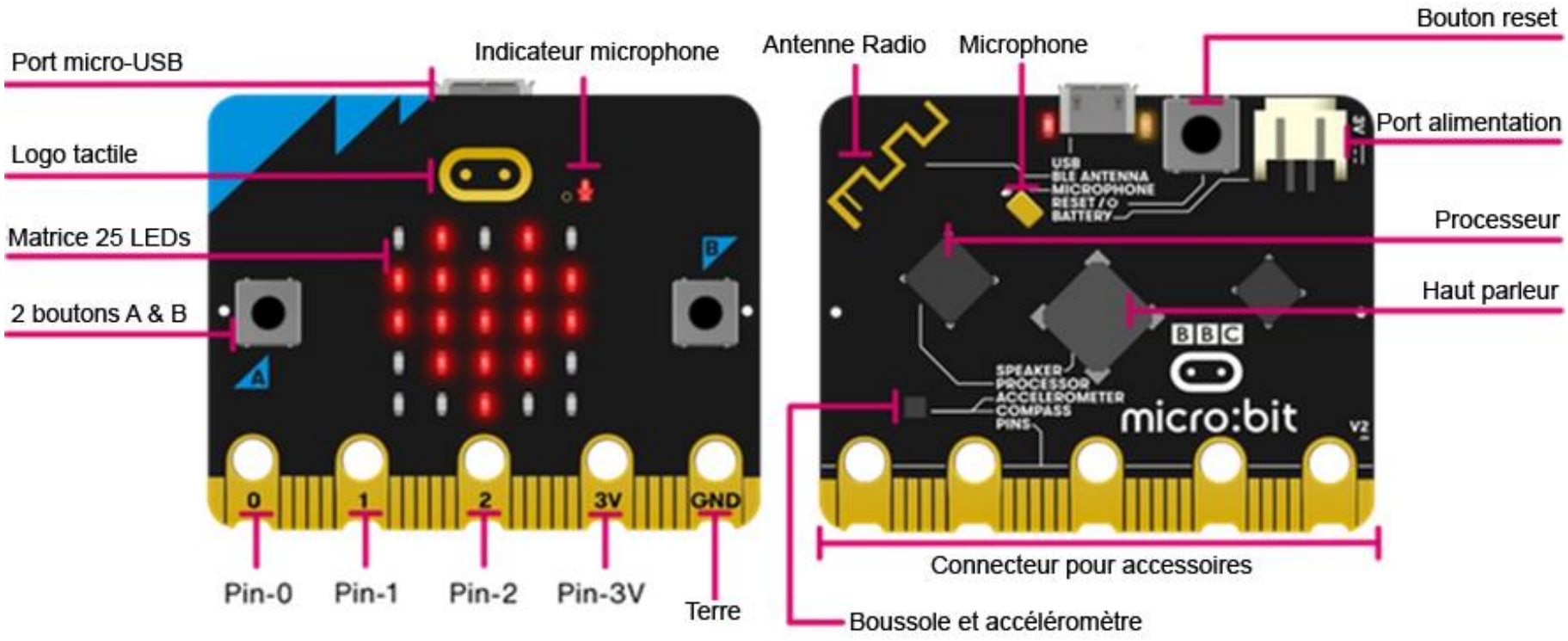
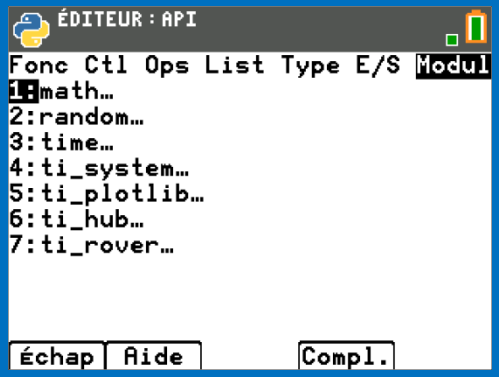
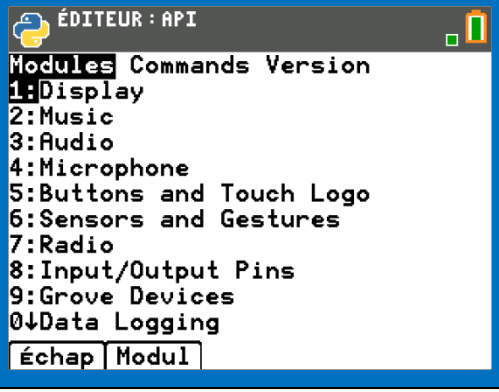
TI-83 Premium CE Edition Python - micro:bit : Guide de Référence

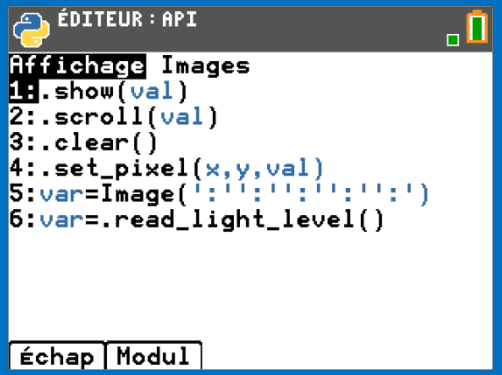
La calculatrice graphique TI-83 Premium CE Edition Python avec le langage de programmation Python 3.x intégré, supporte la liaison aux cartes micro:bit V1 et V2. La fonctionnalité avec la carte micro:bit est facilitée par l'importation des modules microbit dans le programme python. Ces modules sont disponibles sur le site de Texas Instruments micro:bit Support. La syntaxe des modules micro:bit de TI est alignée sur la syntaxe Python de micro:bit publiée sur le site Web de micro:bit. Pour plus d'informations et d'exemples, veuillez consulter les liens des sites Web ci-dessous. Les méthodes Python qui ne sont prises en charge que par les cartes V2 sont surlignés en jaune.

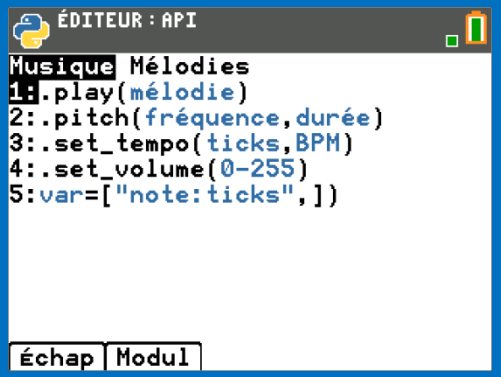
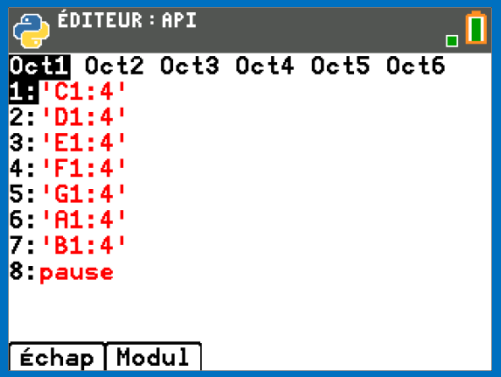


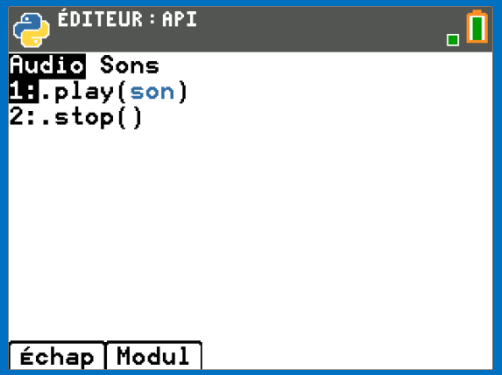
BBC microbit MicroPython API – Il s'agit de la syntaxe Python standard officielle pour le micro:bit.
Micro: bit User Guide – Le site web micro:bit.org avec de nombreux exemples et tutoriels.
Micro:bit Projects Make it: Code it – Idées de projets à utiliser avec la calculatrice. Le code Python est indiqué à l'étape 2 sous l'onglet Python.
Support Micro:bit Texas Instruments – Page contenant les téléchargements du module microbit et du runtime, ainsi qu'un guide de démarrage.
Texas Instruments STEM Projects – De nombreuses activités éducatives STEM intéressantes (site US).

TI-83 Premium CE Edition Python - micro:bit : Guide de Référence

API	Exemple d'utilisation	Notes
Microbit   		
from microbit import *	**from microbit import ***	L'importation est nécessaire pour toutes les fonctionnalités microbit. Une fois que tous les modules micro:bit ont été chargés sur la calculatrice, importez le module microbit à l'aide de l'onglet du menu **Compl.**. Tous les modules complémentaires pris en charge par microbit sont ajoutés à partir du menu Modules microbit.
Modules	**from mb_disp import ***	Les modules micro:bit requis peuvent être sélectionnés dans l'éditeur python à partir du menu microbit. Par exemple, en sélectionnant "Affichage", vous importez le module mb_disp. Une fois le module importé, les menus apparaissent sous l'onglet Fns... dans l'éditeur Python. Les menus n'apparaissent pas dans le Shell de Python. Il est essentiel de n'inclure que les modules nécessaires à un programme. Un trop grand nombre d'importations peut entraîner une erreur pour mémoire insuffisante.
sleep(ms)	**sleep(2000)**	Met l'exécution du programme en pause pendant le nombre de millisecondes indiqué. Notez que l'unité de temps de micro:bit est la ms et qu'elle diffère de celle de la méthode time.sleep() dont l'unité est la seconde.
while not escape()	**While not escape():** ** x = accelerometer.get_x()** ** print(x)**	Donne une boucle infinie d'où on sort en appuyant sur la touche [annul] de la calculatrice. Permet un meilleur contrôle de sortie de boucle qu'avec un while True.

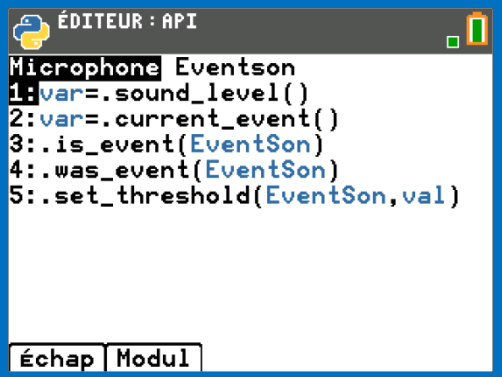
disp_clr()	**disp_clr()**	Efface l'historique du Shell de la calculatrice.
store_list("1" – "6", list)	**my_list = [0,1,2,3] store_list("1",my_list)**	Enregistre une liste Python dans l'une des listes statistiques L1 à L6 de la calculatrice, déterminée par le paramètre "1" - "6".
var = temperature()	**t = temperature()**	Renvoie la température du CPU du micro:bit en degrés Celsius.
Version	**Version 1.x.x or Version 2.x.x**	Affiche la version des modules micro:bit chargés sur la calculatrice.
runtime_version()	**runtime_version()**	Vérifie et renvoie la version de TI_Runtime chargé sur la carte connectée.
Affichage		
Modul->microbit->Affichage	**from mb_disp import ***	Module micro:bit requis pour les méthodes d'affichage.
display.show(val)	**display.show(Image.HAPPY)**	Affiche une image sur la matrice de LEDs. Un délai optionnel peut être saisi pour mettre le programme en pause pendant un temps donné. Par exemple, delay = 500 met le programme en pause pendant 500 ms. display.show(Image.HAPPY, DELAY = 500)
display.scroll(val)	**display.scroll("Hello World!")**	Fait défiler un nombre ou des caractères sur l'écran.
display.clear()	**display.clear()**	Efface l'affichage de la matrice de LEDs.
display.set_pixel(x,y,val)	**display.set_pixel(2,2,9)**	Allume un pixel à l'emplacement x,y (0-4) avec l'intensité souhaitée (0-9). Le pixel central est à (2,2).
var = display.read_light_level()	**I = display.read_light_level()**	intensité = display.read_light_level()
var=Image(':'':'':'':')	**img=Image('99999:' '77777:'**	Construit un objet image nommé img. Chaque pixel des lignes 1 à 5 doit avoir une luminosité comprise entre 0 et

	'55555:' **'33333:'** **'11111:')**	9. Pour saisir les valeurs, placez le curseur devant les deux points et saisissez les cinq valeurs. Repositionnez ensuite le curseur sur la ligne suivante et terminez ainsi pour les cinq lignes. Cette image peut être affichée avec display.show(img).
Image	**"Image.HEART"**	Colle les noms des images intégrées. Pour afficher une image, utilisez la méthode display.show().
Musique & Notes	 	
Importation des modules microbit	**from mb_music import *** **from mb_notes import ***	Active les méthodes de musique et de notes.
music.play(mélodie)	**Music.play('music.ODE')**	Joue une mélodie intégrée ou créée par l'utilisateur. Utilisez var=[note,], puis sélectionnez des notes dans la liste des notes pour composer une mélodie.
music.pitch(fréquence, durée)	**music.pitch(440,2000)**	Joue un son d'une fréquence et d'une durée données.
music.set_tempo(ticks,BPM)	**music.temp(4,220)**	Modifie la vitesse de lecture d'une mélodie. Les ticks sont l'unité de temps de base, par exemple, une noire correspond à 4 ticks. Le BPM est le nombre de battements par minute ; le tempo typique est de 120 BPM.
music.set_volume(0-255)	**music.set_volume (128)**	Règle l'intensité du haut-parleur intégré. Le niveau sonore le plus faible est 0, et le plus fort est 255.
var=[note,]	**Frere_Jaques=['C4:4','D4:4','E4:4',** **'C4:4','C4:4','D4:4','E4:4','C4:4',** **'E4:4','F4:4','G4:8','E4:4','F4:4',** **'G4:8',]**	Une liste des premières notes de Frère Jaques. Cette liste peut être jouée en utilisant music.play(Frere_Jaques).

Mélodies	**'music.ODE'**	Colle les titres des mélodies intégrées. Pour lire une mélodie, utilisez la méthode music.play(). La TI 83 s'écarte de l'API microbit sur le format des noms de mélodies. Le nom d'une mélodie est collé entre guillemets sur la calculatrice ; cela peut provoquer une erreur lors de la copie de programmes provenant d'une autre source qui peut omettre les guillemets simples.
Notes	**'C4:4'**	Colle les notes de l'octave 1 à l'octave 6. Pour coller plus d'une note dans la méthode music.play(), il faut que les notes se trouvent dans une liste. Voir l'aide de var=[note,] ci-dessus.
Audio		
Importation du module microbit	**from mb_audio import ***	Active les méthodes audio et acoustiques.
audio.play(son)	**audio.play("Sound.GIGGLE")**	Joue chacun des sons intégrés. Utilisez le menu Sons pour sélectionner les titres des sons.
audio.stop()	**audio.stop()**	Arrête un son en cours.
Sons	**"Sound.GIGGLE"**	Colle les titres des sons intégrés. Pour lire un son, utilisez la méthode audio.play().

TI-83 Premium CE Edition Python - micro:bit : Guide de Référence

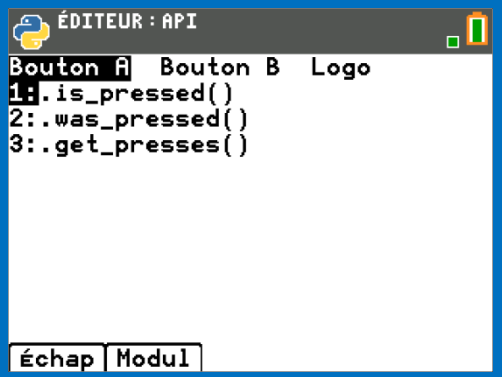
Microphone



Importation du module microbit	**from mb_mic import ***	Active les méthodes relatives au microphone et aux événements sonores.
var=.sound_level()	**i=microphone.sound_level()**	Mesure l'intensité sonore actuelle dans une plage allant de 0 (silencieux) à 255 (fort).
var=.current_event()	**event=microphone.current_event(SoundEvent.LOUD)**	Renvoie l'événement sonore nommé event au moment de l'exécution.
microphone.is_event(EventSon)	**microphone.is_event(SoundEvent.LOUD)**	Renvoie True si le microphone détecte une pression sonore supérieure à SoundEvent.LOUD. La pression sonore de l'événement sonore est définie à l'aide de la méthode microphone.set_threshold(). Cette méthode interroge le microphone lorsqu'elle est exécutée. Pour capter un événement transitoire rapide, utilisez la méthode microphone.was_event().
microphone.was_event(EventSon,val)	**microphone.was_event(SoundEvent.LOUD)**	Renvoie True si le microphone a détecté une pression sonore supérieure à SoundEvent.LOUD. La pression sonore de l'événement sonore peut être définie à l'aide de la méthode microphone.set_threshold(). Cette méthode s'exécute en arrière-plan et l'événement est effacé dès que la méthode est appelée. Pour une interrogation immédiate, utilisez la méthode microphone.is_event().
microphone.set_threshold(EventSon,val)	**microphone.set_threshold(SoundEvent.LOUD, 128)**	Définit le seuil d'un événement sonore de 0 (silencieux) à 255 (fort). Ceci est utile lorsque vous utilisez microphone.current_event() ou microphone.is_event().
EventSon	**SoundEvent.LOUD**	Colle le nom de l'événement dans une méthode.

TI-83 Premium CE Edition Python - micro:bit : Guide de Référence

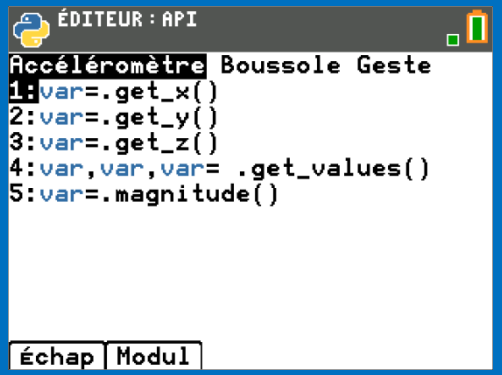
Boutons & Logo



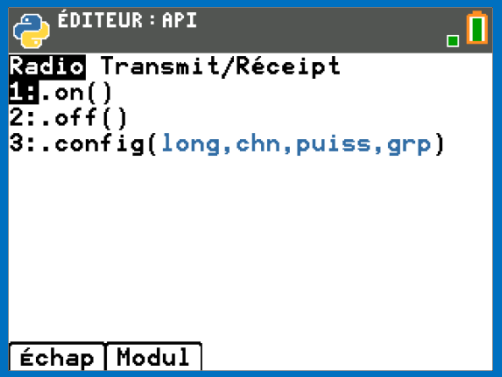
Importation du module microbit	**from mb_butns import ***	Active les boutons et les méthodes tactiles.
button_a.is_pressed()	**while not button_a.is_pressed():** **instructions**	Renvoie True lorsque le bouton A est enfoncé, False sinon. Cette méthode est souvent utilisée comme condition booléenne, lorsque le matériel est utilisé pour modifier le déroulement du programme. La méthode .is_pressed() interroge le bouton et renvoie l'état actuel du bouton.
button_a.was_pressed()	**if button_a.was_pressed():** **instructions**	Renvoie True si le bouton A a été pressé, False sinon. La méthode .was_pressed() s'exécute en arrière-plan et est utile pour détecter les pressions ou les tapotements transitoires sur les boutons. L'état du bouton est immédiatement effacé après l'appel de la méthode.
var=button_a.get_presses()	**presses = button_a.get_presses()**	Renvoie le nombre d'appuis sur le bouton A. L'appel de cette méthode effacera le compteur et remettra la valeur renvoyée à zéro
button_b.is_pressed()	**while not button_b.is_pressed():** **instructions**	Action identique à celle du bouton A pour le bouton B.
button_b.was_pressed()	**if button_b.was_pressed():** **instructions**	Action identique à celle du bouton A pour le bouton B.
var=button_b.get_presses()	**presses = button_b.get_presses()**	Action identique à celle du bouton A pour le bouton B.
pin_logo.is_touched()	**while pin_logo.is_touched():** **instructions**	Renvoie True lorsque le logo tactile est touché, False sinon. Cette méthode est souvent utilisée comme condition booléenne pour modifier le déroulement du programme. La méthode .is_touched() interroge immédiatement le logo et renvoie l'état actuel du capteur.

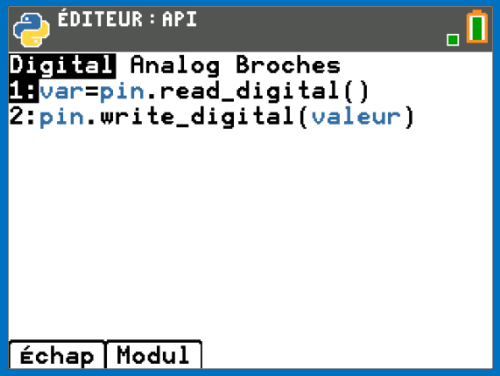
TI-83 Premium CE Edition Python - micro:bit : Guide de Référence

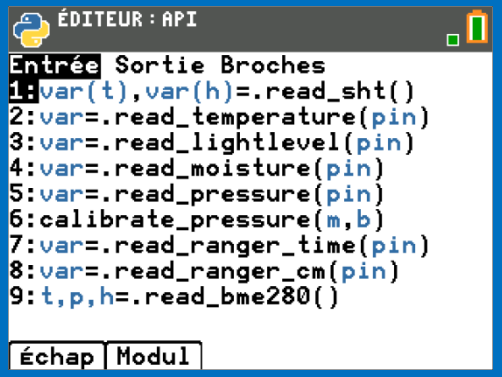
Capteurs & Gestes



Importation du module microbit	**from mb_sensr import ***	Active les méthodes des capteurs internes.
var = accelerometer.get_x()	**x=accelerometer.get_x()**	Renvoie la mesure de l'accéléromètre sur l'axe des x avec une plage de +/- 2000 mg.
var = accelerometer.get_y()	**y=accelerometer.get_y()**	Renvoie la mesure de l'accéléromètre sur l'axe des y avec une plage de +/- 2000 mg.
var = accelerometer.get_z()	**z=accelerometer.get_z()**	Renvoie la mesure de l'accéléromètre sur l'axe des z avec une plage de +/- 2000 mg.
var_x,var_y,var_z = accelerometer.get_values()	**x,y,z=accelerometer.get_values()**	Renvoie simultanément les mesures des trois axes de l'accéléromètre.
var = accelerometer.magnitude()	**mag=accelerometer.magnitude()**	Renvoie la somme des amplitudes normalisées des trois axes de l'accéléromètre. Cette méthode permet de détecter un mouvement tridimensionnel, tel qu'une secousse.
var = compass.heading()	**direction=compass.heading()**	Renvoie le cap de la boussole de 0 à 360 degrés.
var = compass.get_x()	**X=compass.get_x()**	Renvoie la valeur de l'axe des x du magnétomètre en nano Tesla sous la forme d'un nombre entier positif ou négatif, selon la direction du champ.
var = compass.get_y()	**Y=compass.get_y()**	Renvoie la valeur de l'axe des y du magnétomètre en nano Tesla, sous la forme d'un nombre entier positif ou négatif, selon la direction du champ.
var = compass.get_z()	**Z=compass.get_z()**	Renvoie la valeur de l'axe des z du magnétomètre en nano Tesla, sous la forme d'un nombre entier positif ou négatif, selon la direction du champ.
var = compass.is_calibrated()	**is_cal=compass.is_calibrated()**	Renvoie True si la boussole est calibrée, False sinon.

var = compass.get_field_strength()		Renvoie la mesure du champ magnétique autour de la carte en nano Tesla.
compass.calibrate()	**compass.calibrate()**	Force l'étalonnage de la boussole de la carte. Après l'exécution de la méthode, la carte fait défiler les instructions d'étalonnage sur la matrice de LEDs.
compass.clear_calibration()	**compass.clear_calibration()**	Supprime les enregistrements de calibration de la boussole.
accelerometer.current_gesture()	**g=accelerometer.current_gesture()**	Renvoie le geste actuel : 'haut', 'bas', 'gauche', 'droite', 'face en haut', 'face en bas' ou 'secouer'.
accelerometer.is_gesture()	**accelerometer.is_gesture('shake')**	Renvoie True si le geste courant correspond au geste passé en argument, False sinon. Le .is_gesture() effectue un test instantané.
accelerometer.was_gesture()	**accelerometer.was_gesture('up')**	Renvoie True si le geste passé en argument correspond au geste actuel à tout moment depuis le dernier appel de la méthode, False sinon.
Radio		
Importation du module microbit	**from mb_radio import ***	Active les méthodes de la radio interne.
radio.on()	**radio.on()**	Allume la radio 2,4 MHz.
radio.off()	**radio.off()**	Éteint la radio 2,4 MHz.
radio.config(channel=7,power=6,group=0)	**radio.config(channel=3,power=6,group=5)**	Configure les modes de la radio : le canal (fréquence opérationnelle de la radio), la puissance d'émission et le groupe (paramètre de routage des paquets). Pour que 2 micro:bits puissent communiquer, ils doivent partager à la fois le canal et le groupe. Plus la puissance est élevée, plus la distance de diffusion du signal est grande.

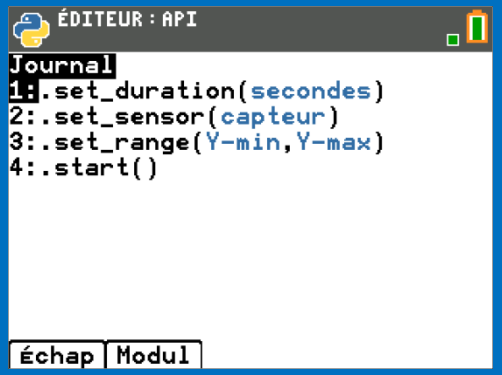
radio.send(”message”)	**radio.send(“Hello World!”)**	Transmet une chaîne de caractères. Doit être couplé avec radio.receive() sur la calculatrice réceptrice.
var = radio.receive()	**msg=radio.receive()**	Reçoit une chaîne de caractères du tampon de la radio. Le tampon peut contenir plusieurs messages dans une configuration de type "premier entré, premier sorti".
radio.send_number (val)	**radio.send_number(3.1415)**	Transmet un entier ou un nombre à virgule flottante. Doit être couplé avec radio.receive_number() sur la calculatrice réceptrice.
var=radio.receive_number()	**number=radio.receive_number**	Reçoit un entier ou un nombre à virgule flottante.
Broches Entrée/Sortie		
Importation du module microbit	**from mb_pins import ***	Active les méthodes analogiques et numériques sur les broches d'E/S.
var = pin0.read_digital()	**state_0 = pin0.read_digital()**	Renvoie l'état numérique de la broche 0 comme 0 (GND) ou 1 (3,3V).
var = pin0.read_analog()	**ADC_0 = pin0.read_analog()**	Renvoie la valeur analogique de la tension sur la broche 0 de 0 (GND) à 1023 (3,3V).
pin0.write_digital(valeur)	**pin0.write_digital(1)**	Règle la tension sur la broche de sortie numérique 0 entre GND (0) et 3,3V (1).
pin0.write_analog(valeur)	**pin0.write_analog(128)**	Règle la tension sur la broche de sortie analogique 0 entre GND (0) et 3,3V (1023).
var = pin1.read_digital()	**state_1 = pin1.read_digital()**	Renvoie l'état numérique de la broche 1 comme 0 (GND) ou 1 (3,3V).
var = pin1.read_analog()	**ADC_1 = pin1.read_analog()**	Renvoie la valeur analogique de la tension sur la broche 1 de 0 (GND) à 1023 (3,3V).
pin1.write_digital(valeur)	**pin1.write_digital(0)**	Règle la tension sur la broche de sortie numérique 1 entre

		GND (0) et 3,3V (1).
pin1.write_analog(valeur)	**pin1.write_analog(511)**	Règle la tension sur la broche de sortie analogique 1 entre GND (0) et 3.3V (1023)
var = pin2.read_digital()	**state_2= pin2.read_digital()**	Renvoie l'état numérique de la broche 2 comme 0 (GND) ou 1 (3,3V).
var = pin2.read_analog()	**ADC_2 = pin2.read_analog()**	Renvoie la valeur analogique de la tension sur la broche 2 de 0 (GND) à 1023 (3,3V).
pin2.write_digital(valeur)	**pin2.write_digital(1)**	Règle la tension sur la broche de sortie numérique 2 entre GND (0) et 3,3V (1).
pin2.write_analog(valeur)	**pin2.write_analog(750)**	Règle la tension sur la broche de sortie analogique 2 entre GND (0) et 3,3 V (1023).
pin.set_analog_period(valeur)	**pin.set_analog_period(10)**	Définit la période du signal PWM émis en millisecondes. La valeur minimale valide est de 1ms.
Capteurs Grove		
Importation du module microbit	**from mb_grove import * from mb_pins import ***	Active les méthodes de capteur et d'actionneur Grove. mb_grove et mb_pins sont tous deux nécessaires.
var(t),var(h) = grove.read_sht35()	**t,h = grove.read_sht35()**	Le capteur de température et d'humidité Grove SHT35 est un dispositif I2C et doit être branché sur un port I2C d'une carte d'extension. La fonction read_sht35() renvoie simultanément la température et l'humidité.
var = grove.read_temperature(pin)	**t = grove.read_temperature(pin0)**	Le capteur de température Grove Temperature sensor V1.2 est un dispositif analogique qui peut être utilisé avec n'importe quelle broche analogique compatible. La méthode .read_temperature() renvoie la température en ◦Celcius et requiert pour argument un numéro de broche.

var = grove.read_lightlevel(pin)	**b = grove.read_lightlevel(pin1)**	Le capteur de lumière Grove Light Sensor V1.2 est un dispositif analogique qui peut être utilisé avec n'importe quelle broche analogique compatible. La méthode .read_lightlevel() renvoie l'intensité de la lumière ambiante incidente en pourcentage de la sensibilité maximale du capteur (0-100 %) et nécessite pour argument un numéro de broche.
var = grove.read_moisture(pin)	**m = grove.read_moisture(pin2)**	La sonde d'humidité Grove Moisture Sensor V1.4 est un dispositif analogique qui peut être utilisé avec n'importe quelle broche analogique compatible. La méthode .read_moisture() renvoie l'humidité du contact sous la forme d'un pourcentage de la sensibilité maximale du capteur (0-100 %) et nécessite pour argument un numéro de broche.
var = grove.read_pressure(pin)	**p = grove.read_pressure(pin0)**	Le capteur de pression intégré Grove Integrated Pressure Sensor est un dispositif analogique qui peut être utilisé avec n'importe quelle broche compatible analogique. La méthode .read_pressure() renvoie la pression du connecteur du tube en kPa et requiert pour argument un numéro de broche.
calibrate_pressure(m,b)	**calibrate_pressure(.15,35)**	Cette méthode permet d'étalonner la sortie linéaire du capteur de pression intégré Grove avec deux coefficients d'étalonnage. Ces coefficients sont dérivés de la pente et de l'ordonnée à l'origine d'un modèle linéaire reliant la sortie ADC et la pression connue du connecteur du tube en kPa.
var = read_ranger_time(pin):	**t = read_ranger_time(pin0):**	Le Grove Ultrasoncie Ranger V2.0 est un dispositif numérique qui peut être utilisé avec n'importe quelle broche numérique compatible. La méthode .read_ranger_time() renvoie le temps de parcours du Ranger ultrasonique en secondes et nécessite pour argument un numéro de broche.
var = read_ranger_cm(pin)	**d = read_ranger_cm(pin0)**	La méthode read_ranger_cm() fonctionne avec le Ranger ultrasonique Grove, renvoie la distance à un objet en cm et requiert pour argument un numéro de broche.

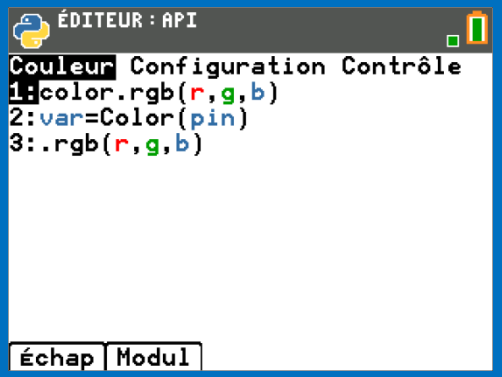
var,var,var = .read_bme280()	**t,p,h = grove.read_bme280()**	Le Grove BME 280 est un capteur de pression baromé-trique, de température et d'humidité. Calculez l'altitude à partir de la pression et de la température en utilisant la formule liant les variations de pression avec l'altitude.
grove.set_power(pin,valeur)	**grove.set_power(pin0,65)**	Le module Grove MOSFET V1.0 est un dispositif PWM qui fournit de l'énergie à partir d'une alimentation externe. La méthode .set_power() requiert deux arguments, le numéro de la broche de sortie analogique et le réglage de la puissance entre 0 % (désactivé) et 100 % (tension de source totale).
grove.set_relay(pin,valeur)	**grove.set_relay(pin1,1)**	Le module Grove Relay V1.2 est un dispositif numérique qui permet d'activer et de désactiver l'alimentation d'un circuit externe. La méthode .set_relay() requiert deux arguments, le numéro de la broche de sortie numérique et l'état du relais OFF(0) et ON(1).
grove.set_servo(pin,deg,min,max)	**grove.set_servo(pin0,45)**	Le moteur de balayage du Grove Servo est un dispositif numérique qui définit la position angulaire du servo. La méthode .set_servo() requiert deux arguments, la broche de sortie numérique et la position angulaire du moteur. Deux autres arguments de calibrage facultatifs définissent la durée minimale et maximale du train d'impulsions en ms.
Pins	**Pin0**	Liste des broches d'E/S valides. 0,1,2,8,13,14,15,16, et haut-parleur.

Enregistrement de données



Importation du module microbit	**from mb_log import ***	Active les méthodes d'enregistrement des données en temps réel.
data_log.set_duration(secondes)	**data_log.set_duration(10)**	Définit la durée de la session d'enregistrement des données en secondes. Chaque session enregistre 100 échantillons, quelle que soit sa durée.
data_log.set_sensor(capteur)	**data_log.set_sensor('accelerometer.get_x')**	Définit le nom du capteur d'enregistrement des données. Les capteurs valides sont les capteurs micro:bit intégrés et tout capteur Grove.
data_log.set_range(Y-min, Y-max)	**data_log.set_range(Y-min, Y-max)**	Définit la portée prévue du capteur sélectionné.
data_log.start()	**data_log.start()**	Commence à collecter et à représenter graphiquement les données du capteur en temps réel. Lorsque l'enregistrement est terminé, les données sont automatiquement stockées dans la liste "L1" (temps) et "L2" (capteur) de la calculatrice. Les deux listes peuvent être utilisées dans l'application statistiques de la calculatrice pour analyse détaillée et représentation graphique.

NeoPixel



Importation du module microbit	**from mb_neopx import ***	Active les méthodes NeoPixel.
color.rgb(r,g,b)	**color.rgb(255,128,0)**	Méthode d'accès rapide pour régler les quatre DELs NeoPixel montées sur la carte d'extension Bit Maker. La méthode requiert une valeur de 0 à 255 pour chaque canal rouge, vert et bleu.

var = Color(pin)	**RGB_LED = Color(pin2)**	Constructeur à utiliser avec un dispositif NeoPixel Grove RGB LED. Cette méthode nécessite un nom d'objet et une affectation de broche.
.rgb(r,g,b)	**RGB_LED.rgb(0,255,255)**	Méthode de la classe Color pour définir les canaux rouge, vert et bleu de 0 à 255.
np = NeoPixel(pin, pixels)	**np = NeoPixel(pin1, 20)**	Constructeur à utiliser avec n'importe quel dispositif NeoPixel, tel que le Grove stick ou ring. Le constructeur requiert une affectation de broche et le nombre de pixels sur le périphérique. Notez que si le nom de l'objet, np par défaut, est modifié, les autres méthodes collées à partir du menu doivent être mises à jour pour tenir compte du nouveau nom.
np[index] = (red,green,blue)	**np[2] = (0,255,0)**	L'objet np se présente comme une liste Python. Chaque élément de la liste correspond à un pixel du périphérique et doit être défini par une valeur r, g, b valide comprise entre 0 et 255.
.show()	**np.show()**	La méthode np.show() illumine le dispositif NeoPixel en fonction des valeurs de la liste np[].
.clear()	**np.clear()**	La méthode np.clear() permet d'éteindre tous les pixels de l'appareil.

Exemples de Programmes		
Module	**Nom du Programme**	**Comportement du Programme**
Affichage	**DISPTEST.8XV**	Affiche du texte, des images et des motifs de pixels.
Musique	**MUSCTEST.8XV**	Joue des mélodies et des sons à différents tempos et volumes.
Audio	**AUDTEST.8XV**	Joue tous les sons audio.
Microphone	**MICTEST.8XV**	Contrôle le niveau sonore du micro et affiche les événements sonores.
Boutons & Logo	**BUTNTEST.8XV**	Contrôle le bouton A, le bouton B et le logo tactile.
Capteurs & Gestes	**SNSRTEST.8XV**	Contrôle l'accéléromètre, le magnétomètre (boussole), la température et la luminosité (nécessite le module mb_disp). Une routine de calibration unique est effectuée si la boussole n'est pas calibrée (suivez les instructions sur la matrice de LEDs de la carte), puis relancez le programme.
	GESTEST.8XV	Contrôle les gestes et teste un événement gestuel.
Radio	**RADITEST.8XV**	Nécessite une deuxième calculatrice avec microbit exécutant le même programme. Les deux calculatrices enverront des messages dans les deux sens.
Broches Entrée/Sortie	**PINSTEST.8XV**	Contrôle les entrées numériques et analogiques sur les broches 0,1 et 2. Il écrit également les sorties numériques et analogiques sur les broches 0, 1 et 2.
Capteurs Grove	**GROVETEST.8XV**	Nécessite une carte d'extension Grove avec des capteurs connectés. Le programme contrôle plusieurs capteurs différents et définit également une puissance de sortie.
Enregistrement de données	**LOGTEST.8XV**	Affiche un graphique de l'accéléromètre X en temps réel et, une fois terminé, enregistre le temps et les valeurs du capteur dans les listes L1 et L2 pour une analyse plus approfondie et la création de graphiques dans l'application statistiques de la calculatrice.
NeoPixel	**NPTEST.8XV**	Nécessite une carte d'extension Grove et une bande NeoPixel. Le programme allume séquentiellement chaque pixel avec une couleur aléatoire.