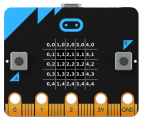
Commande de base pour `Micro:bit` (et python)

Description

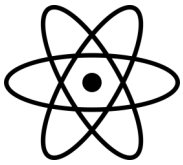
Objectif

Il est possible de programmer la carte `Micro:bit` avec Python. Pour cela, il faut utiliser le module `Micro:bit`.
Dans cette fiche, vous découvrirez les commandes essentielles de ce module.

Matériel

— 1 ×  `Micro:bit`
— 1 × IDE programmation python (Mu) : à télécharger/installer en https ://codewith.mu/

Commandes essentielles

 Durée	 Public	 Maths	 Sciences	 Algo
2 h	2de			affichage ; bouton ; événement.

Le module `microbit`

En Python, les entrées/sorties de la carte `Micro:bit` ne sont pas *nativement* accessibles. Afin de pouvoir utiliser les fonctions prévues à cet effet, il faut appeler le module `microbit` .

Pour appeler ce module, il faut utiliser la commande classique `from ... import ...`

Méthode

```
from microbit import *
```

Astuce

La bibliothèqe `microbit` est déjà téléchargée et préinstallée avec le logiciel `mu-editor`. Mais il faut tout de même l'appeler pour l'utiliser

Le module `micropython` introduit des **classes** (avec instances, méthodes et propriétés) et des **modules** (avec fonctions et constantes).

Exemple de classes :

Image pour créer et manipuler les images. Les propriétés sont des images déjà enregistrées (comme les smileys)

Button avec deux instances `button_a` et `button_b` pour connaître l'état des boutons.

Pin avec différentes instances en fonction du type de broche (digitale, analogique ou de contact comme `pin0`, `pin1` et `pin2`).

Exemple de modules :

display pour gérer l'écran de LED

accelerometer pour interroger l'accéléromètre

compass pour manipuler et interroger la boussole

music pour créer et manipuler de la musique

speech pour faire parler le `Micro:bit`

radio pour communiquer entre `Micro:bit` via un protocole simple

REMARQUE

Pour aller plus loin, vous pouvez consulter la page Microbit Module de la documentation officielle.

Micropython dans la carte `Micro:bit`

Lorsque la carte `Micro:bit` est flashée avec le module `microbit` , elle contient un noyau micropython et peut donc exécuter du code python.

L'IDE `Mu` permet d'accéder au noyau micropython de la carte. Pour **accéder au terminal** de ce noyau, il suffit de cliquer sur l'icône REPL de l'application.

Il est alors tout à fait possible d'écrire du code dans ce terminal qui sera exécuté par la carte `Micro:bit`.

REMARQUE

L'instruction `print()` permet d'écrire dans le terminal REPL.

MÉTHODE

Pour utiliser le terminal de la carte `Micro:bit` :

— Flasher la carte `Micro:bit` avec micropython.

— Ouvrir le terminal de la carte en cliquant sur REPL

— saisir le code ci-dessous :

```
> print("Hello, World!")
```

MÉTHODE

Le programme ci-dessous affiche 10 fois le même texte dans le terminal :

```
from microbit import *
for i in range(10):
    print("Hello, World!")
```

Afficher un texte

Le module `display` permet de gérer l'écran de LED de `Micro:bit`.

La fonction `display.show("string")` permet d'afficher un texte.

La fonction `display.scroll("string")` permet de le faire défiler.

MÉTHODE

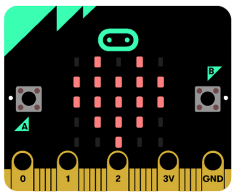
```
from microbit import *
display.scroll("Hello,")
display.show("World!")
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Hello, World! et Display de la documentation officielle.

Des images

La classe `Image` contient comme propriétés de nombreuses images pré-programmées. Par exemple l'image du coeur : `Image.HEART`



Pour afficher une image, il faut utiliser la fonction `display.show()` . La fonction `display.clear()` permet d'effacer l'écran.

Pour afficher successivement des images, il faut mettre le `Micro:bit` en pause. Sinon l'utilisateur n'aura pas le temps de tout voir. La commande `sleep(nombreEntier)` arrête donc la carte pendant la durée (en milliseconde) indiquée.

MÉTHODE

Le code Python ci-dessous affiche une image pendant une seconde puis affiche une deuxième image. Une seconde plus tard, l'écran s'efface.

```
from microbit import *
display.show(Image.HAPPY)
sleep(1000)
display.show(Image.ANGRY)
sleep(1000)
display.clear()
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Images ou Image de la documentation officielle.

Des animations

Lorsque plusieurs images sont stockées dans un **tableau**, il est possible de les faire défiler à un rythme donné. Ceci permet de facilement créer une animation.

Pour faire défiler des images, utiliser l'option `delay=nombreEntier` de la fonction `display.show()` .

Le module `microbit` contient **deux** tableaux d'images :

— `Image.ALL_CLOCKS` qui affiche la grande aiguille d'une montre sous différentes orientations

— `Image.ALL_ARROWS` qui affiche des flèches dans toutes les directions.

MÉTHODE

Le code ci-dessous affiche toutes les 0,200 secondes l'aiguille d'une montre qui tourne :

```
from microbit import *
display.show(Image.ALL_CLOCKS, delay=200)
```

REMARQUE

Pour aller plus loin, vous pouvez consulter la page Introduction » Images de la documentation officielle.

Des boutons

Les états des boutons de la carte `Micro:bit` sont récupérables par le module `micropython` . Il existe pour cela deux classes : `button_a` et `button_b` .

Ces deux classes possèdent 3 méthodes :

— `.get_presses()` retourne le **nombre** de fois que le bouton a été pressé depuis le dernier appel de cette méthode

— `.is_pressed()` retourne un **booléen** qui indique si le bouton est actuellement pressé

— `.was_pressed()` retourne un **booléen** qui indique si le bouton a été pressé depuis le dernier appel de cette méthode.

MÉTHODE

Le code ci-dessous affiche le nombre de fois que le bouton A a été pressé depuis le démarrage.

```
from microbit import *
sleep(10000)
display.scroll(str(button_a.get_presses()))
```

Pour programmer un `Micro:bit`, il est très souvent nécessaire de détecter un **évènement** comme par exemple l'appui sur un bouton.

Une méthode simple consiste à créer une **boucle infinie** `while True:` contenant un test qui, à chaque itération de la boucle, vérifie la réalisation de l'évènement attendu.

Lorsque l'événement se réalise, l'instruction `break` est appelée pour sortir de la boucle.

MÉTHODE

Le code ci-dessous tourne en boucle.

— Lorsque aucun évènement n'est détecté, c'est l'image triste `Image.SAD` qui s'affiche.

— Pendant que le bouton A est pressé, l'image joyeuse `Image.HAPPY` s'affiche.

— Pendant que la broche `pin1` est touchée (en même temps que la broche `GND`), l'image endormie `Image.ASLEEP` s'affiche.

— Lorsque le bouton B est pressé, l'écran s'efface et le programme quitte la boucle.

```
from microbit import *
while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    elif pin1.is_touched():
        display.show(Image.ASLEEP)
    elif button_b.is_pressed():
        display.clear()
        break
    else:
        display.show(Image.SAD)
```

REMARQUE

Les broches instanciées `pin0` , `pin1` et `pin2` peuvent aussi servir de bouton. La méthode `.is_touched()` permet de renvoyer un booléen qui devient vrai lorsqu'une personne en contact avec la masse (broche GND) touche puis relâche la broche en question.

En effet, lorsqu'une instance de ces broches est activée, la carte `Micro:bit` mesure la résistance entre cette broche et la masse (broche GND). Lorsque cette dernière varie et passe d'une valeur quasi infinie à une valeur faible, le test devient vrai. Cet évènement arrive lorsqu'une personne en contact avec GND touche la broche et la relâche.

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Boutons et Buttons de la documentation officielle.

Le hasard

Le module `random` est utilisable avec `Micro:bit`. Une fois importé, il est très simple de générer des nombres aléatoires avec les fonctions `random.random()` ou `random.randint(a,b)` .

MÉTHODE

Le programme ci-dessous affiche très rapidement 50 nombres aléatoires tirés entre 1 et 6. Le dernier nombre tiré est affiché pendant une seconde puis effacé.

```
from microbit import *
import random
for i in range(50):
    display.show(random.randint(1, 6))
    sleep(20)
sleep(1000)
display.clear()
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Hasard et Random Number Generation de la documentation officielle.

Le mouvement

L'accéléromètre du `Micro:bit` est accessible par le module `accelerometer` .

Il est alors possible de récupérer une des coordonnées du vecteur accélération (avec une fonction du type `accelerometer.get_x()` .

MÉTHODE

Le programme ci-dessous affcihe une flèche en fonction de l'inclinaison du `Micro:bit`.
Le bouton B permet de quitter cette boucle.

```
from microbit import *
button_b.was_pressed()
while True:
    capteur = accelerometer.get_x()
    if capteur > 40:
        display.show(Image.ARROW_E)
    elif capteur < -40:
        display.show(Image.ARROW_W)
    else:
        display.show("-")
    if button_b.was_pressed():
        display.clear()
        break
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Mouvement et Accelerometer de la documentation officielle.

Les gestes

Le module `accelerometer` peut aussi détecter des mouvements ou des positions pré-programmés : les gestes (`"up"`, `"down"`, `"left"`, `"right"`, `"face up"`, `"face down"`, `"freefall"`, `"3g"`, `"6g"`, `"8g"`, `"shake"`).

Il y a alors 3 fonctions qui s'applique sur le module `accelerometer` :

- `accelerometer.get_gestures()` retourne un **tuple** contenant l'historique des gestes. Le dernier élément du tuple est le geste le plus récent.Le tuple est réinitialisé à chaque appel de cette fonction.
- `accelerometer.is_gesture("nom")` retourne un **booléen** qui indique si le geste en cours est `"nom"` .
- `accelerometer.was_pressed("nom")` retourne un **booléen** qui indique si le geste `"nom"` a été pressé depuis le dernier appel de cette fonction.

MÉTHODE

Le programme ci-dessous affiche le nombre "8"
Lorsque le `Micro:bit` est secoué, il s'affiche une seconde plus tard de manière aléatoire et équiprobable soit "Oui", soit "Non".
Le jeu recommence sauf si on appui sur le bouton B ce qui fait sortir de la boucle.

```
from microbit import *
import random
button_b.was_pressed()
while True:
    display.show("8")
    if accelerometer.was_gesture("shake"):
        display.clear()
        sleep(1000)
        display.scroll(random.choice(["Oui","Non"]))
    if button_b.was_pressed():
        break
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Gestes et Accelerometer de la documentation officielle.

La radio

Les cartes `Micro:bit` peuvent communiquer entres elles au moyen du module `radio` .

La fonction `radio.send("string")` permet d'envoyer par radio le texte `"string"` .

La fonction `radio.receive()` retourne la chaîne de caractère des données reçus par radio. Si rien n'a été reçu, la chaine vaut `None`

MÉTHODE

Le programme est à téléverser sur 2 cartes `Micro:bit`. Un appui sur les boutons A ou B de l'un des `Micro:bit` affiche les texte "A" ou "B" sur l'autre.

```
from microbit import *
import radio
import random

button_a.was_pressed()
button_b.was_pressed()

while True:
    # émetteur
    if button_a.was_pressed():
        radio.send("A")
    if button_b.was_pressed():
        radio.send("B")

    # récepteur
    incomming = radio.receive()
    if incomming == "A":
        display.scroll("A")
    if incomming == "B":
        display.scroll("B")

    # sortir de la boucle
    if pin0.is_touched():
        display.clear()
        break
    sleep(20)
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Radio et Radio de la documentation officielle.

La boussole

Pour interroger la boussole du `Micro:bit`, il faut utiliser le module `compass` .

Certaines fonctions comme `microbit.compass.get_x()` renvoient une composante du vecteur champ magnétique.

La fonction `microbit.compass.heading()` renvoie l'angle en degré entre l'orientation de la carte `Micro:bit` et le nord magnétique.

REMARQUE

Avant d'utiliser une fonction du module `compass` , il faut obligatoirement calibrer `Micro:bit`.

Pour cela, il est automatiquement demandé à l'utilisateur de bouger la carte dans différentes positions. Il faut que le point rouge qui clignote passe par toutes les LED de l'écran.

Pour programmer une calibration de la carte, il est possible d'utiliser la commande `compass.calibrate()` .

MÉTHODE

Le programme ci-dessous fait office de boussole et indique le nord magnétique.

Attention, le capteur est sensible aux objets tels que téléphones, ordinateurs ou aux lieux tels que ascenseurs ou salle informatique...

```
from microbit import *
compass.calibrate()
button_b.was_pressed()
while True:
    cap = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[cap])
    if button_b.was_pressed():
        display.clear()
        break
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Direction et Compass de la documentation officielle.

DES COMMANDES POUR ALLER PLUS LOIN

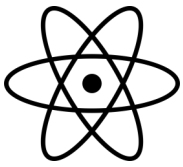

Durée

2 h


Public

2de


Maths


Sciences


Algo

affichage ; bouton ; événement ; son

Représentation graphique avec Mu

Il est possible de tracer un **nuage de points** avec l'IDE Mu.

Pour cela, il faut

— utiliser l'outil Graphique de l'IDE Mu,

— faire afficher par le programme en cours d'exécution un tuple de nombres.

MÉTHODE

Le programme ci-dessous affiche dans le terminal (ou **dans le Graphique** si l'outil de l'IDE est cliqué) des fréquences d'apparitions lors d'une expérience aléatoire.

```
from microbit import *
import random
while True:
    if button_a.was_pressed():
        nb1 = 0
        total = 0
        for i in range(1000):
            # tirage aléatoire
            tirage = random.randint(0,1)
            total = total + 1
            # calcul d'effectifs
            nb1 = nb1 + tirage
            nb0 = total - nb1
            # affichage dans le terminal
            print((i, nb1/total, nb0/total))
```

Des images personnalisées

Les images sont des objets qui peuvent être créées grâce au constructeur `Image("string")` .

Le texte `"string"` permet de décrire, **ligne par ligne**, la luminosité de chaque pixel de l'écran de LED. Pour chaque diode, la valeur varie de 0 (éteinte) à 9 (luminosité maximale). Pour séparer les lignes, il faut utiliser le symbole `":"`

MÉTHODE

Voici une image représentant un bateau avec la coque plus foncée que les 2 mâts :

```
from microbit import *
bateau = Image("05050:"
               "05050:"
               "05050:"
               "99999:"
               "09990")
display.show(bateau)
sleep(1000)
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Images ou Image de la documentation officielle.

Des animations personnalisées

En stockant des images personnalisées dans un tableau, il est possible de créer des animations personnalisées.

MÉTHODE

Créer un image représentant un tableau et stocker cette image dans la variable `bateau1` :

```
from microbit import *
bateau1 = Image("05050:"
                "05050:"
                "05050:"
                "99999:"
                "09990")
```

Créer d'autres images (donnant l'illusion que le bateau s'enfonce) et les stocker dans de nouvelles variables :

```
bateau2 = Image("00000:"
                "05050:"
                "05050:"
                "05050:"
                "99999")
```

```
bateau3 = Image("00000:"
                "00000:"
                "05050:"
                "05050:"
                "05050")
```

```
bateau4 = Image("00000:"
                "00000:"
                "00000:"
                "05050:"
                "05050")
```

```
bateau5 = Image("00000:"
                "00000:"
                "00000:"
                "00000:"
                "05050")
```

```
bateau6 = Image("00000:"
                "00000:"
                "00000:"
                "00000:"
                "00000")
```

Créer un tableau regroupant toutes les images et afficher l'animation :

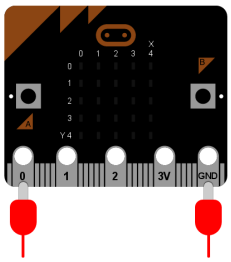
```
tous_bateaux = [bateau1, bateau2, bateau3, bateau4, bateau5, bateau6]
display.show(tous_bateaux, delay=200)
```

REMARQUE

Pour aller plus loin, vous pouvez consulter la page Introduction » Images de la documentation officielle.

Faire parler `Micro:bit`

La carte `Micro:bit` peut aussi **parler**. Avant cela, il faut connecter un casque audio ou un (petit) haut-parleur à la carte.



Le module `speech` contient les fonctions permettant de gérer la voix de `Micro:bit`, dont la fonction `speech.say("string")` qui fera parler.

MÉTHODE

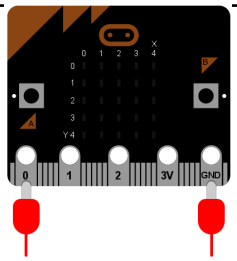
```
from microbit import *
import speech
speech.say("Hello, World")
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Speech ou Speech de la documentation officielle.

De la musique

La carte `Micro:bit` est capable de jouer de la musique. Pour cela, il faut au préalable **connecter un casque audio** ou un petit haut-parleur à la carte.



Le module `music` contient les fonctions permettant de gérer et de contrôler le son. Parmi elles, la fonction `music.play()` joue une mélodie.

Par ailleurs, le module `music` contient aussi un grand nombre de mélodies enregistrée (ie. déjà programmées) comme par exemple :

- `music.DADADADUM`,
- `music.ENTERTAINER` ou encore
- `music.PRELUDE`.

MÉTHODE

Voici comment jouer la musique `POWER_UP` avec `Micro:bit` :

```
from microbit import *
import music
music.play(music.POWER_UP)
```

La carte `Micro:bit` peut aussi générer un son en fonction de sa fréquence. Pour cela, il faut utiliser la fonction `music.pitch(freq, duree)` qui va générer pendant `duree` millisecondes le son de frequence `freq` .

MÉTHODE

Le code ci-dessous génère une sirène qui passe d'un son grave à un son aigu puis qui redevient grave.

Chaque fois que le son redevient grave, la boucle teste si le bouton B a été pressé. Si c'est la cas, le programme quitte la boucle.

```
from microbit import *
import music
button_b.was_pressed()
while True:
    for freq in range(880, 1760, 16):
        music.pitch(freq, 6)
    for freq in range(1760, 880, -16):
        music.pitch(freq, 6)
    if button_b.was_pressed():
        break
```

MÉTHODE

Le programme ci-dessous génère un son dépendant de l'inclinaison du `Micro:bit`.
Le bouton B permet de quitter cette boucle.

```
from microbit import *
import music
button_b.was_pressed()
while True:
    ton = max(0, accelerometer.get_y())
    music.pitch(ton, 10)
    if button_b.was_pressed():
        break
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Musique ou Music de la documentation officielle.

Les fichiers

La carte `Micro:bit` peut stocker des fichiers. Ces derniers ne sont pas accessibles par le système classique de fichiers, mais par l'intermédiaire de l'outil `Fichier` de l'IDE `Mu`.

Il est tout à fait possible d'utiliser Python pour lire, créer, modifier ou effacer un fichier.

MÉTHODE

Pour **lister les fichiers** présents sur la carte `Micro:bit`, il faut utiliser le module `os` .

La fonction `os.listdir()` renvoie alors une liste de chaîne de caractère. Chaque élément de la liste est un nom de fichier.

```
import os
listeFichiers = os.listdir()
```

MÉTHODE

Pour **lire le contenu** d'un fichier (qui doit exister), il faut :

- créer une instance de fichier en mode lecture `'r'`
- utiliser la méthode `.read()` pour copier le contenu du fichier dans une variable.

L'exemple ci-dessous lit le fichier `texte.txt` , copie son contenu dans la variable `txt` et l'afficher dans le terminal :

```
with open('texte.txt','r') as monFichier:
    txt = monFichier.read()
print(txt)
```

MÉTHODE

Pour **écrire un texte** dans un fichier (qui peut ne pas exister), il faut :

- créer une instance de fichier en mode écriture `'w'`
- utiliser la méthode `.write("string")` pour écrire le texte dans le fichier (attention, si le fichier existait déjà, cela supprimera tout ce que le fichier contenait).

L'exemple ci-dessous ouvre le fichier `texte.txt` en écriture puis écrase son contenu avec le contenu de la variable `txt` .

```
txt = "1,2,3\n4,5,6"
with open('texte.txt','w') as monFichier:
    monFichier.write(txt)
```

MÉTHODE

Le programme ci-dessous effectue 50 tirages aléatoires qu'il enregistre dans le fichier `save.txt`.

- importer tous les modules nécessaires
- sauvegarder les 50 tirages dans la variable `txt`
- établir la liste des fichiers
- si le fichier `save.txt` existe, lire le fichier et sauvegarder ses données dans la variable `contenu`
- ouvrir le fichier en mode écriture et l'écraser avec le contenu mis à jour.

```
# sauvegarder 50 nb aléatoires
from microbit import *
import random
import os
```

```
txt = ""
for i in range(50):
    nb = random.randint(1, 6)
    txt = txt + str(nb)
    display.show(nb)
    sleep(20)
```

```
# connaître la liste des fichiers sur MB
listeFichiers = os.listdir()
```

```
# si le fichier 'save.txt' existe, récupérer le contenu
if ('save.txt' in listeFichiers):
    with open('save.txt','r') as monFichier:
        contenu = monFichier.read()
else:
    contenu = ""
```

```
with open('save.txt','w') as monFichier:
    contenu = contenu + txt + "\n"
    print(contenu)
    monFichier.write(contenu)
```

REMARQUE

Pour aller plus loin, vous pouvez consulter les pages Introduction » Storage ou Local Persistent File System de la documentation officielle.

À propos de cette publication

Pourquoi les objets connectés ?

Alors que dans certaines disciplines le temps commence à manquer pour traiter l'ensemble du programme, certains évoquent déjà l'idée d'en faire plus !

En effet, les enseignants utilisent déjà les outils numériques. Par exemple, dans les classes de mathématiques, l'utilité du tableur et de GeoGebra n'est plus à démontrer. Jusqu'à l'introduction de l'algorithmique, ces deux logiciels efficaces et maîtrisés par les enseignants étaient amplement suffisants. Est-ce donc juste un effet de mode de faire cours avec les robots (Thymio, Mbot), les objets programmables et connectés (Arduino, `Micro:bit`, STM education, Raspberry Pi) ou est-ce une nouvelle façon d'aborder notre enseignement ? Ces nouvelles possibilités technologiques, forcément chronophages, nous permettront-elles de traiter un contenu disciplinaire exigeant dans un cadre institutionnel contraignant ?

Nous n'avons bien sûr pas toutes les réponses à ces questions mais nous pensons que lorsqu'il est accompagné de certains de ces outils, notre enseignement a beaucoup à y gagner.

L'introduction de l'algorithmique en lycée professionnel nous interroge. Longtemps il nous a semblé impensable et inenvisageable d'avoir à enseigner un langage de programmation comme Python auprès d'un public d'élèves globalement en difficulté avec les mathématiques. Fort de ce constat, nous avons cherché les moyens de lier les mathématiques à la logique et au raisonnement algorithmique. C'est pourquoi nous avons exploré les potentialités des objets connectés.

Notre postulat est double. Nous pensons que :

— grâce à des situations réelles et concrètes, les objets connectés facilitent la mise en activité de tous les élèves ;

— grâce à des activités simples mais évolutives centrées autour de réalisations matérielles, la dimension affective du travail est valorisée. Soyons fous et espérons que l'élève tisse une histoire personnelle avec l'activité, qu'il soit fier du travail accompli et qu'il prenne également du plaisir à expliquer et à montrer ses réalisations.

En devenant de plus en plus simples, accessibles et facilement utilisables, les objets connectés permettent d'aborder des contenus disciplinaires et de développer des compétences transversales essentielles pour l'élève.

En travaillant à partir des objets connectés, la situation de départ est plus concrète et l'objectif à atteindre suffisamment clair pour l'élève. Plus ou moins guidé selon son niveau d'expertise technique, il est alors libre dans sa démarche. Avec des interfaces de programmation accompagnées parfois de simulateurs, la démarche par essais et erreurs a ici toute sa place. Par ailleurs, l'élève devra clarifier sa pensée avant de verbaliser ses idées en langage naturel. Il pourra ainsi proposer et élaborer un modèle acceptable par la machine pour enfin traduire son algorithme en se pliant à la rigueur du langage de programmation.

Effectuant régulièrement des va-et-vient entre abstraction et réalité, cherchant à valider son algorithme à partir d'un visuel ou d'une exploitation des résultats, l'élève entre progressivement dans la modélisation.

Les scénarios proposés dans cette brochure permettent tout cela : une approche des mathématiques et des sciences qui laisse la place à l'expérimentation : manipulation, programmation et auto-validation.

QUI SOMMES-NOUS ?

Nous sommes des enseignants de maths/sciences regroupés au sein d'un groupe de recherche de l'IREM de Marseille.

groupe Lycée Professionnel innovation, expérimentation et formation de l'IREM d'Aix-Marseille

Notre groupe, Innovation, Expérimentation et Formation en Lycée Professionnel (InEFLP) consacre une partie de son travail à l'enseignement de l'algorithmique en classes de lycée professionnel. Dans le cadre de cette recherche, nous explorons les objets connectés tels que Arduino, `Micro:bit`, `STM32 Éducation` ou mbot.

LIENS UTILES

Page du groupe InEFLP
`http://url.univ-irem.fr/ineflp`

IREM de Marseille Site académique de l'IREM de Marseille
`http://url.univ-irem.fr/mars`

Portail des IREM Site national des IREM
`http://www.univ-irem.fr/`

Formation à l'algorithmique LP et SEGPA Padlet de utilisé lors de nos formations académiques
`http://url.univ-irem.fr/stage-algo`

Collecte de ressources pour `Micro:bit` Padlet sur `Micro:bit` utilisé en formation
`http://url.univ-irem.fr/algo2017-microbit`

Brochure sur `Micro:bit` Publication de la C2i TICE pour une prise en main de `Micro:bit`
`http://url.univ-irem.fr/c2it-mb-t1-pdf`

Description `Micro:bit` Fiche sommaire de description de `Micro:bit`
`http://url.univ-irem.fr/ineflp-microbit`

Site IREM dédié à `Micro:bit` Site de ressources sur `Micro:bit` du groupe
`http://url.univ-irem.fr/o`

Un extrait de la brochure

Les objets connectés pour enseigner l'algorithmique en lycée professionnel

< version du 18 janvier 2020 >