Découverte de Python

GCC ! – Prologin

2020

Introduction à la programmation

Tous les ordinateurs ne sont rien d'autres que des composants électroniques qui exécutent des instructions. Lorsque tu ouvres un navigateur internet, ton ordinateur exécute les instructions du logiciel utilisé, qu'on appelle aussi **programme**. Le but de cet atelier est de te faire découvrir comment nous pouvons créer nos propres programmes, et pour cela il nous faut un moyen d'écrire ces instructions : un **langage de programmation**.

Il existe énormément de langage de programmation, tout comme il existe des milliers de langue dans le monde ! Certains sont plus connus que d'autres et nous allons te faire découvrir Python, un langage facile à prendre en main mais très puissant.

Qu'est-ce qu'un `micro:bit` ?

Pour cet atelier, nous allons utiliser un `micro:bit`, c'est un microcontrôleur de la taille d'une carte bancaire. Nous pouvons écrire des programmes et les lancer dessus, ainsi qu'interagir avec ses composants : l'écran de LED, les boutons, l'accéléromètre (pour détecter des mouvements), etc.

Tu peux garder les microcontrôleurs avec toi, et continuer à écrire tes propres programmes en Python dessus après l'atelier !

Notre environnement de travail

Pour écrire nos programmes, nous utiliserons `Mu`, que vous avez pu découvrir dans le TP0.

Si tu n'as pas réussi à bien installer `Mu`, demande aux organisateurs de t'aider.

Notre premier programme

Un programme en Python est constitué d'une série d'instruction à exécuter par un ordinateur (dans notre cas, le `micro:bit`). Chaque instruction doit être écrite sur une nouvelle ligne, et

le programme sera lu par l'ordinateur de haut en bas. Commençons par analyser un premier exemple de programme :

```
1  from microbit import *
2
3  display.set_pixel(0, 2, 9)
4  sleep(500)
5  display.set_pixel(1, 2, 9)
6  sleep(500)
7  display.set_pixel(2, 2, 9)
8  sleep(500)
9  display.set_pixel(3, 2, 9)
10 sleep(500)
11 display.set_pixel(4, 2, 9)
```

Si tu testes ce programme, il affiche une barre de chargement sur la ligne de diode centrale de ton `micro:bit`.

Ligne à ligne voilà ce que signifie ce programme :

1. `from microbit import *` permet d'indiquer que dans la suite du programme, on va avoir besoin de toutes les fonctionnalités relative à l'utilisation du `micro:bit`. Ici, toutes les instructions qui suivent ne seraient pas valides sans cette ligne.
2. Une ligne vide n'a aucun effet, il ne faut pas hésiter à s'en servir pour espacer son programme (on peut ainsi séparer des morceaux de codes, à la manière dont on sépare des paragraphes dans du texte).
3. `display.set_pixel(0, 2, 9)` allume le pixel situé sur la colonne n°0 et la ligne n°2. Le dernier paramètre (9) indique la luminosité de la diode entre 0 (diode éteinte) et 9 (pleine puissance).
4. `sleep(500)` met l'exécution du programme en pause pendant 500 millisecondes. Essaye de supprimer cette ligne, le programme s'exécute tellement vite que tu n'as pas le temps de voir qu'une diode s'allume avant l'autre !
5. La suite du programme se répète : on allume les diodes des colonnes numéro 1, 2, 3 puis 4.

Une manière d'expliquer du code directement dans le programme est d'utiliser des **commentaires** :

```
1  from microbit import *
2
3  # Ceci est un commentaire
4
5  # La ligne suivante va allumer le pixel situé à la colonne 0 et la ligne 2
6  display.set_pixel(0, 2, 9)
7  # La ligne suivante va attendre 500ms
8  sleep(500)
```


Tout ce qui est écrit après un `#` sera totalement ignoré par Python, ces commentaires sont uniquement à destination des programmeurs, c'est-à-dire toi !

Les variables

Pour l'instant nous avons seulement vu comment créer un programme qui exécute une liste d'instructions assez bêtement, qui donneront toujours exactement le même résultat. Cependant, les ordinateurs (et donc ton `micro:bit` aussi) ont une mémoire que l'on peut manipuler dans un programme pour que celui-ci agisse différemment en fonction de ce qu'il a retenu dans sa mémoire.

Une **variable** est un morceau de la mémoire dans lequel on va pouvoir enregistrer des valeurs. Quand on crée une variable, on commence par lui choisir un nom qui va ensuite pouvoir être utilisé pour lire ou modifier la valeur qui est enregistrée dans la variable.

Utilisation des variables

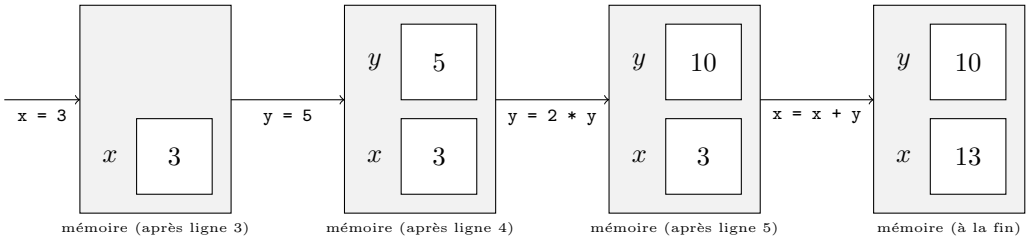
Pour **créer une variable** il suffit d'écrire `nom_de_la_variable = valeur_initiale`, par exemple : `nombre_de_patates = 42`.

Ensuite on peut **réutiliser la valeur** stockée dans la variable en l'identifiant par son nom, par exemple on peut créer une nouvelle variable `prix` qui dépend de la variable qui a été créé précédemment : `prix = nombre_de_patates * 50`.

Pour **modifier une variable** on utilise aussi le symbole d'égalité, par exemple on peut augmenter de 1 la valeur stockée dans une variable : `nombre_de_patates = nombre_de_patates + 1`.

Voici un exemple de programme complet que tu peux tester sur ton `micro:bit` suivi d'un petit schéma qui représente ce qui est enregistré dans les variables après chaque ligne du programme :

```
1 from microbit import *
2
3 x = 3
4 y = 5
5 y = 2 * y
6 x = x + y
7
8 display.scroll(x)
```

Ce programme affichera donc 13 sur l'écran à diodes de ton `micro:bit`.

Exercice Qu'affiche le code suivant sur le `micro:bit` :

```
from microbit import *

a = 12
b = a + 3
x = b - 5
x = x * 2

display.scroll(x)
```

Types de variables

Les valeurs stockées dans des variables peuvent être diverses, comme des nombres, du texte ou un ensemble de valeurs.

Nombres C'est le type de variable le plus courant, on a déjà vu qu'on peut créer des valeurs numériques avec leur écriture naturelle, et qu'on peut utiliser les opérations classiques : `+`, `-`, `*` (multiplication), et `/` (division).

```
x = 3 * 4
y = x / 2
z = 8 + 1
```

Ici `x` vaut 12, `y` vaut 6.0, et `z` vaut 9. Nous pouvons aussi combiner plusieurs opérations ensemble, par exemple :

```
a = (x - y) + z
```

En reprenant les valeurs précédentes, on obtiendrait `a` valant 15.0.

Chaines de caractère On peut créer du texte en mettant son contenu entre guillemets (par exemple : `mon_texte = "Bonjour tous le monde !"`). On peut aussi concaténer des morceaux de textes entre eux avec l'opérateur `+` (par exemple : `mon_texte = mon_texte + "!!"`).

À noter qu'il est souvent très pratique de convertir un nombre en texte pour ensuite l'incorporer dans une phrase, on peut faire ça avec la fonction `str(nombre)`.

```
1 from microbit import *
2
3 nombre_de_patates = 42
4 texte = "Il y a " + str(nombre_de_patates) + " patates !"
5 display.scroll(texte)  # Affiche "Il y a 42 patates !" sur l'écran
```

Booléens Enfin, les booléens servent à exprimer le vrai ou le faux. Il n'y a que deux valeurs possibles pour ce type de variables : `True` (vrai) et `False` (faux).

Des valeurs booléennes sont renvoyées par les opérations de comparaisons : `==` (égalité), `!=` (différence), `<`, `>` (les inégalités strictes), `<=` et `>=` (les inégalités larges). Par exemple `1 < 2` vaut `True` mais `3 != 3` vaut `False`.

Enfin, il est possible de manipuler les valeurs booléennes avec les opérateurs `not`, `and` et `or` :

- `not a` vaut `True` si et seulement si `a` vaut `False` ;
- `a and b` vaut `True` si et seulement si `a` **et** `b` valent `True` ;
- `a or b` vaut `True` si et seulement si `a` **ou** `b` valent `True`.

Exemples

On peut récupérer le nombre d'appuis qui ont été faits sur un bouton du `micro:bit` en utilisant les fonctions `button_a.get_presses()` et `button_b.get_presses()` pour le second bouton.

Voici un exemple de petit jeu qui déclenche un compte à rebours avant de donner 5 secondes pour appuyer autant de fois que possible sur le bouton de gauche.

```python
from microbit import *

# Compte à rebours
display.set_pixel(0, 2, 9)
sleep(500)
display.set_pixel(1, 2, 9)
sleep(500)
display.set_pixel(2, 2, 9)
sleep(500)
display.set_pixel(3, 2, 9)
sleep(500)
display.set_pixel(4, 2, 9)

# Phase de jeu
button_a.get_presses()     # réinitialise le compteur d'appuis
sleep(5000)                # donne 5 secondes pour jouer
a = button_a.get_presses() # récupère le nombre d'appuis

# Affichage du score
display.scroll("Score: " + str(a))
```

Exercice : calculatrice

En t'inspirant de l'exemple précédent, écris un programme qui donne le résultat de la multiplication de deux nombres. Pour récupérer la valeur des deux nombres tu peux donner quelques secondes à l'utilisateur pour appuyer le bon nombre de fois sur chaque bouton. Par exemple, si pendant ce temps, tu appuies 3 fois sur le bouton de gauche et 7 fois sur celui de droite, le programme affichera `3 * 7 = 21` sur l'écran.

Les boucles

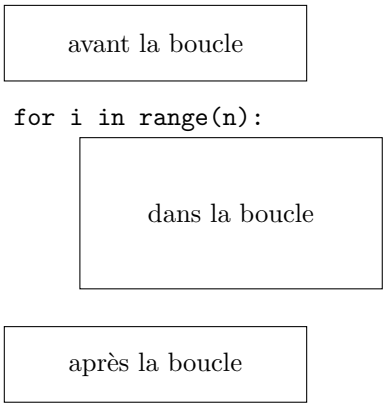
Imaginons que nous souhaitions exécuter notre programme de calculatrice plusieurs fois d'affilée, pour demander différentes opérations à l'utilisateur. Une manière de faire cela sera de tout simplement recopier le programme de calculatrice plusieurs fois, mais tu te doutes bien que cela n'est pas la meilleure façon de faire. Pour résoudre ce problème, nous pouvons utiliser des boucles qui vont se charger de **répéter un bout de code** plusieurs fois.

La boucle `for`

La boucle **for** sert à répéter plusieurs fois le même morceau d'un programme. Elle s'utilise comme suit :

```
1  from microbit import *
2
3  for colonne in range(5):
4      # Ce qui suit va être répété 5 fois, une variable `colonne` est créée,
5      # qui va prendre les valeurs 0, 1, 2, 3 puis 4.
6      display.set_pixel(colonne, 2, 9)
7      sleep(500)
8
9  # Ce qui suit ne va être exécuté qu'une fois
10 display.scroll("fini !")
```

Remarque comme les lignes 4 à 7 sont décalées vers la droite, cette transformation permet d'identifier le morceau du programme qui est concerné par la boucle. À chaque fois que tu écris un `for`, la ligne suivante doit être décalée vers la droite, et le morceau de code qui sera répété par cette boucle s'arrêtera juste avant la première ligne qui ne sera plus décalée vers la droite.



Exercice Modifie ta barre de chargement pour qu'au lieu d'allumer les diodes d'un coup (avec une intensité de 9), elles s'allument en douceur (en faisant progressivement varier l'intensité de 0 à 9).

La boucle `while`

La boucle **while** sert à répéter un morceau de programme tant qu'une condition est vraie. Elle s'utilise comme suit :

```python
from microbit import *

while button_a.get_presses() == 0:
    # Ce qui suit va être exécuté tant que le bouton de gauche n'a pas été
    # pressé.
    display.set_pixel(2, 2, 9)  # Allume la diode
    sleep(500)
    display.set_pixel(2, 2, 0)  # Éteint la diode avant de recommencer
    sleep(500)

display.scroll("fini !")
```

Exercice Modifie ton jeu pour que la partie recommence tant que le joueur n'a pas atteint un score de 20, une fois qu'il a atteint ce score tu peux afficher une image de victoire avec l'instruction `display.show(Image.HAPPY)`.

Exercice Modifie ton jeu pour qu'il affiche une barre de progression, la barre doit être pleine quand il a appuyé 20 fois sur le bouton. À la fin, le jeu affiche le temps qu'il a fallu pour atteindre les 20 appuis.

La fonction `running_time()` retourne le nombre de millisecondes qui se sont écoulées depuis le début de ton programme, tu peux donc t'en servir pour calculer le score du joueur.

Les conditions : `if` ... `elif` ... `else`

L'instruction `if` permet de décider de n'exécuter un morceau de code que lorsqu'une condition est vraie. Pour ce faire la syntaxe est : `if condition:` suivi d'un bloc de code indenté. Le bloc de code en question ne sera alors exécuté que si `condition` s'évalue à `True`. Un `if` peut être accompagné d'un `else` qui exécute un bloc de code uniquement si la condition du `if` est fausse. Enfin, il y a aussi le `elif` qui est la contraction du `else` et du `if`. Le `elif` n'est exécuté que si sa condition est vraie et que les conditions précédentes sont fausses.

Voici un exemple simple pour illustrer :

```python
from random import randint
from microbit import *

x = randint(0, 100)  # assigne un nombre aléatoire à x

if x < 30:
    display.scroll('x est inférieur à 30')
elif x < 50:
    display.scroll('x est inférieur à 50')
elif x < 80:
    display.scroll('x est inférieur à 80')
else:
    display.scroll('x est supérieur à 80')
```

Ou un exemple légèrement plus poussé qui permet de lier les boutons du `micro:bit` à des diodes :

```python
from microbit import *

while True:
    # Ce qui suit est executé indéfiniment

    if button_a.is_pressed():
        # Allume la diode de gauche si le bouton de gauche est enfoncé
        display.set_pixel(1, 2, 9)
    else:
        # Éteint la diode de gauche sinon
        display.set_pixel(1, 2, 0)

    if button_b.is_pressed():
        # Allume la diode de droite si le bouton de droite est enfoncé
        display.set_pixel(3, 2, 9)
    else:
        # Éteint la diode de droite sinon
        display.set_pixel(3, 2, 0)
```

Remarques :

— Un `if` n'est pas nécessairement accompagné d'un `else` (ou d'un `elif`). Dans ce cas, si sa condition est fausse, rien n'est exécuté
— On peut ajouter autant de `elif` que l'on veut après un `if`

Tout est clair ?

Si tu as bien compris, tu devrais pouvoir dire la différence entre notre premier exemple et le code ci-dessous :

```python
from random import randint
from microbit import *

x = randint(0, 100)  # assigne un nombre aléatoire à x

if x < 30:
    display.scroll('x est inférieur à 30')
if x < 50:
    display.scroll('x est inférieur à 50')
if x < 80:
    display.scroll('x est inférieur à 80')
if x > 80:
    display.scroll('x est supérieur à 80')
```

Projets

À partir de maintenant, place à la créativité.

Le but est que tu réalises un petit projet de ton choix. Si tu as une idée précise, tu peux te lancer et les organisateurs t'aideront ! N'hésite pas à aller regarder la section suivante qui détaille comment utiliser toutes les fonctionnalités du `micro:bit` pour trouver des idées. Mais si tu es à court d'idées, voici quelques projets possibles, que tu peux modifier comme tu le souhaites :

Projet Tamagochi

Il s'agit de programmer un petit jeu où l'on doit prendre soin d'un animal. Il a une jauge de faim (stockée dans une variable) qui diminue régulièrement (toutes les 5 secondes par exemple) et qu'on nourrit en appuyant sur le bouton A ou B, ce qui augmente sa jauge de faim. Il part avec une faim à 24, et s'il arrive à une faim de -12, il meurt et on a perdu. Sur l'écran on affiche un smiley très content s'il a plus 20 à sa jauge de faim, s'il a plus de 5 un smiley content, s'il a entre 5 et moins 5 smiley triste et s'il a moins de moins 5 à sa jauge de faim, smiley énervé. En bonus, si on le secoue il perd d'un coup 4 point de jauge de faim.

Pour avoir à la fois un affichage qui change rapidement (quand on le nourrit on veut voir le changement de smiley rapidement) et pouvoir faire diminuer sa jauge de faim régulièrement, nous allons avoir besoin d'une structure un peu spéciale. On va consacrer une variable `temps` au fait de compter combien de demi secondes sont passées. Lorsque 10 demies secondes seront passées, on diminue la jauge de faim. Cela nous donne la structure de code suivante (les phrases en rouge sont les choses que vous devez remplacer).

```
temps = 0
temps_précédent = 0
"autres déclarations de variables"
while "le tamagoch est en vie" :
    sleep(500)
    temps = temps + 1 # Une demie seconde est passée
    if "le bouton A ou B a été appuyé" :
        "la jauge de faim augmente"
    if "le tamagochi est secoué":
        "la jauge de faim diminue"
    if temps == temps_précédent + 10: # 5 secondes sont passées
        "la jauge de faim est diminuée"
        temps_précédent = temps
    "on affiche le smiley qui correspond à la jauge de faim"
```

Projet bulle à niveau

Il s'agit d'afficher une bulle d'air (représentée par un pixel) qui se déplace dans de l'eau, en fonction de comment le `micro:bit` est penché. Si le `micro:bit` est penché vers la droite la bulle sera à gauche de l'écran, et inversement si on penche à gauche elle sera à droite.

Pour faire ce projet, il faut toutes les demies secondes changer l'affichage de la bulle en fonction des données de l'accéléromètre, pour savoir comment utiliser l'accéléromètre, consultez les explications sur les capteurs de la section suivante ! Il vous faudra sauvegarder la position de votre bulle dans des variables et toutes les demies secondes éteindre la diode correspondant à leur position, mettre à jour la position en fonction des données du capteur, et allumer la diode correspondant à cette nouvelle position.

Projet jeu d'agilité

Ici on veut faire un jeu d'agilité où le `micro:bit` nous indique des actions à faire (penchez à droite, à gauche, secouer, appuyer sur A,...), et on perd si on ne fait pas l'action assez vite. Dans ce cas l'écran affiche "Perdu" ainsi que le score (qui correspond au nombre de mouvement réussi).

Pour que le jeu soit intéressant, les mouvements sont demandés dans un ordre aléatoire, et pour cela nous devons générer des nombres au hasard. Il faut rajouter tout en haut du fichier, juste après le `from microbit import *`, `from random import randint`. Après on peut utiliser la fonction `randint(a, b)` qui nous donne un nombre entier aléatoire entre a et b inclus. `r = randint(1, 6)` nous permet donc de stocker dans la variable `r` un nombre aléatoire entre 1 et 6 (inclus).

En fonction de `r` on pourra décider quel mouvement on souhaite, par exemple si `r == 1`, on veut qu'il penche à gauche, on le dit au joueur en affichant une flèche vers la gauche, on

laisse un peu de temps au joueur pour réagir, puis on teste s'il a bien penché à gauche. Pour tester si le joueur fait tel ou tel mouvement vous aurez besoin de lire la partie "capteur" de la section suivante.

Références `micro:bit`

Cette section détaille comment utiliser diverse fonctionnalités du `micro:bit`, si tu souhaites aller encore plus loin on peut trouver des informations plus complète sur la documentation officielle en ligne trouvable ici : https://bbcmicrobitmicropython.readthedocs.io/en/latest/.

Pour utiliser toutes ces fonctions il est nécessaire de les importer depuis le module `micro:bit` en ajoutant `from microbit import *` au début de ton programme.

L'écran

`display.clear()` - effacer l'écran

```
display.clear()  # l'écran est maintenant éteint
```

`display.set_pixel(x, y, value)` - allumer/éteindre un pixel

Change la luminosité d'une diode pour une valeur allant de 0 (diode éteinte), à 9 (luminosité maximale).

La diode est identifiée par sa colonne `x` et sa ligne `y` numérotée de 0 à 4.

```
display.set_pixel(2, 2, 9)  # allume la diode centrale
display.set_pixel(0, 0, 5)  # allume à moitié la diode d'en haut à gauche
display.set_pixel(4, 4, 0)  # éteint la diode d'en bas à droite
```

`display.get_pixel(x, y)` - calcule la luminosité d'un pixel

Réciproquement à `display.set_pixel(x, y, value)`, récupère la valeur de luminosité d'une diode identifiée par sa colonne `x` et sa ligne `y`.

```
display.set_pixel(2, 2, 9)  # allume la diode centrale
x = display.get_pixel(2, 2)  # `x` vaut maintenant 9
```


`display.show(image)` - afficher une image

Cette fonction sert à afficher une image, le plus simple est généralement d'utiliser une image parmi la liste prédéfinie : `Image.HEART`, `Image.HAPPY`, `Image.SMILE`, `Image.SAD`, `Image.YES`, `Image.NO`, … Une liste plus complète peut être trouvée ici :

https://microbit-micropython.readthedocs.io/en/latest/image.html#attributes)

```
display.show(Image.HAPPY)  # affiche un smiley
display.show(Image.HEART)  # affiche un coeur
```

Il est aussi possible de dessiner soi-même une image à partir d'un texte en séparant les ligne avec `:` et assignant une luminosité entre 0 et 9 à chaque diode :

```
bateau = Image('05050:'
               '05050:'
               '05050:'
               '99999:'
               '09990')

# Affiche une image qui ressemblera à peu près à ça:
#         x x
#         x x
#         x x
#        OOOOO
#         OOO
display.show(bateau)
```

`display.scroll(texte)` - afficher du texte

Fait défiler le texte donné en entrée.

```
display.scroll('Salut tous le monde!')  # affiche `Salut [...]!`
display.scroll(42)  # en réalité ça ne fonctionne pas qu'avec le texte!
```

Les boutons

Il y a deux boutons sur le `micro:bit`, ils sont appelés `button_a` et `button_b` et toute fonction qui peut être appelée pour l'un peut aussi être appelée pour l'autre.

`button_a.is_pressed()` - état du bouton

Retourne `True` si le bouton est actuellement enfoncé.

```python
while True:
    if button_b.is_pressed():
        # Allume la diode centrale si le bouton de droite est enfoncé
        display.set_pixel(2, 2, 9)
    else:
        # Éteint la diode si le bouton de droite n'est plus enfoncé
        display.set_pixel(2, 2, 0)
```

`button_a.was_pressed()` - le bouton a été enfoncé

Retourne `True` si le bouton a été enfoncé depuis la dernière fois que cette fonction a été appelée.

```python
display.scroll('Appuyez sur le bouton de gauche pour arrêter le programme')

# Si le bouton n'a pas été enfoncé pendant que le texte défilait, on peut
# continuer le programme
if not button_a.was_pressed():
    display.scroll('Et bien continuons le programme ...')
    # ...
```

`button_a.get_presses()` - nombre d'appuis sur le bouton

Retourne le nombre total d'appuis sur le bouton depuis la dernière fois que cette fonction a été appelée.

```python
sleep(5000)
nb_appuis = button_b.get_presses()
display.scroll(
    'Vous avez appuyé '
    + str(nb_appuis)
    + ' fois sur le bouton b en 5 secondes.'
)
```

Capteurs (boussole, accéléromètre)

`compass.heading()` - boussole

Retourne l'angle du `micro:bit` par rapport au nord (en degrés, donc de 0 à 360).

```
angle = compass.heading()

if 170 <= angle <= 190:
    display.scroll('Le micro:bit est dirigé vers le sud')
```

`accelerometer.current_gesture()` - geste / position

Retourne le nom du mouvement actuellement appliqué au `micro:bit`. Ce nom est une chaîne de caractère parmi les suivantes :

- `"up"` - haut
- `"down"` - bas
- `"left"` - gauche
- `"right"` - droite
- `"face up"` - à l'endroit
- `"face down"` - à l'envers
- `"freefall"` - chute libre
- `"3g"`, `"6g"`, `"8g"`
- `"shake"` - secoué

Il existe aussi des fonctions `accelerometer.is_gesture(nom)` et `accelerometer.was_gesture(nom)` qui fonctionnent de façon similaire à `button_a.is_pressed()` et `button_a.was_pressed()`.

Pour les mouvements haut, bas, gauche et droite, la fonction `accelerometer.current_gesture()` peut être assez imprécise et nous vous recommandons de plutôt utiliser `accelerometer.get_x()` et `accelerometer.get_y()` (fonction suivante).

```
# Si le micro:bit est face contre ciel, on affiche un smiley content.

if accelerometer.current_gesture() == 'face up':
    display.show(Image.HAPPY)

if accelerometer.current_gesture() == 'face down':
    display.show(Image.SAD)
```

`accelerometer.get_x()` (pour `x`, `y` ou `z`) - accéléromètre

Retourne l'accélération suivant l'axe x (existe aussi pour `y` et `z`). L'axe x correspond à l'axe gauche-droite, x est positif vers la droite et négatif vers la gauche. L'axe y correspond à l'axe haut-bas du `micro:bit`, si y est positif le `micro:bit` est penché vers l'avant, s'il est négatif il est penché vers l'arrière. L'axe z correspond à l'axe de la hauteur à laquelle se trouve le `micro:bit`.

Si le `micro:bit` est à l'horizontal, x est proche de 0, mais l'accéléromètre est très sensible donc x vaut rarement 0, On considère donc que s'il est entre -100 et 100, il est à l'horizontal. C'est une valeur arbitraire, on peut la changer pour plus ou moins de sensibilité.

```
# On affiche une flèche dans la direction où le microbit est penché.

while True:
    x = accelerometer.get_x()
    if x > 100:
        # Le micro:bit est penché vers la droite
        display.show(Image.ARROW_E)
    if x < -100:
        # Le micro:bit est penché vers la gauche
        display.show(Image.ARROW_W)
```

Radio

Pour utiliser le module de radio il faut ajouter en haut de votre fichier : `import radio`, votre code doit donc commencer par :

```
import radio
from microbit import *

# Votre code ici !
```

`radio.on()` - allume la radio

Cette fonction doit être appelée car la radio consomme pas mal d'énergie.

`radio.send(message)` - envoyer un message

Envoie un message sur le canal de radio.

```
radio.on()
radio.send('Salut, ça va?')
```

`radio.receive()` - recevoir un message

Retourne un message reçu sur le canal de radio. Si aucun nouveau message n'a été reçu, retourne `None`.

```python
radio.on()
message = None

# Attends jusqu'à avoir reçu un message
while message is None:
    message = radio.receive()

display.scroll(message)
```

`radio.config(...)` - paramétrer la radio

Permet de changer différent paramètres de la radio :

- `length` : la taille maximale d'un message, au maximum 251 (défaut : 32)
- `queue` : le nombre maximal de messages en attente d'être lu avec la fonction `radio.receive()` (défaut : 3)
- `channel` : le canal de communication à utiliser, la radio ne peut communiquer qu'avec des `micro:bit` qui utilisent le même paramètre, entre 0 et 83 (défaut : 7)

```python
radio.config(channel=42, length=251)
radio.send(
    "Wow, maintenant je peux envoyer des messages super longs sur le channel "
    "42, c'est vraiment génial d'avoir cette chance ! "
    "Merci radio.config() !"
)
```

Continuer la programmation chez soi

Si tu souhaites installer un logiciel de programmation sur ton ordinateur plutôt que d'utiliser un site web, nous te recommandons vraiment Mu editor (`https://codewith.mu/`), disponible sur Mac, Windows et Linux, facile à utiliser et possède un mode spécial pour les `micro:bit`.

Ce TP couvre les bases essentielles de la programmation, pour continuer nous te conseillons les ressources suivantes :

- France ioi (`http://www.france-ioi.org/`) : un site d'entrainement à la programmation qui propose des leçons suivis de problèmes à résoudre, la difficulté y est très progressive.
- Girls Can Code ! : Nous organisons des stages courts et des stages longs, tous les TPs des stages sont disponibles ici : https ://github.com/prologin/gcc-resources et bientôt ils seront détaillés dans une nouvelle section de notre site : `https://gcc.prologin.org/`.

— Prologin : un concours d'informatique pour les moins de 20 ans que nous organisons en plus des stages Girls Can Code !. Candidater ou s'entraîner sur les archives peuvent être un bon moyen de continuer à programmer. Si tu es sélectionnée les épreuves régionales seront l'occasion de revoir les organisateurs que tu as rencontré et la finale est un super événement, mais ce n'est pas facile !

Si un domaine de l'informatique t'intéresse en particulier n'hésite pas à demander aux organisateurs des références plus spécifiques.

Bonne continuation !