Math-Sciences & Météo

Tutoriel

Nicolas NOWAK

Martial ANDRÉ

Conseils techniques : Léo BRIAND (Vittascience)

Région académique
HAUTS-DE-FRANCE

Janvier 2020

Sommaire

Composition du kit « Ballon solaire »

Le kit « Ballon Solaire » ou le kit « Station Météo » disponibles sur le site de Vittascience qui est accessible à l'adresse suivante : https://fr.vittascience.com/

L'avantage de ces kits, c'est qu'ils sont prêts à l'emploi, tout y est :

Pour le kit « Station Météo » :

- 1 carte microbit avec câble USB
- 1 capteur de température, pression et altitude (3 en 1) : BMP280
- 1 capteur de qualité de l'air (CO_2 et COV) : SGP30
- 1 capteur de luminosité
- 1 écran LCD monochrome 32 caractères
- 1 carte µSD avec lecteur et adaptateur USB
- 1 breadboard et des câbles
- Piles

Pour le kit « Ballon Solaire » :

- 2 kits « Station Météo »
- Bâche en polyéthylène noir 30 m × 1 m
- 2 poignées avec 45 m de corde
- 1 paire de ciseaux
- 2 rouleaux adhésifs
- 1 patron de fuseau
- Guide d'utilisation

L'utilisation de ces matériels, a pour objectif d'initier les élèves à la programmation via la technologie Microbit tout en les sensibilisant aux enjeux du développement durable.

Durant l'expérience, le rôle de vos chercheurs sera de construire une nacelle et l'enveloppe solaire du ballon, programmer divers capteurs : (CO_2, température, altitude, pression …) et d'analyser les données collectées durant l'envol du ballon solaire.

Côté réglementation

Le ballon solaire exploite l'énergie solaire pour s'envoler, et peut porter jusqu'à un kilogramme de charge utile avec des conditions météorologiques optimales.

Le ballon doit être tenu attaché par deux cordes de 50 mètres (incluses dans le kit), ce qui permet de faire voler le ballon jusqu'à 50 mètres de hauteur. **La réglementation pour les ballons captifs est issue de l'arrêté du 11 décembre 2014 et autorise le vol captif jusqu'à 50 mètres sans autorisation spécifique.**

Le ballon peut voler indéfiniment une fois construit, en revanche, on ne peut le déconstruire puis le reconstruire. Bien que fragile, il est très simple à réparer à l'aide de ruban adhésif. Une fois dégonflé il rentre dans un sac à dos d'étudiant, le transport et le stockage ne posent donc pas des problèmes.

Carte microcontrôleur : microbit

Qu'est-ce qu'un microcontrôleur ?

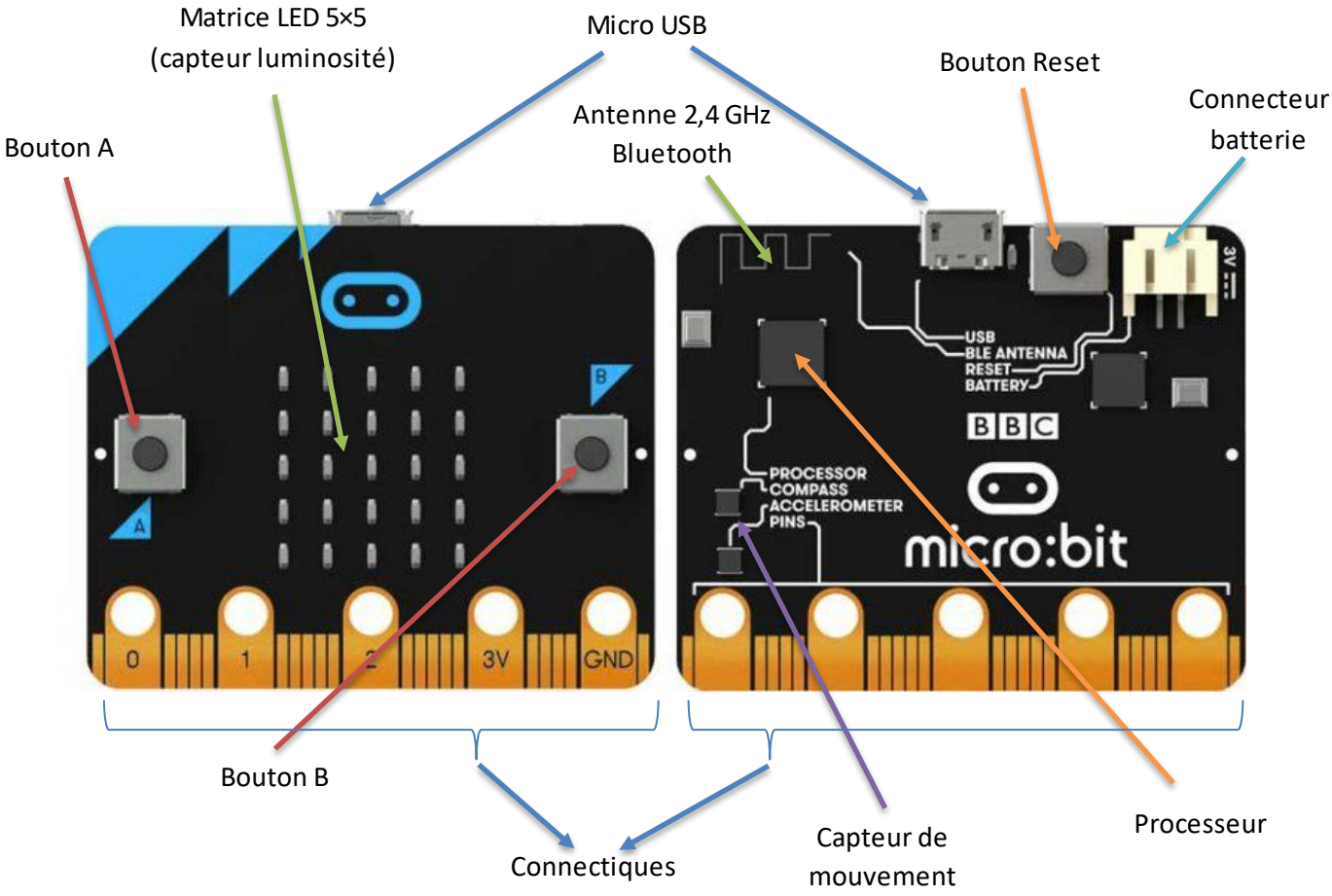
Le terme de microcontrôleur est bien moins connu que celui de microprocesseur, ce composant est pourtant omniprésent dans notre environnement (voiture, électroménager, télévision, Smartphone, GPS, ...).

Dans un seul et même circuit intégré, appelé "microcontrôleur" sont embarqués :

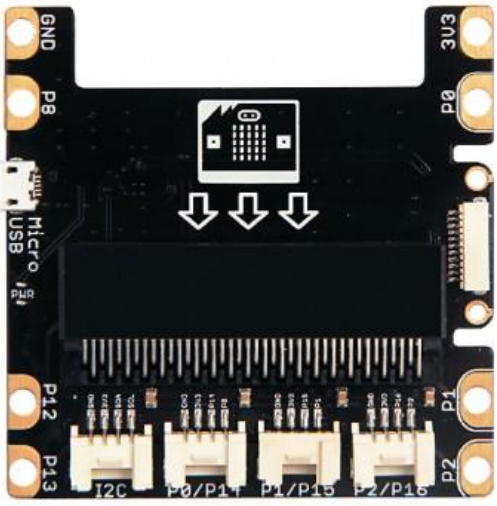
- Un microprocesseur (initialement sur 8 bits ... bien souvent sur 32 bits maintenant)
- De la mémoire de type RAM
- De la mémoire non volatile pour stocker le programme (c'est l'équivalent du disque dur pour un ordinateur). Cette mémoire est maintenant reprogrammable ce qui permet de mettre à jour le programme ("firmware")
- Des ports de communication de type parallèle ou série : UART (pour la communication RS232), USB, I2C, CAN (Controller Area Network), Ethernet...voire même un module radio (Bluetooth, wifi, ...)
- Des convertisseurs analogique-numériques (CAN) ...

Le même composant peut ainsi être affecté à des taches très différentes.

Description de la carte microbit.



Pour faciliter les branchements, il est utile d'utiliser un « Shield » et des capteurs Grove.



Shield Grove pour microbit

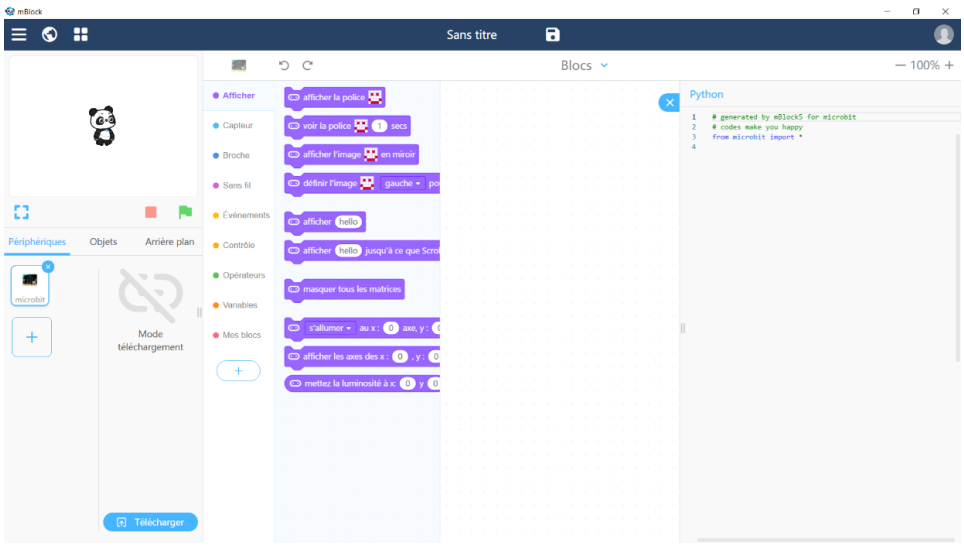
Capteurs Grove

Câbles de connexion

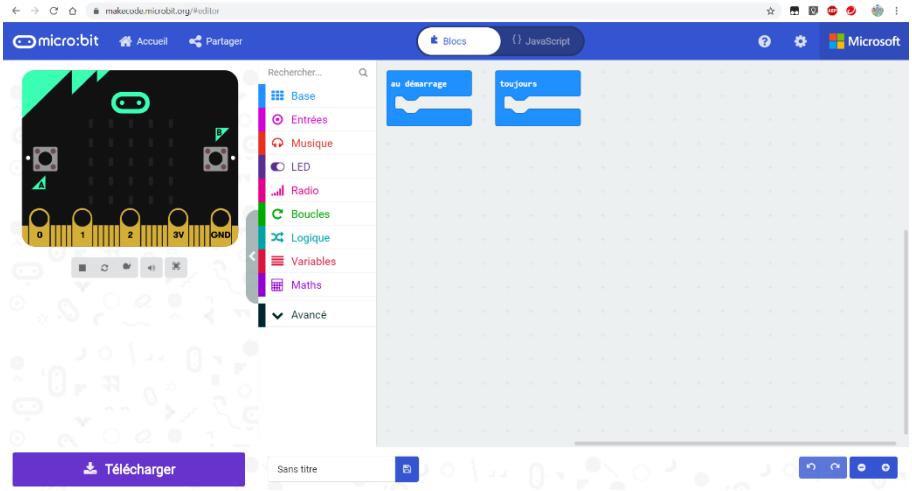
Programmation de la carte microbit.

La carte peut se programmer à l'aide de plusieurs logiciels ou en ligne via une plateforme de programmation :

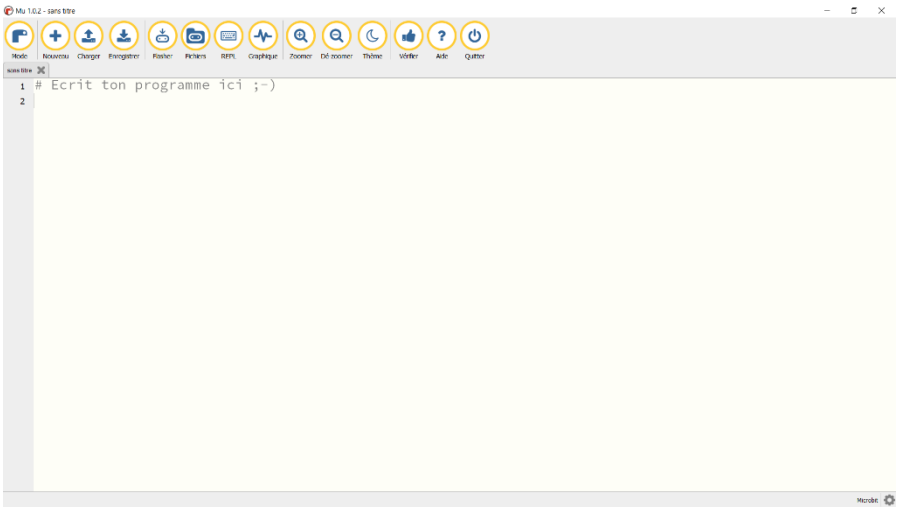
- Logiciel MBlock (https://www.mblock.cc/en-us/) : Programmation en blocs et en Python (utilisation uniquement des capteurs de la carte)



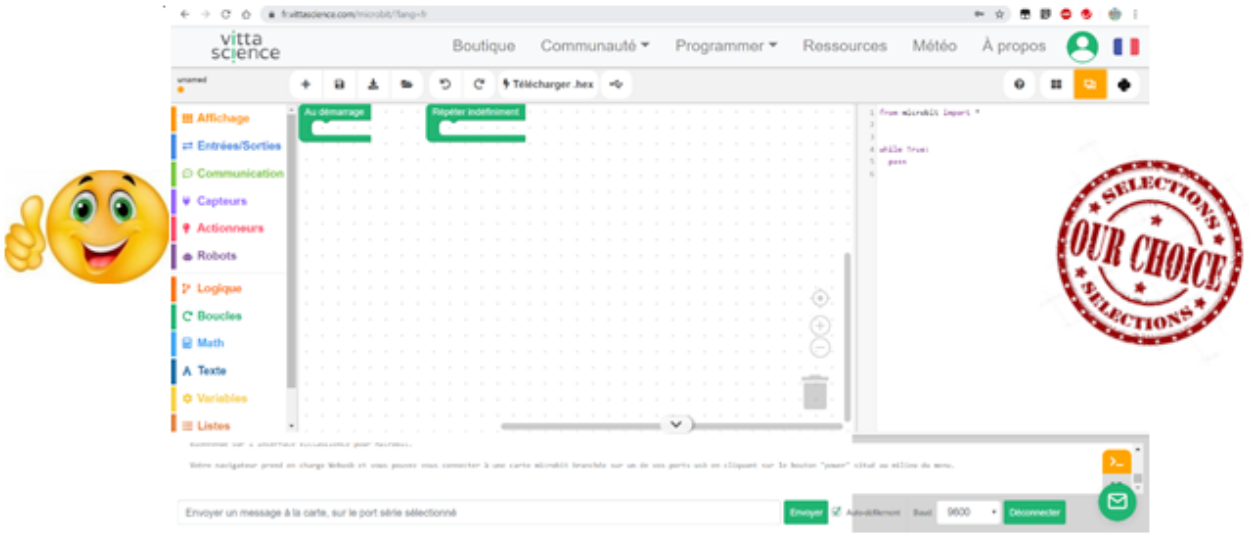
- L'éditeur MakeCode for microbit (https://makecode.microbit.org/) : Programmation en blocs avec « émulateur de la carte ».



- Le logiciel Mu-Editor (https://codewith.mu/) : Programmation en Python et visualisation en direct des mesures.



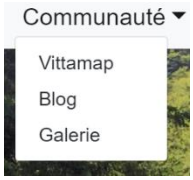
- La plateforme de Vittascience (https://fr.vittascience.com/) : Programmation en blocs ou en Python et visualisation en direct des mesures.



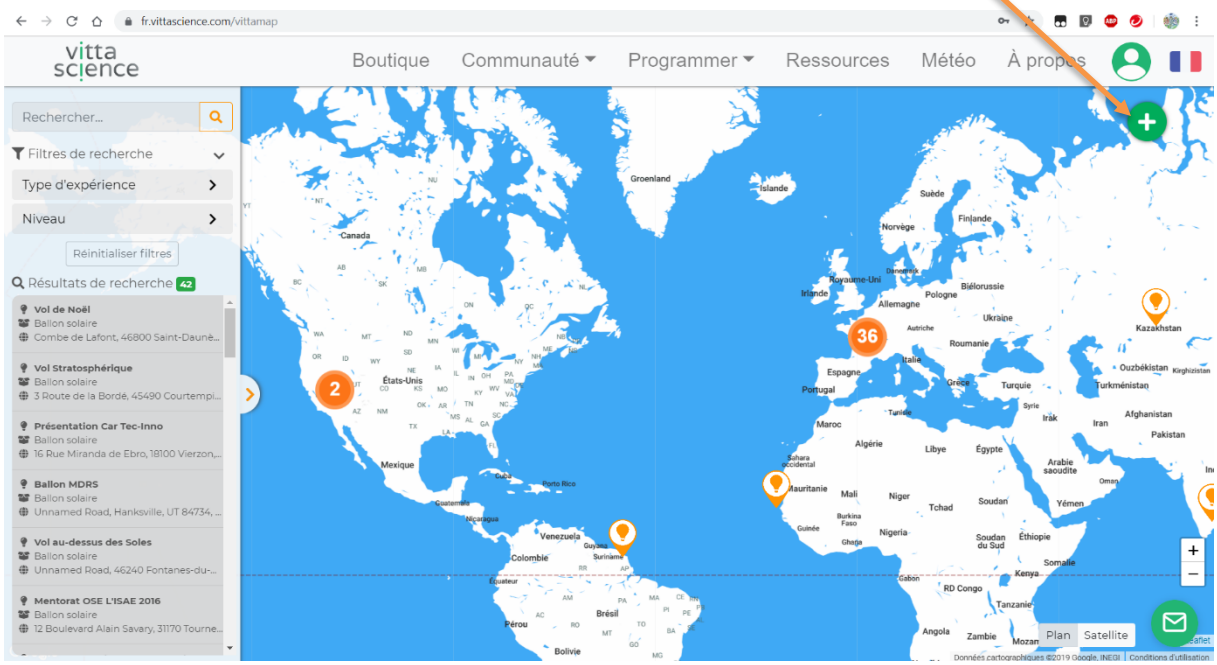
Site Vittascience

Il est accessible à l'adresse https://fr.vittascience.com/.

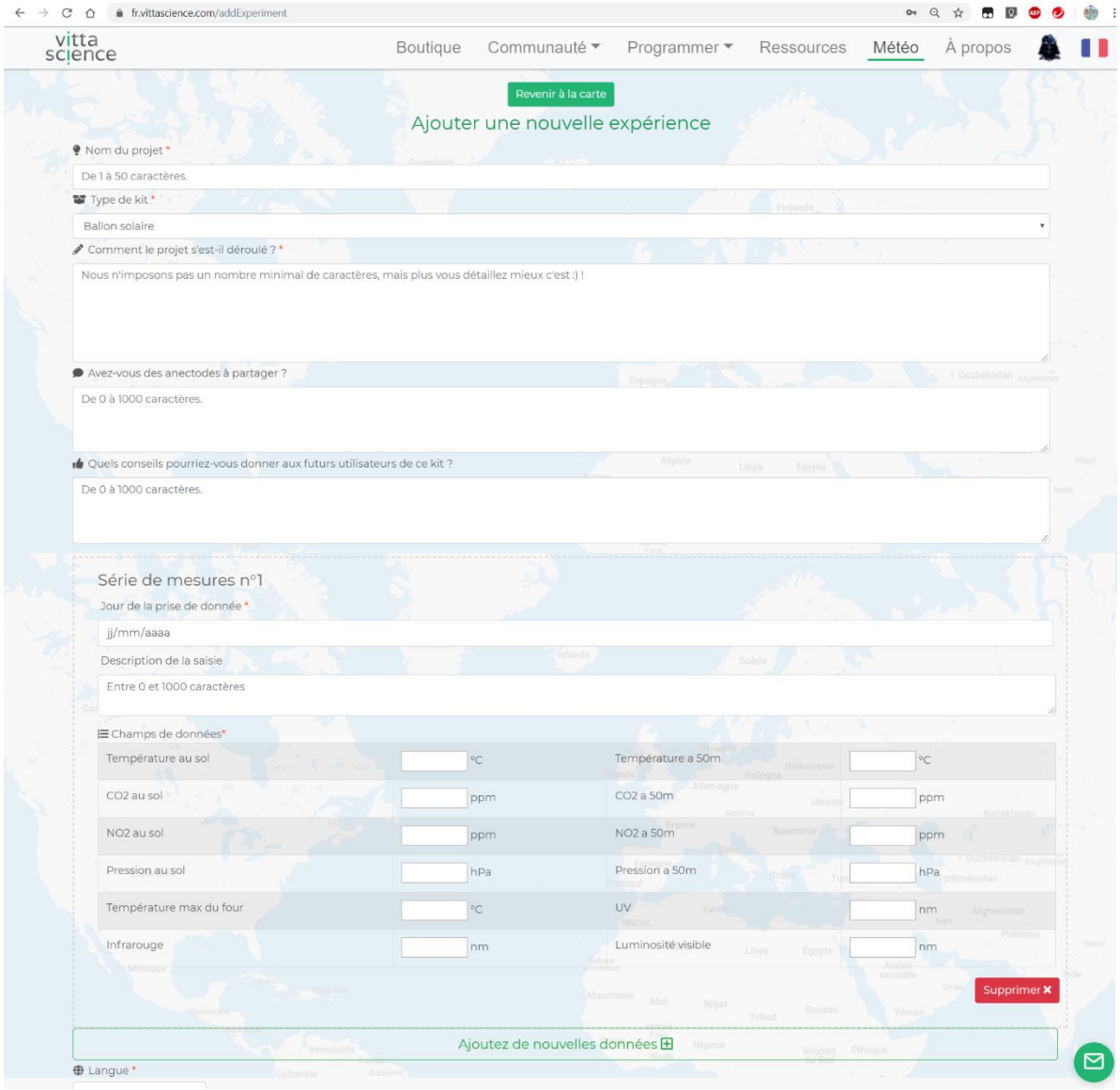
Vittamap.



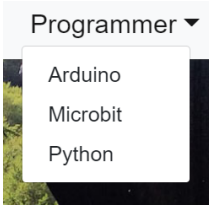
Ajouter une nouvelle expérience (Voir ci-après)



Ajouter une nouvelle expérience.

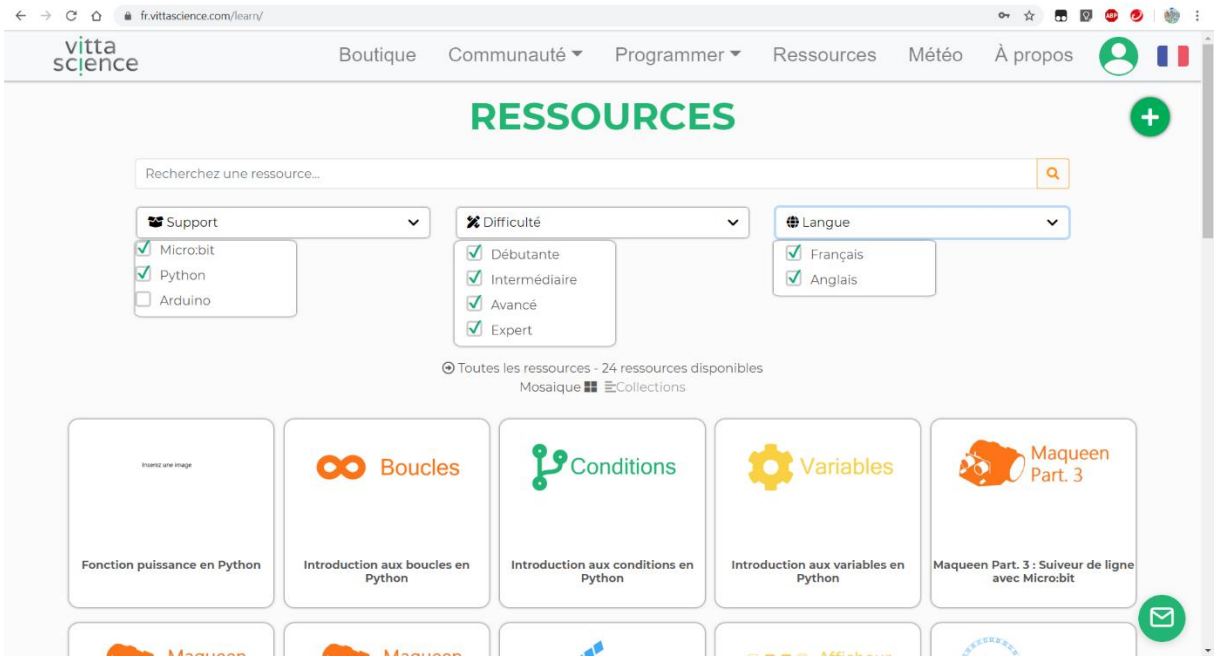


Onglet « Programmer ».

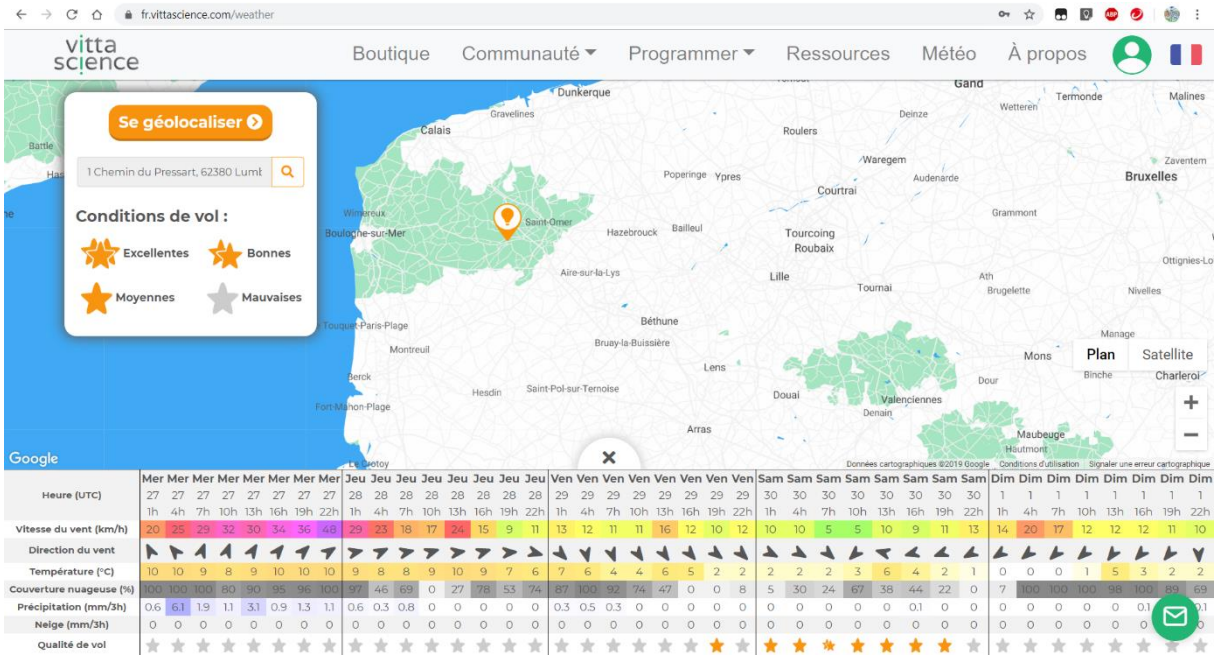


Voir section « Plateforme de programmation pour la carte microbit »

Onglet « Ressources ».



Onglet « Météo ».

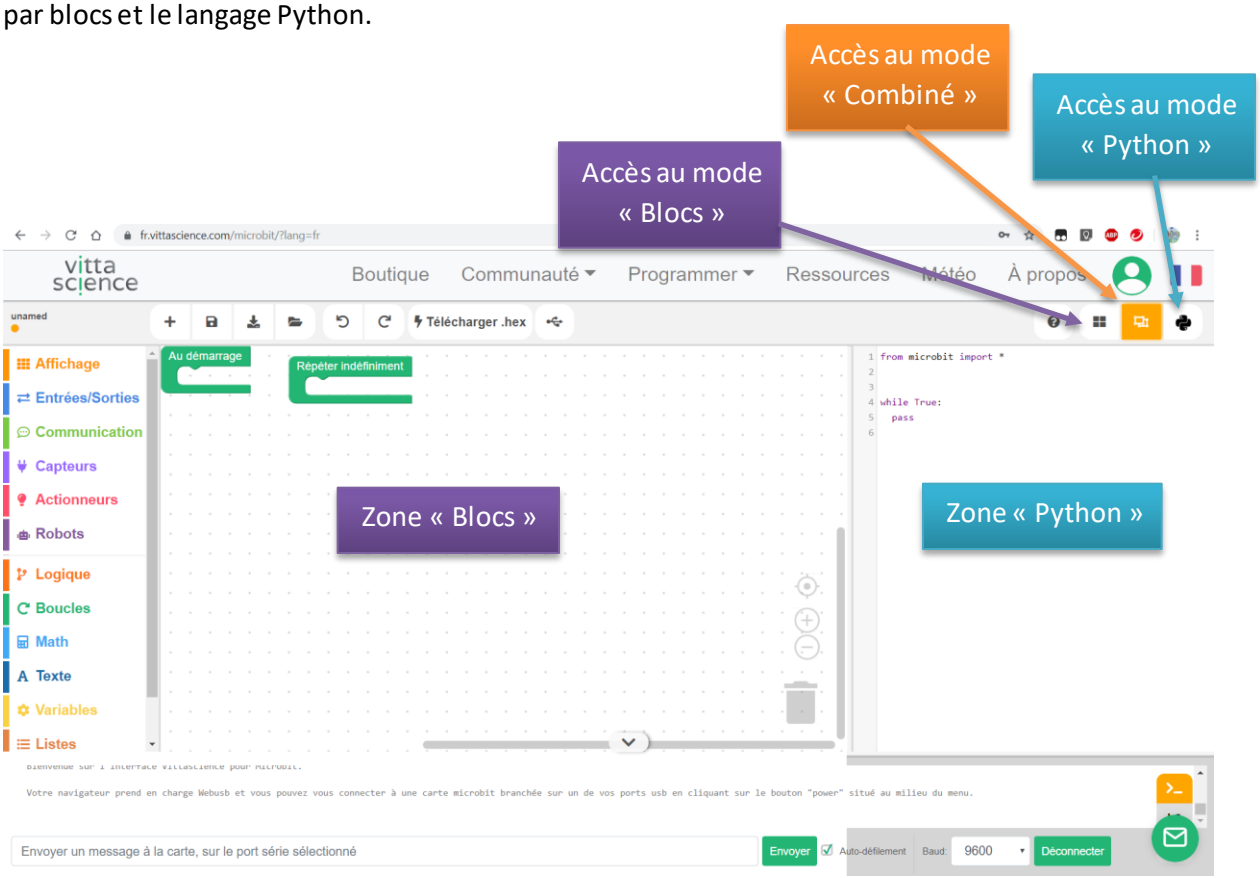


Plateforme de programmation pour la carte microbit

La plateforme est accessible à l'adresse suivante : https://fr.vittascience.com/microbit/.

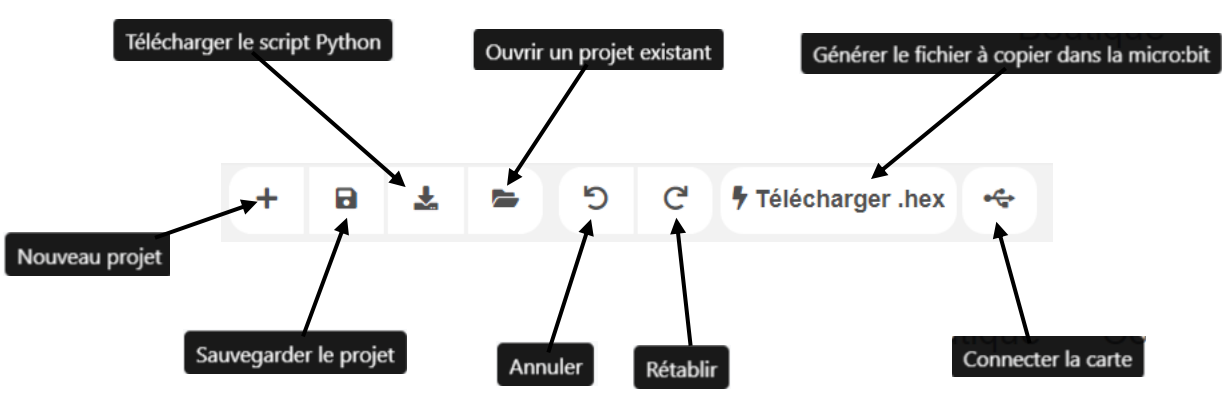
Pour utiliser le mode connecté de la carte et visualiser en direct le tracé des capteurs, on sera amené à utiliser le navigateur Google Chrome (https://www.google.com/intl/fr_fr/chrome/).

La programmation de la carte microbit peut se faire en blocs (basé sur Scratch) ou en Python. La plateforme Vittascience propose un mode « combiné » afin de faciliter la transition entre le langage par blocs et le langage Python.



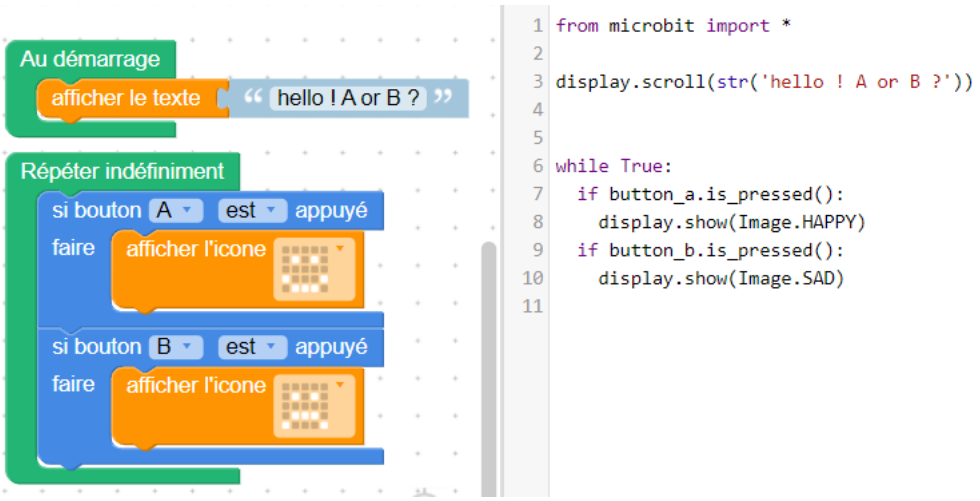
Plateforme de programmation en mode « combiné »

Menu de la plateforme



Exemple de programme en blocs avec sa correspondance en Python.

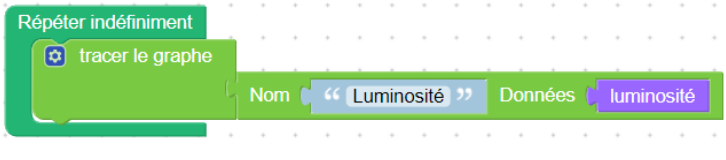
Lorsque l'on programme en blocs, la correspondance en Python se fait automatiquement (Attention la réciproque n'est pas vraie !)

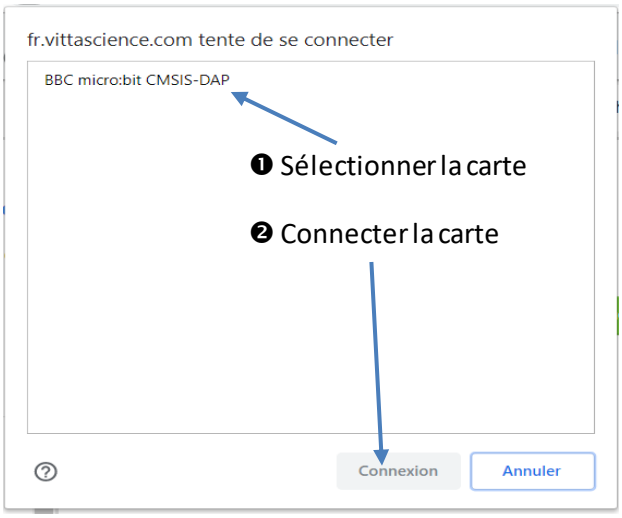


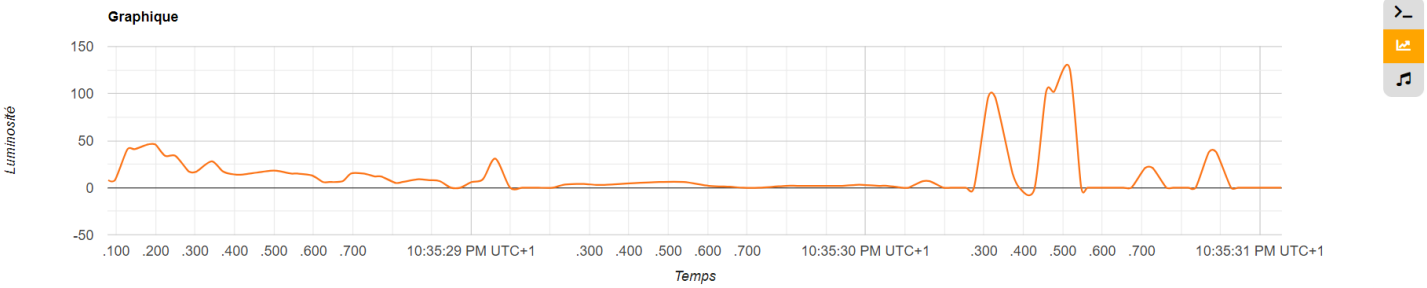
Zone graphique.

Il y a possibilité de visualiser en temps réel dans la zone graphique, les valeurs lues par les capteurs.

Pour cela, après avoir écrit le script, il faut connecter la carte à l'aide de l'outil  .
La fenêtre suivante apparaît :



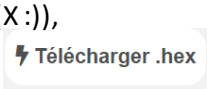




Programmation de la carte microbit

Comment téléverser un script dans la carte microbit ?

C'est une étape essentielle pour que la carte exécute le script. L'objectif est de convertir le script en langage « machine ». La manipulation génère un fichier à copier dans la carte.

- Connecter la carte microbit à un port USB de l'ordinateur,
- La carte est reconnue comme disque amovible (repérer la lettre du lecteur par exemple MICROBIT (X :)),
- Cliquer sur 

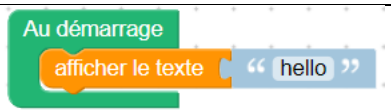
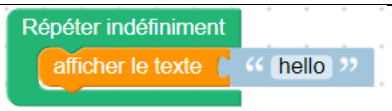
,
- Copier le fichier téléchargé dans le lecteur « MICROBIT » (il se trouve dans le dossier téléchargement ou download),
- L'exécution du script est automatique.

Programmation par blocs.

L'écriture des scripts se fait de la même façon qu'avec Scratch ou MBlock

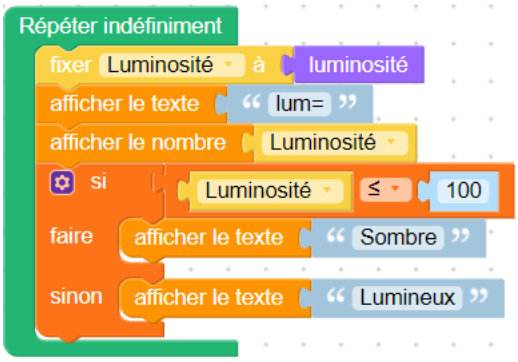
- Quelle est la différence entre ces 2 blocs ?

Exemple :

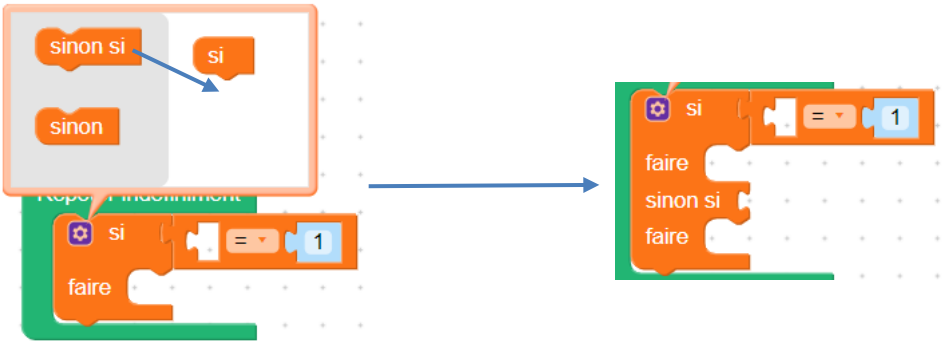
Script	Analyse
	Fait défiler le texte « hello » sur la matrice LED **1 seule fois**
	Fait défiler le texte « hello » sur la matrice LED **indéfiniment**

- Écrire un script qui permet d'afficher à l'écran la valeur de la température et le tester. (Le capteur température est celui du processeur)

- Que fait ce script ?



- Modifier le script précédent pour faire 4 niveaux de luminosité (C'est la matrice LED qui joue le rôle de capteur de luminosité, les valeurs s'échelonnent entre 0 et 255)
Blocs à utiliser :



Programmation en Python.

Le tutoriel est basé sur le document consultable et téléchargeable à l'adresse : https://microbit-micropython.readthedocs.io/fr/latest/tutorials/introduction.html

Comme tous les scripts écrits en Python, il faut faire appel aux bibliothèques afin de pouvoir utiliser les bons arguments de programmation.

Ici, il faut faire appel à la bibliothèque « microbit » en tapant la ligne de code :

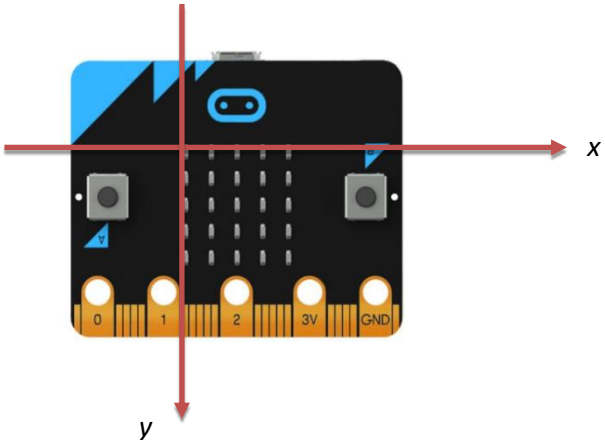
```
1 from microbit import *
```

Lorsque l'on écrit un programme en python, il faut faire très attention à l'indentation du code, indentation que l'on réalisera par 4 espaces.

L'indentation se réalise toujours après un double-point (« : ») pour délimiter l'entrée dans un bloc du programme. Toutes les lignes de code de ce bloc doivent être alignées sur la même verticale. Ceci est indispensable à l'interpréteur Python, mais nous est aussi très utile pour bien comprendre le code.

Matrice LED 5 × 5.

L'afficheur 5 × 5 nous est perçu comme étant un objet réel. La carte microbit fournit un objet au sens Python, nommé « **display** » sur lequel on peut appliquer des méthodes en suivant la syntaxe : **objet.méthode** dans le but d'agir sur l'objet réel qu'est la matrice LED



Objet	Méthode	Description
display.	set_pixel(x,y,b)	Permet de régler la luminosité b (entre 0 et 9) d'un pixel (LED) de coordonnées (x,y)
	scroll('texte')	Fait défiler le texte
	scroll(str(x))	Fait défiler une valeur numérique x (convertie en chaîne de caractère)
	show(Image)	Affiche « l'Image » (prédéfinie ou créée)
	clear()	Efface la matrice

Premier script en Python : Un grand classique !

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

- Saisir le script ci-contre
- Téléverser-le dans la carte en suivant le tutoriel du début du chapitre

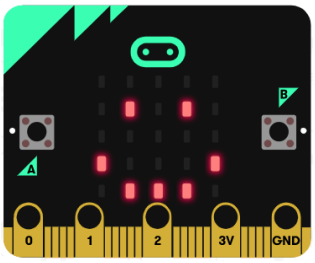
Images natives – Images personnalisées.

Images natives.

La bibliothèque « microbit » comprend beaucoup d'images prédéfinies.

Pour les coder, on utilise l'objet « display » avec la méthode « show » :

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

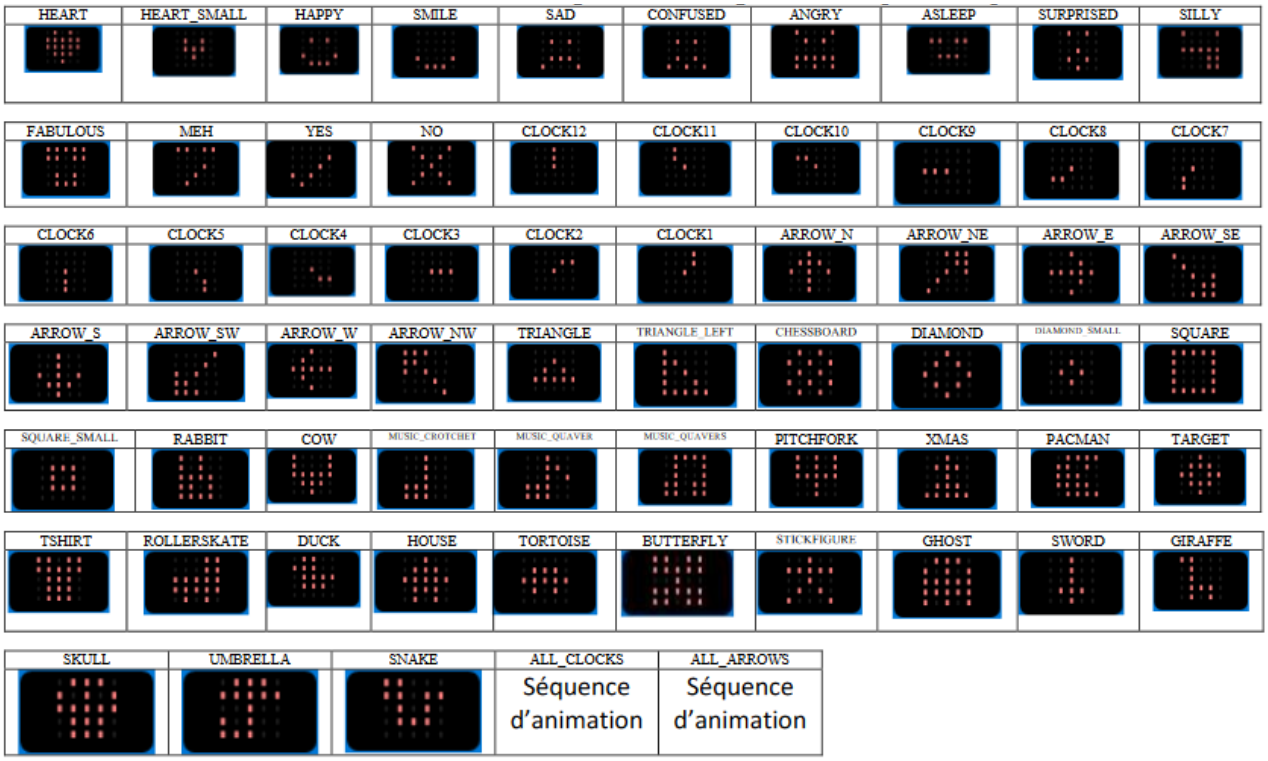


L'image « happy » que nous voulons afficher fait partie de l'objet Image et est appelée HAPPY.

Pourquoi ne pas modifier le code qui donne à microbit l'air heureux pour voir à quoi ressemblent les autres images intégrées ?

Liste des images natives :

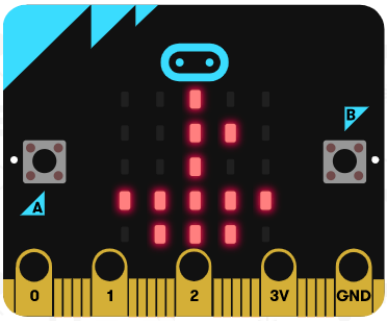
Image.HEART
Image.HEART_SMALL
Image.HAPPY
Image.SMILE
Image.SAD
Image.CONFUSED
Image.ANGRY
Image.ASLEEP
Image.SURPRISED
Image.SILLY
Image.FABULOUS
Image.MEH
Image.YES
Image.NO
Image.CLOCK12
Image.CLOCK11
Image.CLOCK10
Image.CLOCK9
Image.CLOCK8
Image.CLOCK7
Image.CLOCK6
Image.CLOCK5
Image.CLOCK4
Image.CLOCK3
Image.CLOCK2
Image.CLOCK1
Image.ARROW_N
Image.ARROW_NE
Image.ARROW_E
Image.ARROW_SE
Image.ARROW_S
Image.ARROW_SW
Image.ARROW_W
Image.ARROW_NW
Image.TRIANGLE
Image.TRIANGLE_LEFT
Image.CHESSBOARD
Image.DIAMOND
Image.DIAMOND_SMALL
Image.SQUARE
Image.SQUARE_SMALL
Image.RABBIT
Image.COW
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
Image.SKULL
Image.UMBRELLA
Image.SNAKE



Images personnalisées.

On a la possibilité de créer sa propre image. Pour cela, il nous suffit de travailler sur les LED une par une.

Essayons de réaliser l'image suivante :



La première méthode qui nous vient à l'esprit est d'allumer les LED voulues avec une ligne de commande du type :

display.set_pixel(2,0,5) (pour allumer la 3ème LED de la 1ère ligne avec l'intensité 5)

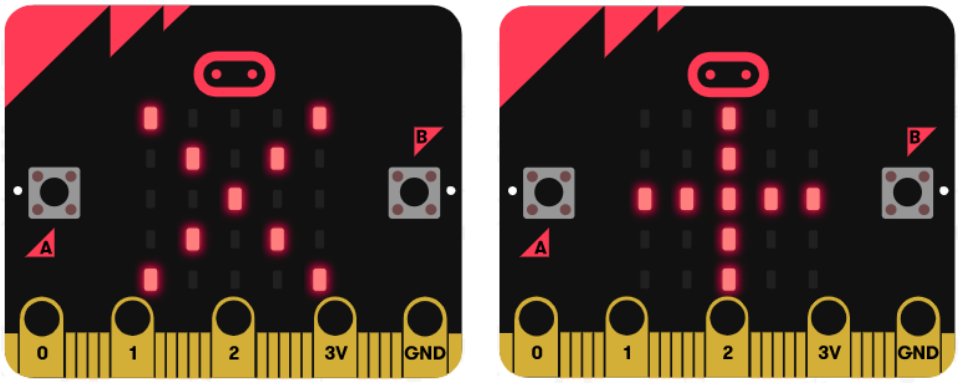
Une autre méthode consiste à créer une variable « Boat » qui sera notre image, puis d'afficher cette image avec l'instruction display.show(Boat). On pourra ainsi la réutiliser à d'autres endroits dans le script.

```
from microbit import *

# Création de la variable image "Boat"
Boat=Image("00500:"
           "00550:"
           "00500:"     # Allume les différentes LED avec l'intensité désirée
           "99999:"
           "09990")

display.show(Boat)      # Affiche l'image créee
```

Remarque : Si vous n'avez pas besoin de la vision matricielle, on peut aussi écrire :

```
from microbit import *

Boat=Image("00500:00550:00500:99999:09990")

display.show(Boat)
```

Application : Créer ces 2 images et afficher-les sur la carte microbit



Animation (temporisation).

Les animations sont une succession d'image avec un temps de pause entre chacune d'elles.

La ligne de code permettant cette attente est : `sleep(n)` où n est donné en millisecondes

Nous allons utiliser le script des 2 images précédentes, faire une pause de 0,5 s et faire l'animation 5 fois avec une boucle FOR (détaillée plus loin).

```
from microbit import *

Fois=Image("90009:"
           "09090:"
           "00900:"
           "09090:"
           "90009")

Plus=Image("00900:"
           "00900:"
           "99999:"
           "00900:"
           "00900"

for i in range(5):
    display.show(Fois)
    sleep(500)
    display.show(Plus)
    sleep(500)
```

Une autre façon est de créer une liste dans laquelle on met les images et on demande de les afficher avec un délai entre chaque image.

```
from microbit import *

Fois=Image("90009:09090:00900:09090:90009")
Plus=Image("00900:00900:99999:00900:00900")

Fois_Plus=[Fois,Plus]   # on définit une liste contenant les 2 images

for i in range(5):
    display.show(Fois_Plus,delay=500)
```

Les images des « flèches heures » et des « flèches boussole » sont rangées dans 2 listes : « ALL_CLOCKS » et « ALL_ARROWS » (respectivement).

Exemple :

```
from microbit import *

while True:             # on crée une boucle infinie
  display.show(Image.ALL_CLOCKS,delay=100)   # on affiche les flèches heures successivement toutes les 0,1 s
  pass
```

Applications :

- Modifier le script précédent pour afficher successivement les flèches boussoles toutes les 0,2 s.
- Créer une animation pour faire couler le bateau.

Boutons.

Jusqu'à maintenant nous avons créé du code qui fait faire quelque chose à l'appareil. C'est ce qu'on appelle une **sortie** ou **output**.

Cependant, nous avons aussi besoin que l'appareil réagisse à quelque chose. C'est ce qu'on appelle des **entrées** ou **inputs**.

C'est facile à retenir : une sortie est ce que l'appareil fait ressortir vers le monde extérieur tandis qu'une entrée c'est qui provient du monde extérieur et entre dans l'appareil.

Exemple :

```
from microbit import *

display.show(Image.ASLEEP)  # affiche l'image dodo
sleep(10000)
display.scroll(str(button_a.get_presses()))     # compte le nombre de fois où le bouton A a été appuyé
```

Tout ce que fait ce script c'est de dormir pendant dix mille millisecondes, (c'est-à-dire 10 secondes) et ensuite de faire défiler le nombre d'appuis sur le bouton A. C'est tout !

Objet	Méthode	Description
button_a. button_b.	get_presses()	Compte le nombre de fois où le bouton (A ou B) a été pressé. **Renvoie une valeur numérique**
	is_pressed()	Attend la pression sur le bouton (A ou B). Utile dans les tests

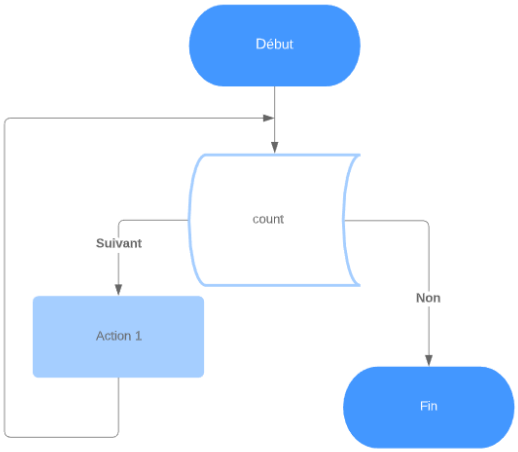
Boucles.

Dans de nombreux algorithmes, il est nécessaire de répéter un certain de nombre fois les mêmes instructions.
Pour éviter d'écrire ces répétitions, on utilise en algorithmique des **boucles**.

Il existe principalement deux types de boucles, les boucles finies « **Pour** » (FOR) et les boucles conditionnelles « **Tant Que** » (WHILE).

For (Pour).

La boucle « Pour » est utilisée **lorsque le nombre d'itérations est connu.**
On dit alors que la boucle est bornée.



L'instruction en Python correspondant aux boucles bornées est :

```
for i in range(...):
  Action 1
  ...
```

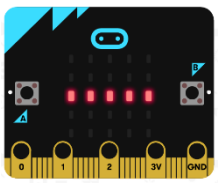
Attention à l'indentation, « Action 1 » est répétée plusieurs fois. Une fois le compte fini, on sort de la boucle.

L'instruction **range()** peut être codée de plusieurs façon :

- range(5) → 0 1 2 3 4
- range(3,8) → 3 4 5 6 7
- range(2,12,3) → 2 5 8 11
- range(20,5,-5) → 20 15 10

Exemple :

On veut allumer toutes les LED de la 3ème ligne avec une intensité lumineuse de 7.



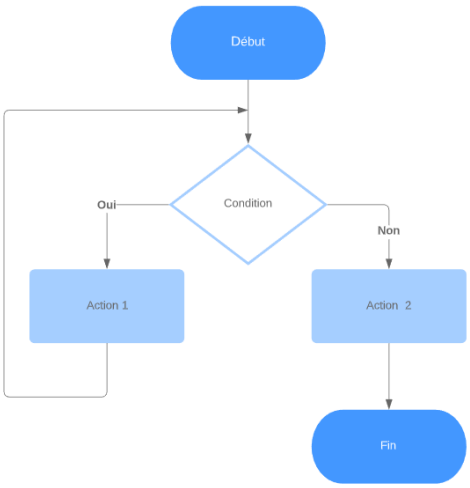
```
1 from microbit import *
2
3 for i in range(5):
4   display.set_pixel(i,2,7)
```

Application :

Écrire un script permettant d'obtenir un « plus » avec une intensité lumineuse de 4, en utilisant une boucle « for ».

While (Tant Que).

En algorithmique, les boucles « Tant Que » permettent **de répéter des instructions tant qu'une condition est vérifiée**.



Cela permet ainsi d'itérer des instructions sans que l'on connaisse le nombre de répétitions à effectuer.

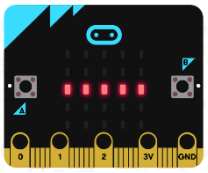
L'instruction permettant de coder un « Tant Que » est :

```
while Condition:
  Action 1

Action 2
```

L'indentation est très importante, « Action 1 » est dans la boucle, alors que « Action 2 » est la 1ère instruction au sortir de la boucle.

Exemple :

On veut allumer toutes les LED de la 3ème ligne avec une intensité lumineuse de 7.



```
1 from microbit import *
2 i=0     # on affecte la valeur 0 à la variable i
3 while i<5:
4   display.set_pixel(i,2,7)
5   i=i+1     # on incrémente la variable de 1
```

Application :

Écrire un script permettant d'obtenir un « plus » avec une intensité lumineuse de 4, en utilisant une boucle « while ».

Boucle infinie.

Pour exécuter des instructions une « infinité » de fois, on utilise une boucle dite « infinie » :

```
while True:
  Action 1
  Action 2
  ...
```

L'indentation est très importante, « Action 1 » est exécutée, puis « Action 2 » , etc…

L'instruction « while » vérifie si quelque chose est « True » pour déterminer s'il doit exécuter son corps. Puisque « True » est évidemment True tout le temps, on obtient une boucle infinie !

Conditions.

Dans de nombreux algorithmes, certaines instructions ne doivent être réalisées que sous certaines conditions, on utilise alors **des instructions conditionnelles**.

Les instructions conditionnelles se font en 2 ou 3 étapes.

- Étape 1 : Le test : on vérifie si une condition est réalisée.
- Étape 2 : L'instruction conditionnelle : il s'agit de l'instruction à effectuer si la condition est vérifiée.
- Étape 3 : Sinon : il s'agit de l'instruction à effectuer lorsque la condition n'est pas vérifiée.

Remarque :

Une condition est une expression logique dont le résultat est soit « vrai », soit « faux ».

L'étape 3 n'est pas toujours nécessaire lorsque l'on ne veut rien faire lorsque la condition n'est pas vérifiée.

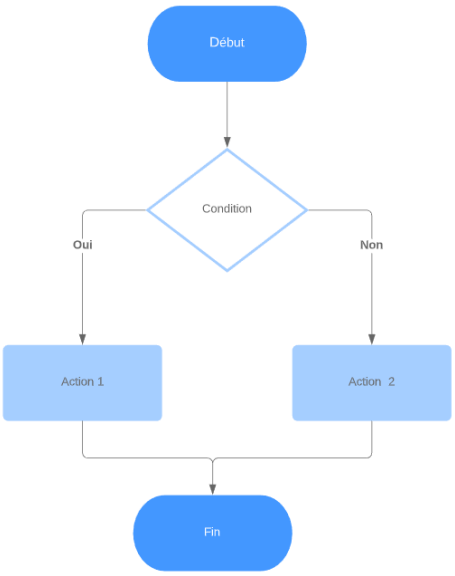
If – Else (Si – Sinon).

L'instruction pour tester si une condition est vraie est **if condition :**

Pour exécuter des instructions lorsque la condition n'est pas vérifiée, on ajoute une ligne avec l'instruction **else:**

Les lignes **if condition:** et **else:** se terminent obligatoirement par deux points et les instructions conditionnelles sont indentées d'un niveau.

```
if condition:
  Action 1
else:
  Action 2
```



Remarque :

On ne peut pas utiliser l'opérateur «=» pour effectuer des tests car il s'agit de l'opérateur pour l'affectation, on utilisera plutôt «==» pour tester une égalité.

Les opérateurs de comparaison que l'on utilisera en Python sont :

- < (inférieur)
- <= (inférieur ou égal)
- > (supérieur)
- >= (supérieur ou égal)
- == (égal)
- != (différent)
- in (contient)

Exemple :

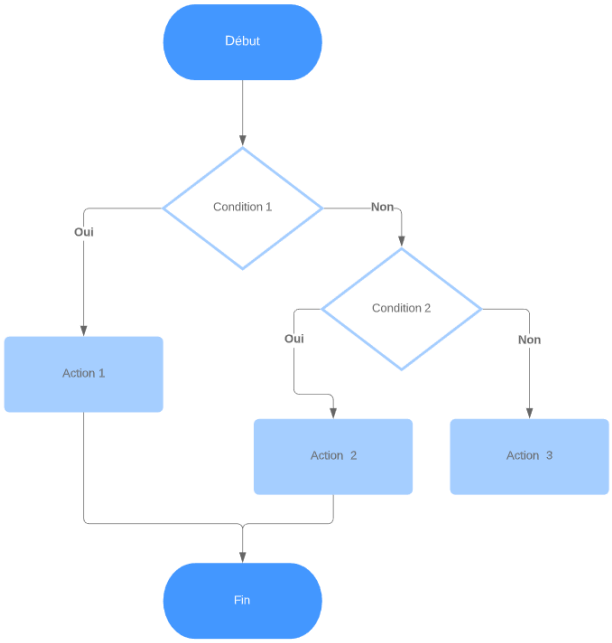
Que fait ce script ?

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

If – Elif – Else (Si – Sinon Si - Sinon).

On sera souvent amené à enchaîner les boucles conditionnelles. Pour cela, on utilise l'architecture suivante :

```
if condition 1:
  Action 1
elif Condition 2:
  Action 2
else:
  Action 3
```



Exemple :

Que fait ce script ?

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

Application :

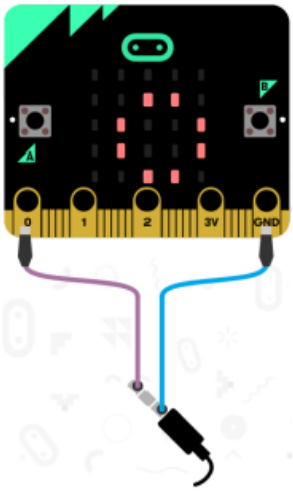
On souhaite réaliser un dé électronique en modélisant un dé cubique classique à 6 faces.

Cahier des charges :

- Afficher une image « SLEEP »
- Le dé se lance quand le bouton A est actionné
- Lancer une petite animation « DIAMOND_SMALL – DIAMOND » 10 fois avec un intervalle de 0,1 s
- Afficher la face tirée au hasard pendant 2 s

```
import random              # importation du module random
nbre=None                  # création de la variable nbre
nbre=random.randint(1,6)   # tire un entier au hasard entre 1 et 6 et l'affecte à nbre
```

Jouons de la musique !



Par exemple, pour connecter des enceintes ou une paire d'écouteurs à la carte, il suffit de deux câbles avec pinces crocodiles de la façon suivante :

Il faut avant tout importer la bibliothèque « music » : `import music`

Le module « music » contient les méthodes utilisées pour produire et contrôler le son.

Liste des mélodies native :

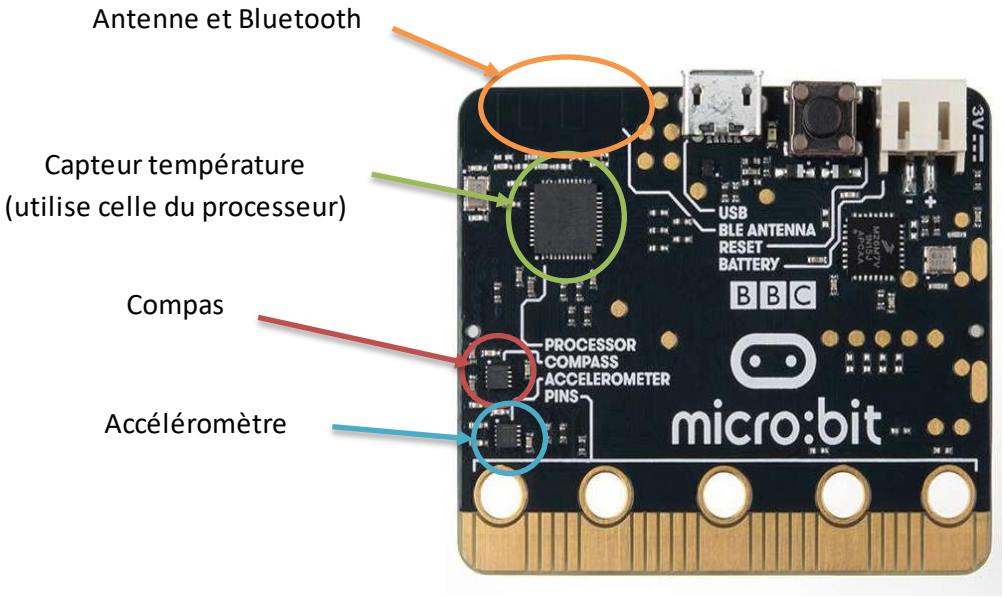
`DADADADUM` - the opening to Beethoven's 5th Symphony in C minor.
`ENTERTAINER` - the opening fragment of Scott Joplin's Ragtime classic « The Entertainer ».
`PRELUDE` - the opening of the first Prelude in C Major of J.S.Bach's 48 Preludes and Fugues.
`ODE` - the « Ode to Joy » theme from Beethoven's 9th Symphony in D minor.
`NYAN` - the Nyan Cat theme (http://www.nyan.cat/). The composer is unknown. This is fair use for educational porpoises (as they say in New York).
`RINGTONE` - something that sounds like a mobile phone ringtone. To be used to indicate an incoming message.
`FUNK` - a funky bass line for secret agents and criminal masterminds.
`BLUES` - a boogie-woogie 12-bar blues walking bass.
`BIRTHDAY` - « Happy Birthday to You… » for copyright status see : http://www.bbc.co.uk/news/world-us-canada-34332853
`WEDDING` - the bridal chorus from Wagner's opera « Lohengrin ».
`FUNERAL` - the « funeral march » otherwise known as Frédéric Chopin's Piano Sonata No. 2 in B minor, Op. 35.
`PUNCHLINE` - a fun fragment that signifies a joke has been made.
`PYTHON` - John Philip Sousa's march « Liberty Bell » aka, the theme for « Monty Python's Flying Circus » (after which the Python programming language is named).
`BADDY` - silent movie era entrance of a baddy.
`CHASE` - silent movie era chase scene.
`BA_DING` - a short signal to indicate something has happened.
`WAWAWAWAA` - a very sad trombone.
`JUMP_UP` - for use in a game, indicating upward movement.
`JUMP_DOWN` - for use in a game, indicating downward movement.
`POWER_UP` - a fanfare to indicate an achievement unlocked.
`POWER_DOWN` - a sad fanfare to indicate an achievement lost.

Exemple de script :

```
1 from microbit import *
2 import music
3
4 music.play(music.NYAN)
```

Pourquoi ne pas faire chanter « Joyeux anniversaire » à votre carte microbit ? Ou «La Marche funèbre » ? Ou encore «L'Ode à la joie » ?

Capteurs internes de la carte microbit.



Température.

Le capteur température est celui du processeur.

Le modèle de script à utiliser est du type :

```
from microbit import *

while True:
  display.scroll("T=")
  display.scroll(temperature())
```

Application :

Réaliser une jauge qui se remplira avec la température (25, 26, 27, 28, 29, 30, Youpi !)

Capteur de lumière.

Le capteur de luminosité est basé sur la matrice LED

Le modèle de script à utiliser est du type :

```
from microbit import *

while True:
  display.scroll("L=")
  display.scroll(display.read_light_level())
```

Application :

Réaliser une jauge qui se remplira avec la luminosité (de 0 à 255).

Accéléromètre – Magnétomètre.

La carte microbit possède un accéléromètre. Il mesure le mouvement selon trois axes :

- X - l'inclinaison de gauche à droite.
- Y - l'inclinaison d'avant en arrière.
- Z - le mouvement haut et bas.

Objet	Méthode	Description
accelerometer.	get_x() get_y() get_z()	Donne la mesure de l'alignement selon les axes x, y ou z.
	current_gesture()	Renvoie l'un des gestes : • haut→ up • bas → down • gauche → left • droite → right • face en haut → face up • face en bas → face down • chute libre → free fall • accélération correspondant à 3, 6 ou 8 fois celle de la chute libre → 3g, 6g ou 8g • secousse → shake

Il y a une méthode pour chaque axe, qui renvoie un nombre positif ou négatif et indique une mesure en milli-g.

Lorsque la lecture est de 0, la carte est « alignée » selon cet axe.

Par exemple, voici un « niveau à bulle » très simple qui utilise l'objet « accelerometer » avec la méthode « get_x » pour mesurer l'alignement de l'appareil selon l'axe X :

```
1  from microbit import *
2
3  while True:
4    lecture = accelerometer.get_x()
5    if lecture > 20:
6      display.show("D")
7    elif lecture < -20:
8      display.show("G")
9    else:
10     display.show("-")
```

Gestes.

L'effet le plus intéressant d'un accéléromètre est la détection des gestes.

Les gestes sont toujours représentés par des chaînes de caractères.

Exemple :

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

Encore une fois, puisque nous voulons que l'appareil réagisse à des circonstances changeantes, nous utilisons une boucle **while**. À l'intérieur du corps de la boucle, le geste est lu et stocké dans la variable **geste**. L'instruction conditionnelle **if** vérifie si **geste** est égal à **face up** (Python utilise « == » pour tester une égalité car un simple signe égal « = » est utilisé pour l'affectation - tout comme lorsque nous affectons le geste lu à l'objet geste). Si le **geste** est égal à **face up** alors on utilise l'affichage pour montrer un visage heureux. Sinon, l'appareil a l'air mécontent.

Direction.

Pour réaliser une girouette, on peut utiliser la boussole (le compas) de la carte microbit.

Objet	Méthode	Description
compas.	calibrate()	Calibre le compas. (Bouger dans tous les sens la microbit pour le réaliser)
	heading()	Donne la direction.

Exemple :

Le script suivant permet d'indiquer le Nord.

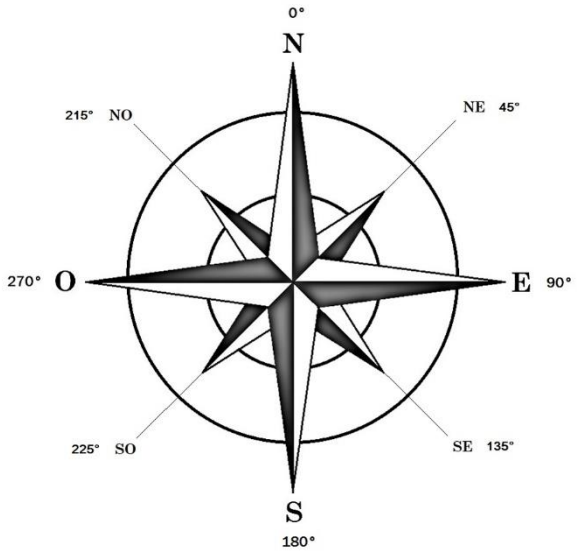
```
from microbit import *

compass.calibrate()      # calibre la boussole avant utilistion
                         # pour cela il faut la bouger selon tous les axes

while True:
  aiguille = ((15 - compass.heading()) // 30) % 12  # division entière (//) avec modulo %
  display.show(Image.ALL_CLOCKS[aiguille])
```

Application :

Chaque orientation correspond à une mesure angulaire d'un cercle (en degré). Le Nord correspondant à 0°.
Observer les directions de la rose des vents et les angles associés.

Boussole à 4 directions :

```
from microbit import *
compass.calibrate()
Direction = None

while True:
  Direction = compass.heading()
  if Direction > 215 or Direction <= 45:
    display.show(Image.ARROW_N)
  elif Direction > 45 and Direction <= 135:
    display.show(Image.ARROW_E)
  elif Direction > 135 and Direction <= 225:
    display.show(Image.ARROW_S)
  else:
    display.show(Image.ARROW_W)
  sleep(100)
```



Application :

Réaliser une boussole à 8 directions.

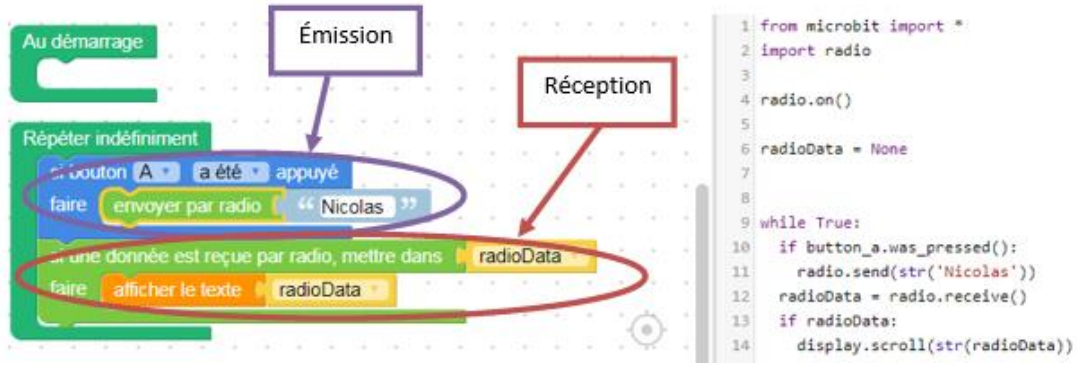
Communication entre 2 cartes microbit.

Le microprocesseur qui a permis l'essor de la technologie microbit est le Nordic SemiConductor nrf51822. À l'origine, ce semi-conducteur était destiné à la communication entre microcontrôleurs à haute fréquence et à moindre coût. Ce n'est que plusieurs années après l'exploitation de ce composant que sa versatilité et sa légèreté en ont fait un champion dans le domaine éducatif.

Ainsi, par construction, toutes les cartes microbit disposent d'un émetteur radio à bande étroite réglable entre 2400 et 2499 Mhz sur une portée de plusieurs dizaines de mètres en extérieur. Il est à noter qu'aucun protocole de sécurité ou d'adressage n'est utilisé par le processeur. En d'autres termes, toutes les cartes microbit peuvent communiquer naturellement entre elles en utilisant le même protocole public.

Pour le projet de ballon solaire, la communication radio est très intéressante car elle permet de monitorer en direct les mesures faites par les capteurs à distance. Ainsi, avec une microbit au sol et une microbit dans le ballon captif, il est possible d'afficher la pression, la température, l'humidité, ou la luminosité au fur et à mesure qu'il s'élève.

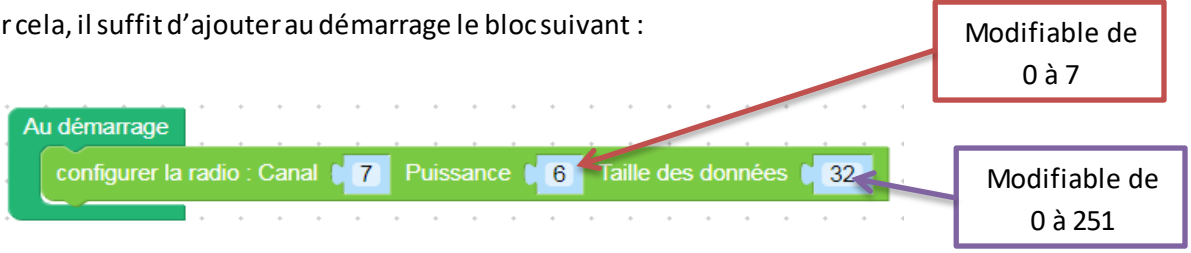
L'utilisation des bibliothèques "radio" se fait très simplement, comme le montre l'exemple ci-dessous :



Dans cet exemple, l'élève Nicolas a programmé sa carte microbit pour qu'elle émette son prénom. Les autres élèves auront fait de même mais en utilisant leur propre prénom. Ainsi, à chaque action du bouton A, le signal se répand dans la classe et on peut identifier l'émetteur à son prénom qui s'affiche sur l'écran ! La carte est programmée à la fois en émission et en réception.

Portée de la radio.

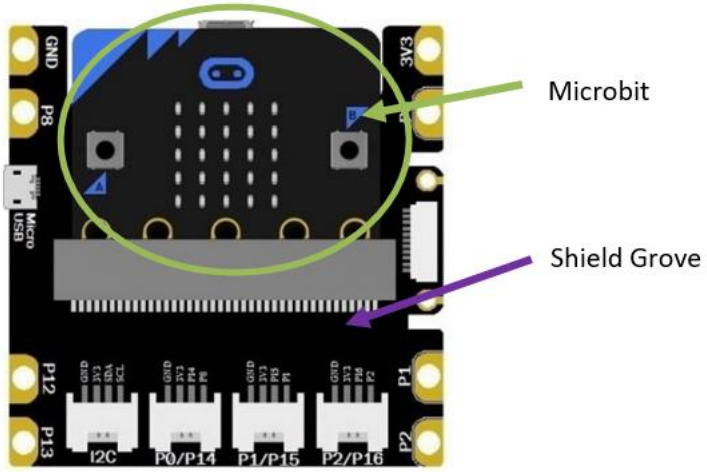
Il est également possible de paramétrer la puissance de l'émission/transmission de la radio. La puissance joue sur la distance maximale entre les cartes où l'on peut recevoir des données. Il est possible de définir la puissance de 0 à 7 (inclus), sachant qu'une augmentation de puissance augmente la portée mais aussi la consommation énergétique de la carte. Pour cela, il suffit d'ajouter au démarrage le bloc suivant :



Application : Utiliser une batterie externe pour alimenter une des 2 cartes microbit. Éloignez-vous et tenter régulièrement d'envoyer des données. Au bout de combien de mètre la transmission ne s'effectue plus, c'est-à-dire que la carte n'affiche plus la chaîne de caractères ?

Capteurs externes (Grove).

L'utilisation des capteurs externes nécessite l'ajout du Shield Grove pour microbit et éventuellement du Hub I2C (ajout de plusieurs capteurs I2C).

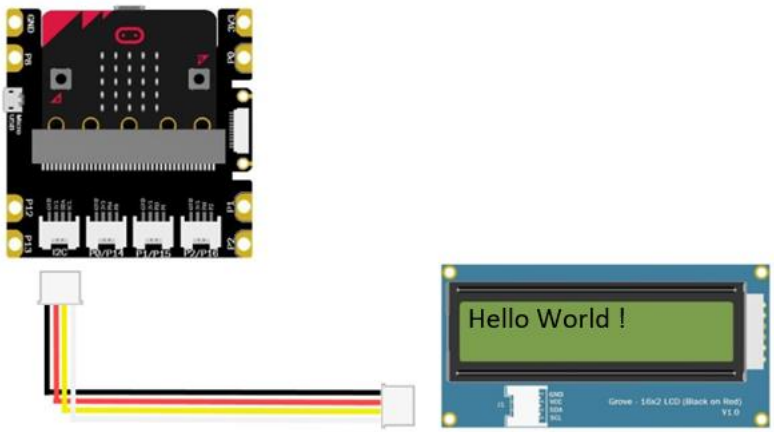


Pour les coder, on utilise toujours la solution ***Objet.Méthode*** en important les bibliothèques

	Objet	Méthode	Description
Écran LCD from lcd_i2c import LCD1602	lcd.	setCursor(***c***,***l***)	Place le curseur à la colonne ***c*** et à la ligne ***l***
		writeTxt(« ***texte*** »)	Affiche le « ***texte*** »
		clear()	Efface l'écran
Capteur T, p et h from bmp280 import BMP280	bmp280.	Temperature()	Lit la température en °C du capteur
		Pressure()	Lit la pression en Pa du capteur
		Altitude()	Donne l'altitude en m par différence de pression (nécessite la donnée de h0)
Capteur CO_2 et COV from sgp30 import SGP30	sgp30.	eCO2()	Lit le taux de CO_2 en ppm
		TVOC()	Lit le taux de COV en ppb

Écran LCD.

L'écran LCD se branche sur la broche I2C du Shield.



Pour le codage de l'écran LCD, il faut importer le module « LCD1602 » de la bibliothèque « lcd_i2c ».

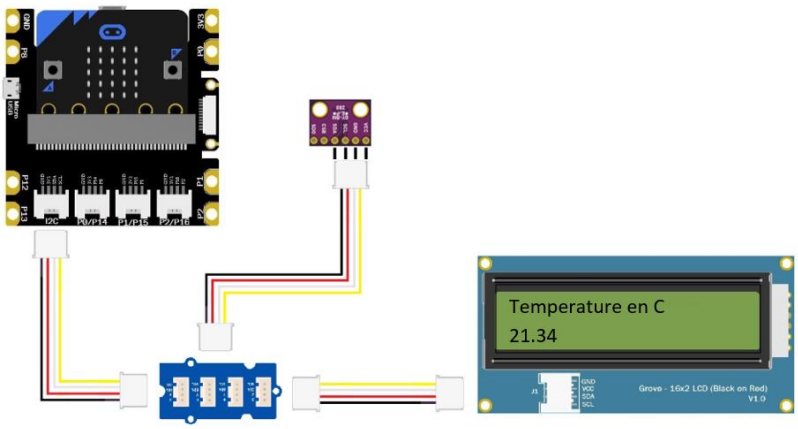
Écrire le script suivant :

```
from microbit import *
from lcd_i2c import LCD1602     # importation du module LCD1602

lcd = LCD1602()         # création de la variable lcd


while True:
  lcd.setCursor(0,0)        # on place le curseur (colonne 0,ligne 0)
  lcd.writeTxt("Hello World !")
```

Dans la suite du document, on utilisera l'écran LCD pour faciliter la lecture des données et le Hub I2C pour connecter l'écran et les capteurs.

Capteur Température – Pression – Altitude : BMP 280.

Le capteur se branche en I2C. Il mesure la température en °C (au dixième près), la pression en Pa et l'altitude en m (à l'aide de la variation de la pression h – h0, où h0 est la hauteur calculée avec la pression à t = 0).



Pour le codage du capteur BMP 280, il faut importer le module « BMP280 » de la bibliothèque « bmp280 ».

Écrire le script suivant :

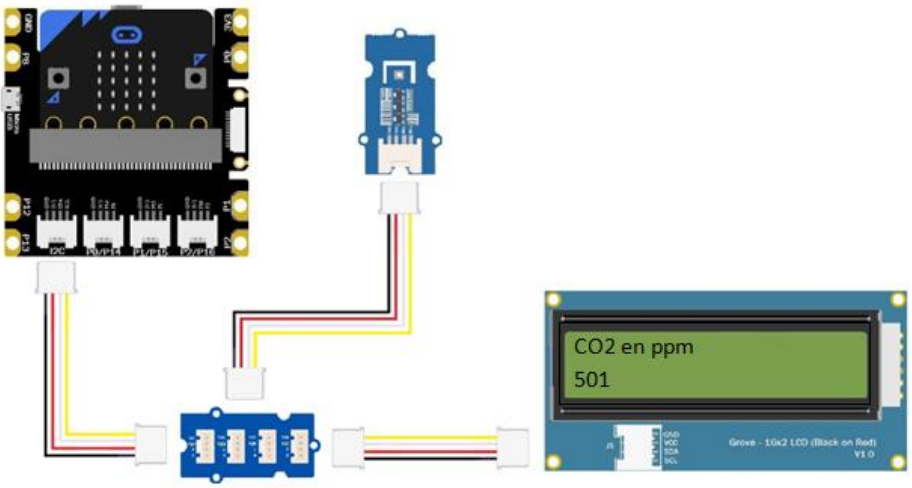
```
from microbit import *
from lcd_i2c import LCD1602     # importation de l'écran
from bmp280 import BMP280       # importation du capteur

lcd = LCD1602()                 # création de la variable lcd
bmp280 = BMP280(0x76)           # création de la variable bmp280


while True:
  lcd.setCursor(0,0)            # curseur en (colonne 0,ligne 0)
  lcd.writeTxt("Temperature en C")
  lcd.setCursor(0,1)            # curseur en (colonne 0,ligne 1)
  lcd.writeTxt(str(bmp280.Temperature()))   # affiche la valeur de la température convertie en chaîne de caractères
```

Application :

Écrire un script pour afficher la température sur la 1ère ligne (avec l'unité) et la pression sur la 2ème ligne (avec l'unité).

Capteur Qualité de l'air (CO_2 – COV) : SGP 30.

Le capteur se branche en I2C. Il mesure le taux de CO_2 en ppm (parts per million) et le taux de COV (Composés organiques volatils) en ppb (parts per billion).



Pour le codage du capteur SGP30, il faut importer le module « SGP30 » de la bibliothèque « sgp30 ».

Écrire le script suivant :

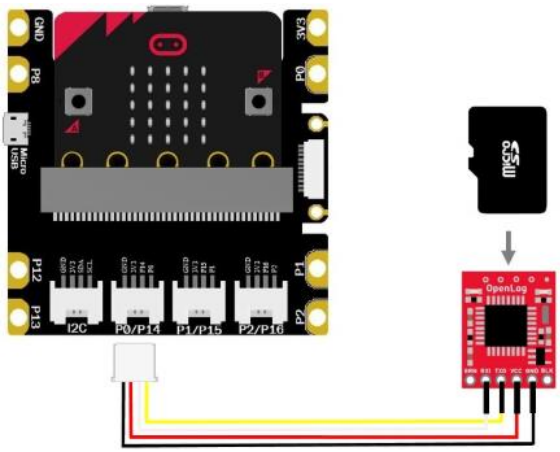
```
from microbit import *
from lcd_i2c import LCD1602
from sgp30 import SGP30

lcd=LCD1602()
sgp30=SGP30()

while True:
  lcd.setCursor(0,0)
  lcd.writeTxt("CO2 en ppm")
  lcd.setCursor(0,1)
  lcd.writeTxt(str(sgp30.eCO2()))
  sleep(1000)
```

Application :

Écrire un script pour afficher le taux de CO_2 sur la 1ère ligne (avec l'unité) et le taux de COV sur la 2ème ligne (avec l'unité).

Enregistrement sur µSD et récupération des données.

Le module Openlog permet d'enregistrer des données au format texte sur une carte µSD. On pourra ainsi les récupérer et les analyser avec un tableur type Excel. Le fichier généré est un fichier .txt enregistré sous un nom commençant par LOG…



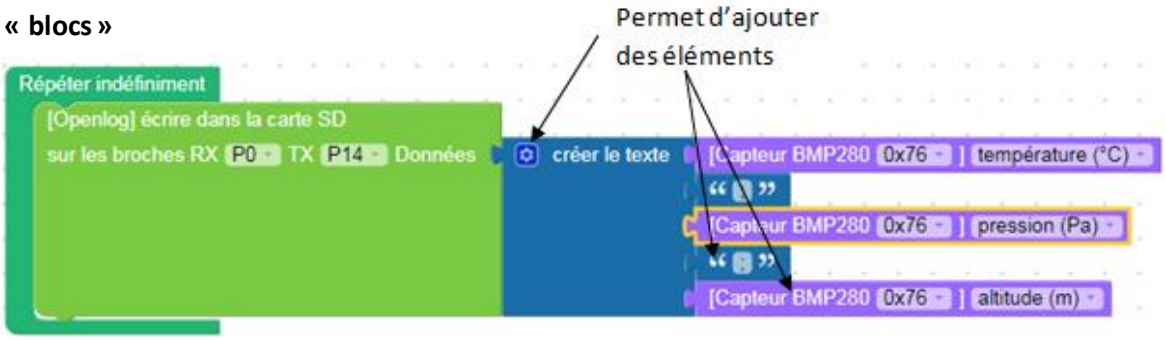
Enregistrement sur µSD.

Le code en langage Python est assez compliqué. Il est toutefois possible de le retrouver en passant en mode « combiné » en en programmant en blocs.

Exemple :

On veut enregistrer la température, la pression et l'altitude à l'aide du capteur BMP280 (branché sur le port I2C) toutes les secondes.

En « blocs »



En « Python »

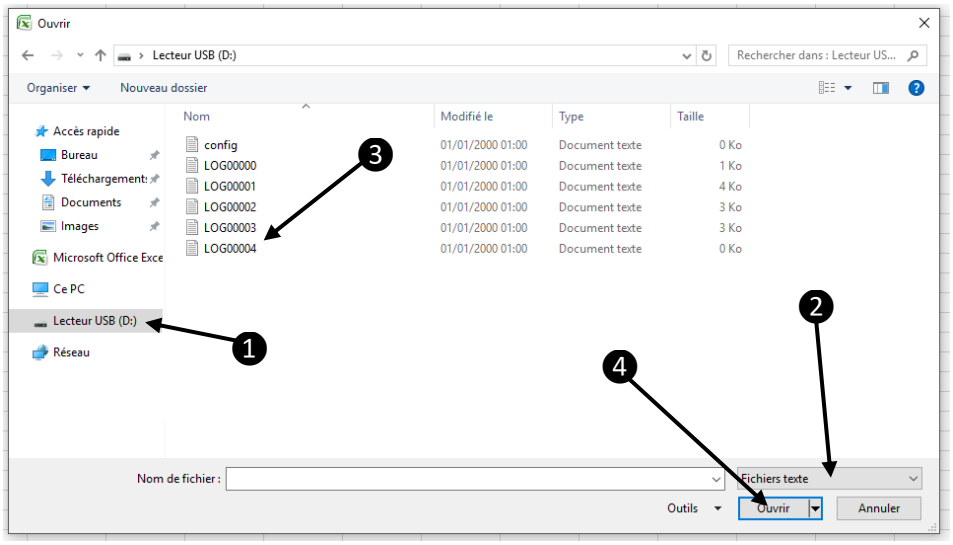
```
from microbit import *
from bmp280 import BMP280

bmp280 = BMP280(0x76)

uart.init(baudrate=4800, tx=pin14, rx=pin0)
h0 = bmp280.Altitude()


while True:
  uart.write(''.join([str(x) for x in [bmp280.Temperature(), ';', bmp280.Pressure(), ';', bmp280.Altitude()-h0]]) + '\n')
```

Récupération des données.

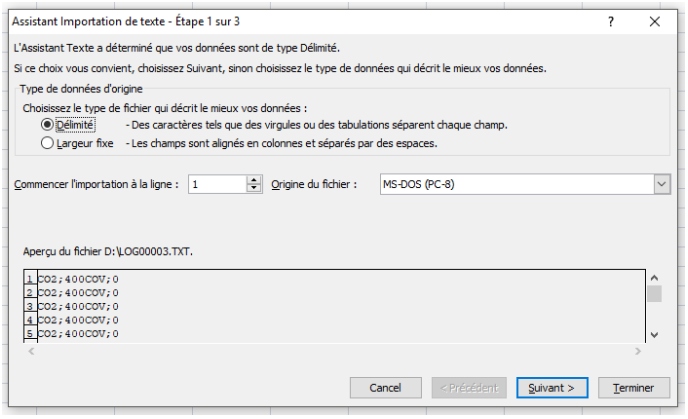
Il est possible de récupérer les données afin de les analyser dans Excel. Il faut, pour cela, ouvrir le fichier texte (.txt) généré lors de l'enregistrement des données.

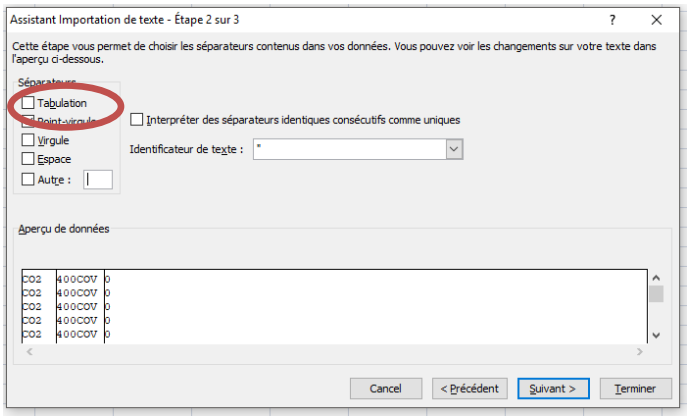
Une fois Excel ouvert :

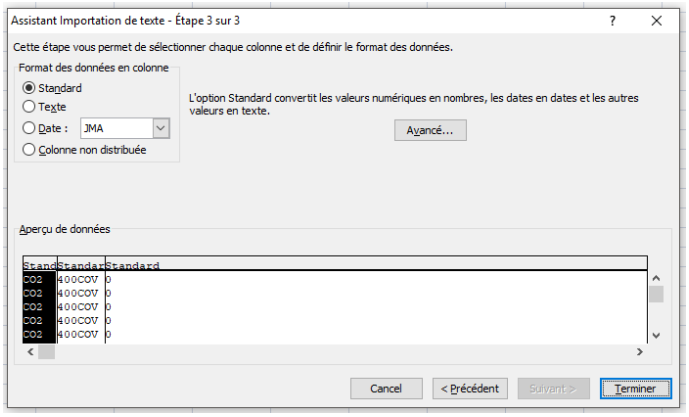
- Fichier, Ouvrir
- Choisir le lecteur USB ❶, le format « texte » ❷ et le dernier fichier LOG… généré ❸



- Suivre les étapes de l'assistant

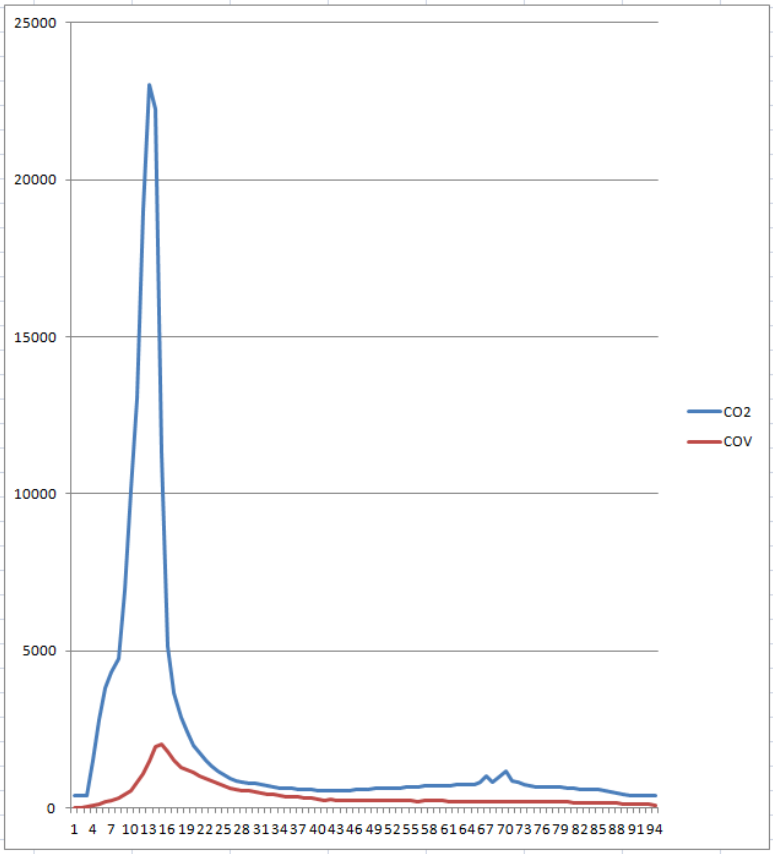






- Il ne reste plus qu'à arranger le tableau et à effectuer les représentations graphiques souhaitées.

	A	B
1	CO2	COV
2	400	0
3	400	5
4	400	33
5	1524	86
6	2803	138
7	3839	188
8	4329	236
9	4742	318
10	6946	417
11	10120	555
12	13048	803
13	19017	1107
14	23042	1503
15	22256	1951
16	11308	2032
17	5140	1780
18	3677	1517
19	2894	1306
20	2413	1229
21	1990	1124
22	1754	1008
23	1519	948
24	1315	855
25	1177	784
26	1056	711
27	932	639
28	860	577
29	820	557
30	788	530
31	766	492
32	726	466
33	699	447
34	663	423
35	646	397
36	615	372
37	619	359
38	598	340
39	584	321
40	574	299



Application :

Réaliser la partie électronique de la nacelle, pour relever les températures, pressions, altitude, taux de CO_2 et COV. Ces données seront transmises à une 2ème carte microbit via une liste et on visualisera les données sur la console graphique.

Memento Python

Mémento Bases Python 3

Documentation officielle Python sur http://docs.python.org/py3k

Types de base

entier, flottant, booléen, chaîne

```
  int  783    0    -192
float  9.23   0.0   -1.7e-6        (10^-6)
 bool  True   False
  str  "Un\nDeux"   'L\'âme'
```

retour à la ligne — ' échappé

multiligne `"""X\tY\tZ 1\t2\t3"""` — tabulation

non modifiable, séquence ordonnée de caractères

Types Conteneurs

- séquences ordonnées, accès index rapide, valeurs répétables

```
  list  [1,5,9]   ["x",11,8.9]   ["mot"]      []
 tuple  (1,5,9)   11,"y",7.4     ("mot",)     ()
```

non modifiable — expression juste avec des virgules

`str` en tant que séquence ordonnée de caractères

- sans ordre *a priori*, clé unique, accès par clé rapide ; clés = types de base ou tuples

```
  dict  {"clé":"valeur"}                                  {}
```

dictionnaire *couples clé/valeur* `{1:"un",3:"trois",2:"deux",3.14:"π"}`

ensemble

```
   set  {"clé1","clé2"}   {1,9,3,0}   set()
```

Identificateurs

pour noms de variables, fonctions, modules, classes…

`a..zA..Z_` suivi de `a..zA..Z_0..9`

- accents possibles mais à éviter
- mots clés du langage interdits
- distinction casse min/MAJ

☺ `a toto x7 y_max BigOne`
☹ ~~`8y and`~~

Affectation de variables

```
x = 1.2+8+sin(0)
```

valeur ou expression de calcul — nom de variable (identificateur)

```
y,z,r = 9.2,-7.6,"bad"
```

noms de variables — conteneur de plusieurs valeurs (ici un tuple)

`x+=3` incrémentation — décrémentation `x-=2`

`x=None` valeur constante « non défini »

Conversions

`type(expression)`

`int("15")` on peut spécifier la base du nombre entier en 2nd paramètre

`int(15.56)` troncature de la partie décimale (`round(15.56)` pour entier arrondi)

`float("-11.24e8")`

`str(78.3)` et pour avoir la représentation littérale → `repr("Texte")`

voir au verso le formatage de chaînes, qui permet un contrôle fin

`bool` → utiliser des comparateurs (avec `==`, `!=`, `<`, `>`, …), résultat logique booléen

`list("abc")` → *utilise chaque élément de la séquence en paramètre* → `['a','b','c']`

`dict([(3,"trois"),(1,"un")])` → `{1:'un',3:'trois'}`

`set(["un","deux"])` → *utilise chaque élément de la séquence en paramètre* → `{'un','deux'}`

`":".join(['toto','12','pswd'])` → `'toto:12:pswd'`

chaîne de jointure — séquence de chaînes

`"des mots espacés".split()` → `['des','mots','espacés']`

`"1,4,8,2".split(",")` → `['1','4','8','2']`

chaîne de séparation

Indexation des séquences

pour les listes, tuples, chaînes de caractères,…

```
index négatif   -6   -5   -4   -3   -2   -1
index positif    0    1    2    3    4    5
lst=[11, 67, "abc", 3.14, 42, 1968]
tranche positive 0    1    2    3    4    5    6
tranche négative -6   -5   -4   -3   -2   -1
```

`len(lst)` → `6`

accès individuel aux éléments par `[index]`

`lst[1]` → `67` `lst[0]` → `11` *le premier*

`lst[-2]` → `42` `lst[-1]` → `1968` *le dernier*

accès à des sous-séquences par `[tranche début:tranche fin:pas]`

`lst[:-1]` → `[11,67,"abc",3.14,42]`
`lst[1:-1]` → `[67,"abc",3.14,42]`
`lst[::2]` → `[11,"abc",42]`
`lst[:]` → `[11,67,"abc",3.14,42,1968]`

`lst[1:3]` → `[67,"abc"]`
`lst[-3:-1]` → `[3.14,42]`
`lst[:3]` → `[11,67,"abc"]`
`lst[4:]` → `[42,1968]`

Indication de tranche manquante → à partir du début / jusqu'à la fin.

Sur les séquences modifiables, utilisable pour suppression `del lst[3:5]` *et modification par affectation* `lst[1:4]=['hop',9]`

Logique booléenne

Comparateurs: `<` `>` `<=` `>=` `==` `!=` (≤ ≥ = ≠)

`a and b` et logique — *les deux en même temps*

`a or b` ou logique — *l'un ou l'autre ou les deux*

`not a` non logique

`True` valeur constante vrai

`False` valeur constante faux

Blocs d'instructions

```
instruction parente:
    bloc d'instructions 1…
    …
instruction parente:
    bloc d'instructions 2…
    …
instruction suivante après bloc 1
```

indentation !

Instruction conditionnelle

bloc d'instructions exécuté uniquement si une condition est vraie

```
if expression logique:
    bloc d'instructions
```

combinable avec des sinon si, sinon si... et un seul sinon final, exemple :

```
if x==42:
    # bloc si expression logique x==42 vraie
    print("vérité vraie")
elif x>0:
    # bloc sinon si expression logique x>0 vraie
    print("positivons")
elif bTermine:
    # bloc sinon si variable booléenne bTermine vraie
    print("ah, c'est fini")
else:
    # bloc sinon des autres cas restants
    print("ça veut pas")
```

Maths

nombres flottants... valeurs approchées ! — angles en radians

Opérateurs: `+` `-` `*` `/` `//` `%` `**` (× ÷ ↑ ↑ a^b)

÷ entière — reste ÷

`(1+5.3)*2` → `12.6`
`abs(-3.2)` → `3.2`
`round(3.57,1)` → `3.6`

`from math import sin,pi…`
`sin(pi/4)` → `0.707…`
`cos(2*pi/3)` → `-0.4999…`
`acos(0.5)` → `1.0471…`
`sqrt(81)` → `9.0` ✓
`log(e**2)` → `2.0` *etc. (cf doc)*

Instruction boucle conditionnelle

bloc d'instructions exécuté tant que la condition est vraie

```
while expression logique :
    bloc d'instructions
```

```
s = 0
i = 1
```
*initialisations **avant** la boucle*

condition avec au moins une valeur variable (ici `i`)

```
while i <= 100:
    # bloc exécuté tant que i ≤ 100
    s = s + i**2
    i = i + 1
```
faire varier la variable de condition !

```
print("somme:",s)
```
résultat de calcul après la boucle

attention aux boucles sans fin !

$$s = \sum_{i=1}^{i=100} i^2$$

Contrôle de boucle

`break` *sortie immédiate*

`continue` *itération suivante*

Instruction boucle itérative

bloc d'instructions exécuté pour chaque élément d'un conteneur ou d'un itérateur

```
for variable in séquence :
    bloc d'instructions
```

Parcours des **valeurs** de la séquence

```
s = "Du texte"
cpt = 0
```
*initialisations **avant** la boucle*

variable de boucle, valeur gérée par l'instruction `for`

```
for c in s:
    if c == "e":
        cpt = cpt + 1
print("trouvé",cpt,"'e'")
```
Comptage du nombre de `e` dans la chaîne.

boucle sur dict/set = boucle sur séquence des clés

utilisation des tranches pour parcourir un sous-ensemble de la séquence

Parcours des **index** de la séquence

- changement de l'élément à la position
- accès aux éléments autour de la position (avant/après)

```
lst = [11,18,9,12,23,4,17]
perdu = []
for idx in range(len(lst)):
    val = lst[idx]
    if val> 15:
        perdu.append(val)
        lst[idx] = 15
print("modif:",lst,"-modif:",perdu)
```
Bornage des valeurs supérieures à 15, mémorisation des valeurs perdues.

Parcours simultané **index** et **valeur** de la séquence:

```
for idx,val in enumerate(lst):
```

Affichage / Saisie

```
print("v=",3,"cm :",x,",",y+4)
```

éléments à afficher : valeurs littérales, variables, expressions

Options de `print`:

- `sep=" "` (séparateur d'éléments, défaut espace)
- `end="\n"` (fin d'affichage, défaut fin de ligne)
- `file=f` (print vers fichier, défaut sortie standard)

```
s = input("Directives:")
```

`input` retourne toujours une **chaîne**, la convertir vers le type désiré (cf encadré *Conversions* au recto).

Opérations sur conteneurs

`len(c)` → nb d'éléments

`min(c)` `max(c)` `sum(c)`

`sorted(c)` → *copie* triée

*Note: Pour dictionnaires et ensembles, ces opérations travaillent sur les **clés**.*

`val in c` → booléen, opérateur `in` de test de présence (`not in` d'absence)

`enumerate(c)` → *itérateur* sur (index,valeur)

*Spécifique aux **conteneurs de séquences** (listes, tuples, chaînes) :*

`reversed(c)` → *itérateur* inversé `c*5` → duplication `c+c2` → concaténation

`c.index(val)` → position `c.count(val)` → nb d'occurences

Opérations sur listes

modification de la liste originale

`lst.append(item)`	ajout d'un élément à la fin
`lst.extend(seq)`	ajout d'une séquence d'éléments à la fin
`lst.insert(idx,val)`	insertion d'un élément à une position
`lst.remove(val)`	suppression d'un élément à partir de sa valeur
`lst.pop(idx)`	suppression de l'élément à une position et retour de la valeur
`lst.sort()` `lst.reverse()`	tri / inversion de la liste *sur place*

Opérations sur dictionnaires

`d[clé]=valeur` `d.clear()`

`d[clé]`→*valeur* `del d[clé]`

`d.update(d2)` *mise à jour/ajout des couples*

`d.keys()` `d.values()` `d.items()` *vues sur les clés, valeurs, couples*

`d.pop(clé)`

Opérations sur ensembles

Opérateurs:

`|` → union (caractère barre verticale)

`&` → intersection

`-` `^` → différence/diff symétrique

`<` `<=` `>` `>=` → relations d'inclusion

`s.update(s2)`

`s.add(clé)` `s.remove(clé)`

`s.discard(clé)`

Génération de séquences d'entiers

très utilisé pour les boucles itératives `for`

```
range([début,] fin [,pas])
```
par défaut 0 — non compris

```
range(5)      → 0 1 2 3 4
range(3,8)    → 3 4 5 6 7
range(2,12,3) → 2 5 8 11
```

`range` retourne un « générateur », faire une conversion en liste pour voir les valeurs, par exemple:

```
print(list(range(4)))
```

Définition de fonction

nom de la fonction (identificateur) — paramètres nommés

```
def nomfct(p_x,p_y,p_z):
    """documentation"""
    # bloc instructions, calcul de res, etc.
    return res
```
valeur résultat de l'appel. si pas de résultat calculé à retourner : `return None`

les paramètres et toutes les variables de ce bloc n'existent que *dans* le bloc et *pendant* l'appel à la fonction (« *boite noire* »)

Appel de fonction

```
r = nomfct(3,i+2,2*i)
```
un argument par paramètre

récupération du résultat retourné (si nécessaire)

Fichiers

stockage de données sur disque, et relecture

```
f = open("fic.txt","w",encoding="utf8")
```

variable fichier pour les opérations

nom du fichier sur le disque (+chemin…)

mode d'ouverture
- `'r'` lecture (read)
- `'w'` écriture (write)
- `'a'` ajout (append)…

encodage des caractères pour les fichiers textes: `utf8` `ascii` `latin1` …

cf fonctions des modules `os` et `os.path`

en **écriture**

```
f.write("coucou")
```
*fichier texte → lecture / écriture de **chaînes** uniquement, convertir de/vers le type désiré*

en **lecture**

```
s = f.read(4)
```
chaîne vide si fin de fichier

si nb de caractères pas précisé, lit tout le fichier

```
s = f.readline()
```
lecture ligne suivante

```
f.close()
```
ne pas oublier de refermer le fichier après son utilisation !

Fermeture automatique Pythonesque : `with open(…) as f:`

très courant : boucle itérative de lecture des lignes d'un fichier texte :

```
for ligne in f :
    bloc de traitement de la ligne
```

Formatage de chaînes

directives de formatage — valeurs à formater

```
"modele{} {} {}".format(x,y,r) → str
"{sélection:formatage!conversion}"
```

- **Sélection** :
  - `2`
  - `x`
  - `0.nom`
  - `4[clé]`
  - `0[2]`
- **Formatage** :

Exemples:
```
"{:+2.3f}".format(45.7273)
→'+45.727'
"{1:>10s}".format(8,"toto")
→'      toto'
"{!r}".format("L'ame")
→'"L\'ame"'
```

car-rempl. alignement signe larg.mini.précision~larg.max type

`< > ^ =` `+ - espace` `0` au début pour remplissage avec des 0

entiers: `b` binaire, `c` caractère, `d` décimal (défaut), `o` octal, `x` ou `X` hexa…

flottant: `e` ou `E` exponentielle, `f` ou `F` point fixe, `g` ou `G` approprié (défaut)

`%` pourcentage

chaîne : `s` …

- **Conversion** : `s` (texte lisible) ou `r` (représentation littérale)

Annexes

Flèches de boussole tournantes :

```
from microbit import *

while True:
  display.show(Image.ALL_ARROWS,delay=200)
  pass
```

Bateau qui coule :

```
from microbit import *

Boat1=Image("00700:00770:00700:99999:09990")
Boat2=Image("00000:00700:00770:00700:99999")
Boat3=Image("00000:00000:00700:00770:00700")
Boat4=Image("00000:00000:00000:00700:00770")
Boat5=Image("00000:00000:00000:00000:00700")
Boat6=Image("00000:00000:00000:00000:00000")

All_Boat=[Boat1,Boat2,Boat3,Boat4,Boat5,Boat6]
display.scroll("les Gaugau !!!")
display.show(All_Boat,delay=1000)
```

For :

```
from microbit import *

for i in range(5):
  display.set_pixel(i,2,4)
  display.set_pixel(2,i,4)
```

While :

```
from microbit import *
i=0     # on affecte la valeur 0 à la variable i
while i<5:
  display.set_pixel(i,2,4)
  display.set_pixel(2,i,4)
  i=i+1     # on incrémente la variable de 1
```

Lancer de dé :

```
from microbit import *
import random

nbre = None
image0 = Image("00000:99099:00000:09990:00000")
image1 = Image("00000:00000:00900:00000:00000")
image2 = Image("00000:09000:00000:00090:00000")
image3 = Image("00000:09000:00900:00090:00000")
image4 = Image("00000:09090:00000:09090:00000")
image5 = Image("00000:09090:00900:09090:00000")
image6 = Image("00000:09090:09090:09090:00000")

while True:
  display.show(image0)
  display.show("A pour Lancer")
  if button_a.is_pressed():
    nbre = random.randint(1, 6)
    for i in range(10):
      display.show(Image.DIAMOND_SMALL)
      sleep(100)
      display.show(Image.DIAMOND)
      sleep(100)
    sleep(500)
    if nbre == 1:
      display.show(image1)
    elif nbre == 2:
      display.show(image2)
    elif nbre == 3:
      display.show(image3)
    elif nbre == 4:
      display.show(image4)
    elif nbre == 5:
      display.show(image5)
    elif nbre == 6:
      display.show(image6)
    sleep(2000)
```

Boussole à 8 directions :

```
from microbit import *

Direction = None


while True:
  Direction = compass.heading()
  if Direction > 337.5 or Direction <= 22.5:
    display.show(Image.ARROW_N)
  elif Direction > 22.5 and Direction <= 67.5:
    display.show(Image.ARROW_NE)
  elif Direction > 67.5 and Direction <= 112.5:
    display.show(Image.ARROW_E)
  elif Direction > 112.5 and Direction <= 157.5:
    display.show(Image.ARROW_SE)
  elif Direction > 157.5 and Direction <= 202.5:
    display.show(Image.ARROW_S)
  elif Direction > 202.5 and Direction <= 247.5:
    display.show(Image.ARROW_SW)
  elif Direction > 247.5 and Direction <= 292.5:
    display.show(Image.ARROW_W)
  elif Direction > 292.5 and Direction <= 337.5:
    display.show(Image.ARROW_NW)
  else:
    pass
  sleep(0)
```

Jauge de température :

```
from microbit import *

# Création des images pour la jauge
image0=Image.ASLEEP             # Affiche un endormi
image1=Image("00000:00000:00000:00000:99999")
image2=Image("00000:00000:00000:99999:66666")
image3=Image("00000:00000:99999:66666:44444")
image4=Image("00000:99999:66666:44444:22222")
image5=Image("99999:66666:44444:22222:22222")

while True:
  temp=temperature()            # Lecture du capteur température de la carte (variable temp)
  if temp<=25:                  # Affichage de la jauge en fonction de la température
    display.show(image0)
  elif 25<temp<=26:
    display.show(image1)
  elif 26<temp<=27:
    display.show(image2)
  elif 27<temp<=28:
    display.show(image3)
  elif 28<temp<=29:
    display.show(image4)
  elif 29<temp<=30:
    display.show(image5)
  elif 30<temp:
    for i in range(15):         # Célébration de la victoire
      display.show(Image.DIAMOND_SMALL)
      sleep(100)
      display.show(Image.DIAMOND)
      sleep(100)
  sleep(500)
  pass
```

Jauge de luminosité :

```
from microbit import *

def plotBarGraph(val, max):     # On définie la fonction qui dessinera la jauge
  if val > 0 and val < max/5:
    led_image = Image('00000:00000:00000:00000:99999')
    display.show(led_image)
  if val >= max/5 and val < 2*max/5:
    led_image = Image('00000:00000:00000:99999:66666')
    display.show(led_image)
  if val >= 2*max/5 and val < 3*max/5:
    led_image = Image('00000:00000:99999:66666:44444')
    display.show(led_image)
  if val >= 3*max/5 and val < 4*max/5:
    led_image = Image('00000:99999:66666:44444:22222')
    display.show(led_image)
  if val >= 4*max/5 and val <= max:
    led_image = Image('99999:66666:44444:22222:22222')
    display.show(led_image)


while True:
  plotBarGraph(display.read_light_level(), 255)
  sleep(300)
```

Affichage température et pression (BMP 280) :

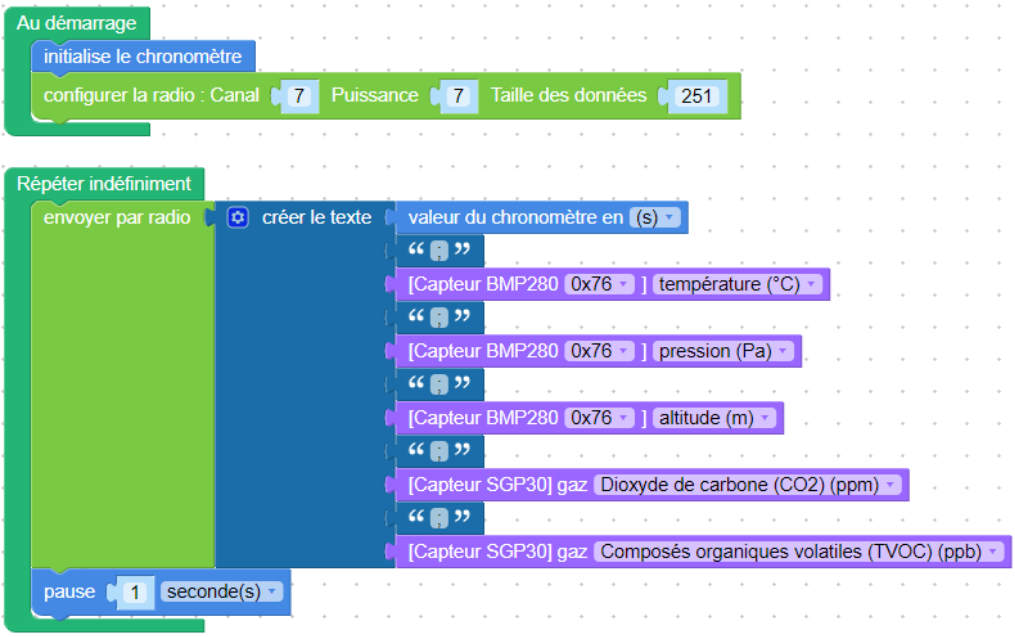
```
from microbit import *
from lcd_i2c import LCD1602
from bmp280 import BMP280

lcd=LCD1602()
bmp280=BMP280()

while True:
  lcd.setCursor(0,0)
  lcd.writeTxt(str(bmp280.Temperature())+" C")
  lcd.setCursor(0,1)
  lcd.writeTxt(str(bmp280.Pressure())+" Pa")
  sleep(1000)
```

Affichage taux CO2 et taux COV (SGP30) :

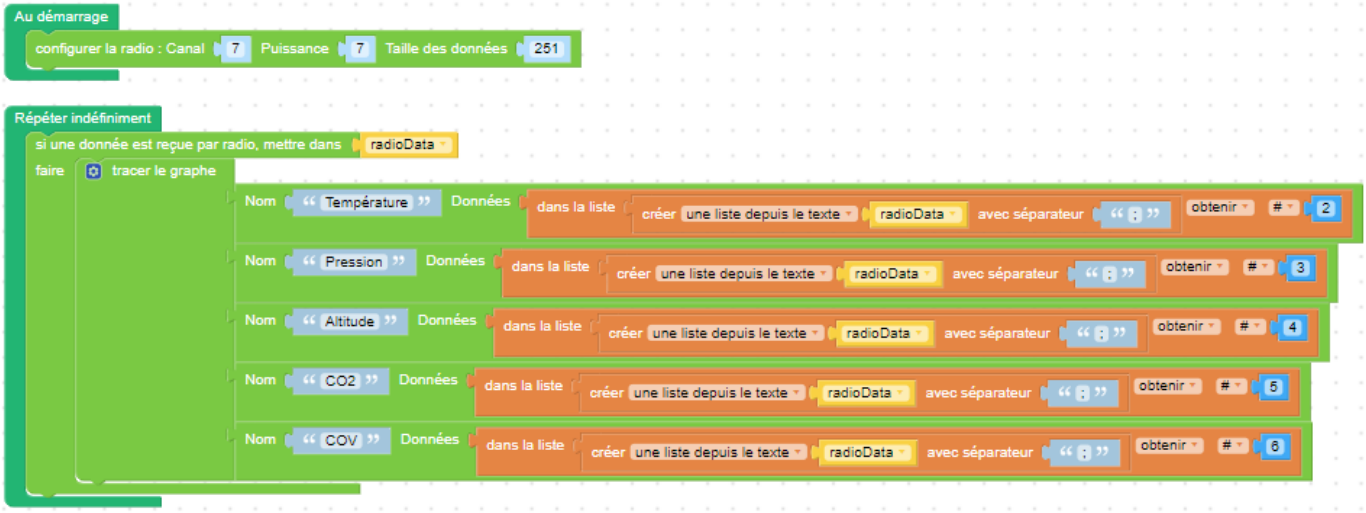
```
from microbit import *
from lcd_i2c import LCD1602
from sgp30 import SGP30

lcd=LCD1602()
sgp30=SGP30()

while True:
  lcd.setCursor(0,0)
  lcd.writeTxt(str(sgp30.eCO2())+" ppm")
  lcd.setCursor(0,1)
  lcd.writeTxt(str(sgp30.TVOC())+" ppb")
  sleep(1000)
```

Programmation de la partie électronique de la nacelle :

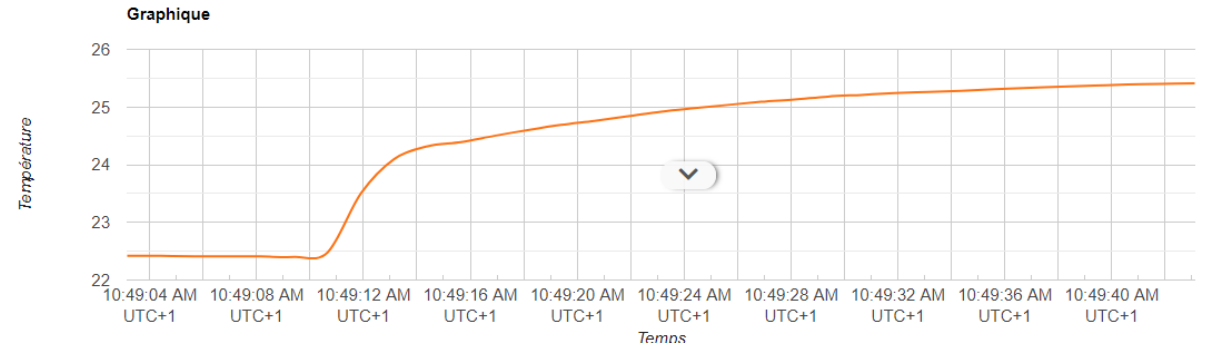
Carte émettrice :

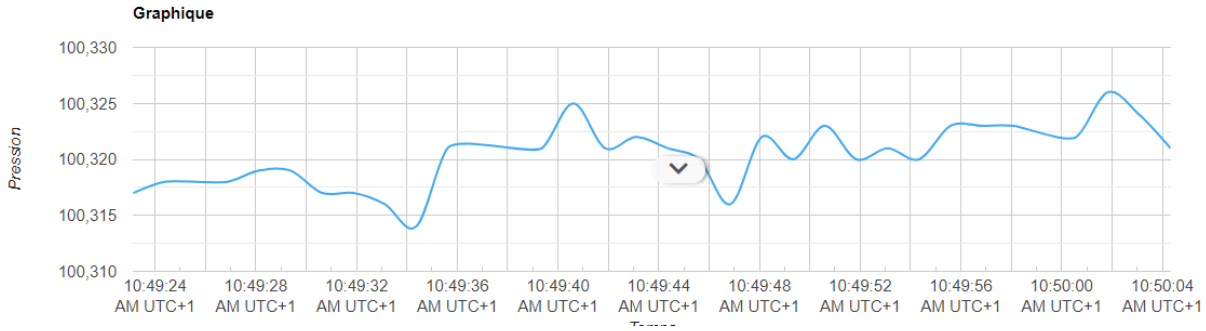


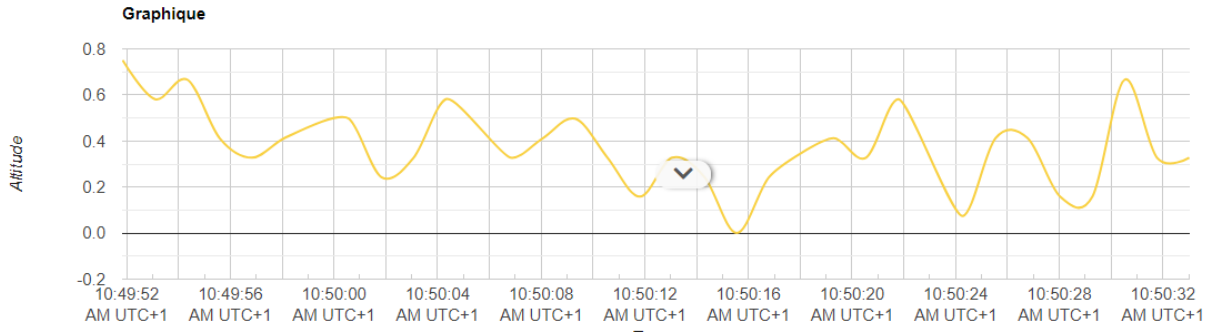
Carte réceptrice :

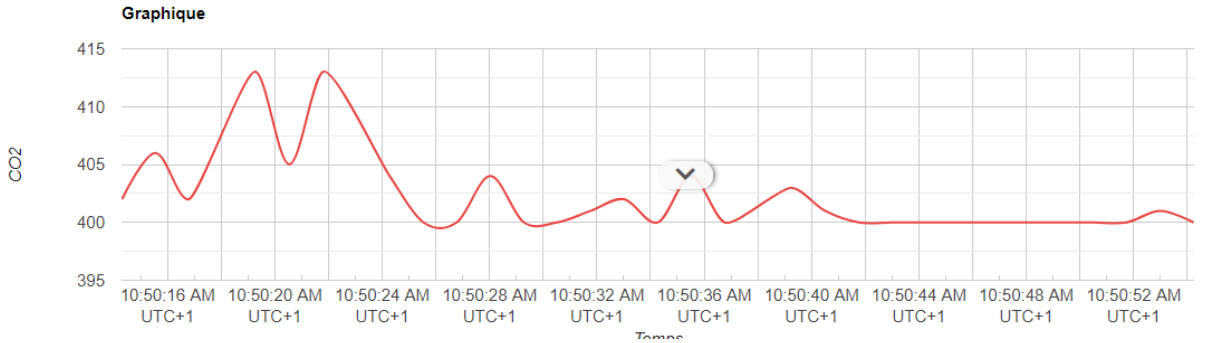


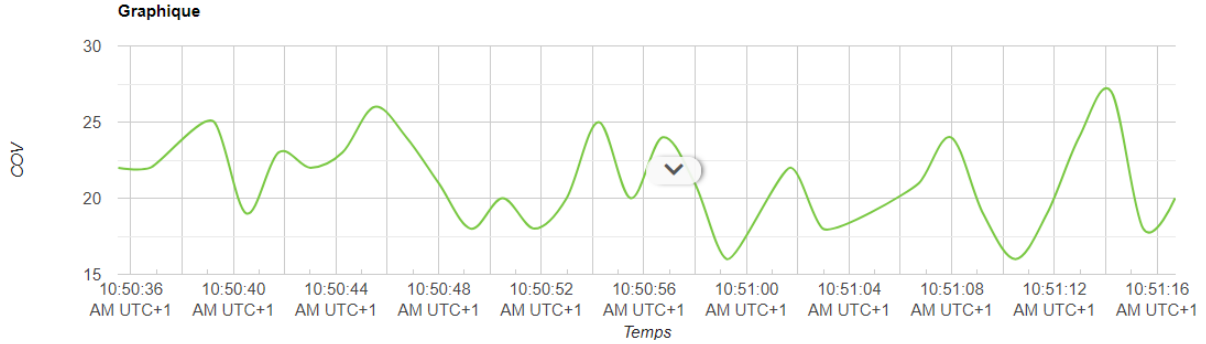
Visualisation des graphes :











Contacts – Références

Contacts

- Nicolas NOWAK : nicolas-raymond.nowak@ac-lille.fr
- Martial ANDRE : martial-fabien.andre@ac-lille.fr
- Léo BRIAND (Vittascience) : leo@vittascience.com
- Patrick HAMPTAUX (Météo à l'école) : patrick.hamptaux@sfr.fr

Références

- Tutoriel officiel de la carte microbit

https://microbit-micropython.readthedocs.io/fr/latest/tutorials/introduction.html

- Site la boite à physique :

https://laboiteaphysique.fr/site2/application/files/5215/5828/1840/1_Decouvrir_Python_Microcontroleur_live.pdf

https://laboiteaphysique.fr/site2/application/files/2915/5828/3495/2_Etendre_Fonctionnalites_Microcontroleur_live.pdf

- Site Vittascience

https://fr.vittascience.com/learn/

- Site Office For Climate Education (Ressources pédagogiques)

http://www.oce.global/sites/default/files/2019-04/1.5degree_FR_final_LR.pdf