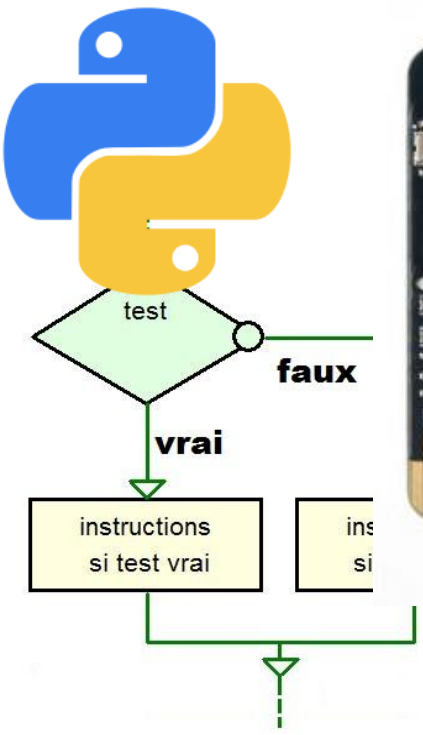
Langage Python et programmation du microcontrôleur micro:bit



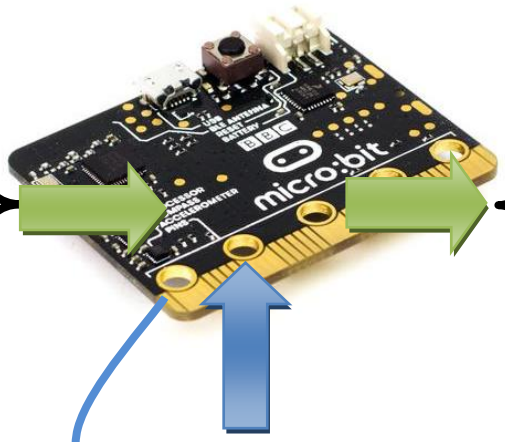
Le microcontrôleur Micro : bit : caractéristiques et utilisation.

La carte ***BBC Micro:bit*** est un micro-ordinateur de poche programmable qui peut être utilisé pour toutes sortes de créations : des robots aux instruments de musique – les possibilités sont infinies.

Cette carte permet de gérer des capteurs et des actionneurs.

Capteurs sur la carte :
Boussole
Accéléromètre
2 boutons poussoirs A et B
Capteur de température

Entrées extérieures :
Capteurs
Microphone,
Boutons poussoirs
Signal Radio
Bluetooth...



Sortie sur la carte :
Matrice de LED 5x5

Sorties extérieures :
Relais
LED extérieures
Signal Radio
Bluetooth
Etc...

port USB

Programmation :
Blocks,
Javascript,
Python,
Scratch...

I.D.E.

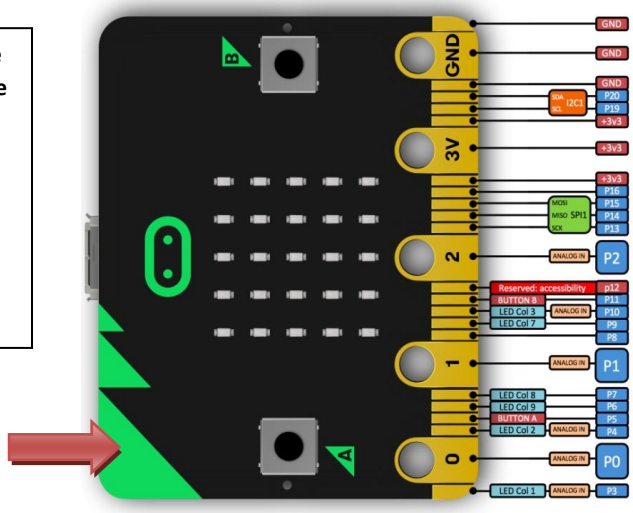
Mu Python
MakeCode Editor
Python Editor...

En ligne ou en local

Lorsque la carte est connectée au port USB, elle est alimentée et peut fonctionner.

Lorsque la carte n'est pas connectée au port USB, elle doit être alimentée par une alimentation extérieure 3V (piles).

Alimentation extérieure



La structure d'un programme Python

la structure d'un programme doit être la suivante :

```
from microbit import *
import neopixel
from random import randint
```

importation des modules

```
np = neopixel.NeoPixel(pin0, 8)
```

définition des variables

```
def repeated_frame(frame, count):
    for i in range(count):
        yield frame
```

définition des fonctions

```
while True:

    for pixel_id in range(0, len(np)):
        red = randint(0, 60)
        green = randint(0, 60)
        blue = randint(0, 60)
        np[pixel_id] = (red, green, blue)

        np.show()
        sleep(100)
```

programme principal

Le langage Python.

1) Conventions de style d'écriture

Indentation

En Python, les structures de blocs d'instructions sont déterminées par l'*indentation*. (espace après retour à la ligne)
Utilisez quatre espaces pour chaque niveau d'indentation comme dans l'exemple ci-contre.
	`a = 15`
	`while a != 1:`
	`    if a % 2 == 0:`
	`        a = a // 2`
	`    else:`
	`        a = 3 * a + 1`

Conventions de nommage

Constantes

Les constantes doivent être nommées par des identificateurs en lettres capitales, avec éventuellement des blancs soulignés ou chiffres.

```
RACINE_DEUX = 1.414
```

Variables

Les variables doivent être nommées par des identificateurs ne comprenant que des lettres avec éventuellement des blancs soulignés ou chiffres.

```
joueur1 = 'Philippe'
joueur2 = 'René'
numero_partie = 5
```

Fonctions

Les fonctions doivent être nommées en lettres, éventuellement avec des blancs soulignés ou chiffres.

```
def celsius_en_fahrenheit(celsius):
    fahrenheit = 9 / 5 * celsius + 32
    return fahrenheit
```

Mots-clés en Python

Un *mot-clé* est un identificateur ayant un sens prédéfini dans le langage de programmation. Les mots-clés ne peuvent pas être utilisés comme identificateur de constantes, variables, fonctions, etc. ..., sous peine de provoquer une erreur de syntaxe

```
And  as  assert  break
class  continue  def  del
else elif except  false
finally  for  from  global
if import in  is lambda
none  nonlocal  not  or
pass raise  return  true
try  while  with  yield
```

Commentaires

Tout ce qui suit le caractère # est un commentaire, et est ignoré par Python.

```
x = 0.035  # variable
```

2) Variables

Les entiers (integer) int	

Les entiers sont obtenus tout naturellement pas leur écriture habituelle en base 10.
On peut aussi donner des valeurs littérales dans d'autres bases que la base décimale. On préfixe alors l'écriture littérale par `0b` pour le binaire (base 2), `0o` pour l'octal (base 8) ou `0x` pour l'hexadécimal (base 16)
	`123  -123  etc...`
	
	`0b1111011  123`
	`0o173  123`
	`-0x7B  -123`

En Python3, aucune limitation sur la taille des entiers (hormis la taille de la mémoire de la machine sur laquelle le programme s'exécute).
Les littéraux entiers ne contiennent jamais de point. Il ne faut pas confondre `123` qui est un entier (type int) et `123.` qui est un flottant (type float).

Les flottants (float)	

Les littéraux flottants s'écrivent uniquement en base 10.
Ils comprennent tous un point dans leur écriture.
Ils peuvent être exprimés en notation scientifique :
	`123.5`
	
	`123e100`
	`1.23e+102`

Les flottants sont limités à environ 17 chiffres significatifs, et les flottants non nuls sont, en valeur absolue, compris entre 10^{-308}$ et 10^{308} environ.

Les chaînes de caractères (string)	
Les littéraux chaînes de caractères sont délimités par deux apostrophes (`'`) ou deux guillemets anglais (`"`).	`'Timoleon est un grec.'` `"Timoleon est un grec."`

Les booléens (boolean)	
Deux valeurs littérales seulement pour les booléens : `True` et `False`. Ce sont aussi deux mots-clés du langage	a=True

.Attention à la majuscule ! `TRUE`, `FALSE`, `true`, ... ne sont pas des littéraux booléens.

Les listes	
Une **liste** (ou **list** / **array**) en **python** est une variable dans laquelle on peut mettre plusieurs variables.	Exemple : Liste1 = ["a", "b", "c"] Liste2 = [1,2,3]

Les tuples	
Un **tuple** est une **liste** qui ne peut plus être modifiée. Pour créer un **tuple** , on utilise la syntaxe suivante : mon_tuple = ()	Tuple1 = (1, "ok", "super")

Les complexes	
La partie imaginaire est indiquée grâce à la lettre « j » ou « J ».	`b = 1 + 1j`

3) Les opérateurs

Opérateurs arithmétiques		Opérateurs relationnels	
+	Addition	<	Inférieur
-	Soustraction	>	Supérieur
*	Multiplication	<=	Inférieur ou égal
**	Exponentiation	>=	Supérieur ou égal
/	Division flottante	==	Égal
//	Division euclidienne : quotient	!=	Différent
%	Division euclidienne : reste (modulo)		

Opérateurs booléens		Classement des opérateurs par priorité décroissante :

`and`	Et logique	`**`
		`*, /, //, %`
		`+, -`
		`<, >, <=, >=, ==, !=`
		`Not`
		`And`
		`or`

`or`	Ou logique	
`not`	Négation	

4) Fonctions prédéfinies

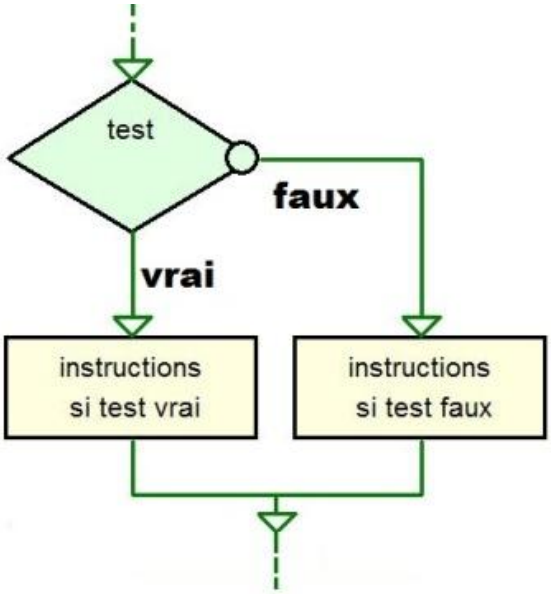
`abs(x)`	renvoie la valeur absolue du nombre `x`.
`chr(n)`	renvoie le caractère de numéro Unicode `n`.
`cmp(x, y)`	compare les deux valeurs `x` et `y` et renvoie une valeur négative si `x < y`, zéro si `x == y` et une valeur positive si `x > y`.
`float(arg)`	renvoie le flottant représenté par `arg` qui doit être un entier ou une chaîne de caractères.
`format(value,format_spec)`	renvoie une chaîne de caractères dont le contenu est donné par `value` et le format par `format_spec` (voir plus loin).
`help(arg)`	imprime une aide sur l'objet désigné par `arg`.
`input([prompt])`	invite l'utilisateur à entrer une donnée, et renvoie cette donnée sous forme d'une chaîne de caractères.
`int(arg)`	renvoie l'entier représenté par `arg` qui doit être un nombre ou une chaîne de caractères
`len(arg)`	renvoie la longueur de `arg` qui doit être une valeur séquentielle comme une liste ou une chaîne de caractères par exemple
`list(arg)`	renvoie une liste dont les éléments sont ceux de `arg` qui doit être une structure de données séquentielles.
`max(arg)`	renvoie le plus grand élément de `arg`.
`min(arg)`	renvoie le plus petit élément de `arg`
`ord(c)`	renvoie le numéro Unicode du caractère `c`.
`print(arg1, arg2, ...)`	imprime les arguments `arg1`, `arg2`, ... (voir plus loin).
`range([start], stop, [step])`	renvoie un générateur de nombres entiers compris entre `start` (0 si `start` est omis) inclus et `stop` exclus. Si `step` est précisé, sa valeur indique l'incrément de progression dans l'énumération.
`str(arg)`	renvoie une chaîne de caractères représentant `arg`
`type(arg)`	renvoie le type de `arg`

5) Entrées/sorties standards

Entrée au clavier `Var=input([message])` Cette fonction affiche le message "message" et lit la donnée entrée par l'utilisateur depuis le clavier et renvoie cette donnée sous forme d'une chaîne de caractères	Exemple : `nom = input('Entrez votre nom : ')`
Affichage `print(arg1, arg2, ...,'texte').`	

6) Instruction conditionnelle (`if`)

If <condition1>:
Instructions
Elif <condition2>:
Instructions
Else :
Instructions
 	
Il peut n'y avoir aucune partie `elif` ni même `else`
Le bloc d'instructions exécuté est le premier pour lequel la condition est vraie. Si aucune condition n'est vraie, c'est alors le bloc d'instructions suivant le `else` qui est exécuté s'il y a un `else`
Les instructions conditionnelles peuvent être imbriquées
	Exemple :
	
	`if a % 2 == 0:`
	`  a = a // 2`
	`else:`
	`  a = 3 * a + 1`



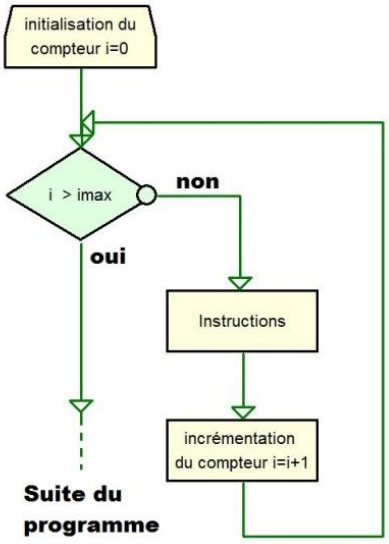
7) Boucles

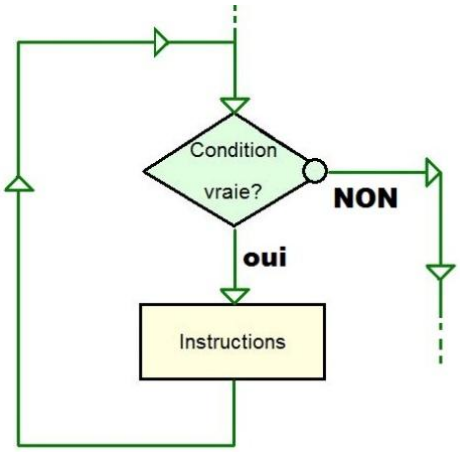
Répétitions bornées (for)

`For nom in range(e) :` `  instructions`	La variable nom varie de 0 à e-1
`For nom in range(e1, e2) :` `  instructions`	La variable nom varie de e1 à e2-1
`For nom in range(e1,e2,e3) :` `  instructions`	La variable nom varie de e1 à e2-1 par pas de e3

Répétition conditionnelle (`while`)

`White condition :` `  instructions`	Répète les instructions tant que la condition est vraie





9) Modules

Un *module* est un fichier contenant des définitions de constantes, fonctions, ...
La bibliothèque standard de Python fournit de base un grand nombre de modules consacrés à divers thèmes.

Importation dans le programme

From *nomDuModule* **import** nomDeLobjet	importe uniquement les fonctions nomDeLobjet
Import nomDuModule *	importe toutes les définitions du module.

Modules existants

math	**Random** **(voir page XX)**	**Microbit** **(voir pages 9 à 10)**	**time** **(voir page XX)**
music **(voir page 11)**			

Instructions Python pour le microcontrôleur Micro bit

Le microcontrôleur utilise les instructions Python classiques. Certaines instructions sont spécifiques à cette carte **Micro : bit** et sont contenues dans une bibliothèque qu'il faut importer au début du programme. Le programme doit commencer par importer la bibliothèque microbit.

Importer une bibliothèque

from *nom* import *	Importe la bibliothèque ***nom***
from microbit import *	Importe la bibliothèque de ***Microbit***

Lecture des capteurs internes

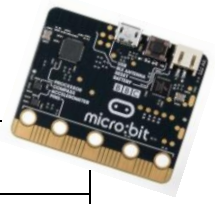
Boutons programmables A et B

button_a.is_pressed()	Retourne Vrai si le bouton A est pressé au moment de l'appel
button_a.was_pressed()	Retourne Vrai si le bouton A a été pressé depuis le dernier appel
button_b.is_pressed()	Retourne Vrai si le bouton B est pressé au moment de l'appel
button_b.was_pressed()	Retourne Vrai si le bouton B a été pressé depuis le dernier appel
Accéléromètre Les résultats varient de -2000 à +2000 (2g)	
accelerometer.get_x()	Donne l'accélération en millième de g selon l'axe transversal
accelerometer.get_y()	Donne l'accélération en millième de g selon l'axe longitudinal
accelerometer.get_z()	Donne l'accélération en millième de g selon vertical
Boussole	
compass.calibrate()	Calibration du compas
compass.heading()	Retourne le cap dans lequel est orientée la carte de 0 (NORD) à 360°

Lecture/Ecriture des entrées/sorties sur le connecteur

pin0.read_digital()	Renvoie la valeur 0 ou 1 suivant la valeur de la tension sur la broche 0
pin0.write_digital(v)	Ecrit la valeur logique (0 ou 1) sur la broche 0
pin0.read_analog()	Mesure la tension en entrée et renvoie un entier entre 0 et 1023
pin0.write_analog(v)	Génère un signal proportionnel à la valeur v comprise entre 0 et 1023

Les bornes utilisables sont nommées pin0 à pin2 et pin8 et pin16 (voir page précédente)



Contrôle de l'exécution

sleep(n)	pour suspendre l'exécution du programme pendant ***n*** millisecondes
Running_time()	Donne le temps en millisecondes passé depuis le démarrage de la carte

Affichage des LED de l'écran

display.set_pixel(x, y , z)	Allume la LED de coordonnées (x, y) à l'intensité n (0≤n≤9)
display.scroll(message)	Fait défiler un message
display.show(image)	Affiche une image sur les LED (voir ci dessous)
image=Image("aaaaa:" "aaaaa:" "aaaaa:" "aaaaa:" "aaaaa:")	Pour créer son image. Remplacer les ***a*** par un chiffre de 0 à 9. 0= LED éteinte, 9= LED éclairée au maximum.

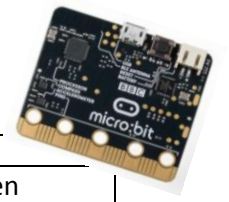
Liste des images existantes

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

Communication Radio

import radio	Importe la bibliothèque radio
radio.on()	Permet d'allumer l'émetteur de la carte
radio.send("message")	Permet d'envoyer un message texte via la liaison radio
radio.receive()	Permet de recevoir un texte depuis la liaison radio

Jouer des sons et de la musique



Il faut connecter un haut parleur entre la masse et la sortie n°0.

`import` **`music`**	`Importe le module` **`music`**
music.pitch(*freq*,*t*)	Joue la note de fréquence **freq** pendant la durée **t** en millisecondes Si t n'est pas indiquée, la note est jouée en continu.
music.play("noteOctave : t")	Joue la note ***note*** en notation anglaise de l'octave ***Octave*** pendant la durée ***t*** (0≤t≤8) Exemple : music.play("***A1:6"***)
music.play(nomDeLaMusique)	Joue la musique préprogrammée ***nomDeLaMusique***

le module ***music*** possède un grand nombre de mélodies préprogrammées :

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

pour créer sa mélodie :

melodie **= [note1,note2,...]** **music.play(*melodie*)**	Joue à la suite les notes continues dans la variable ***melodie*** ***Exemple :*** melodie = ["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"] music.play(***melodie***)

Remarque : si l'octave et la durée ne changent pas à la note suivante, il n'est pas nécessaire de les réécrire.

Fréquences des notes (en hertz) dans la gamme tempérée :

Notation anglaise	Note \ octave	0	1	2	3	4	5	6
C	Do	33	65	131	262	523	1047	2093
C#	Do# ou ré b	35	69	139	277	554	1109	2217
D	ré	37	73	147	294	587	1175	2349
D#	Ré# ou mi b	39	78	156	311	622	1245	2489
E	Mi	41	82	165	330	659	1319	2637
F	Fa	44	87	175	349	698	1397	2794
F#	Fa#	46	93	185	367	734	1480	2960
G	Sol	49	98	196	392	784	1568	3136
G#	Sol#	52	104	208	415	831	1661	3322
A	La	55	110	220	440	880	1760	3520
A#	La #	58	117	233	466	932	186	3729
B	si	62	123	247	494	988	1976	3951
R	pause	Pas de note jouée						

une mélodie connue : E4:6", "E4:6", "E4:6", "C4:4", "G4,2", "E4:6", "C4:4", "G4,2", "E4:6", "B4 :6", "B4:6", "B4 :6", "C5:4", "G4:2", "E4:6", "C4:4", "G4:2", "E4:6"

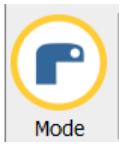
L'IDE Mu Python pour programmer

Le programme utilisé pour programmer la carte Microbit et l'IDE (Integrated Development Environment) Mu Python. Il peut être installé sur l'ordinateur ou être sur une clé USB.

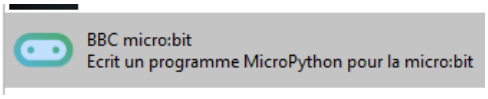
- Connecter d'abord le microcontrôleur au port USB

- Lancer Mu python.

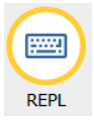
- Sélectionner Mode puis BBC microbit.



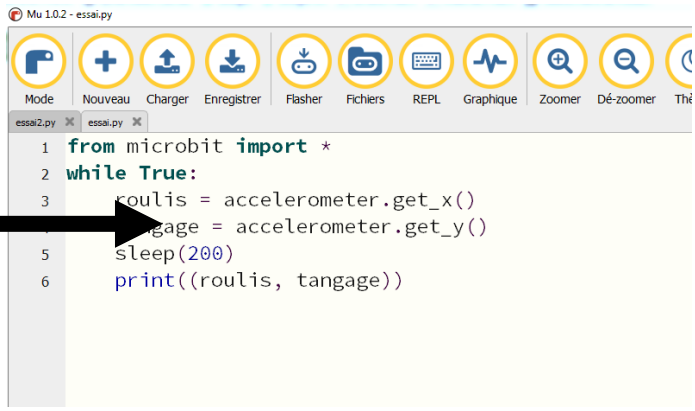
puis



- Ouvrir l'interface de communication



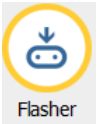
- Ecrire votre programme.



- Vérifier votre programme et corriger les erreurs.

- Enregistrer votre programme.

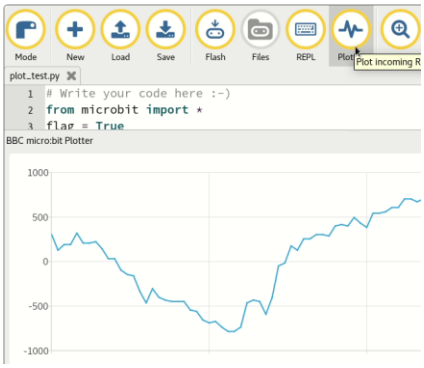
- Transférer votre programme à la carte.



Pour afficher les données sous forme de graphique, sélectionner 

.

Les données sont affichées avec l'instructions print((variable,))



Exemples de programmes

Lire une entrée

Entrée digitale

```
# TP microntroleur Lycée A Camus Rillieux
# lectrue d'une entrée digitale (logique) pin0 toutes les 0.5s

from microbit import *

n = ()

while True:
    n = pin0.read_digital()
    print(n)
    sleep(500)

```

entrée analogique

```
# TP microcontroleur Lycée A Camus Rillieux
# lecture d'une entrée analogique pin0 toutes les 0.5s

from microbit import *
while True:
    n = pin0.read_analog()
    print((n,))
```

Attention à la syntaxe pour afficher le graphe !

texte défilant

```
# TP microcontroleur Lycée A Camus Rilleux
# Texte défilant
from microbit import *


while True:
    display.scroll('Lycee A Camus')
    sleep(2000)

```

nombres au hasard

```
# Tp microcontroleur Lycee A Camus Rillieux
# nombre au hasard entre 0 et 100

import random
import time

while True:
    time.sleep(0.05)
    print((random.randint(0, 100),))
```

Allumer une LED extérieure

```
# Tp microcontroleur Lycee A Camus Rillieux
#  Allumage progressif d'une LED Extérieure sur pin1
from microbit import *

for i in range(1023):
    pin1.write_analog(i)
    print(i)
    sleep(10)

sleep(500)

for i in range(1023):
    pin1.write_analog(1023-i)
    print(1023-i)
    sleep(10)
```