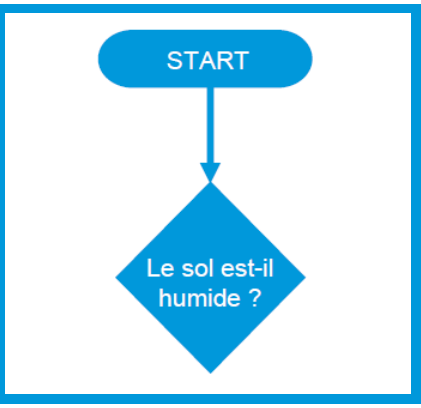
Belgium

esero

Fiche pour l'enseignant

Des plantes sur Mars

Construction d'un système d'arrosage automatique



Les élèves élaborent et programment un système d'arrosage automatique qui mesure l'humidité du sol et actionne un dispositif d'arrosage en conséquence, en utilisant un microcontrôleur Pyboard et le langage Python. Les bases de la programmation en Python sont introduites à l'aide de l'environnement Thonny.
Adaptation de la ressource T09 *Un potager sur Mars.*

esa

belspo

La Scientothèque
L'ÉGALITÉ DES CHANCES PAR LES SCIENCES

ULB

Table des matières

INFORMATIONS PRATIQUES

Tranche d'âge	14-18 ans
Type	Activité pratique (individuel ou en groupe)
Durée	6,5 heures
Lieu	Intérieur
Matériel nécessaire	Matériel informatique, électrique et de bricolage divers (détails dans le document)
Matières principales	Physique (électricité), géographie, biologie, technologie, programmation
Objectifs	• Aborder/revoir les notions de base en électricité : tension, courant, résistance • Comprendre la loi d'Ohm et le diviseur de tension • Faire le lien entre un schéma électrique et un circuit réel • Se familariser avec un multimètre, la breadboard, la Pyboard • Se familiariser avec la programmation – Notions abordées : code, programme, boucle, conditions, variables, fonction • Bases de la programmation en Python • Utiliser des capteurs pour prendre des mesures • Réaliser un montage complexe à partir d'éléments connus

Auteurs : Expérimentarium de physique de l'ULB (ESERO Belgium)
La Scientothèque (ESERO Belgium)

Date de publication : Décembre 2021

BIENVENUE SUR MARS

30 min

Introduction

Cette activité présente aux élèves le contexte d'une mission spatiale sur Mars, et les défis qui pourraient être associés à la vie sur Mars. Les différences entre la Terre et Mars et ce que cela signifie pour les êtres vivants sont discutées, et les élèves sont invités à réfléchir à ce qui est nécessaire pour maintenir la vie.

Informations générales

D'après ce que nous savons déjà sur Mars, il serait difficile d'imaginer que la vie qui a évolué sur Terre puisse survivre dans l'environnement martien. Malgré une inclinaison axiale proche de celle de la Terre (25° contre 23°), offrant des saisons similaires à celles que nous connaissons sur Terre, l'absence d'océans pour aider à réguler la température de surface et une atmosphère mince (environ 1 % de la densité de celle de la Terre) signifie que la température varie beaucoup du jour à la nuit.

L'orbite de Mars est également beaucoup plus excentrique (elliptique) que celle de la Terre, ce qui signifie qu'à certaines périodes de l'année, elle est beaucoup plus proche du Soleil qu'à d'autres, aggravant le problème des variations extrêmes de température. La minceur de l'atmosphère et le manque d'ozone, combinés à l'absence de protection contre un champ magnétique, font que la surface de Mars est bombardée de rayons UV nocifs et de vents solaires. Les recherches d'une ressource vitale, l'eau liquide, à la surface de Mars, ont jusqu'à présent été infructueuses. Il y a cependant des signes d'une quantité importante de glace d'eau.

Pour aggraver encore les problèmes, le CO2 représente environ 96% de l'atmosphère, certainement trop élevé pour les animaux sur Terre, et trop élevé pour de nombreuses plantes. Si nous voulons faire pousser des plantes sur Mars, nous devrons peut-être intégrer nos technologies et outils modernes afin de créer des plantes sophistiquées, les habitats artificiels et les systèmes d'irrigation.

Toutefois, il existe plusieurs facteurs positifs. Tout d'abord, la durée d'un jour martien est très proche de celle d'un jour terrestre, puisqu'elle est de 24 heures et 37 minutes. Cela signifie que les cycles de photosynthèse-respiration des plantes resteraient largement les mêmes. De plus, bien qu'elle soit plus éloignée du Soleil que la Terre, Mars reçoit encore suffisamment de lumière solaire pour permettre à une plante de faire de la photosynthèse. Combiné à l'eau qui pourrait éventuellement être extraite des pôles glacés de Mars, nous aurions deux des composants vitaux nécessaires pour soutenir une plante. Cela pourrait potentiellement réduire la quantité de matériaux à transporter à bord de l'engin spatial.

Enfin, la zone des « Godilocks » ou « Boucle d'or » est définie comme la zone habitable autour du Soleil, car dans cette zone, la plage de température permet à l'eau d'exister sous forme liquide sur une planète en orbite. Le nom vient du conte de fées *Boucle d'or et les trois ours*, dans lequel une petite fille choisit de manger un bol de porridge qui n'est ni trop chaud, ni trop froid, mais juste à la bonne température !

Réponses à l'exercice

1. *Pour commencer à y réfléchir plus en détail, énumérez quelques-unes des choses dont les plantes et autres organismes vivants ont besoin pour survivre.*

   Voici les principales choses dont les plantes et autres organismes vivants ont besoin pour survivre et que les élèves doivent identifier :
   - Une source d'énergie (nourriture pour les animaux et lumière du soleil pour les plantes)
   - Eau
   - Nutriments
   - Oxygène
   - Dioxyde de carbone (nécessaire à la photosynthèse des plantes)

   Ils peuvent également discuter de choses telles que l'abri, la chaleur et la sécurité dans leur environnement. Ces aspects sont tous pertinents et peuvent être liés à une discussion plus approfondie sur les écosystèmes et l'environnement.

2. *Discutez avec vos camarades de classe et votre professeur des réponses que vous pensez avoir aux questions suivantes sur la Terre.*
   - *Quelles sont les causes des saisons sur la Terre ?*
   - *Quelle est la forme de l'orbite de la Terre autour du Soleil ?*
   - *Quels sont les principaux éléments présents dans l'atmosphère terrestre ?*
   - *Qu'est-ce que la zone de la Boucle d'or et la Terre se trouve-t-elle à l'intérieur de cette zone ?*

   Les questions de cet exercice visent à vérifier que les élèves comprennent déjà certaines caractéristiques de base de la Terre. Ils devraient consolider leurs connaissances existantes, mais vous pourriez en profiter pour clarifier les malentendus courants, en particulier la cause des saisons terrestres, en utilisant les informations de base ci-dessus.

3. *Décidez si les affirmations suivantes sont vraies ou fausses.*

   En utilisant les informations de base ci-dessus et les résultats des discussions de l'exercice précédent, les élèves doivent remplir le tableau ci-dessous et examiner les justifications de leurs réponses.

Déclarations	Vrai ou Faux
Mars connaît des saisons, tout comme la Terre.	Vrai
L'orbite de Mars a une forme similaire à celle de la Terre, ce qui signifie que la température à la surface est assez constante. **L'orbite de Mars est beaucoup plus excentrique, ce qui signifie que la température varie beaucoup plus que sur Terre.**	Faux
L'atmosphère de Mars est épaisse et retient la chaleur du Soleil. **L'atmosphère de Mars est très mince, ce qui signifie que la température baisse considérablement pendant la nuit.**	Faux
Mars n'a pas de champ magnétique, ce qui signifie qu'il y a moins de protection contre les rayons UV nocifs et les vents solaires.	Vrai

Nous avons trouvé de l'eau liquide à la surface de Mars. **Nous avons trouvé des signes d'eau gelée près des pôles, mais pas d'eau liquide.**	Faux
L'atmosphère de Mars a une composition similaire à celle de la Terre. **L'atmosphère martienne contient un pourcentage de CO_2 beaucoup plus élevé que l'atmosphère terrestre et presque pas d'oxygène.**	Faux
Les plantes de Mars devraient s'adapter aux cycles diurnes et nocturnes très différents sur Mars. **Le jour martien est de 24 heures et 37 minutes, les cycles du jour et de la nuit sont donc très similaires à ceux de la Terre.**	Faux
Mars n'existe pas à l'intérieur de la zone "Boucle d'or" (habitable), il est donc impossible que de l'eau liquide existe à la surface. **Mars existe juste à la limite de la zone habitable, il est donc possible que de l'eau liquide existe à sa surface.**	Faux

PRÉPARATION DES COMPOSANTS, CONCEPTION ET PREMIERS TESTS 1h

Les élèves doivent réfléchir à la manière dont ils concevraient un système d'arrosage automatique. Ils reçoivent une liste des matériaux fournis et des connaissances sur le fonctionnement de chacun des composants.

Liste du matériel :

- Pyboard
- Ordinateur portable + câble micro-USB
- Capteur d'humidité du sol
- Servo-moteur (mini 3-5 V)
- Breadboard
- Un seau
- Tube (un tube d'irrigation fin est parfait)
- Sol/plante
- Câbles Dupont
- Ciseaux/couteaux d'artisanat
- Bouteille vide
- Pâte adhésive/papier collant
- Attaches de câble

Réponses à l'exercice

Vous devez vous attendre à un large éventail de propositions dans le cadre de cet exercice. Si certaines idées ne sont pas réalisables, beaucoup d'autres pourraient être mises en oeuvre. Il est probable que le projet des étudiants ne soit pas leur projet final, et les étudiants ne doivent pas être découragés s'ils doivent modifier leur plan tout au long des activités, car cela fait partie du processus. En tant qu'enseignant, vous devez vérifier s'ils ont réfléchi aux questions posées et si leur proposition a du sens. Dans la logique du projet, il est important que certains éléments figurent sur leurs croquis, peu importe la forme : un réservoir d'eau, un système de fermeture/ouverture automatique, une mesure de l'humidité du sol.

Conception et test des modèles des élèves

Dans cette activité, les élèves introduisent de l'eau dans leurs prototypes de systèmes pour voir comment leur conception se comporte. Ils sont guidés dans la construction d'un modèle spécifique à adapter selon leur conception. Cela permet aux élèves de passer par un processus scientifique itératif de conception et de construction d'un système.

PRISE EN MAIN DE PYBOARD 2h

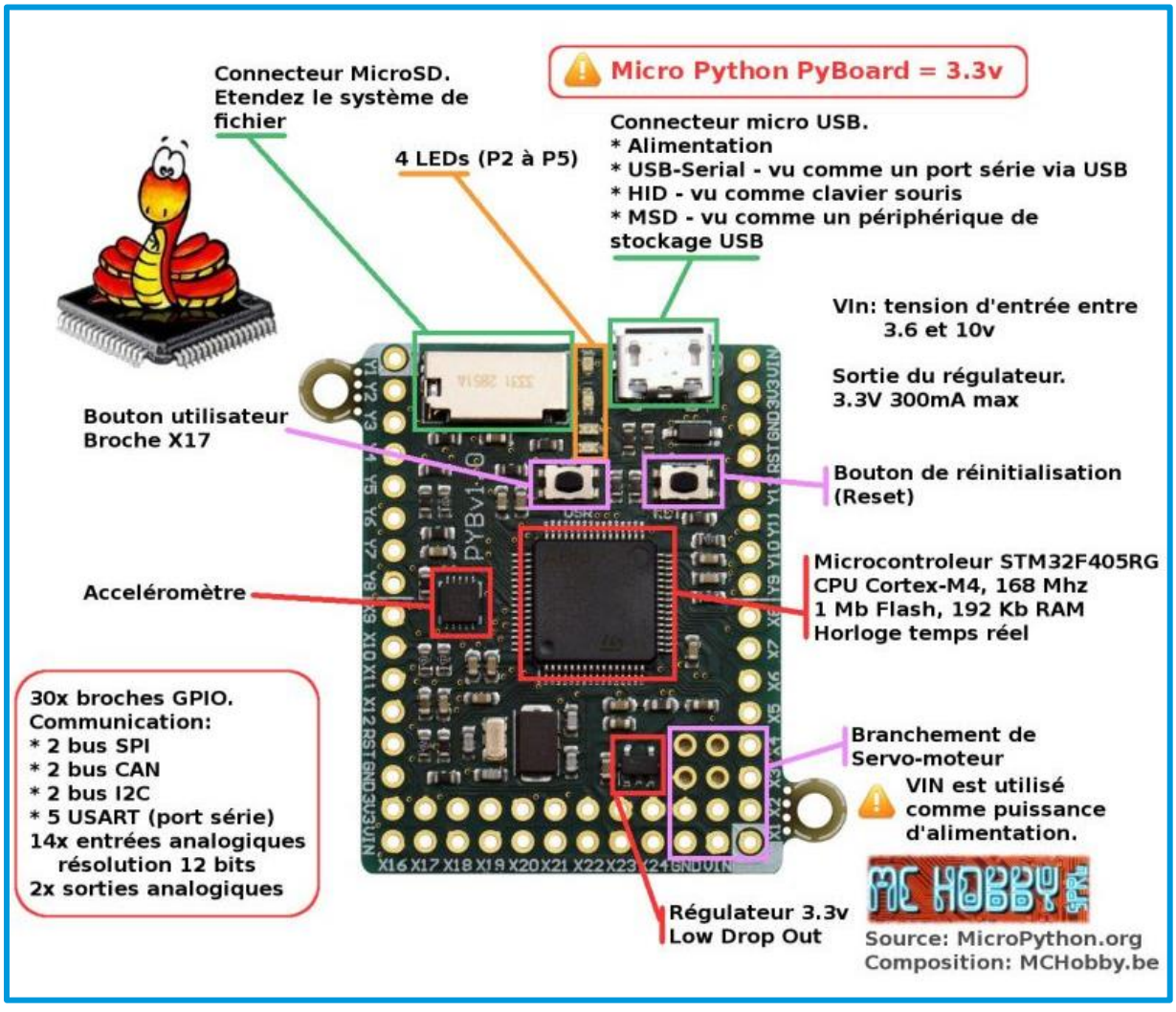
Pyboard

La présentation que nous faisons ici est très succincte et directement en lien avec ce projet. Pour avoir une vision complète et détaillée, nous vous invitons à lire le livre « Micropython et Pyboard » de D. Meurisse.

Pyboard est une carte de type microcontrôleur conçue pour le développement sous MicroPython. Elle est très facile à utiliser grâce à son système de fichiers (elle se comporte comme une clef USB quand on la branche) et son interpréteur python (accessible sur un port série) permet de prototyper très rapidement.

Le port micro-usb est l'alimentation et doit être connecté à un ordinateur via un câble micro-USB pour télécharger le code de l'ordinateur vers la carte.

Elle possède 4 LEDS intégrées et un accéléromètre. Les broches permettent de brancher des éléments extérieurs à la carte. Ce sont les « entrées » et « sorties » (soit digitales soit analogiques). Typiquement un capteur est une entrée et un actionneur est une sortie. Notons



qu'elle possède des entrées spécifiques pour les servo-moteurs, lesquels nous seront utiles dans ce projet.

Le bouton « user » est un élément programmable, contrairement au bouton « reset » qui n'est pas programmable et sert à réinitialiser la carte.

Logiciel

Thonny est un environnement de développement intégré (IDE, *Integrated Development Environment*) pour Python mis au point par l'Université de Tartu en Estonie. Il est spécialement conçu pour les débutants.
Téléchargement et installation sur l'ordinateur : https://thonny.org/.

Matériel

Par groupe/participant :

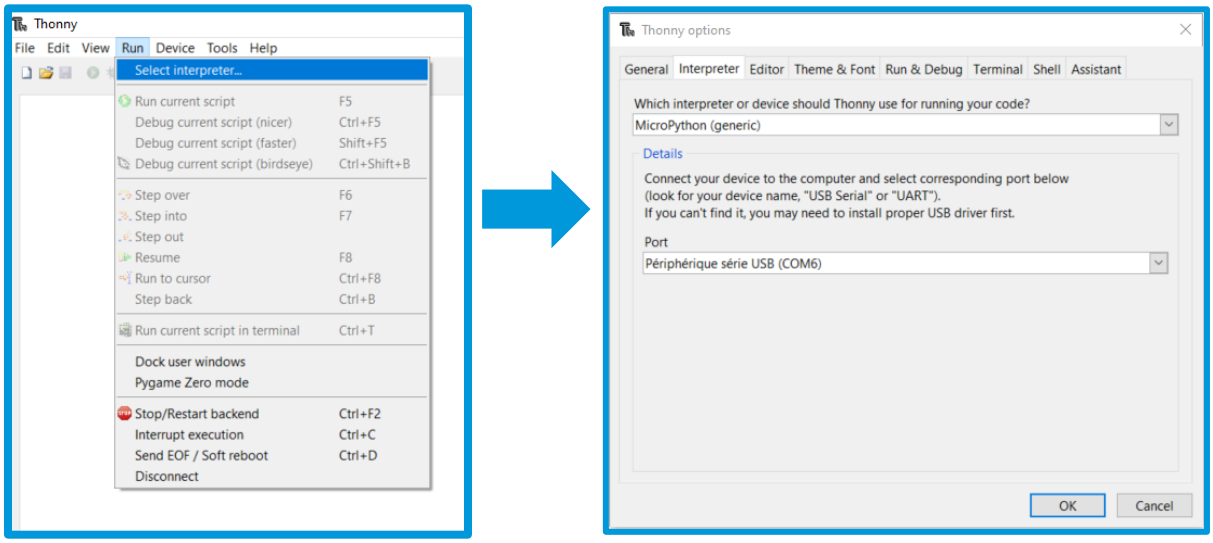
- 1 Pyboard
- 1 PC avec Thonny installé
- 1 câble Micro-USB

Défi 0 : Brancher Pyboard à l'ordinateur et le sélectionner comme interpréteur

Par défaut, Thonny utilise le programme comme interpréteur (c'est-à-dire comme entité qui exécutera le code). Or nous voulons utiliser Pyboard.

Afin de définir l'interpréteur, suivez les instructions suivantes :

a. Allez dans l'onglet *Exécuter* (ou *Run* si le programme est en anglais) et, dans le menu déroulant, cliquez sur « Sélectionner l'interpréteur ». Une fenêtre s'ouvrira sur l'onglet *Interpréteur* (si ce n'est pas le cas, sélectionnez l'onglet *Interpréteur*).
b. Dans le menu déroulant « Quel interpréteur ou appareil Thonny doit-il utiliser pour exécuter votre code ? », sélectionnez « MicroPython (generic) ».
c. Si ce n'est pas encore fait, connectez Pyboard à l'ordinateur à l'aide du câble Micro-USB.
d. Dans le menu déroulant « Port », sélectionnez « Périphérique série USB (COM6) » (le numéro après COM dans les parenthèses sera différent).



Remarque : Thonny garde toujours le dernier interpréteur sélectionné. Si, par la suite, vous voulez ré-utiliser le programme comme interprétateur, vous devez donc refaire la procédure pour le sélectionner.

Défi 1 : Écrire une phrase dans la console

La fonction « print » permet d'écrire dans la console. Ici nous choisissons d'écrire du texte mais on peut imprimer des valeurs de variables, de mesures, etc.

```
print ('Hello world')
# Ceci est un commentaire
# Ce code ne fonctionnerait pas : print (bienvenue à tous)
# Celui-ci non plus : print 'bienvenue à tous'
```

Placer le signe # permet d'insérer un commentaire dans le code, c'est-à-dire des lignes qui ne sont pas du code à proprement parler et seront ignorées lors de l'exécution du programme mais qui peuvent servir à expliquer une ligne de code à destination des autres utilisateur·rices. Dans l'exemple ci-dessus, les 3 dernières lignes de couleur grise sont des commentaires qui donnent des explication supplémentaires. Elles ne doivent pas être recopiées lors de l'écriture du programme.

Insistez sur l'importance de la syntaxe : bien écrire tous les caractères, respecter les majuscules, etc.

→ **Exercice :** écrire une phrase au choix à l'aide de la fonction « print ».

→ **Solution :** ils recopient la première ligne de l'exemple ci-dessus en remplaçant la phrase « Hello world » par celle de leur choix.

Défi 2 : Allumer les leds intégrées à la carte

« Import » permet d'importer des fonctions spécifiques. Ici, nous allons importer « pyb » qui est l'ensemble des fonctions de Pyboard.

Certaines fonctionnalités sont incluses dans d'autres « registres » : par exemple, « LED » est l'ensemble des fonctions des leds de Pyboard ; il se trouve dans le registre « pyb ».

```
import pyb # permet d'utiliser des fonctionnalités propres à Pyboard
from pyb import LED # permet d'utiliser les fonctions liées aux leds intégrées sur Pyboard
LED(1).on() # allume la led numéro 1 (il y a 4 leds au total).
```

→ **Exercice :** identifier la couleur de chacune des 4 leds.

→ **Solution :** ils recopient le code en changeant le numéro de la led (de 1 à 4) et identifient la couleur de la led qui s'allume dans chaque cas afin d'associer une couleur à chaque numéro de led.
On trouve : LED(1) : rouge – LED(2) : vert – LED(3) : jaune – LED(4) : bleu

Notez que, une fois les quatre programmes exécutés, les leds restent allumées. En effet, la carte ne revient pas automatiquement à l'état de départ. Si la dernière instruction donnée à une led est de s'allumer, elle restera allumée jusqu'à ce qu'une instruction contraire soit donnée ou que la carte soit réinitialisée.

Défi 3 : Éteindre toutes les leds de la carte

→ **Exercice :** éteindre toutes les leds en même temps.

→ **Solution :** remplacer « on » par « off » pour chaque led.

```
import pyb
from pyb import LED
LED(1).off()
LED(2).off()
LED(3).off()
LED(4).off()
```

Défi 4 : Insérer des délais

La bibliothèque « time » comprend des fonctions liées au temps telles que lancer un chronomètre, poser des délais, etc. Dans ce défi, nous aurons recours à la fonction « sleep » qui permet d'introduire des délais dans l'exécution des instructions du programme.

```
import pyb
import time # importe l'ensemble des fonctions liées au temps
from pyb import LED
print ('0')
time.sleep(1) # met le programme "en pause" pour 1 seconde : le programme ne s'arrête pas vraiment, mais attend 1 seconde avant d'exécuter l'instruction suivante. L'écriture "time.sleep" signifie que l'on va chercher dans la bibliothèque "time" la fonction « sleep »
print ('1')
time.sleep(1)
print ('2')
```

→ **Exercice :** allumer la led 1 pendant 1 seconde puis l'éteindre et allumer la led 2 pendant 2 secondes, l'éteindre, attendre 1 seconde et allumer la led 3.

→ **Solution :**

```
import pyb
import time
from pyb import LED
LED(1).on()
time.sleep(1)
LED(1).off()
LED(2).on()
time.sleep(2)
LED(2).off()
time.sleep(1)
LED(3).on()
```

Insistez bien sur la différence entre « éteindre et allumer » et « éteindre, attendre et allumer » pour mettre en évidence l'importance du placement du « sleep », et de l'ordre des instructions de façon générale.

Défi 5 : Utiliser une variable

Une variable est une donnée dont la valeur est susceptible de changer durant l'exécution du code. Attention à ne pas faire d'amalgame avec une variable mathématique. En programmation, une variable a une valeur précise. On ne le connait pas spécialement à tout moment (en particulier dans un cas comme le nôtre où nous utilisons des valeurs aléatoires) mais elle est définie en amont, soit dans le code (comme dans le cas présenté ici), soit via des instruments de mesures (thermomètre, capteur de lumière, etc.).
Dans l'exemple ci-dessous, nous introduisons une variable que nous nommons « temps ».

```
import pyb
import time
from pyb import LED
from time import sleep # importe la fonction « sleep » ou attendre. Cela permet de ne pas avoir à taper « time.sleep » dans le code ; on peut simplement écrire « sleep ». L'unité est en secondes.
from random import * # importe la bibliothèque « * » des fonctionnalités aléatoires
temps = random() # on définit la variable « temps » : sa valeur est donnée par la fonction « random() » qui génère un nombre aléatoire dans la plage [0,0 ; 1,0].
# insérer une fonction
sleep(temps)
# insérer une fonction
```

→ **Exercice :** allumer les leds rouge, verte et jaune à intervalles réguliers, mais aléatoires, et imprimer la valeur de l'intervalle dans la console.

→ **Solution :**

```
import pyb
import time
from pyb import LED
from time import sleep
from random import *
temps = random()
print(temps) # écrit la valeur de la variable « temps » dans la console. Notez qu'il ne faut pas mettre de guillemets autour de temps, sinon c'est le mot temps qui sera écrit dans la console.
sleep(temps)
LED(1).on()
sleep(temps)
LED(2).on()
sleep(temps)
LED(3).on()
```

Défi 6 : Utiliser des « boucles »

La fonction « boucle » n'existe pas en tant que telle pour Pyboard mais il y a plusieurs façons de programmer une boucle.
Deux programmation de boucle sont présentées ci-dessous ; chacune a ses avantages et ses inconvénients.

Faire une boucle avec « for » :

```
import pyb
import time
from pyb import LED
from time import sleep
from random import *
temps = random()
i=1
for i in range(1,4):
    print(i)
    sleep(temps)
```

Faire une boucle avec « while » :

```
import pyb
import time
from pyb import LED
from time import sleep
from random import *
temps = random()
i=1
while(i < 5):
    print(i)
    sleep(temps)
    i=i+1
```

Si l'on traduit en texte la boucle établie avec la fonction « for », cela donne :

> *Pour i allant de 1 à 4, écrire sur la console la valeur de i, puis attendre un nombre de secondes égal à la valeur de la variable « temps ».*

Pour la boucle établie avec la fonction « while », on lit :

> *Tant que i est plus petit que 5, écrire sur la console la valeur de i, puis attendre un nombre de secondes égal à la valeur de la variable « temps », puis ajouter 1 à la valeur de i.*

La fonction « for » incrémente automatiquement la variable de 1 (la valeur de la variable augmente de 1). L'utilisation de « while » nécessite de redéfinir la variable en fin de boucle, ce qui représente une ligne de code supplémentaire mais permet de travailler avec des décimales.

Notez que les lignes sous les fonctions « for » et « while » sont décalées, on parle d'« indentation » : on met des espaces en début de ligne. Toutes les lignes indentées sont incluses dans la ligne supérieure, elles forment le bloc lié à la fonction. Dans le cas des boucles ci-dessus, les lignes indentées sont les lignes qui seront répétées. Pour sortir du bloc et continuer à écrire le code, il suffit d'écrire les lignes sans indentation, sans mettre d'espaces au début de la ligne donc.

Discutez de l'avantage d'utiliser ce type de procédure par rapport à simplement répéter plusieurs fois la séquence désirée : dans notre cas, copier/coller fonctionne. Mais si on doit répéter mille fois une séquence, le code ne sera plus très lisible et ce

sera très long à écrire (et si on doit répéter une séquence à l'infini, on n'y arrivera jamais !).

→ **Exercice :** faire clignoter 4 fois une led avec un intervalle de temps aléatoire.

→ **Solution :**

```
import pyb
import time
from pyb import LED
from time import sleep
from random import *
temps = random()
print(temps)
i=1
while(i < 5):
    print(i)
    LED(1).on()
    sleep(temps)
    LED(1).off()
    sleep(temps)
    i=i+1
```

ou

```
import pyb
import time
from pyb import LED
from time import sleep
from random import *
temps = random()
print(temps)
i=1
for i in range(1,6):
    print(i)
    LED(1).on()
    sleep(temps)
    LED(1).off()
    sleep(temps
```

Défi 7 : Faire un blink infini

L'idée est que les élèves cherchent un peu par eux-mêmes une façon de rendre une boucle infinie. L'astuce consiste à utiliser la fonction « while » avec un argument qui est toujours vrai. On arrive parfois à des solutions telles que : « while(1=1) », ce qui est une bonne façon de procéder car le code se répètera tant que 1 est égal à 1, et 1 est toujours égal à 1.

Cependant, il existe en Python la boucle « while True » qui s'exécute sans condition. Celle-ci est utilisée dans la solution de l'exercice ci-dessous (mais les solutions faisant appel à des aurguments toujours vrais trouvées par les élèves sont tout aussi valable).

→ **Exercice :** faire clignoter une led indéfiniment.

→ **Solution :**

```
import pyb
import time
from pyb import LED
from time import sleep
from random import *
temps = random()
print(temps)
while True:
    LED(1).on()
    sleep(temps)
    LED(1).off()
    sleep(temps)
```

Défi 8 : Utiliser des conditions

Les conditions permettent de réaliser des actions différentes en fonction du résultat d'une mesure, d'un état, …
S'il n'y a qu'une condition, on emploiera juste un « if », et si il n'y en a que 2, on ajoutera un « else ». Si les conditions se multiplient, il faudra utiliser des « elif » entre les deux fonctions précédentes. Il remplit la même fonction que les « if » mais vient après ceux-ci. Si on ne mettait que des « if », plusieurs peuvent s'actionner en même temps (c'est parfois ce qu'on recherche mais pas dans notre cas). Ici, si le « if » s'active, le code ne prendra pas les « elif » ni le « else » en compte.

```
import pyb

while True:

    sleep(1)

    test=randint(1,6) # on crée une variable appelée test qui a une valeur aléatoire entre 1 et 6.

    if 0 <test<=2: # « if » permet d'établir une condition. Si la condition est remplie, tout ce qui est indenté dans le « if » sera exécuté.

    # Il y a plusieurs possibilités pour écrire cette instruction : « if 1<=test<=2 » est correct aussi, etc.

        print(1)

    elif 2 <test<=4: # « elif » est la contraction de « else » et « if ». Il agit comme un « if », mais doit être précédé d'un « if ». Il peut y avoir plusieurs « elif ».

        print(2)

    else # else veut dire « sinon » et couvre tous les cas qui ne rentrent pas dans les conditions des « if » et « elif » qui le précèdent.

        print(3)
```

→ **Exercice :** Tirer un nombre aléatoire entre 1 et 10 et, en fonction du résultat :
1 ou 2 : allumer la led verte pendant 2 s
3 ou 4 : allumer la led jaune pendant 2 s
5 ou 6 : allumer la led rouge pendant 2 s
7 ou 8 : allumer la led bleue pendant 2 s
9 ou 10 : guirlande (= blink vert, jaune, rouge, bleu ; délai 0,25 s)

→ **Solution :**

```
import pyb
while True:
    sleep(1)
    test= randint(1,10)
    print(test)
    if 0<test<=2:
        LED(2).on()
        sleep(2)
        LED(2).off()
    elif 2<test<=4:
        LED(3).on()
        sleep(2)
        LED(3).off()
    elif 4<test<=6:
        LED(1).on()
        sleep(2)
        LED(1).off()
    elif 6<test<=8:
        LED(4).on()
        sleep(2)
        LED(4).off()
    else:
        LED(2).on()
        sleep(0.25)
        LED(2).off()
        LED(3).on()
        sleep(0.25)
        LED(3).off()
        LED(1).on()
        sleep(0.25)
        LED(1).off()
        LED(4).on()
        sleep(0.25)
        LED(4).off()
```

Défi 9 : Utiliser des états pour un capteur

Nous allons utiliser ici le bouton poussoir dont la sortie est un état logique : les valeurs possibles sont soit « True » (lorsqu'il est enfoncé) soit « False » (lorsqu'il est relâché).

```
import pyb
from pyb import LED
from pyb import Switch # la fonction « Switch » permet d'utiliser les boutons poussoirs intégrés à Pyboard.
sw= Switch()
while True:
    print(sw.value()) # les "valeurs" possibles sont soit « True » (enfoncé), soit « False » (relâché).
```

→ **Exercice :** Utiliser le bouton poussoir pour allumer ou éteindre une led.

→ **Solution :**

```
import pyb
from pyb import LED
from pyb import Switch
sw= Switch
while True:
    if sw.value()== True:
        LED(4).on()
    else:
        LED(4).off()
```

Programmation d'éléments extérieurs à Pyboard

2h

Matériel

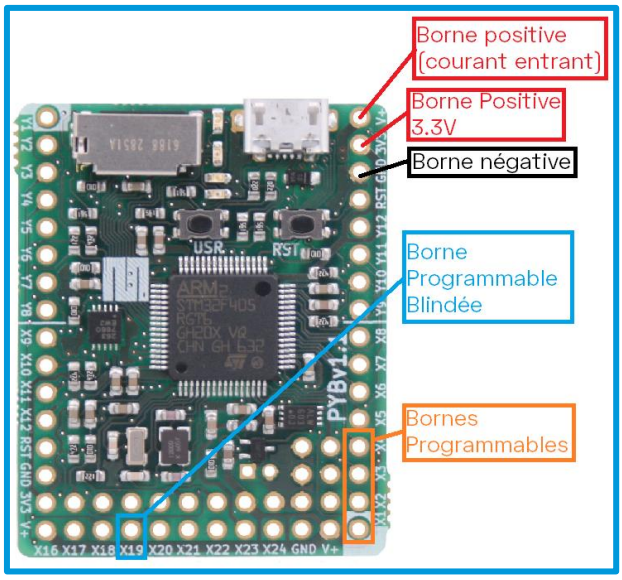
- Pyboard
- PC avec Thonny installé par groupe
- Câble Micro-USB
- Breadboard
- Potentiomètre (1 ou 2 kOhm)
- capteur d'humidité de sol
- Micro servo-moteur
- Des leds
- Des résistances de 220 Ω
- Jeu de câbles Dupont

Défi 0 : Connecter une breadboard à Pyboard

En plus des éléments utilisés précédemment, nous serons amenés à brancher différents composants électriques aux bornes de Pyboard.
Nous distinguons 3 types de bornes qui seront utilisées dans les défis suivants :

- Les bornes positives (3V3 et V+) et négative (GND)
- Les bornes programmables
- Les bornes programmables blindées (X19 à X22)

Les bornes que nous utiliserons dans la suite de l'activité sont indiquées ci-dessous.



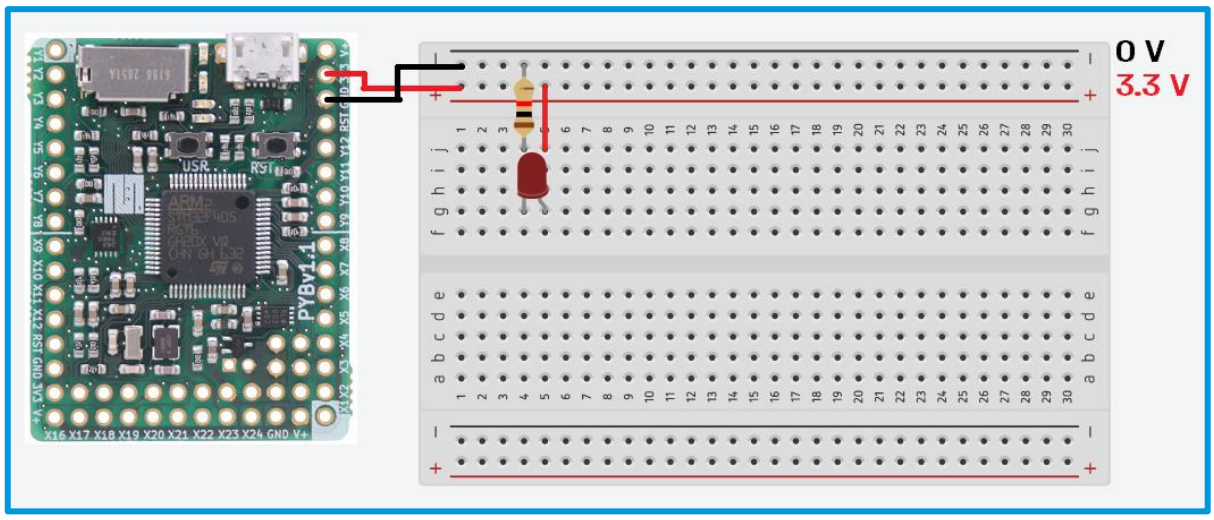
Les bornes GND, 3V3 et V+ ne sont pas programmables. Leur seule fonction est de servir de borne, comme une pile. La borne GND a une tension de 0 V, la borne 3V3 délivre 3,3 V tandis

que la tension délivrée par la borne V+ dépend du courant entrant (si la carte Pyboard est branchée à un port USB d'un ordinateur, la tension de la borne V+ sera de 5 V).
Parmi les bornes programmables, nous utiliserons les bornes X1 à X4 qui permettent, entre autres, de commander des servo-moteurs.
Les bornes programmables dites « blindées » (comme la X19) sont moins sensibles aux interférences.

Pour brancher la plaque d'essai (breadboard) à Pyboard et réaliser le premier circuit, suivez les étapes suivantes (schéma du montage ci-dessous) :

a. Connecter la borne positive 3V3 à la ligne « + » de la breadboard et la borne négative GND à la ligne « - ». Pour respecter les conventions, utilisez des connecteurs rouges pour relier des éléments à la borne positive et bleus ou noirs pour la borne négative.
b. Placez une led (*light-emitting diode* ou diode électroluminescente) reliée d'un côté au pôle négatif et de l'autre au pôle positif. La longue patte de la led, l'anode, doit être connectée au pôle « + ».
c. On met une résistance dans le circuit pour limiter le courant qui traverse la led afin de ne pas dépasser son courant nominal, ce qui aurait pour conséquence de la griller. La résistance peut se mettre avant ou après la led ; tant que les deux éléments sont en série, il n'y a pas de problème (attention ce n'est pas vrai pour tous les composants : notamment, si l'on veut faire des mesures, se trouver avant ou après une résistance peut tout changer).
d. Brancher Pyboard à l'ordinateur.

La LED s'allume normalement sans la moindre programmation. Elle est branchée sur une borne négative d'un côté et positive de l'autre, comme à une pile.



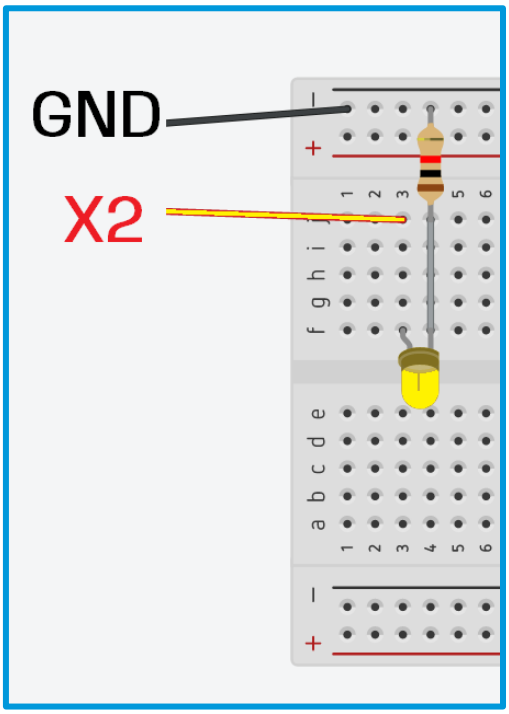
Avant de brancher Pyboard à l'ordi et de lancer des programmes, insistez pour que les élèves demandent la vérification du montage. A priori, il y a des sécurités, mais des courts-circuits peuvent malgré tout endommager la carte ou le PC.

Rem. : Vous trouverez une description plus complète de la breadboard et des leds dans la ressource *Meet Arduino!*.

Défi 1 : Allumer une led extérieure à la carte

Les leds se branchent sur les broches X1 à X4. Dans l'exemple suivant, la led est connectée à la broche X2.

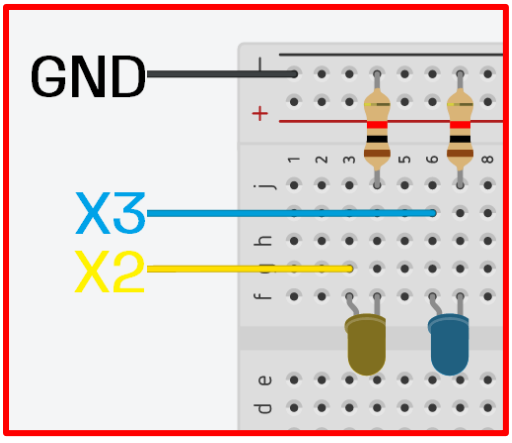
```
import pyb
from pyb import Pin # importe les fonctions relatives aux "Pins", càd les broches
Y = Pin.board.X2 # on donne le nom « Y » à la broche X2
pyb.Pin(Y , Pin.OUT) # on définit « Y » comme une sortie, c'est-à-dire qu'on pourra lui envoyer des informations.
while True:
    Y.value(1) # on définit la valeur de « Y » sur 1, ce qui veut dire un état haut, c'est-à-dire que le courant passe : la Led branchée sur cette broche sera allumée. Si on veut que la led soit éteinte, donc que le courant ne passe pas, on définit la valeur de la broche sur 0 (état bas).
```



→ **Exercice :** Brancher une led supplémentaire sur la broche X3, l'allumer et éteindre Y.

→ **Solution :**

```
import pyb
from pyb import Pin
Y = Pin.board.X2
B = Pin.board.X3
pyb.Pin(Y , Pin.OUT)
pyb.Pin(B , Pin.OUT)
while True:
    B.value(1)
    Y.value(0)
```

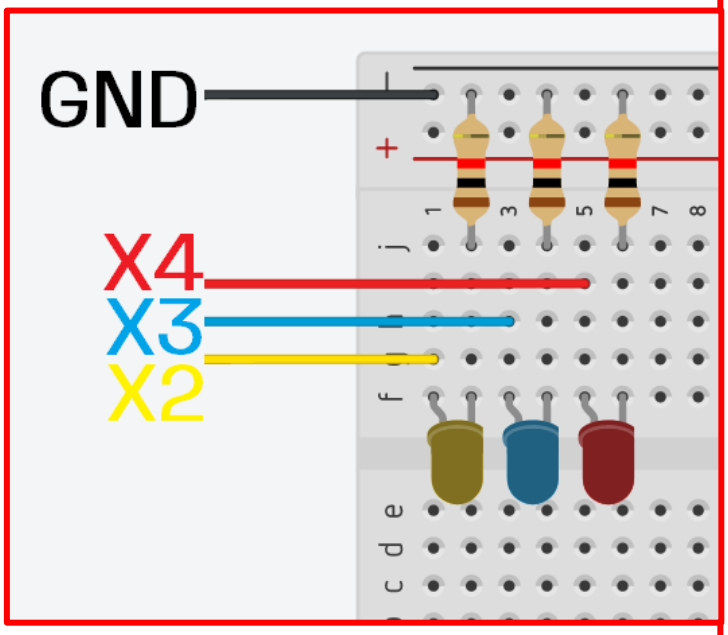


Défi 2 : Allumer 4 leds en alternance

→ **Exercice :** Brancher une troisième led (sur la broche X4) et allumer les 3 leds sur la breadboard et la led 1 intégrée à Pyboard en alternance pendant 0,5 s.

→ **Solution :**

```
import pyb
import time
from pyb import Pin
from pyb import LED
from time import sleep
Y = Pin.board.X2
B = Pin.board.X3
R = Pin.board.X4
pyb.Pin(Y , Pin.OUT)
pyb.Pin(B , Pin.OUT)
pyb.Pin(R , Pin.OUT)
while True:
    Y.value(1)
    sleep(0.5)
    Y.value(0)
    B.value(1)
    sleep(0.5)
    B.value(0)
    R.value(1)
    sleep(0.5)
    R.value(0)
    LED(1).on()
    sleep(0.5)
    LED(1).off()
```



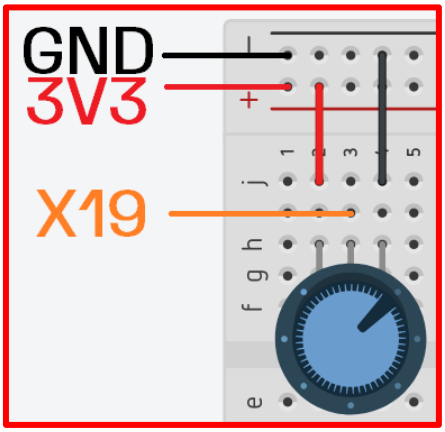
Défi 3 : Prendre des mesures

```
import pyb
from pyb import Pin
from pyb import ADC # ADC (Analog to Digital Converter) est un ensemble de fonctions particulier qui permet de lire des valeurs analogiques et les convertit en valeurs digitales.
adc = ADC( 'X19' ) # on définit la broche  X19 comme « lecteur », qui sera traduit avec les fonctions ADC
import time
from time import sleep
while True:
    A = adc.read() # on crée une variable « A » qui est égale à la valeur obtenue via adc défini plus haut. Il est important de définir cette valeur dans la boucle sinon la valeur est fixée avant et, une fois dans la boucle, même si la valeur enregistrée varie (adc), la valeur « A » ne changera pas.
    print(A)
    sleep(1) # on définit ici un sleep pour que le print ne se fasse pas en continu, ce qui au bout d'un moment fait planter le programme.
```

La broche X19 est une entrée analogique qui permet de lire la valeur d'une tension (située entre 0 et 3,3 V) présente sur la broche. Le convertisseur analogique (ADC) convertit la valeur analogique en une valeur numérique. En informatique, une valeur est décrite en bits. Un bit, c'est deux valeurs possibles : 0 ou 1. Si une information est stockée sur 2 bits, on a 2^2 = 4 combinaisons possibles, sur 3 bits, 2^3 = 8 combinaisons, etc. 2. Dans le cas de Pyboard, l'information est stockée sur 12 bits, ce qui fait 2^12 combinaisons, soit 4096. Les valeurs numériques renvoyées par le convertisseur varient donc de 0 à 4095 pour des tensions entre 0 et 3,3 V. Ainsi 0 correspond à une mesure de 0 V et 4095 à 3,3V.

→ **Exercice :** Brancher un potentiomètre sur la broche X19 et observer la variation des valeurs mesurées en fonction de la position du curseur.

→ **Solution :** Le code ne change pas par rapport à celui donné ci-dessus. Il faut juste observer les valeurs dans la console.



Défi 4 : Tester le capteur d'humidité

Afin d'automatiser entièrement le système d'arrosage des plantes, nous devons savoir quand la plante a besoin d'être arrosée. Dans cette activité, les élèves sont donc initiés au capteur d'humidité du sol et vont le calibrer pour pouvoir déterminer la valeur qu'il faudra utiliser par la suite pour déclencher/stopper l'arrosage.

→ **Exercice :** Remplacer le potentiomètre par le capteur d'humidité et mesurer l'humidité de différents milieux.

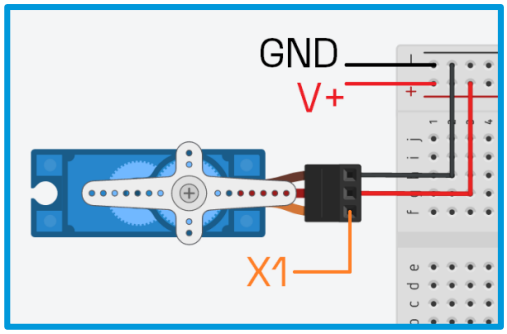
→ **Solution :** Le code est le même que pour le défi 3, la seule différence étant le capteur d'humidité qui est branché à la place du potentiomètre.
Les élèves plongent le capteur dans des pots contenant de la terre ou du sable avec des niveaux d'humidité différents et observent les valeurs obtenues. Une lecture sèche est réalisée dans l'air sec et une lecture humide dans l'eau (ne pas plonger le capteur entièrement…).

Les valeurs que les élèves obtiennent pour les lectures sèches et humides varient d'un capteur à l'autre. La valeur choisie pour passer de "on" à "off" doit se situer entre les deux valeurs. Les lectures à différents niveaux d'humidité permettent d'affiner le choix de la valeur.

Défi 5 : Contrôler un servomoteur

```
import pyb
from pyb import Pin
from pyb import Servo # importe les fonctions de Pyboard pour les servomoteurs.
servo1position = Servo(1) # on définit la première broche, X1, pour notre servomoteur.
while True:
    servo1position.angle(90) # on positionne le servomoteur à un angle de 90°.
```

Le servomoteur se branche sur la broche V+.



Les broches X1 à X4 sont des broches servo. Quand on parle de la broche « Servo(1), » la carte sait qu'on parle de « X1 » et pas d'une autre broche, d'où le fait que « X1 » n'apparaisse pas dans le code.

Dépendant du servomoteur et de la position à laquelle il est arrêté lorsqu'on le débranche, l'angle 0 du servo n'est pas nécessairement en bout de course du servo. Il faut

donc prendre le temps d'identifier sa position. C'est important pour plus tard car un servomoteur ne peut pas faire de tour complet. Il couvre généralement un angle de plus ou moins 180°, et, si le 0 n'est pas au 0 « physique », les angles à entrer dans le code seront différents. On peut par exemple avoir des angles négatifs : si j'utilise un servomoteur qui est décalé de 75°, pour avoir un angle de 0° réels, je devrai régler mon servo, dans le programme, sur -75°, et pour 180° réels, sur 105°.

→ **Exercice :** Trouver la position zéro du servomoteur.

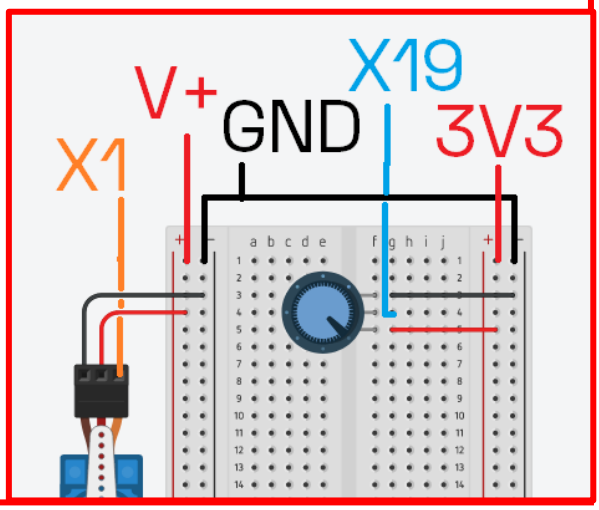
→ **Solution :** Recopier le code, et tester des angles. Ils peuvent tester plusieurs angles en une seule exécution du programme, en enchainant la commande « servolposition.angle() » et en insérant un « sleep » entre chaque changement de position pour avoir le temps d'observer (dans ce cas, n'oubliez pas d'importer la bibliothèque « time » et la fonction « sleep » comme vu précédemment).

Défi 6 : Contrôler le servomoteur à l'aide du potentiomètre

→ **Exercice :** Régler la position du servomoteur en fonction du potentiomètre.

→ **Solution :**

```
import pyb
from pyb import Pin
from pyb import Servo
servo1position = Servo(1)
from pyb import ADC
adc = ADC('X19')
while True:
    A = adc.read()
    if A<=1350:
        servo1position.angle(-15)
    elif 1350<A<=2700:
        servo1position.angle(45)
    else:
        servo1position.angle(105)
```



Tenez évidemment compte du décalage observé lors du défi précédent pour vos angles.

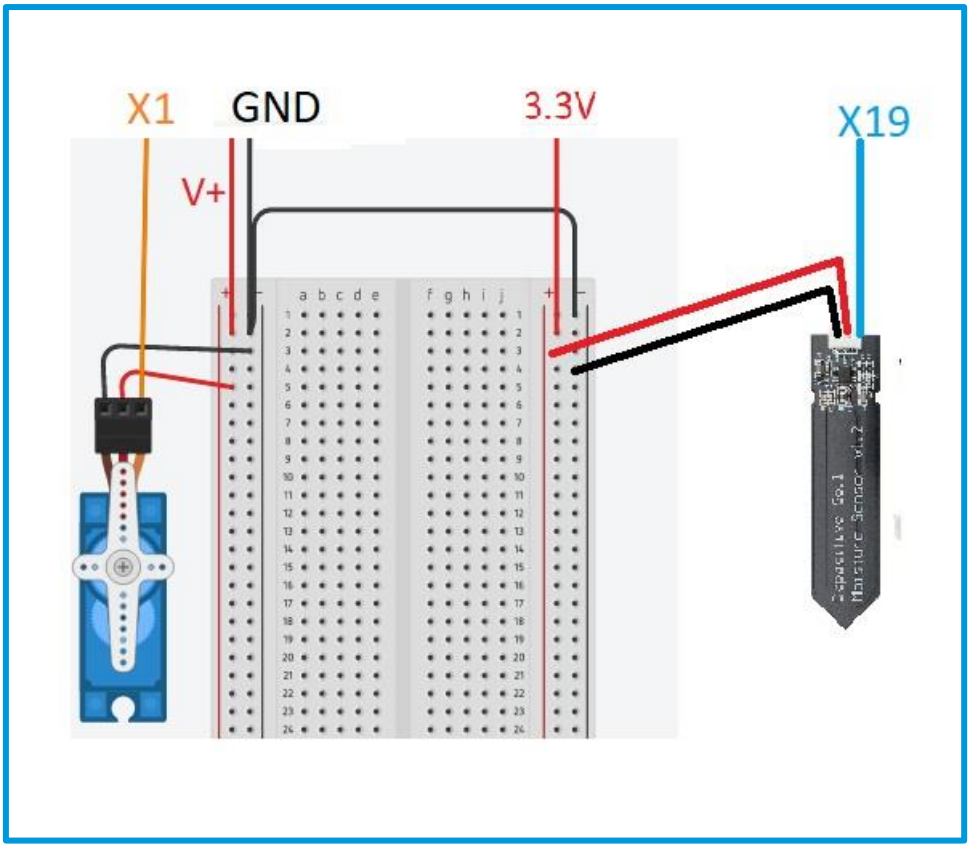
Rappel : attention à ne brancher que le servomoteur sur V+. Le potentiomètre doit être branché sur la broche 3V3.

MONTAGE FINAL

1h

Les élèves disposent de toutes les connaissances nécessaires à la réalisation du montage de tous les éléments et la programmation du système.

Connectez les composants



Selon le capteur d'humidité utilisé, la disposition ou la couleur des broches peut varier. Pensez à consulter la fiche technique du fabricant ; on peut également retrouver des incations près du port.

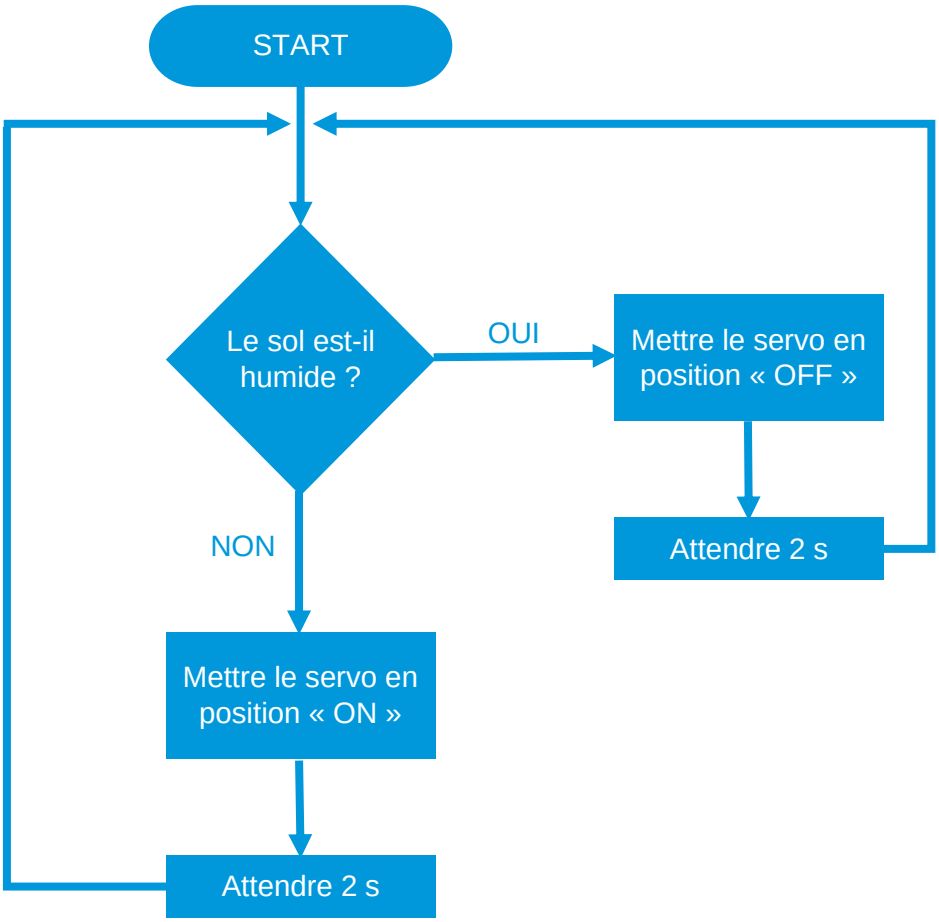
Programmez votre système

Maintenant que les élèves ont connecté tous les composants, il est temps de programmer Pyboard pour qu'il fasse fonctionner tous les composants automatiquement.

→ **Exercice 1 :** Essayez d'écrire votre "code" sous la forme d'un organigramme.

→ **Solution :** Bien que les processus de réflexion et la conception des systèmes d'arrosage proposés par vos élèves varient, il est presque certain qu'ils incluront l'utilisation d'un énoncé "si, sinon" dans leur code. L'organigramme doit donc utiliser une case "décision", indiquée par un diamant.

Un simple diagramme de ce que nous voulons que notre code fasse est présenté ci-dessous.



Le programme est relativement simple et peut se résumer par le pseudo-code suivant :

Initialisation position servo (= indiquer la broche à laquelle le servo est connecté)

Répèter indéfiniment

Lire capteur humidité

attendre (0.1 s)

Si humidité < 3000

Alors mettre le servo en position « ON »

attendre (2 s)

Sinon

mettre le servo en position « OFF »

attendre (2 s)

→ **Exercice 2 :** Définir les angles des positions « ON » (arrosage) et « OFF » (arrêt de l'arrosage) du servomoteur.

→ **Solution :** Appliquer la procédure du défi 5 (servomoteur) pour déterminer les valeurs des angles à utiliser dans le programme pour les positions d'arrosage et d'arrêt.

Les élèves peuvent maintenant rédiger leur programme en faisant appel à tout ce qui a été vu au cours des défis :

- importer les bibliothèques et fonctions nécessaires pour programmer Pyboard, des pauses, des mesures et le contrôle d'un servomoteur
- faire une boucle infinie
- utiliser des conditions
- définir une variable
- faire une mesure
- contrôler un servomoteur
- utiliser les valeurs définies lors du défi 4 avec le capteur d'humidité et de l'exercice 2 ci-dessus (angles du servo).

```
import pyb
from pyb import Servo
from time import sleep
from pyb import ADC
capteur = ADC( 'X19' )
servo1position = Servo(1)
while True:
    humid = capteur.read()
    sleep(0.1)
    print(humid)
    if humid < 3000: # mettez la valeur seuil d'humidité définie lors du défi 4 pour activer
    l'arrosage
        print('soif')
        servo1position.angle(60, 100) # utilisez la valeur correspondant à la position
        « ON » définie lors de l'exercice 2 ci-dessus. La seconde valeur correspond au
        temps de déplacement angulaire en millisecondes. Ce paramètre n'est pas
        indispensable mais cela permet de contrôler la vitesse du mouvement pour
        ménager le servomoteur.
        sleep(2)
    else:
        print('OK')
        servo1position.angle(-80, 100) # utilisez la valeur correspondant à la position
        « ON » définie lors de l'exercice 2 ci-dessus
        sleep(2)
```

Tester le programme avec le montage, sans eau dans un premier temps, et le corriger ou l'adapter si besoin. Lorsque le programme conduit au résultat souhaité et une fois celui-ci enregistré sur Pyboard, déconnecter Pyboard de l'ordinateur et utiliser 4 piles de 1,5 V pour l'alimenter. Le système automatique d'arrosage est prêt pour fonctionner tout seul (il faut juste penser à remplir le réservoir d'eau et changer les piles de temps en temps) !

LIENS

L'espace en classe

Un potager sur Mars – Ressource pour les classes, document Profs :
https://eserobelgium.be/wp-content/uploads/2020/06/T09_Potager_sur_Mars_Document_Prof.pdf

Un potager sur Mars – Ressource pour les classes, document Élèves :
https://eserobelgium.be/wp-content/uploads/2020/06/T09-_Potager_sur_Mars_Document_Eleve.pdf

Meet Arduino – Ressource pour les classes :
https://www.esa.int/Education/CanSat/Meet_Arduino!_Introduction_to_Arduino_computing_using_C_Teach_with_Space_T04.1

Missions sur Mars

Mission Mars Express :
https://www.esa.int/Science_Exploration/Space_Science/Mars_Express

Mission ExoMars :
https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/Exploration/ExoMars

MicroPython et IDE

https://micropython.org/

https://thonny.org/