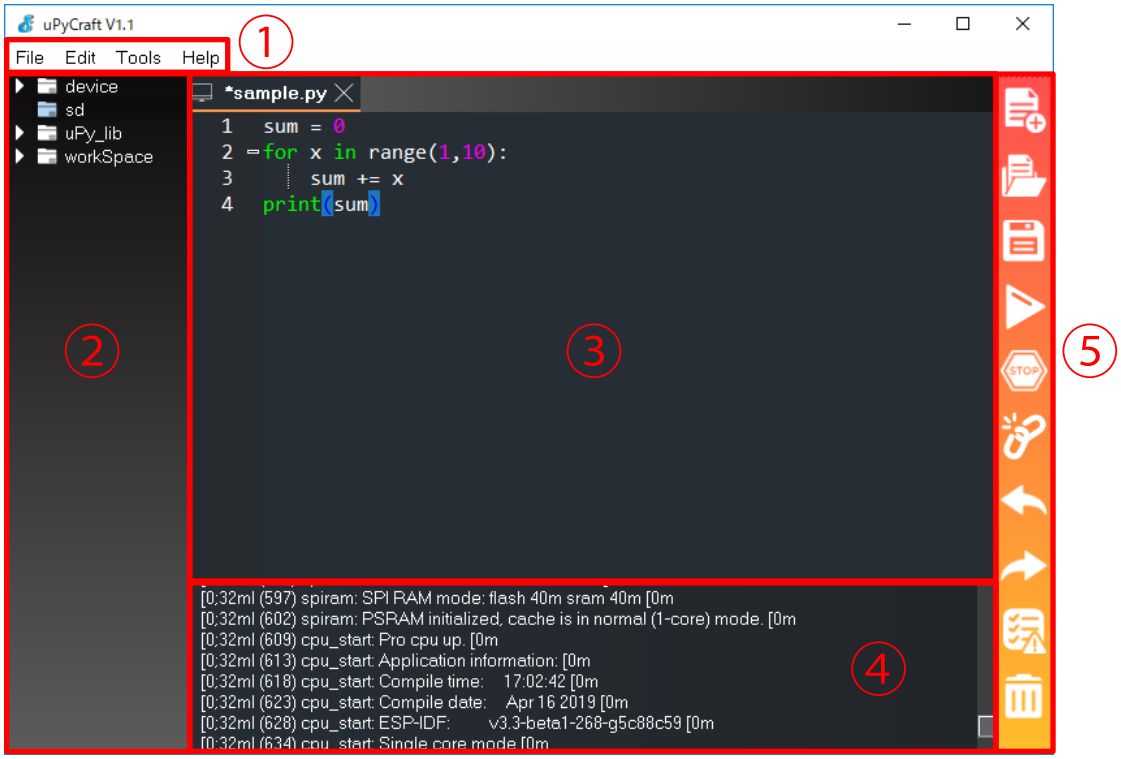
display	Panneau LED
Image (Class)	Panneau LED (image)
buzzer	Buzzer
button_a, button_b	Boutons A/B
lightsensor	Capteur de lumière
temperature	Capteur de température
accelerometer	Accéléromètre
gyro	Gyroscope
compass	Boussole
P0 -16, P19, P20	Connecteur

5.2. uPyCraft

uPyCraft est un IDE MicroPython (un environnement de développement intégré) compatible avec le microcontrôleur ESP32 de Studuino:bit. Il fonctionne sous Windows, Linux et Mac. Téléchargez le logiciel à cette adresse : http://docs.dfrobot.com/upycraft/

5.2.1. Interface

L'interface du logiciel uPycraft est présentée ci-dessous.

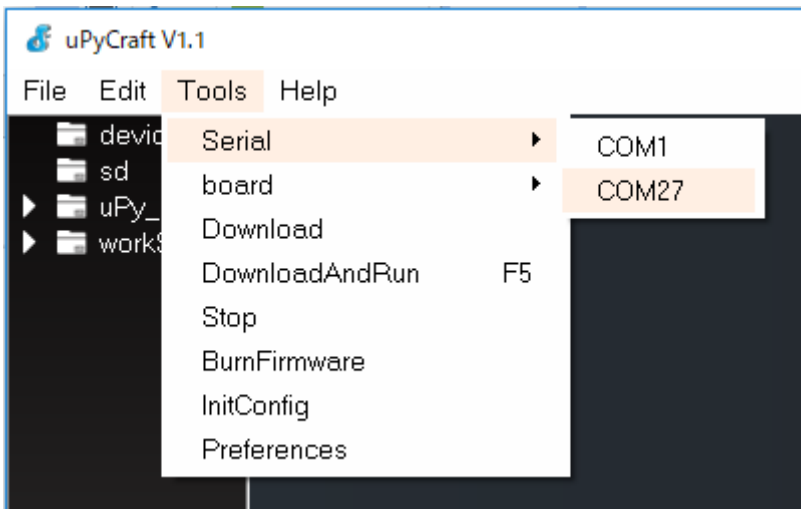


Les cinq parties de l'interface sont la barre de menu (1), l'explorateur de fichiers (2), l'éditeur de texte (3), le terminal (4) et le menu des raccourcis (5).

5.2.2. Construire un programme

1. Réglages des ports série

Lancer uPycraft. Connectez votre carte Studuino:bit à votre PC, choisissez **Serial** du menu **Tools**, puis sélectionnez le port série auquel votre Studuino:bit est connecté.



2. Démarrer les communications série

Cliquez sur cette icône du menu des raccourcis pour ouvrir les communications série avec Studuino:bit.

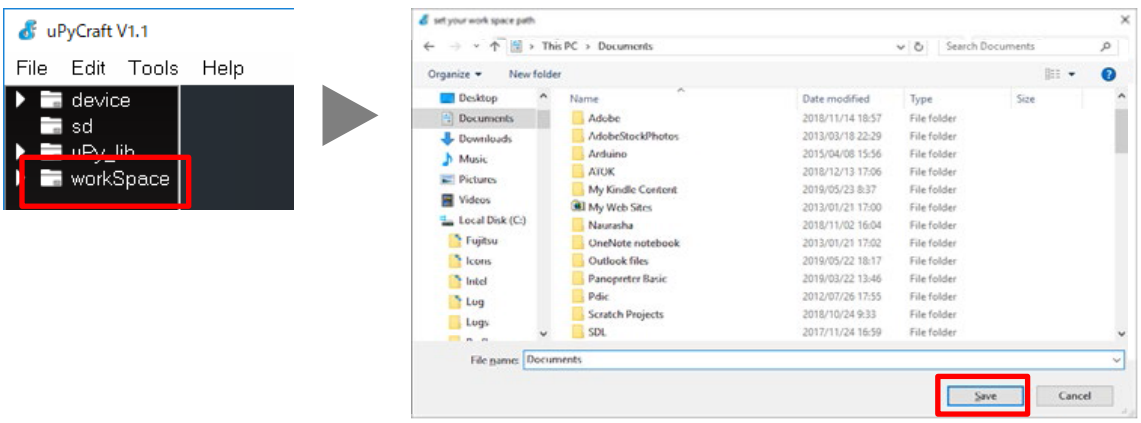
Cliquez sur cette icône quand les communications sont éteintes pour les démarrer.

Cliquez sur cette icône quand les communications sont ouvertes pour les couper.

Si la connexion est établie avec succès, les caractères >>> apparaîtront sur le terminal vous autorisant à utiliser le REPL.

3. Réglages de l'espace de travail

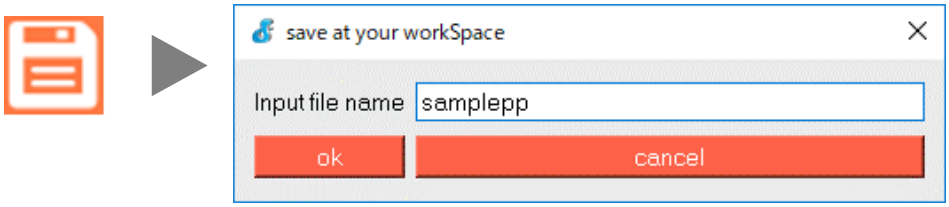
Lors du 1er démarrage, vous devrez créer un espace de travail pour sauvegarder vos programmes. Cliquez sur **workSpace** dans l'explorateur de fichiers, puis choisissez n'importe quel dossier. Après avoir sélectionné votre dossier, créer un nouveau dossier à l'intérieur de celui-ci nommé **workSpace**.



4. Créer et sauvegarder des fichiers de programme

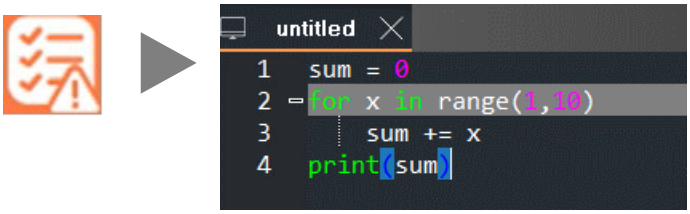
Cliquez sur cette icône dans le menu des raccourcis pour créer un nouveau fichier.

Cliquez sur l'icône suivante dans le menu des raccourcis pour sauvegarder votre programme. Lors du 1er enregistrement d'un nouveau programme, une boîte de dialogue apparaîtra pour que vous nommiez votre programme. Le fichier de votre programme une fois nommé se sauvegardera dans le dossier workSpace créé à l'étape 3.



5. Vérifier les erreurs de syntaxe

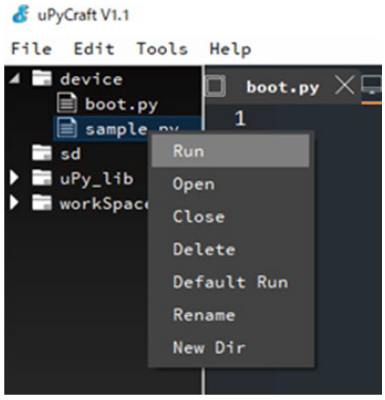
Cliquez sur cette icône dans le menu des raccourcis pour vérifier toutes les erreus de syntaxe de votre programme. Si des erreurs sont trouvées,, la ligne de code concernée sera mise en évidence comme montré ci-dessous.



6. Transférer et exécuter des programmes

Cliquez sur cette icône dans le menu des raccourcis pour transférer votre programme sur le Studuino:bit.

Quand le transfert est terminé, votre programme sera listé dans le dossier device de l'explorateu de fichiers. Le programme s'exécutera automatiquement une fois transféré. S'il ne s'exécute pas, ouvrez le fichier du programme dans le dossier device, faites un clic droit, puis cliquez sur **Run**.



*Si le transfert du programme échoue, peut-être est-ce parce que le Studuino:bit est en état “occupé”. Appuyer sur le bouton Reset du Studuino:bit résoudra le problème dans la plupart des cas.

7. Arrêter le programme
Cliquez sur cette icône dans le menu des raccourcis pour arrêter un programme en cours d'exécution.

5.3. Son de sortie

Voici les nombres des notes MIDI et les noms des notes utilisés dans la classe StuduinoBitBuzzer :

MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note
48	C3	60	C4	72	C5	84	C6	96	C7	108	C8	120	C9
49	CS3	61	CS4	73	CS5	85	CS6	97	CS7	109	CS8	121	CS9
50	D3	62	D4	74	D5	86	D6	98	D7	110	D8	122	D9
51	DS3	63	DS4	75	DS5	87	DS6	99	DS7	111	DS8	123	DS9
52	E3	64	E4	76	E5	88	E6	100	E7	112	E8	124	E9
53	F3	65	F4	77	F5	89	F6	101	F7	113	F8	125	F9
54	FS3	66	FS4	78	FS5	90	FS6	102	FS7	114	FS8	126	FS9
55	G3	67	G4	79	G5	91	G6	103	G7	115	G8	127	G9
56	GS3	68	GS4	80	GS5	92	GS6	104	GS7	116	GS8		
57	A3	69	A4	81	A5	93	A6	105	A7	117	A8		
58	AS3	70	AS4	82	AS5	94	AS6	106	AS7	118	AS8		
59	B3	71	B4	83	B5	95	B6	107	B7	119	B8		

5.4. Images incluses

Voici les images incluses dans la bibliothèque Studuino:bit :

StuduinoBitImage.HEART

StuduinoBitImage.HEAR T_SMALL

StuduinoBitImage.HAPPY

StuduinoBitImage.SMILE

StuduinoBitImage.SAD

StuduinoBitImage.CONFUSED

StuduinoBitImage.ANGRY

StuduinoBitImage.ASLEEP

StuduinoBitImage.SURPRISED

StuduinoBitImage.SILLY

StuduinoBitImage.FABULOUS

StuduinoBitImage.MEH

StuduinoBit Image.YES

StuduinoBitImage.NO

StuduinoBitImage.CLOCK12,

StuduinoBitImage.CLOCK11,

StuduinoBitImage.CLOCK10,

StuduinoBitImage.CLOCK9,
StuduinoBitImage.CLOCK8,
StuduinoBitImage.CLOCK7,
StuduinoBitImage.CLOCK6,
StuduinoBitImage.CLOCK5,
StuduinoBitImage.CLOCK4 ,
StuduinoBitImage.CLOCK3,
StuduinoBitImage.CLOCK2,
StuduinoBitImage.CLOCK1
StuduinoBitImage.ARROW_N,
StuduinoBitImage.ARROW_NE,
StuduinoBitImage.ARROW_E,
StuduinoBitImage.ARROW_SE,
StuduinoBitImage.ARROW_S,
StuduinoBitImage.ARROW_SW,
StuduinoBitImage.ARRO W_W,
StuduinoBitImage.ARROW_NW
StuduinoBitImage.TRIANGLE
StuduinoBitImage.TRIANGLE_LEFT
StuduinoBitImage.CHESSBOARD
StuduinoBitImage.DIAMOND
StuduinoBitImage.DIAMOND_SMALL
StuduinoBitImage.SQUARE
StuduinoBitImage.SQUARE_SMALL
StuduinoBitImage.RABBIT
Studui noBitImage.COW
StuduinoBitImage.MUSIC_CROTCHET
StuduinoBitImage.MUSIC_QUAVER
StuduinoBitImage.MUSIC_QUAVERS
StuduinoBitImage.PITCHFORK
StuduinoBitImage.XMAS
StuduinoBitImage.PACMAN
StuduinoBitImage.TARGET
StuduinoBitImage.TSHIRT
StuduinoBitImage.ROLLERSKAT E

StuduinoBitImage.DUCK

StuduinoBitImage.HOUSE

StuduinoBitImage.TORTOISE

StuduinoBitImage.BUTTERFLY

StuduinoBitImage.STICKFIGURE

StuduinoBitImage.GHOST

StuduinoBitImage.SWORD

StuduinoBitImage.GIRAFFE

StuduinoBitImage.SKULL

StuduinoBitImage.UMBRELLA

Studuin oBitImage.SNAKE

5.5. Tableau des classes

Voici la bibliothèque complète des classes de Studuino:bit.

Fonction	Classe	Type	Instructions
Bouton	StuduinoBitButton	_init_(ab)	Utilisé pour créer des objets qui font fonctionner les boutons A/B. Le paramètre **ab p**eut être réglé sur A ou B.
		get_value()	Retourne 0 si le bouton est pressé et 1 s'il ne l'est pas.
		is_pressed()	Retourne True si le bouton est en train d'être pressé.
		was_pressed()	Les objets Button stockent l'information que le bouton a été pressé. Cette méthode retourne True si le bouton a été pressé. Le compteur est remis à 0 quand la méthode est appelée.
		get_presses()	Les objets Button stockent l'information que le bouton a été pressé. Cette méthode retourne le nombre de fois qu'un bouton a été pressé. Le compteur est remis à 0 quand la méthode est appelée.
Panneau LED (couleurs)	StuduinoBitDisplay	_init_()	Crée un objet display.
		get_pixel(x, y)	Retrouve la couleur d'une LED à la ligne x et à la colonne y. La couleur est indiquée en (R,G,B).
		set_pixel(x, y, color)	Règle la couleur de la LED à la ligne x de la colonne y. Le paramètre peut être réglé en (R,G,B), [R,G,B] ou #RGB.
		clear()	Règle la luminosité de toutes les LED du panneau à 0 (off).
		show(iterable, delay=400, *, wait=True, loop=False, clear=False, color=None)	Affiche le paramètre **iterable** (une image, lettres ou nombre) dans l'ordre.
		scroll(string, delay=150, *, wait=True, loop=False, color=None)	Fais défiler le paramètre **value** (lettres/nombres) dans l'ordre sur le panneau horizontalement.
		on()	Allume l'alimentation du panneau LED.
		off()	Eteint l'alimentation du panneau LED. (Cela vous permet d'utiliser les terminaux GPIO connectés au panneau pour d'autres choses)
		is_on()	Retourne True si l'alimentation du panneau est sur ON et False si elle est sur OFF.

Image	StuduinoBitImage	_init_(string, color=None) _init_(width=None, height=None, buffer=None, , color=None)	Crée des objets image dont le motif est écrit en 0 (LED OFF) et 1 (LED ON) dans le paramètre string. StuduinoBitImage('01100:10010:11110:10010:10010:', color=(0,0,10)) Crée un objet image en utilisant des espaces avec une largeur (paramètre **width**) et une hauteur (paramètre **height**) spécifiées. StuduinoBitImage(2, 2,bytearray([0,1,0,1]) StuduinoBitImage(3, 3) Comme pour micro:bit, tous les nombres entre 0 et 9 peuvent être utilisés. 1-9 se traduisent tous en ON et 0 en OFF. Si la variable **color** n'a pas été réglée, (RBG)=(31,0,0).
		width()	Retourne la largeur de l'image en colonnes.
		height()	Retourne la hauteur de l'image en lignes.
		set_pixel(x, y, value)	Le paramètre **value** règle la valeur du pixel aux coordonnées (x,y) dans l'image. Le paramètre value est réglé soit sur 0 (OFF), soit sur 1 (ON).
		set_pixel_color(x, y, color)	Le paramètre **color** règle la couleur du pixel aux coordonnées (x,y) dans l'image. Le paramètre color peut être réglé avec (R, G, B), [R, G, B] ou #RGB.
		get_pixel(x, y)	Retrouve la valeur du pixel aux coordonnées (x, y) de l'image.
		get_pixel_color(x, y, hex=False)	Retrouve la couleur du pixel aux coordonnées (x, y) de l'image. Si le paramètre **hex** est réglé sur False, la couleur sera écrite en (R, G, B). S'il est réglé sur True, elle sera écrite en #RGB.
		set_base_color(self, color)	Le paramètre **color** règle la couleur de tous les pixels de l'image. Le paramètre color peut être réglé avec (R, G, B), [R, G, B] ou #RGB.
		shift_left(n)	Crée une nouvelle image en décalant l'image en cours à gauche du nombre d'espaces réglé dans le paramètre **n**.
		shift_right(n)	Même chose que shift_left(-n).
		shift_up(n)	Crée une nouvelle image en décalant l'image en cours vers le haut du nombre d'espaces réglé dans le paramètre **n**.
		shift_down(n)	Même chose que shift_up(-n).
		copy()	Retourne une copie complète de l'image.
		repr(image)	Obtient une chaîne compacte représentant l'image.
		str(image)	Obtient une chaîne lisible représentant l'image.
		+	Combine tous les pixels de deux images pour en faire une nouvelle.
			Images incluses (voir **5.4**)

Entrée/sortie numérique (★)	StuduinoBitDigitalPin	write_digital(value)	Règle les signaux numériques en High si le paramètre **value** est 1, en Low s'il est à 0.
		read_digital()	Lit les signaux numériques. Retourne High si la valeur reçue est 0 et Low si c'est 1.
		write_analog(value)	Envoie les signaux PWM via le port. Le paramètre **value** peut être réglée entre 0 (0%) et 1023 (100%).
		set_analog_period(period, timer=−1)	Règle la période de signal PWM en millisecondes.
		set_analog_period_microseconds (period, timer=−1)	Règle la période de signal PWM en microsecondes.
		set_analog_hz(hz, timer=−1)	Règle la période de signal PWM en fréquence.
		status()	Affiche l'état d'utilisation actuelle des PWM.
Entrée analogique (★)	StuduinoBitAnalogDigitalPin	read_analog(mv=False)	Lit le voltage du port et, si le paramètre **mv** est réglé sur False, le retourne en nombre entier entre 0 (0V) et 4096 (3.3V). Si mv=True, le voltage sera écrit en mV.
Entrée/sortie numérique et analogique (★)	StuduinoBitAnalogDigitalPin	write_digital(value)	Règle les signaux numériques sur High si le paramètre **value** est de 1 et sur Low s'il est de 0.
		read_digital()	Lit les signaux numériques. Retourne High si la valeur reçue est 0 et Low si c'est 1.
		write_analog(value)	Envoie des signaux PWM via le port. Le paramètre **value** peut être réglé de 0 (0%) à 1023 (100%).
		set_analog_period(period, timer=−1)	Règle la période du signal PWM en millisecondes.
		set_analog_period_microseconds (period, timer=−1)	Règle la période du signal PWM en microsecondes.
		set_analog_hz(hz, timer=−1)	Règle la période du signal PWM en fréquence.
		status()	Affiche l'état d'utilisation actuelle des PWM.
		read_analog(mv=False)	Lit le voltage du port et, si le paramètre **mv** est réglé sur False, le retourne en nombre entier entre 0 (0V) et 4096 (3.3V). Si mv=True, le voltage sera écrit en mV.
Port	StuduinoBitTerminal	__init__(pin)	Crée un objet terminal qui s'applique à un port désigné du Studuino:bit entre P0 et P20 (exceptés P17 et P18).
Buzzer	StuduinoBitBuzzer	__init__()	Crée un objet qui contrôle le Buzzer.
		on(sound, *, duration=−1)	Règle le son du buzzer dans les paramètres **sound** et **duration**. **sound** peut être réglée avec des noms de note (C3-G9), des nombres de note MIDI (48-127) ou une fréquence (en nombre entier). **duration** peut être réglée en ms. Si le paramètre **duration** reste non spécifié, le buzzer continuera de jouer jusqu'à ce que la méthode **off** soit appelée. bzr.on('50',duration=1000) # Le buzzer jouera la note correspondant au nombre de la note MIDI 50 pendant 1 seconde.

			bzr.on('C4', duration=1000) # Le buzzer jouera la note C4 pendant 1 seconde.
			bzr.on(440)

		off()	Arrête le buzzer.
Capteur de température	StuduinoBitTemperature	_init_()	Crée un objet qui contrôle le capteur de température.
		get_value()	Retrouve la valeur (0-4095) du capteur de température de la carte.
		get_celsius()	Retrouve la valeur du capteur de température de la carte en degrés Celsius.
Capteur de lumière	StuduinoBitLightSensor	_init_()	Crée un objet qui contrôle le capteur de lumière.
		get_value()	Retrouve la valeur (0-4095) du capteur de lumière de la carte.
Accéléromètre	StuduinoBitAccelerometer	_init_(fs='2G', sf='mg2')	Crée un objet Accelerometer en réglant le paramètre **fs** à 2g, 4g, 8g ou 16g pour régler l'accélération maximale à mesurer et le paramètre **sf** en mg ou ms2 pour choisir vos unités de mesure. Les réglages par défaut sont fs=2G et sf=ms2.
		get_x	Mesure l'accélération le long de l'axe x.
		get_y	Mesure l'accélération le long de l'axe y.
		get_z	Mesure l'accélération le long de l'axe z.
		get_values()	=(get_x(), get_y(), get_z())
		set_fs(value)	Règle l'accélération maximale mesurable avec 2g, 4g, 8g ou 16g.
		set_sf(value)	Règle les unités de mesure avec mg ou ms2.
Gyroscope	StuduinoBitGyro	_init_(fs='250dps', sf='dps')	Crée un objet Gyroscope en réglant le paramètre **fs** à 250dps, 500dps, 1000dps ou 2000dps pour régler la vitesse angulaire maximale à mesurer et le paramètre **sf** en dpsou rps pour choisir vos unités de mesure. Les réglages par défaut sont fs=250dps et sf=dps.
		get_x	Mesure la vitesse angulaire le long de l'axe x.
		get_y	Mesure la vitesse angulaire le long de l'axe y.
		get_z	Mesure la vitesse angulaire le long de l'axe z.
		get_values()	=(get_x(), get_y(), get_z())
		set_fs(value)	Règle la vitesse angulaire maximale mesurable avec 250dps, 500dps, 1000dps ou 2000dps.
		set_sf(value)	Règle les unités de mesure avec dps ou rps.

Boussole	StuduinoBitCompass	_init_()	Crée un objet qui contrôle la boussole. La valeur maximale mesurable est ±4500 μT et les unités sont μT (micro teslas).
		get_x	Trouve la force magnétique le long de l'axe x.
		get_y	Trouve la force magnétique le long de l'axe y.
		get_z	Trouve la force magnétique le long de l'axe z.
		get_values()	=(get_x(), get_y(), get_z())
		calibrate()	Démarre le processus de calibration de la boussole. Tournez la carte jusqu'à ce que le panneau LED dessine un tour complet de son bord.
		is_calibrated()	Retourne True si la boussole a été calibrée et False si elle ne l'a pas été.
		clear_calibration()	Efface les données de calibration existantes.
		heading()	Trouve la direction à laquelle la boussole fait face en nombres entiers entre 0 et 360° dans le sens des aiguilles d'une montre, 180° indiquant le Nord.
BLE	StuduinoBitBLE		T.B.D
Communication sans fil	StuduinoBitRadio	_init()_	Crée un objet qui contrôle les communications sans fil de la carte.
		on()	Allume la communication sans fil. La communication sans fil doit être explicitement appelée car elle consomme de l'énergie et occupe la mémoire qui peut être nécessaire pour d'autres fonctions.
		off()	Eteint la communication sans fil, en conservant la puissance et la mémoire de la carte.
		Les méthodes suivantes ne peuvent pas être utilisées à moins que la communication sans fil ait été allumée.	
		start(group)	Règle le paramètre **group** sur un nombre entre 0 et 255 pour choisir un groupe et commencer la communication sans fil.
		send_number(n)	Diffuse un nombre. Le nombre sera un nombre entier 32-bit.
		send_value(s, n)	Diffuse un nombre et un nom de variable. Le nombre sera un nombre entier 32-bit et le nom de la variable peut aller jusqu'à 13 caractères.
		send_string(s)	Diffuse une chaîne de caractères. La chaîne peut aller jusqu'à 19 caractères.
		send_buffer(buf)	Diffuse une séquence de bytes. Elle peut aller jusqu'à 19 bytes.
		recv()	Reçoit des données. Si aucune donnée n'est réceptionnée, retourne None.
		group(group=−1)	Règle le paramètre **group** sur un nombre entre 0 et 255 pour choisir un groupe.
Wi−Fi	StuduinoBitWiFi		T.B.D

(★) Les entrées/sorties numériques individuelles, les classes des entrées analogiques et les classes d'entrée/sortie numérique et analogique ne sont pas destinées à être transformées en instances. Pour elles, faites des instances avec la classe Terminal.