Ecole Robots
Apprendre à coder avec les robots éducatifs

Kit de programmation
CURSUS EDUCATION NATIONALE

Python - Niveau collège

Les bases de la programmation
Mouvement et détection

Chapitre 1 : Programmer des LED et des buzzers
Chapitre 2 : Programmer des voitures

Prénom : _____ Nom : _____ Classe : _____

ARTEC®
Speechi

Les programmes nécessaires à la réalisation des robots sont disponibles en téléchargement sur le site www.ecolerobots.com.

Toutes les boîtes et les pièces détachées sont aussi disponibles sur le site www.ecolerobots.com.

Sommaire

Introduction

La programmation en action

Notions abordées

- Les systèmes informatisés dans la vie de tous les jours
- Utiliser un environnement de programmation

1 . La technologie de tous les jours

Les appareils électroniques que nous utilisons tous les jours fonctionnent généralement automatiquement grâce aux systèmes de contrôle et aux instruments de mesure qui y sont intégrés. Leurs capteurs et ordinateurs internes leur permettent de répéter des processus pré-établis, de détecter des informations externes et de s'y adapter. Ces processus reposent sur 3 éléments dans la machine : les capteurs, les ordinateurs et les actionneurs. Prenons l'exemple des climatiseurs.

Ex. Les climatiseurs

Capteurs

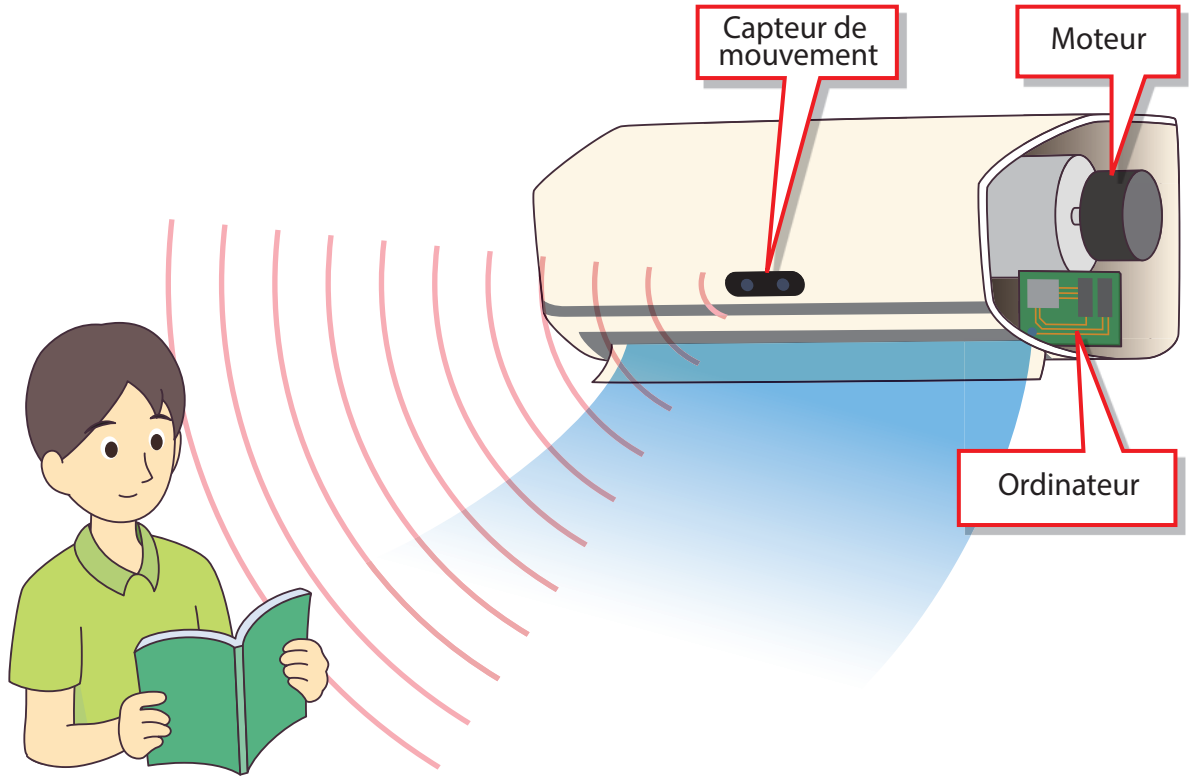
Les capteurs infrarouges et de chaleur détectent la présence de personnes dans une pièce. Les thermomètres mesurent la température actuelle d'une pièce.

Ordinateur

L'ordinateur reçoit et traite les informations provenant des capteurs. Il utilise ces informations pour déterminer quelle commande envoyer aux moteurs.

Actionneur

Les moteurs ajustent la température de la pièce en tournant en fonction des commandes reçues de l'ordinateur.



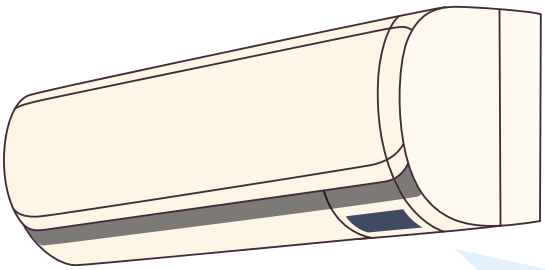
Trouve d'autres appareils de la vie quotidienne qui utilisent des capteurs et des ordinateurs pour fonctionner...

Appareils	Les capteurs détectent...	Et l'ordinateur...
Chauffe-eau	la quantité d'eau	commence ou arrête de chauffer l'eau
Robot aspirateur	les murs et autres obstacles	éloigne le robot des obstacles

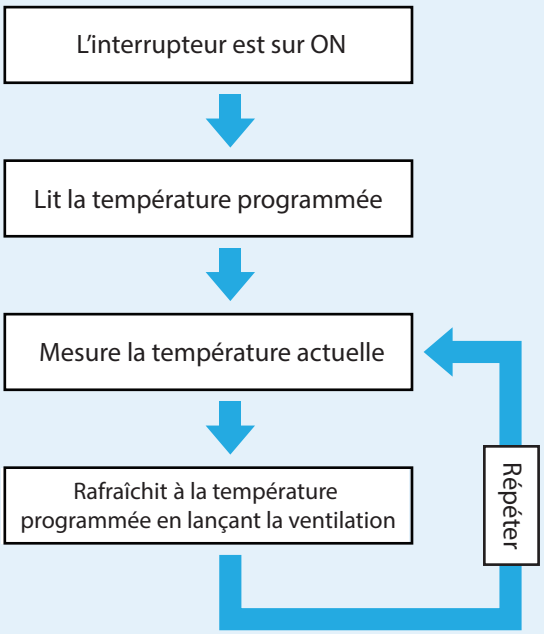
2 . Quelle est l'importance de la programmation ?

Les machines avec ordinateurs intégrés (comme les climatiseurs) fonctionnent en suivant des instructions préparées par des humains et en utilisant les informations recueillies par leurs capteurs pour savoir quelles instructions suivre.

Ex. Mettre en route un climatiseur



Un humain a déterminé les étapes suivantes :



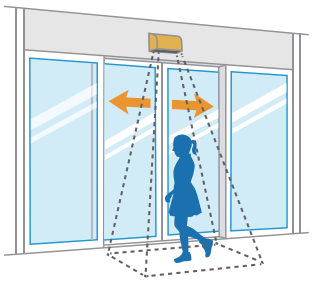
L'ensemble des instructions qu'un ordinateur suit s'appelle un **programme**. Les programmes d'un ordinateur ne sont pas écrits en langage parlé, mais dans des langages spécifiques que des ordinateurs peuvent comprendre. Ces langages sont ce qu'on appelle des **langages de programmation**.
Écrire un programme s'appelle **programmer**. Dans ce manuel, tu apprendras à écrire des programmes dans un langage de programmation appelé **Python**.

[Un programme pour allumer/éteindre une LED, écrit en Python]

```
1 from pystubit.board import display
2 import time
3
4 display.set_pixel(1, 1, display.GREEN)
5 time.sleep(1)
6 display.clear()
```

3. Déplacement et détection avec les pièces de la carte

Il est possible de mettre au point soi-même des instruments de mesure informatisés et des systèmes de contrôle en se servant de nos pièces.

Un système de porte automatique	Un système de signalisation par bouton-poussoir
	

Capteurs

Ils collectent des informations sur leur environnement.

Les photoréflecteurs IR
détectent la présence d'objets.

Les boutons
détectent s'ils ont été pressés.

Ordinateurs

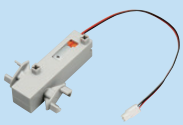
Ils envoient des directives aux actionneurs en se basant sur des informations obtenues grâce aux capteurs.

La carte programmable

Pièces opérationnelles

Ils exécutent les directives envoyées par l'ordinateur.

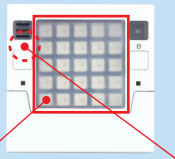
Actionneur



Moteurs CC

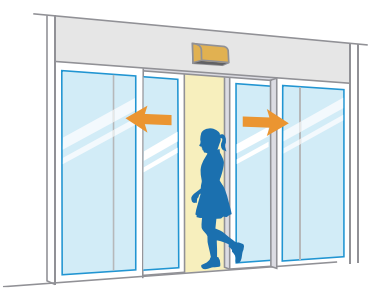
Servomoteurs

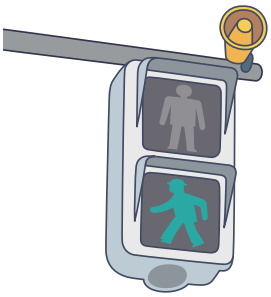
transforment l'électricité en force de rotation



LED
transforme l'électricité en lumière

Buzzer
transforme l'électricité en son





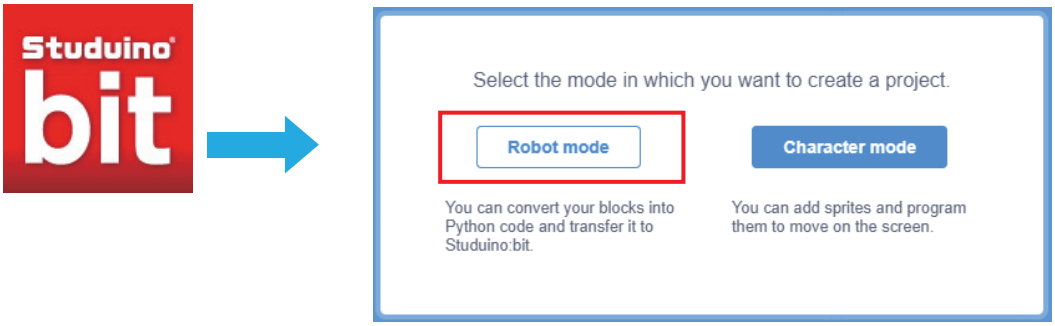
★Le mot **actionneur** désigne un appareil qui transforme l'énergie en mouvement. Les LED et les buzzers ne sont donc pas des actionneurs !

4. L'environnement de programmation

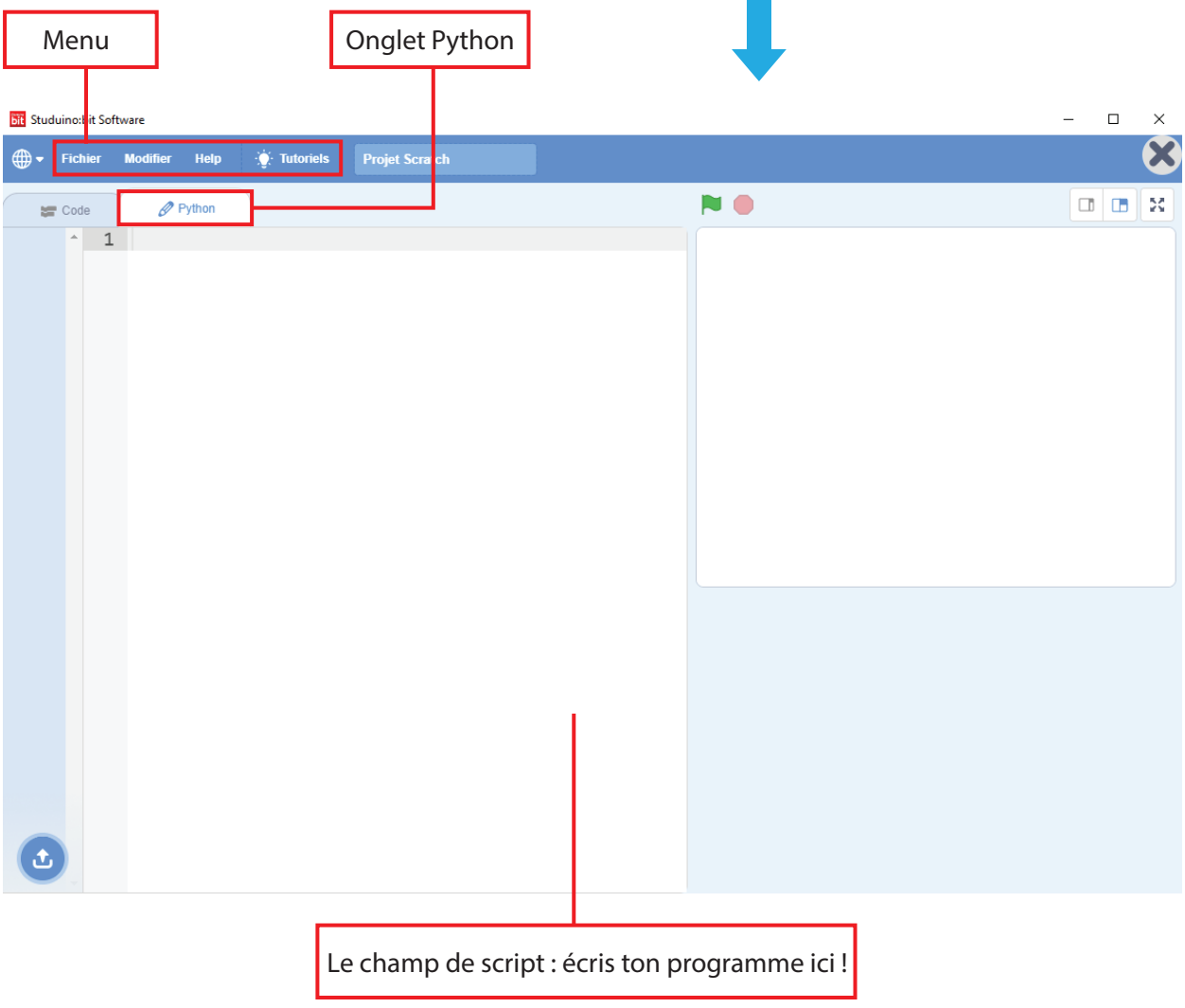
Dans ce cours, nous utiliserons l'éditeur Python du logiciel Studuino:bit pour écrire des programmes en Python.

4.1 Démarrage du logiciel

Ouvre le logiciel Studuino:bit et sélectionne **Robot mode**.

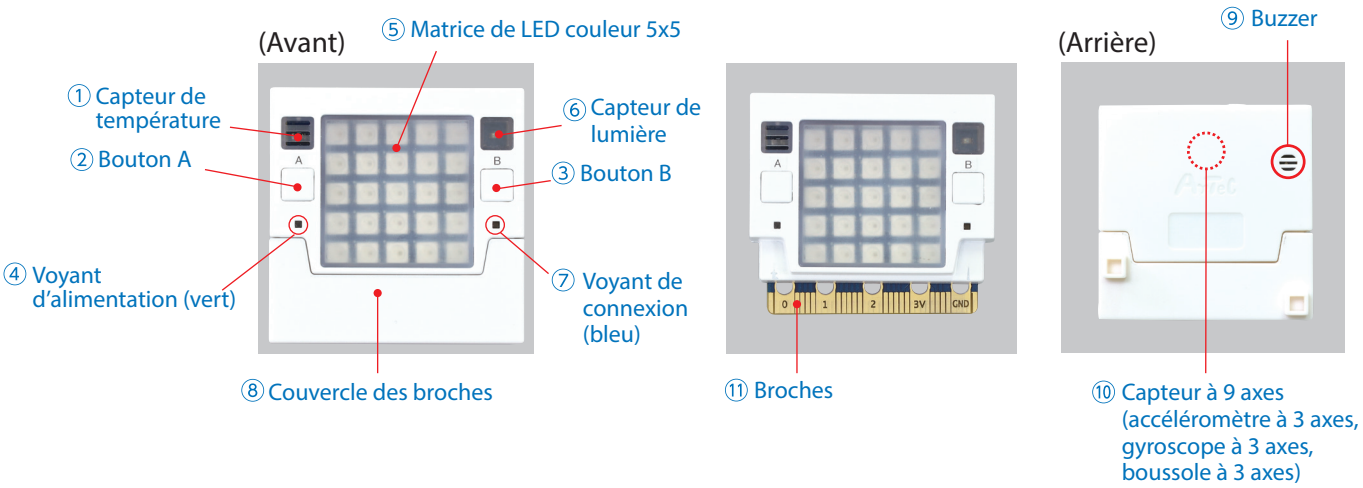


Rends-toi dans l'onglet Python pour ouvrir le champ de script.



4.2 La carte programmable et sa bibliothèque (Studuino:bit)

La carte programmable intègre les éléments suivants :



La bibliothèque Studuino:bit permet de contrôler la carte facilement en définissant des **objets** (*) pour chacun de ses éléments. Le tableau ci-dessous met en correspondance les éléments de la carte avec leur objet.

#	Élément	Objet	Ce qu'il fait...
①	Capteur de température	temperature	Retrouve les valeurs du capteur de température.
②	Bouton A	button_a	Détecte si le bouton A est sur on ou off.
③	Bouton B	button_b	Détecte si le bouton B est sur on ou off.
⑤	Matrice LED de couleurs 5x5	display	Allume et éteint des LED, lance des animations, etc.
⑥	Capteur de lumière	lightsensor	Retrouve les valeurs du capteur de lumière.
⑨	Buzzer	buzzer	Fait jouer au buzzer une note spécifique.
⑩	Accéléromètre à 3 axes	accelerometer	Retrouve les valeurs de l'accéléromètre.
	Gyroscope à 3 axes	gyro	Retrouve les valeurs du gyroscope.
	Boussole à 3 axes	compass	Retrouve les valeurs de la boussole.
⑪	Broches	P0 - P16, P19, P20	Contrôle les entrées/sorties des broches.

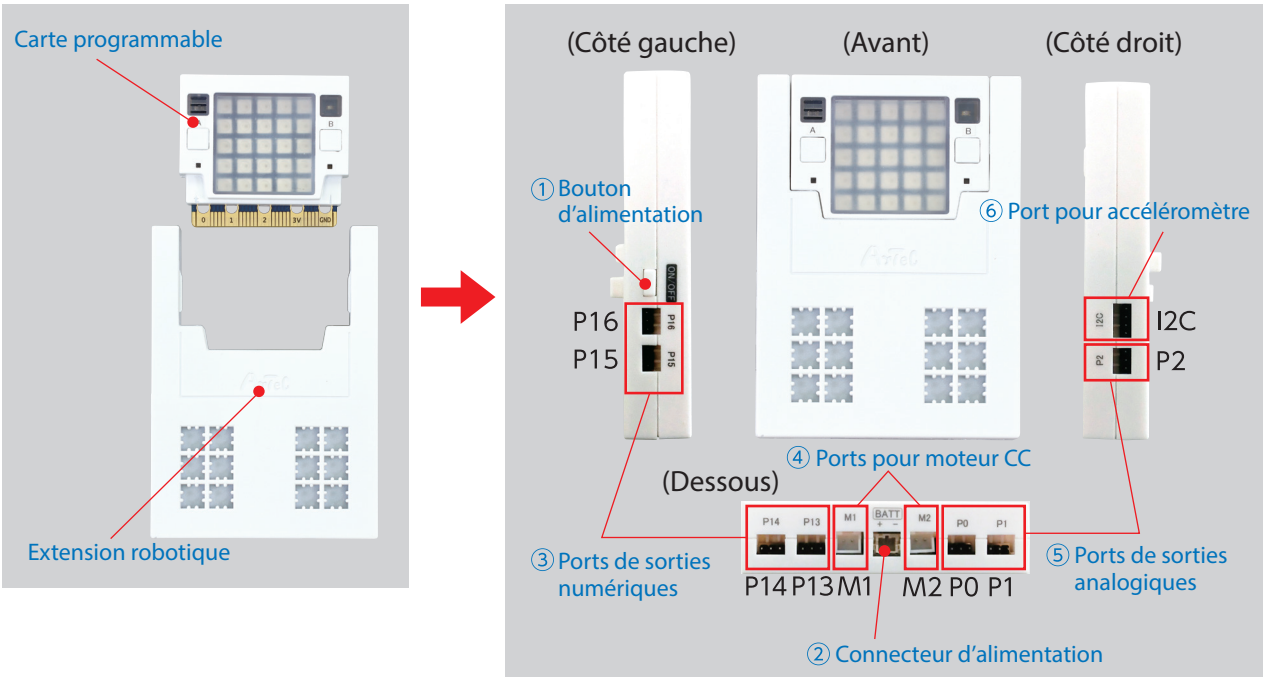
(*) En programmation, un "objet" est un ensemble de données et/ou d'instructions qu'on peut utiliser dans un programme. Les objets de la bibliothèque Studuino:bit te permet d'écrire des programmes qui contrôlent les éléments de la carte !

En Python, la bibliothèque Studuino:bit est appelée **pystubit**. La ligne de code ci-dessous permettra d'utiliser, dans ton programme, tous les objets de la bibliothèque Studuino:bit.

```
from pystubit.board import *
```

4.3 L'extension robotique et sa bibliothèque (ArtecRobo 2.0)

L'extension peut être connectée à la carte pour utiliser des moteurs et des capteurs.



Des éléments additionnels comme les moteurs et capteurs peuvent être branchés sur les ports de l'extension. Se référer au tableau ci-dessous pour trouver quels éléments fonctionnent avec quels ports.

#	Ports	Éléments	Ce qu'il fait...
③	Ports de sortie numérique	LED	Allume et éteint les LED.
		Buzzer	Joue des sons.
		Servomoteur	Tourne à un angle spécifique.
④	Ports pour moteur CC	Moteur CC	Tourne dans une direction et à une vitesse spécifiques.
⑤	Ports d'entrée analogique	Capteur de lumière	Mesure les niveaux de lumière environnante.
		Capteur de son	Mesure le volume des sons.
		Capteur tactile	Détecte si le commutateur est sur on ou off.
		Photoréflecteur IR	Détecte l'emplacement et la couleur d'un objet proche.
⑥	Ports pour accéléromètre	Accéléromètre	Mesure l'inclinaison/accélération.

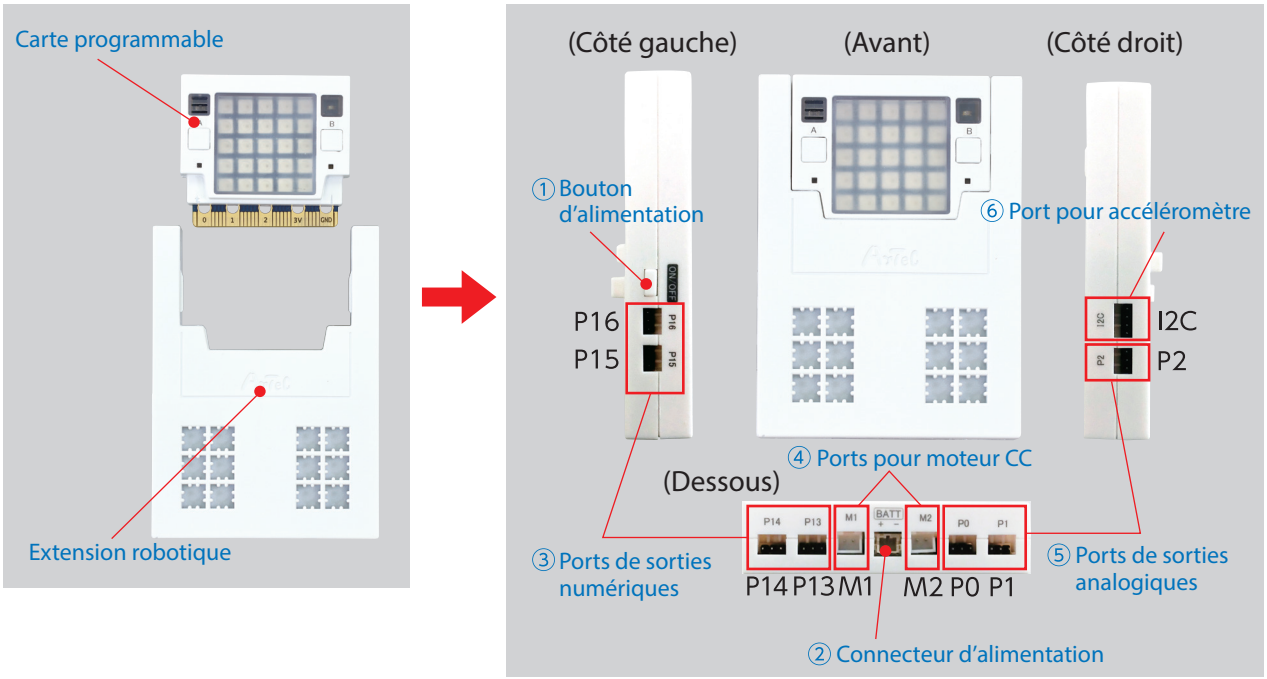
La bibliothèque ArtecRobo 2.0 te permet de contrôler une grande variété d'éléments que tu peux connecter à l'extension.

En Python, la bibliothèque ArtecRobo 2.0 est appelée **pyatcrobo2**. La ligne de code ci-dessous te permettra d'utiliser, dans ton programme, tous les éléments de la bibliothèque ArtecRobo 2.0.

```
from pyatcrobo2.parts import *
```

4.3 L'extension robotique et sa bibliothèque (ArtecRobo 2.0)

L'extension peut être connectée à la carte pour utiliser des moteurs et des capteurs.



Des éléments additionnels comme les moteurs et capteurs peuvent être branchés sur les ports de l'extension. Se référer au tableau ci-dessous pour trouver quels éléments fonctionnent avec quels ports.

#	Ports	Éléments	Ce qu'il fait...
③	Ports de sortie numérique	LED	Allume et éteint les LED.
		Buzzer	Joue des sons.
		Servomoteur	Tourne à un angle spécifique.
④	Ports pour moteur CC	Moteur CC	Tourne dans un sens et à une vitesse spécifiques.
⑤	Ports d'entrée analogique	Capteur de lumière	Mesure les niveaux de lumière environnante.
		Capteur de son	Mesure le volume des sons.
		Capteur tactile	Détecte si le commutateur est sur on ou off.
		Photoréflecteur IR	Détecte l'emplacement et la couleur d'un objet proche.
⑥	Ports pour accéléromètre	Accéléromètre	Mesure l'inclinaison/accélération.

La bibliothèque ArtecRobo 2.0 te permet de contrôler une grande variété d'éléments que tu peux connecter à l'extension.

En Python, la bibliothèque ArtecRobo 2.0 est appelée **pyatcrobo2**. La ligne de code ci-dessous te permettra d'utiliser, dans ton programme, tous les éléments de la bibliothèque ArtecRobo 2.0.

```
from pyatcrobo2.parts import *
```

1. Les bases de la programmation

Programmer des LED

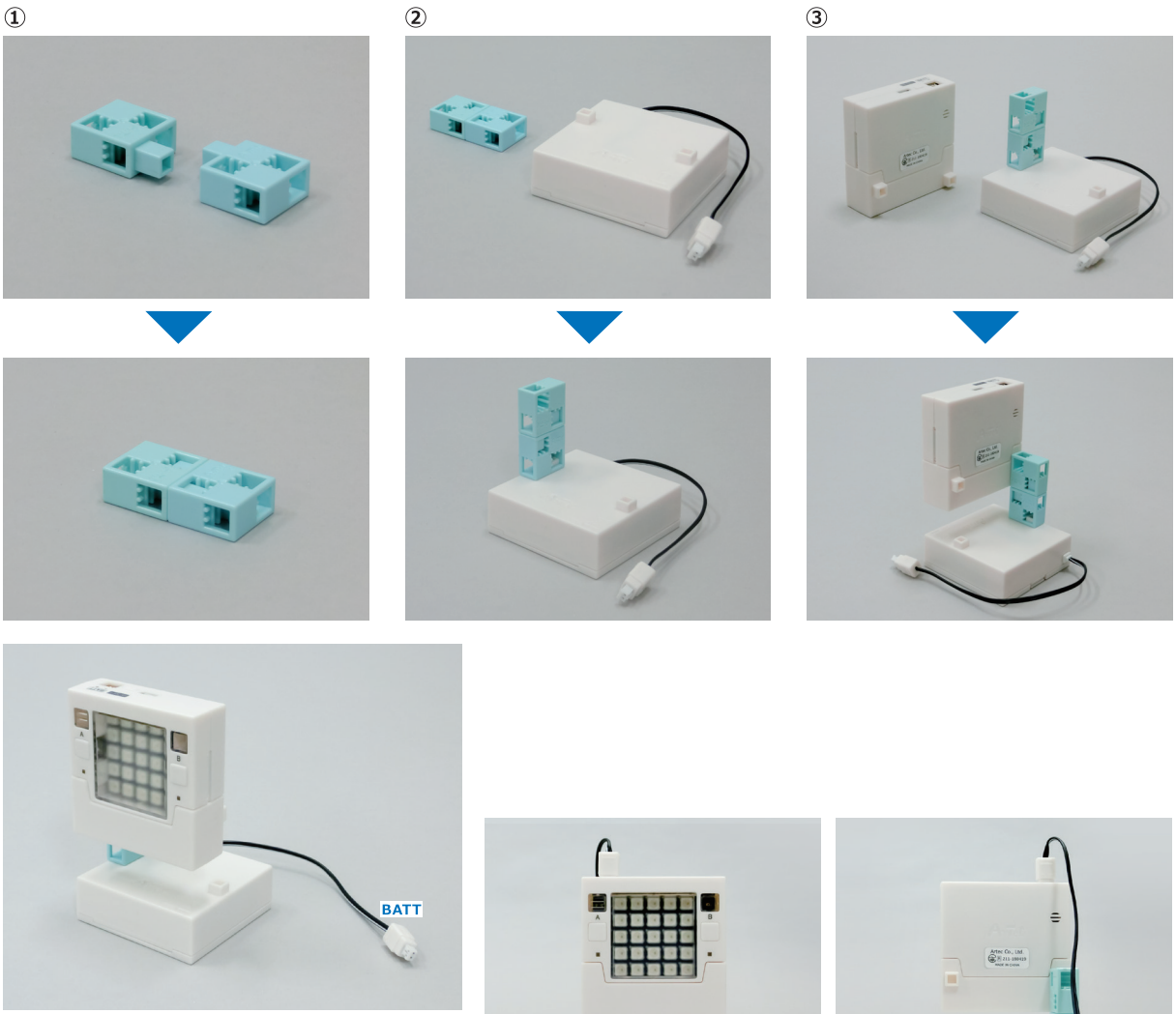
Notions abordées

- Programmer des LED et des buzzers
- Séquences
- Boucles
- Conditions

1. Programmer des séquences

Un programme séquentiel est un programme qui donne une série de commandes à l'ordinateur dans l'ordre dans lequel ils sont écrits. Mets au point un feu de signalisation avec des LED et programme-le pour fonctionner avec un programme séquentiel.

1.1 Construire un feu de signalisation



1.2 Allumer une LED

Utilise les lignes de code ci-dessous pour envoyer la commande qui allume une LED.

```
from pystubit.board import display

display.set_pixel(1, 1, display.GREEN)
```

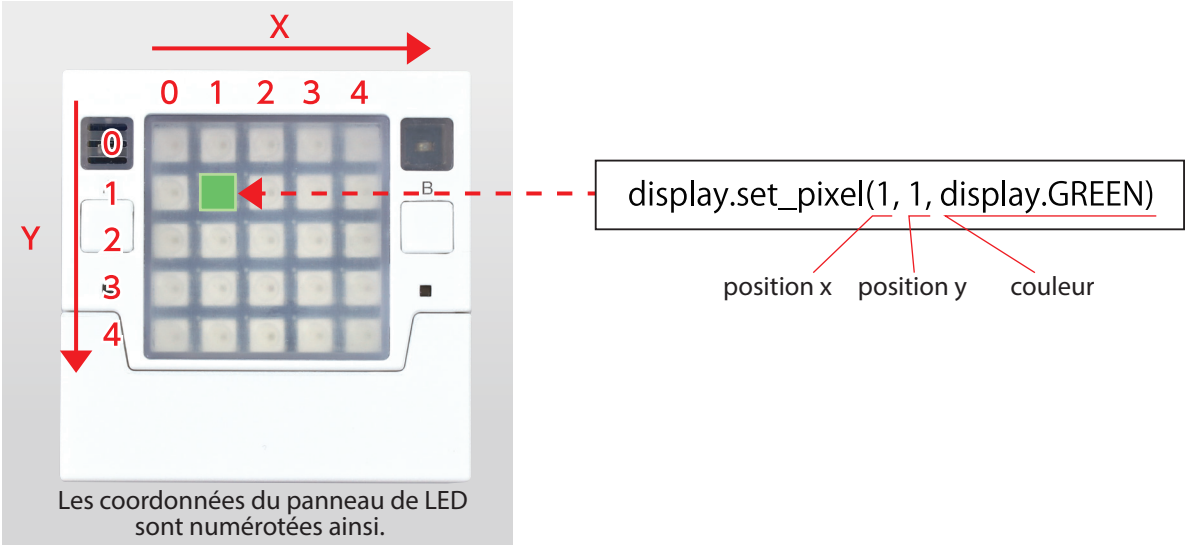
Nous utiliserons la bibliothèque Studuino:bit pour contrôler les éléments de la carte. La ligne 1 du programme, **from pystubit.board import display**, demande à l'ordinateur d' "importer" l'objet **display** du module de la carte pystubit (**board**) de la bibliothèque Studuino:bit. Cela te permettra d'utiliser l'objet **display** dans ton programme. L'objet **display** sert à contrôler l'affichage des LED (les allumer, changer leurs couleurs, etc.). Réfère-toi à l'**Annexe A** (p.29) pour en savoir plus sur l'objet **display**.

Tu peux accéder aux informations (appelées **propriétés**) d'un objet en écrivant le nom de la propriété après le nom de l'objet et en les séparant d'un point. Si tu veux que ton programme fasse quelque chose à l'objet, tu peux exécuter une **méthode** en procédant de la même façon, mais en écrivant cette fois le nom de la méthode au lieu de la propriété.

```
Object.Property
Object.Method(Parameter)
```

```
set_pixel(x, y, color)
```

Les valeurs de la méthode entre parenthèses sont appelées **paramètres**. Tu peux les ajuster pour changer ce que fait la méthode. Les paramètres dans **set_pixel** servent à spécifier une LED du panneau et la couleur dans laquelle la LED s'allumera. La ligne 3 du programme ci-dessus, **display.set_pixel(1,1,display.GREEN)**, fait s'allumer la LED située aux coordonnées (1,1) du panneau en vert **(display.GREEN)**.



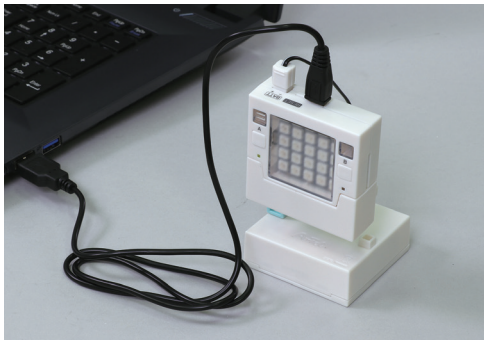
Les coordonnées du panneau de LED sont numérotées ainsi.

1.2.1 Tester un programme

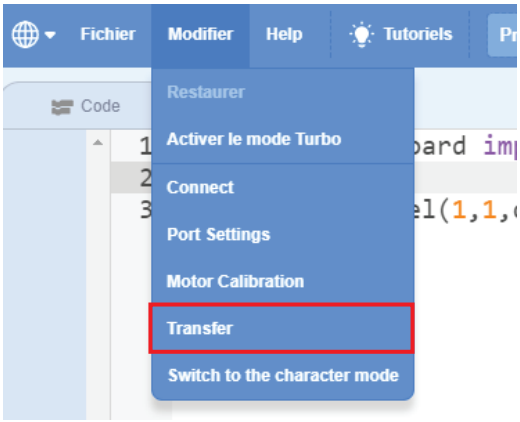
Suis ces étapes pour transférer ton programme vers la carte.

Depuis Windows/Mac

① Connecte ta carte au PC avec un câble USB.

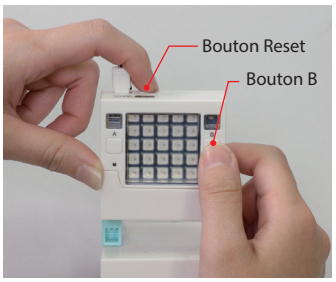


② Ouvre le menu **Modifier** et clique sur **Transfer**.

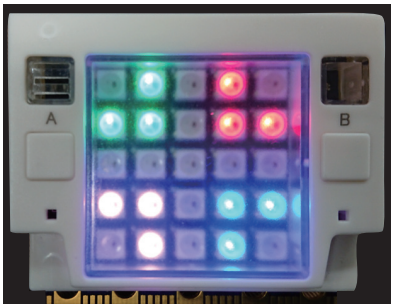


Depuis Android/iPad/Chromebook

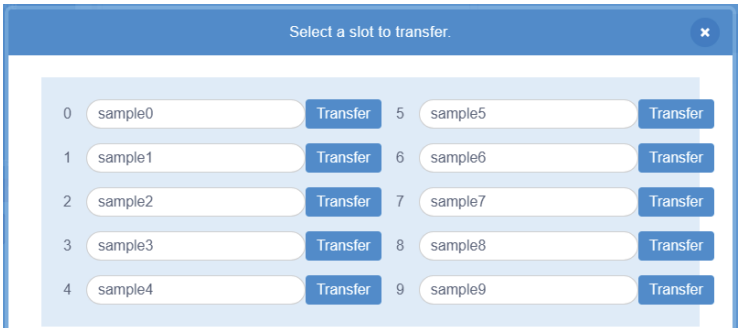
① Suis les instructions de ton PC, maintiens appuyé le bouton B de ta carte et appuie sur Reset.



② Sélectionne l'appareil indiqué par le schéma du panneau LED de la carte.



③ Sélectionne un emplacement pour envoyer ton programme.
★Tu peux choisir entre les emplacements **0 à 9**.

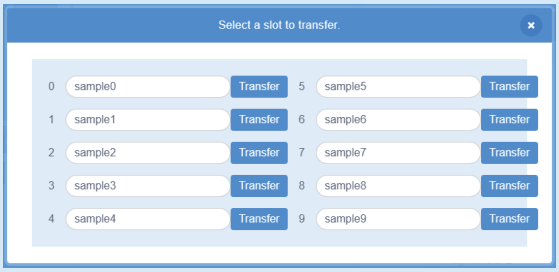


④ Quand le transfert est terminé, ton programme se lancera automatiquement.
Si tu veux redémarrer ton programme, appuie sur Reset.

Transférer plusieurs programmes à la fois

Tu peux transférer plusieurs programmes vers la carte en une seule fois en spécifiant plusieurs emplacements (numérotés de 0 à 9) pour les y envoyer.

Normalement le dernier programme transféré se lancera automatiquement, mais ces étapes te permettront de choisir parmi les programmes stockés dans la carte celui que tu veux lancer.



① Brancher l'alimentation (démarrage standard)

(1) Branche la batterie ou le câble USB sur la carte.

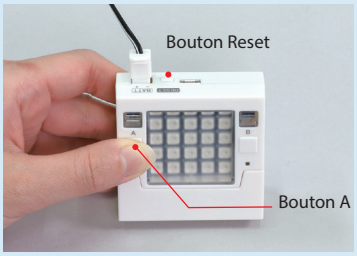
(2) L'alimentation se mettra automatiquement sur ON et le voyant vert s'allumera.

② Aller dans le mode de sélection d'un programme

(1) Maintiens appuyé le bouton A et appuie sur Reset. Reset redémarrera l'appareil. Le voyant d'alimentation s'éteindra donc brièvement.

(2) Quand le voyant d'alimentation s'allumera à nouveau 3 secondes après avoir relâché le bouton Reset, relâche le bouton A.

(3) Si le panneau montre un 0 en vert, tu es entré dans le mode de sélection d'un programme.



③ Sélectionner/démarrer un programme

(1) Dans le mode de sélection d'un programme, le bouton A change le numéro affiché (0, 1, 2, etc.). Le nombre peut aller jusqu'à 9 avant de retourner à 0.

(2) Le bouton B sert à sélectionner et démarrer le programme assigné au numéro choisi.

La prochaine fois que tu procéderas à un démarrage standard, le programme sélectionné en ③ se lancera.

★ Lors du tout 1er démarrage, l'appareil contiendra des exemples de programme pour tester la carte.
★ Lorsque tu transféreras un nouveau programme, celui-ci effacera et remplacera ces exemples de programme.

1.3 Allumer une LED pendant 1 seconde

Le programme ci-dessous permet d'allumer une LED en vert pendant exactement 1 seconde.

```
1 from pystubit.board import display
2 import time
3
4 display.set_pixel(1, 1, display.GREEN)
5 time.sleep(1)
6 display.clear()
```

La ligne 2, **import time**, indique que le programme utilisera le module **time** de la bibliothèque standard de Python.

Le module **time** peut garder trace du temps durant lequel la carte a été allumée, mettre la carte en pause/en veille, etc. Le code **time.sleep(sec)** met en pause le programme durant le nombre de secondes spécifié dans le paramètre appelé **sec.** La ligne 5, **time.sleep(1),** met donc en pause le programme pendant 1 seconde ! La ligne 6, **display.clear()**, éteint les LED du panneau. La méthode **clear** n'a pas de paramètre à régler, c'est pourquoi rien n'est écrit entre ses parenthèses.

Tu peux changer la valeur écrite dans le paramètre **sec** dans **time.sleep(sec)** pour régler le nombre de secondes durant lequel la LED restera allumée. Tu peux définir **sec** sur le nombre de secondes que tu veux, même des décimales !

Ton programme se lancera toujours du haut vers le bas.

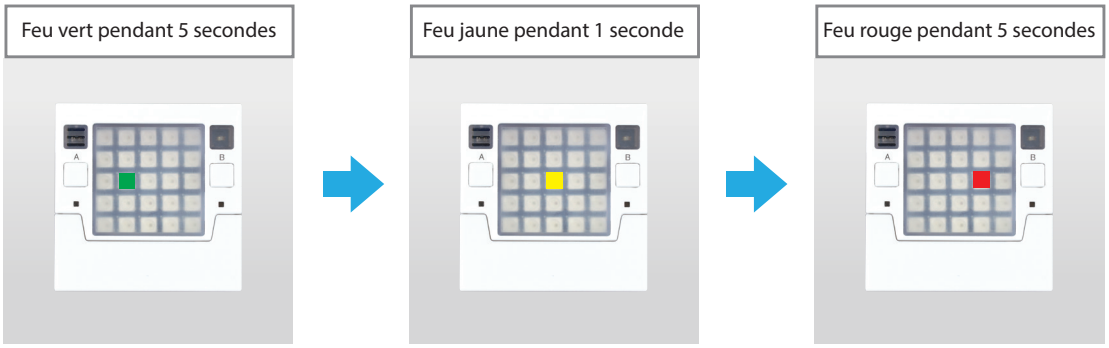
```
from pystubit.board import display
import time

display.set_pixel(1, 1, display.GREEN)
time.sleep(1)
display.clear()
```

Sais-tu pourquoi nous avons besoin de la ligne de code **time.sleep** ? Si tu exécutes un programme sans cette ligne de code (comme le programme de la page suivante), la LED ne s'allumera pas. Cela arrive parce que la carte exécute les programmes très rapidemment. Dès qu'elle envoie la commande d'allumer la LED, elle envoie immédiatement la commande suivante d'éteindre la LED. La LED s'éteint donc aussitôt qu'elle s'allume !

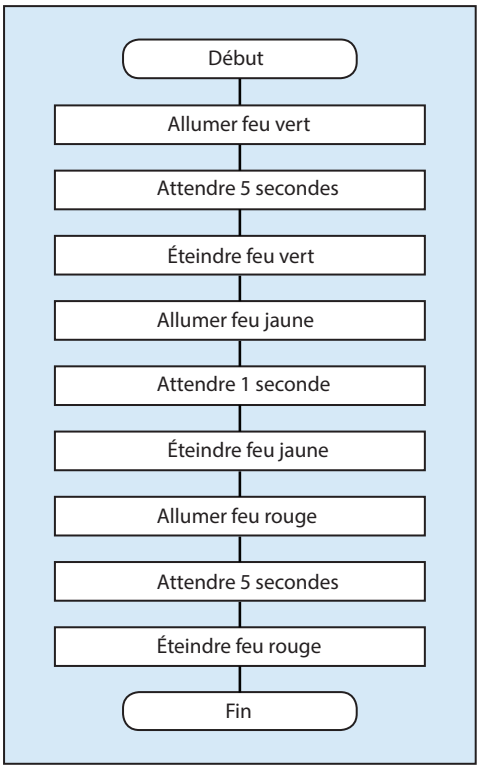
```
from pystubit.board import display
import time

display.set_pixel(1, 1, display.GREEN)
display.clear()
display.set_pixel(1, 1, dipslay.GREEN)
display.clear()
```

Exercice de programmation : programmer un feu de signalisation

Dessine un organigramme qui reproduit la séquence d'allumage présentée ci-dessous, puis essaie d'en écrire le programme (réfère-toi à l'Annexe A, p.29, si besoin).



Exemple de programme




```
1  from pystubit.board import display
2  import time
3
4  display.set_pixel(1, 2, display.GREEN)
5  time.sleep(5)
6  display.clear()
7  display.set_pixel(2, 2, display.YELLOW)
8  time.sleep(1)
9  display.clear()
10 display.set_pixel(3, 2, display.RED)
11 time.sleep(5)
12 display.clear()
```

2 . Programmer des boucles

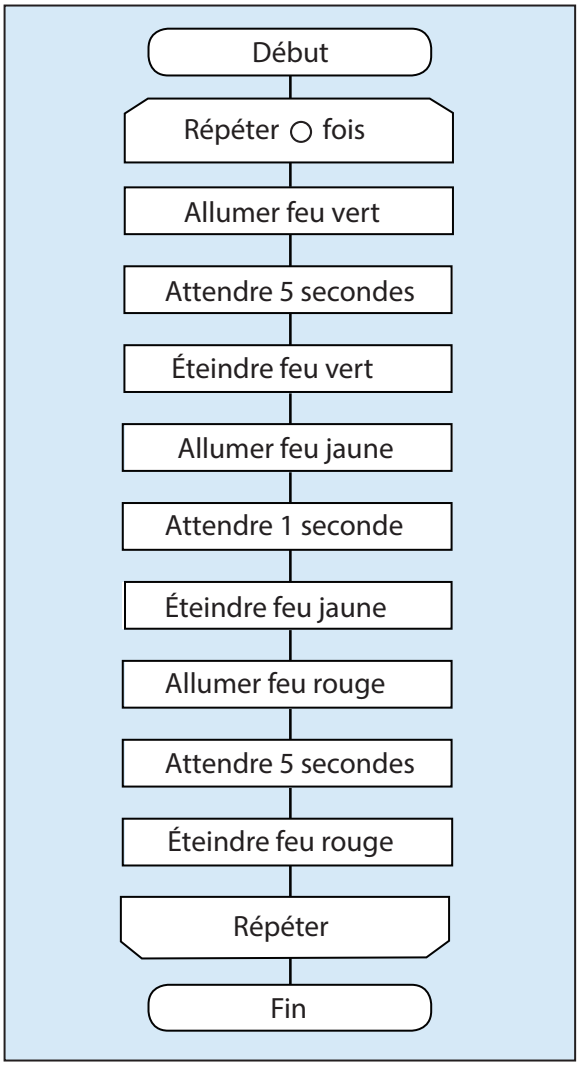
Un programme qui répète une séquence de commandes indéfiniment est appelée une **boucle**. Le programme du feu de signalisation que nous avons fait à la page précédente éteint le feu une fois la LED rouge éteinte, puis s'arrête complètement. Pour qu'un feu de signalisation répète la même séquence d'allumage des feux indéfiniment comme les vrais feux de signalisation, nous aurons besoin d'une boucle.

Les boucles peuvent être divisées en deux catégories : les boucles infinies qui continuent tant que le programme s'exécute et les boucles conditionnelles qui continuent jusqu'à ce qu'une condition particulière soit remplie.

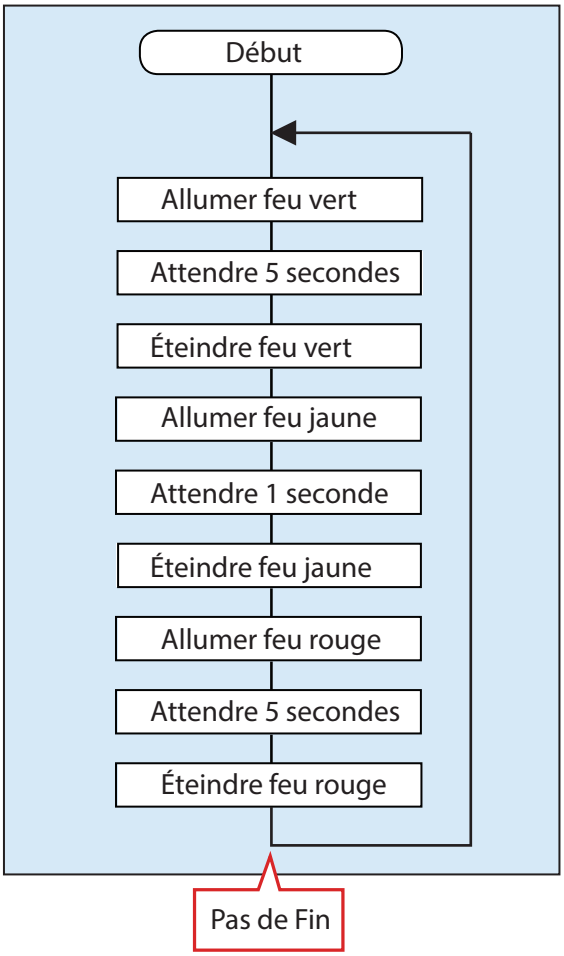
2.1 Faire un organigramme

Dessine un organigramme pour un programme de feu de signalisation qui tourne en boucle. Quand tu crées un programme sous forme d'organigramme, tu peux utiliser les segments suivants pour indiquer une boucle :

Répéter ○ fois	Début d'une boucle
Répéter	Fin d'une boucle



Si tu souhaites boucler indéfiniment, enlève "Fin" et dessine une flèche qui montre comment le programme retourne au début de la boucle, comme ceci :



2.2 Améliorer ton programme (ajouter des boucles)

En Python, tu peux répéter en boucle des programmes en utilisant deux types d'instruction : les instructions **for** et les instructions **while**. Les instructions **for** font se répéter une boucle un nombre de fois qui a été fixé. Tu peux les écrire comme suit :

for variable **in range(** nombre d'itérations **)**:

Programme à répéter en boucle

Veille à indenter toutes les lignes du programme que tu veux répéter ! Dans le programme ci-dessous, nous voulons faire répéter en boucle l'ensemble du programme du feu de signalisation. C'est pourquoi toutes les lignes sont indentées. Tu peux indenter une ligne en appuyant sur la touche Tab (ou sur la touche Espace 4 fois) à son début.

```
from pystubit.board import display
import time

for i in range(10):
    display.set_pixel(1, 2, display.GREEN)
    time.sleep(5)
    display.clear()
    display.set_pixel(2, 2, display.YELLOW)
    time.sleep(1)
    display.clear()
    display.set_pixel(3, 2, display.RED)
    time.sleep(5)
    display.clear()
```

Indentation

Les instructions **while** font des boucles infinies. Tu peux les écrire comme suit :

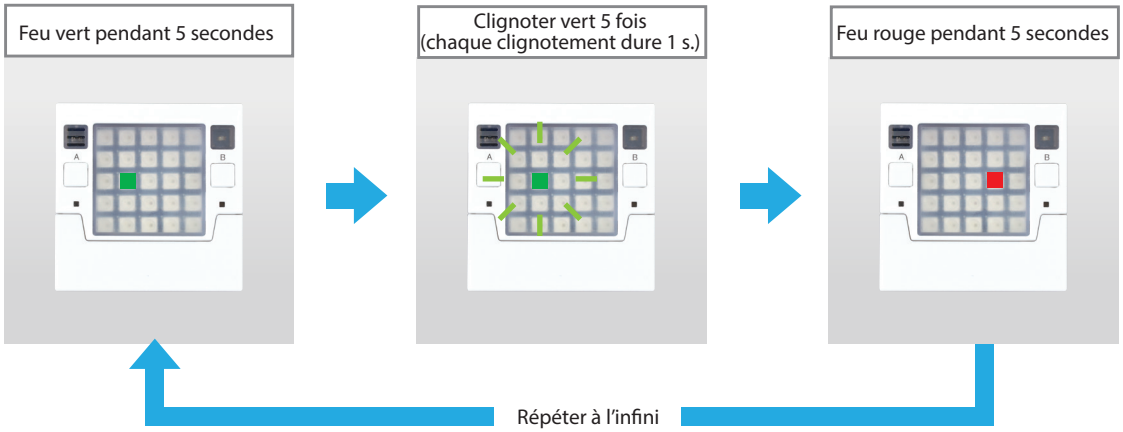
while True:

Programme à répéter en boucle

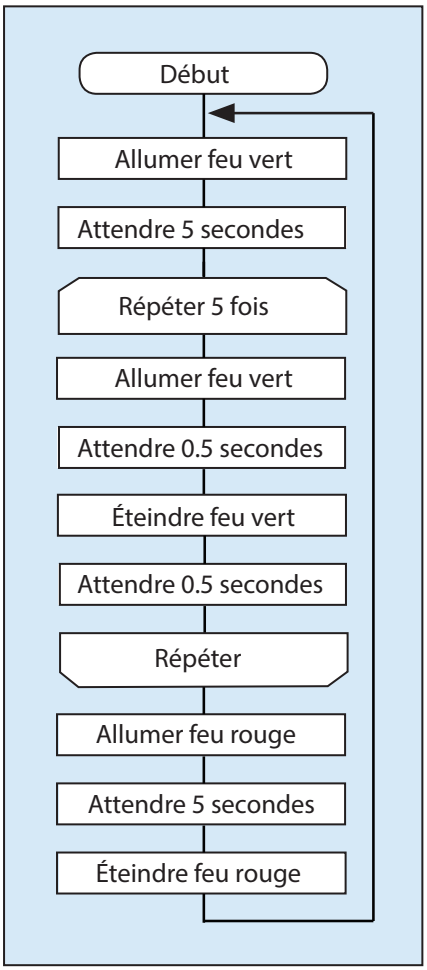
Veille à indenter l'ensemble du programme que tu veux répéter, comme dans les instructions **for**.

Exercice de programmation : programmer un feu piéton

Dessine un organigramme qui reproduit la séquence d'allumage montrée ci-dessous, puis essaie d'en écrire le programme.



Exemple de programme




```
1  from pystubit.board import display
2  import time
3
4  while True:
5      display.set_pixel(1, 2, display.GREEN)
6      time.sleep(5)
7      for i in range(5):
8          display.set_pixel(1, 2, display.GREEN)
9          time.sleep(0.5)
10         display.clear()
11         time.sleep(0.5)
12     display.set_pixel(3, 2, display.RED)
13     time.sleep(5)
14     display.clear()
```

3 . Programmer des conditions

Une **condition** est une instruction qui propose différentes commandes selon que la condition est vraie ou fausse. Découvrons les conditions et comment elles font en sorte que les programmes prennent différentes directions en programmant un feu de signalisation pour piéton avec un bouton poussoir.

3.1 Valeurs des boutons

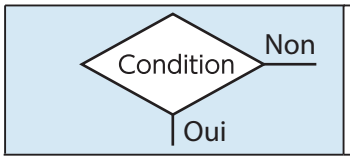
L'objet **button_a** de la bibliothèque Studuino:bit te permet d'écrire des programmes qui utilisent le bouton A de la carte. Sers-toi de la ligne de code ci-dessous pour rendre disponibles l'objet **button_a** dans ton programme, comme nous l'avons fait avec l'objet **display**.

```
from pystubit.board import button_a
```

Quand tu exécutes la méthode **get_value()** sur l'objet **button_a**, tu récupéreras la valeur **1** ou **0**.
Cette valeur t'indique si le bouton A est actuellement pressé ou non.

3.2 Organigrammes et syntaxe des conditions

Utilise un segment comme celui-ci pour dessiner une condition dans l'organigramme d'un programme :



Divise le programme en branches en fonction de la condition.

En Python, les conditions sont créées avec des instructions **if-else**. Tu peux écrire une instruction **if-else** comme suit :

```
if instruction conditionnelle :
    Programme qui se lance quand la condition est remplie.
else :
    Programme qui se lance quand la condition n'est PAS remplie.
```

Les deux programmes que tu veux lancer doivent être indentés !
Tu peux écrire des instructions conditionnelles en utilisant les opérateurs relationnels suivants :

Opérateur relationnel	Instruction conditionnelle	Signification
==	A == B	A est égal à B
!=	A != B	A n'est pas égal à B
>	A > B	A est supérieur à B
>=	A >= B	A est supérieur ou égal à B
<	A < B	A est inférieur à B
<=	A <= B	A est inférieur ou égal à B

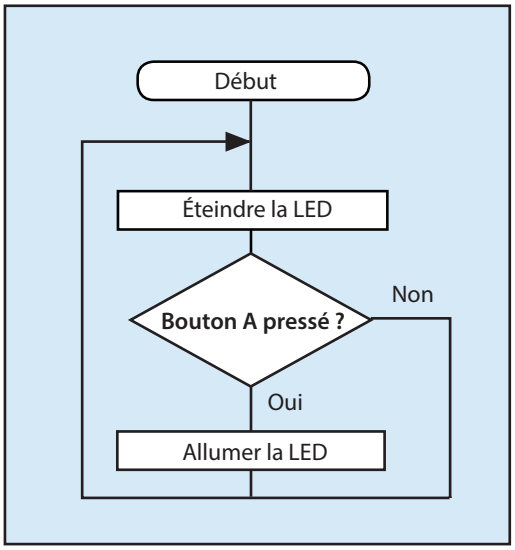
Voici l'exemple d'une instruction **if-else** :

```
if A >= B:
    Branche C
else:
    Branche D
```

Dans ce programme, la branche C s'exécutera si A est supérieur ou égal à B, sinon (c'est-à-dire si A est inférieur à B), la branche D s'exécutera.

3.3 Appuyer sur le bouton A pour allumer une LED

Tu peux le faire avec l'organigramme/le programme suivant.

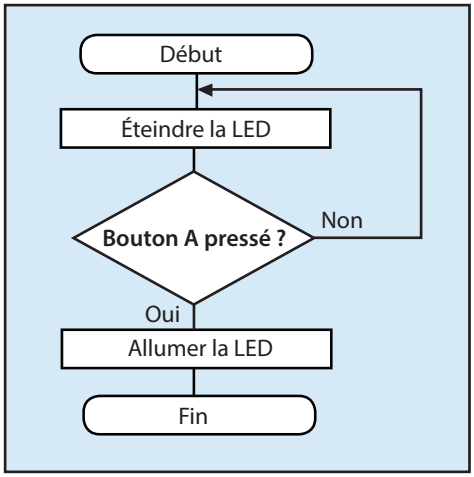



```
1 from pystubit.board import display, button_a
2
3 while True:
4     if button_a.get_value() == 0:
5         display.set_pixel(0, 0, display.RED)
6     else:
7         display.clear()
```

La ligne 1 de ce programme, **from pystubit.board import display, button_a**, importe les deux objets **display** et **button_a**, ce qui te permettra de les utiliser pour contrôler à la fois l'affichage des LED et le bouton A dans ce programme.

La méthode **get_valued()** utilisée avec l'objet **button_a** à la ligne 4 te permet de trouver l'état actuel du bouton A. Si le bouton A est pressé, la valeur récupérée sera **0** et remplira la condition. Quand la condition est remplie, la LED aux coordonnées (0,0) sur le panneau s'allumera en rouge (ligne 5 du programme). Si la valeur récupérée n'est pas 0 (le bouton A n'est donc pas pressé), le panneau de LED s'éteindra (ligne 7 du programme).

Qu'arrive-t-il si on ne fait pas de ce programme une boucle ? Observe l'organigramme et le programme dans lesquels l'instruction **while** a été retirée.



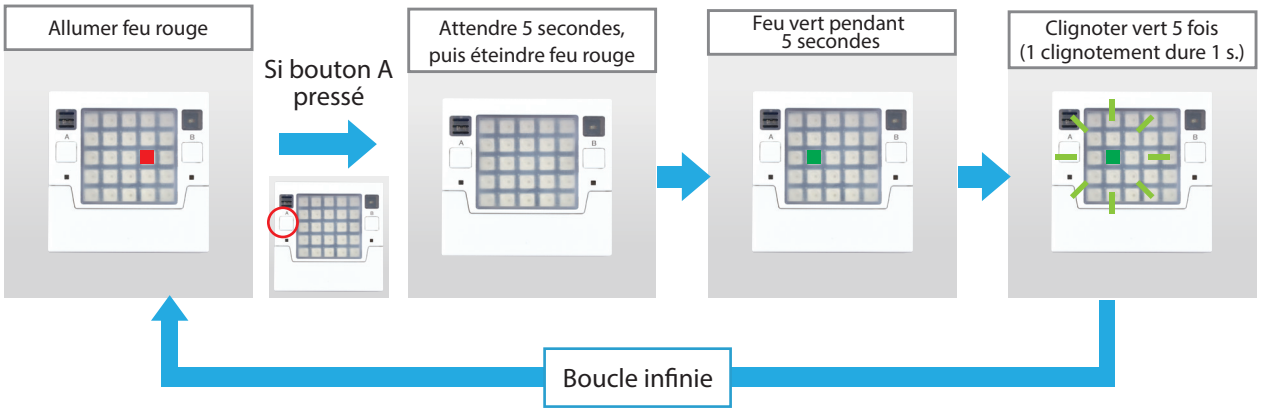

```
1 from pystubit.board import display, button_a
2
3 if button_a.get_value() == 0:
4     display.set_pixel(0, 0, display.RED)
5 else:
6     display.clear()
```

Si tu lances le programme tel quel, la carte ne répondra pas du tout quand tu appuies sur le bouton A ! Encore une fois, cela est causé par la vitesse d'exécution de ton programme par la carte - le programme finit de s'exécuter alors même que tu viens de le démarrer. C'est pourquoi tu as besoin de l'instruction **while** pour que le programme ne s'arrête jamais de vérifier si le bouton A est pressé.

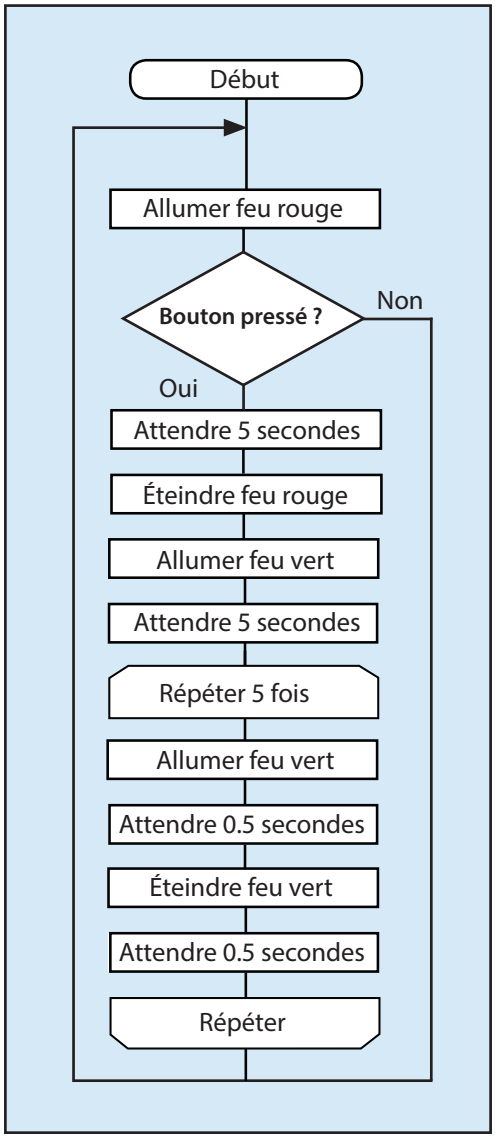
Exercice de programmation

Programmer un feu de signalisation avec bouton-poussoir

Dessine un organigramme qui reproduit la séquence d'allumage montrée ci-dessous, puis essaie d'en écrire le programme.



Exemple de programme



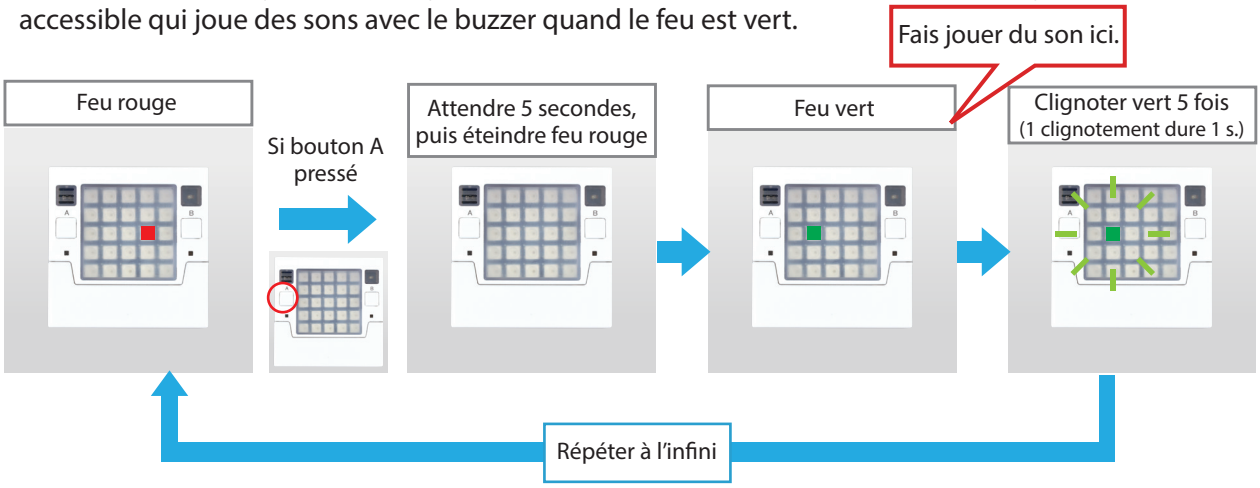

```
1  from pystubit.board import display, button_a
2  import time
3
4  while True:
5      display.set_pixel(3, 2, display.RED)
6      if button_a.get_value() == 0:
7          time.sleep(5)
8          display.clear()
9          display.set_pixel(1, 2, display.GREEN)
10         time.sleep(5)
11         for i in range(5):
12             display.set_pixel(1, 2, display.GREEN)
13             time.sleep(0.5)
14             display.clear()
15             time.sleep(0.5)
```

4. Apprentissages avancés : Programmer des buzzers

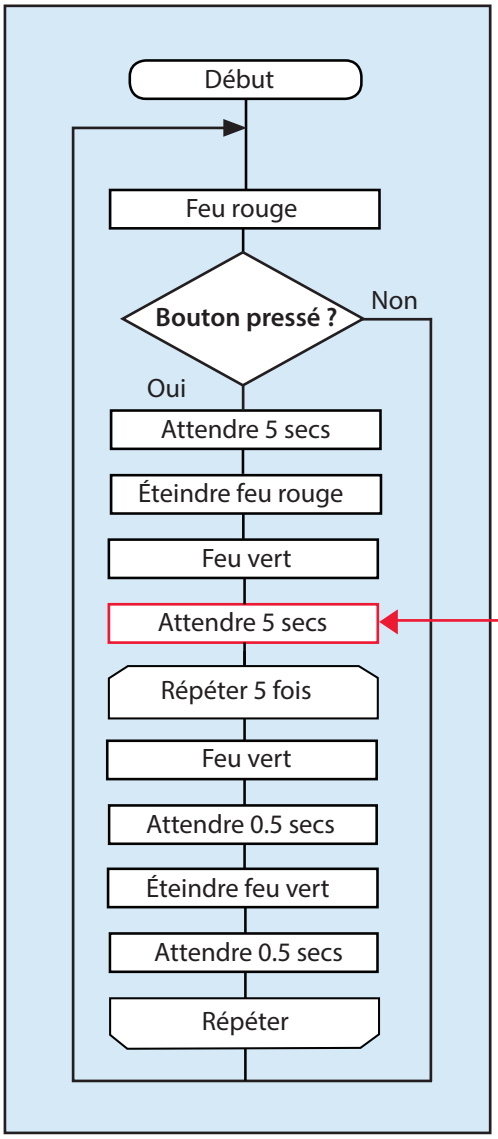
Programmer un feu de signalisation accessible

Certains feux de signalisation lancent des sons quand ils changent de couleur pour permettre aux personnes malvoyantes de savoir quand traverser la rue en toute sécurité. On dit que ce type de signalisation est accessible parce qu'elle sert à plus de gens que la signalisation uniquement visuelle !

Développons le programme de signalisation à bouton-poussoir pour en faire une signalisation accessible qui joue des sons avec le buzzer quand le feu est vert.



Exemple de programme



Nous ajouterons le programme qui lance le buzzer ici.

4.1 Lancer le buzzer

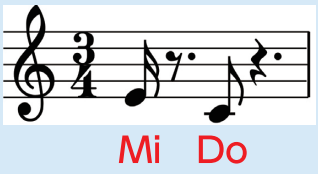
Nous aurons besoin d'un objet **buzzer** de la bibliothèque Studuino:bit pour pouvoir programmer le buzzer de la carte. Le programme montré ci-dessous fait jouer du son au buzzer. Réfère-toi à l'**annexe C** (p.30) pour en savoir plus sur les objets **buzzer**.

```
1 from pystubit.board import buzzer
2 import time
3
4 buzzer.on('60')
5 time.sleep(1)
6 buzzer.off()
```

La ligne 1 de ce programme, **from pystubit.board import buzzer**, importe un objet buzzer de la bibliothèque Studuino:bit pour pouvoir programmer le buzzer. **buzzer.on('60')** à la ligne 4 fait jouer au buzzer la note Do (60). À la ligne 5, **time.sleep(1)** met en pause le programme pendant 1 seconde avant que **buzzer.off()** n'arrête le buzzer à la ligne 6. Le nombre utilisé pour définir la valeur du paramètre dans la méthode **on** est une note MIDI (*Musical Instrument Digital Interface*). Ces nombres équivalent aux notes du solfège : 60 est un Do, 61 est un Do# et ainsi de suite. Réfère-toi à l'**annexe E** (p.30) pour consulter la liste des notes MIDI et jouer avec la méthode **on** des objets **buzzer** et la méthode **sleep** des objets **time**.

4.2 Faire un son de coucou

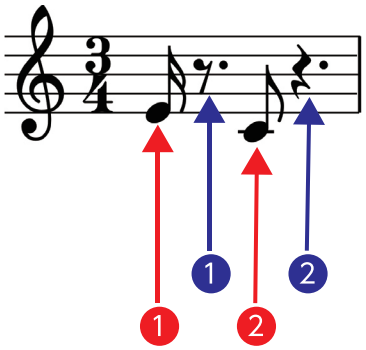
Pour faire un son de coucou, tu as besoin des notes : **Mi (64)** et **Do (60)**.



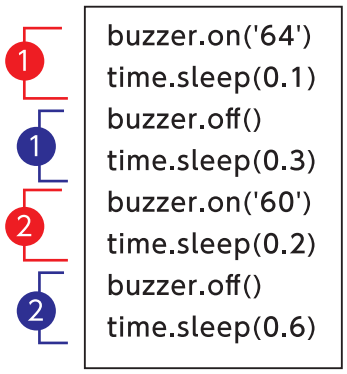
Quelle est la longueur de chaque note ?

Les musiciens lisent des partitions pour connaître la longueur des notes à jouer et les pauses. Tu auras besoin de la partition ci-contre pour jouer le son du coucou. Programme **deux notes** et **deux pauses** aux durées différentes.

La partition du son du coucou



Le programme du son du coucou



Pour jouer 5 fois le son du coucou, tu peux définir une boucle en utilisant une instruction **for**, comme montré ci-contre.

```
1 from pystubit.board import buzzer
2 import time
3
4 for i in range(5):
5     buzzer.on('64')
6     time.sleep(0.1)
7     buzzer.off()
8     time.sleep(0.3)
9     buzzer.on('60')
10    time.sleep(0.2)
11    buzzer.off()
12    time.sleep(0.6)
```

4.3 Jouer un son au feu vert

Mettre une séquence de commandes dans une **fonction** rend la lecture de ton programme plus facile. Enregistre une nouvelle fonction comme suit :

```
def fonction (paramètre):
    Programme
```

Tout comme pour les méthodes, certaines fonctions comportent des paramètres pour lesquelles tu peux régler différentes valeurs et d'autres non. Veille à indenter le programme dans une fonction !

```
1  from pystubit.board import buzzer
2  import time
3
4  def coucou():
5      for i in range(5):
6          buzzer.on('64')
7          time.sleep(0.1)
8          buzzer.off()
9          time.sleep(0.3)
10         buzzer.on('60')
11         time.sleep(0.2)
12         buzzer.off()
13         time.sleep(0.6)
```

La fonction **coucou** ne comporte aucun paramètre à régler. Il te suffit donc juste d'écrire **def coucou():** à la ligne 4 de ton programme. Une fois ta fonction créée, il te suffira d'écrire ceci dans ton programme à chaque fois que tu voudras l'exécuter :

```
fonction (paramètre)
```

Remplace la ligne 10 de ton programme de signalisation à bouton-poussoir de l'**Exercice de programmation** de la page 22 (**time.sleep(5)**, après le feu vert) par ta nouvelle fonction.

```
1  from pystubit.board import display, button_a, buzzer
2  import time
3
4  def coucou():
5      for i in range(5):
6          buzzer.on('64')
7          time.sleep(0.1)
8          buzzer.off()
9          time.sleep(0.3)
10         buzzer.on('60')
11         time.sleep(0.2)
12         buzzer.off()
13         time.sleep(0.6)
14
15 while True:
16     display.set_pixel(3, 2, display.RED)
17     if button_a.get_value() == 0:
18         time.sleep(5)
19         display.clear()
20         display.set_pixel(1, 2, display.GREEN)
21         coucou()
22         for i in range(5):
23             display.set_pixel(1, 2, display.GREEN)
24             time.sleep(0.5)
25             display.clear()
26             time.sleep(0.5)
```

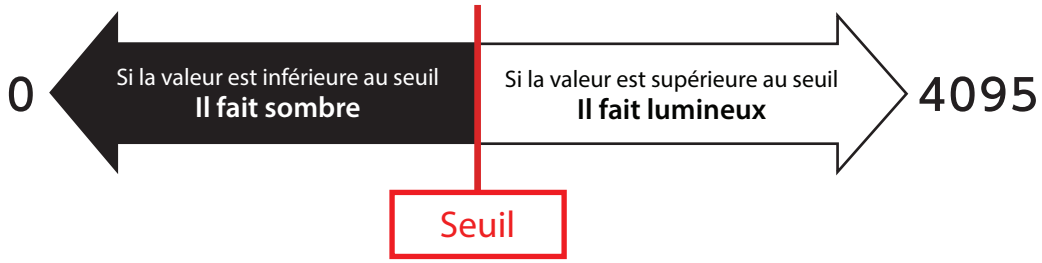
À la ligne 21, **coucou()** exécute la fonction **coucou** créée à la ligne 4.

5. Apprentissages avancés : Programmer des capteurs de lumière

Programmer un capteur de lumière

Programme un capteur de lumière qui allume la lumière quand il fait sombre.

Nous utiliserons le capteur de lumière intégré dans la carte pour mesurer la luminosité. Le capteur de lumière mesure la luminosité sur une échelle de **0** (noir) à **4095** (lumineux). Notre capteur de lumière servira à déterminer le moment où il fera suffisamment sombre pour allumer une LED. Le nombre servant à déterminer le moment où une condition est remplie est un **seuil**.



Pour choisir un seuil, tu auras besoin d'utiliser le panneau LED de la carte pour lire les valeurs du capteur de lumière. Utilise la méthode **scroll** de l'objet **display** pour faire défiler un texte sur le panneau LED. Commence par lancer l'exécution du programme ci-dessous :

```
1  from pystubit.board import display
2
3  while True:
4      display.scroll('Hello')
```

À la ligne 4, **display.scroll('Hello')** fera défiler le mot "Hello" sur le panneau LED. Tu peux écrire du texte dans ses paramètres en utilisant des guillemets simples ou doubles, comme suit :

```
'Text' or "Text"
```

Nous aurons besoin de l'objet **lightsensor** de la bibliothèque Studuino:bit pour pouvoir programmer le capteur de lumière de la carte. Tu peux récupérer les valeurs du capteur de lumière en utilisant la méthode **get_value()**. Réfère-toi à l'annexe D (p.30) pour en savoir plus sur l'objet **lightsensor**.

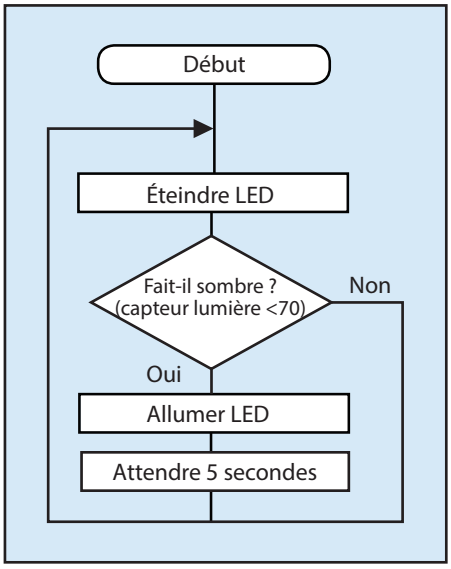
Le paramètre de la méthode **scroll** peut seulement contenir une chaîne de caractères. Or la méthode **get_value()** de l'objet **lightsensor** ne renvoie que des valeurs numériques. Les deux ne sont donc pas compatibles automatiquement. C'est pourquoi nous aurons besoin d'utiliser la fonction **str()** (de la bibliothèque standard Python) pour transformer une valeur numérique en chaîne de caractères.

```
1  from pystubit.board import display, lightsensor
2
3  while True:
4      display.scroll(str(lightsensor.get_value()))
```

Le paramètre de la méthode **display.scroll** à la ligne 4 est défini par **str(lightsensor.get_value())**. Il crée une chaîne en utilisant la méthode **str()** qui convertit la valeur trouvée par la méthode **get_value()** de l'objet **lightsensor** en texte.
Lance ce programme sur ta carte, puis couvre de ta main le capteur de lumière pour observer la façon dont changent ses valeurs.

Ci-dessous un exemple de programme avec un capteur de lumière.

Organigramme




```
1 from pystubit.board import display, lightsensor
2 import time
3
4 while True:
5     display.clear()
6     if lightsensor.get_value() < 700:
7         display.set_pixel(0, 0, display.RED)
8         time.sleep(5)
```

Annexes

Annexe A. Méthodes de l'objet display

Méthode	Ce qu'elle fait...
get_pixel(x, y)	Trouve la couleur de la LED à la ligne x/colonne y. La valeur renvoyée sera dans un format (R/G/B).
set_pixel(x, y, color)	Règle la couleur de la LED à la ligne x/colonne y avec le paramètre **color**. **color** peut être paramétré aux formats (R,G,B), [R,G,B] ou #RGB.
clear()	Règle la luminosité de toutes les LED à 0 (off).
show(iterable, delay=400, *, wait=True, loop=False, clear=False, color=None)	Affiche le paramètre **iterable** sur le panneau LED (images, texte et nombres peuvent tous être réglés dans ce paramètre).
scroll(string, delay=150, *, wait=True, loop=False, color=None)	Fait défiler le paramètre **value** sur le panneau LED horizontalement (chaînes de caractères et de chiffres peuvent être réglées dans ce paramètre).
on()	Allume l'écran d'affichage.
off()	Éteint l'écran d'affichage (cela te permet d'utiliser les ports GPIO connectés au panneau pour d'autres choses).
is_on()	Retourne **True** si l'écran d'affichage est allumé et **False** s'il est éteint.

L'objet **display** contient aussi les propriétés suivantes qui peuvent être utilisées pour régler des LED de couleur : BLACK (elle éteint la LED), WHITE, RED, LIME, BLUE, YELLOW, CYAN, MAGENTA, SILVER, GRAY, MAROON, OLIVE, GREEN, PURPLE, TEAL, NAVY.

Annexe B. Méthodes de l'objet button_a

Méthode	Ce qu'elle fait...
get_value()	Retourne **0** si le bouton est pressé et **1** s'il ne l'est pas.
is_pressed()	Retourne **True** si le bouton est pressé.
was_pressed()	Les objets button stockent des informations sur le moment où leurs boutons sont pressés. Cette méthode retournera **True** si le bouton a été pressé par le passé. Cette information sera réinitialisée après que tu appelles cette méthode dans un programme.
get_presses()	Les objets button stockent des informations sur le moment où leurs informations ont été pressés. Cette méthode retournera le nombre de fois que le bouton a été pressé par le passé. Ce nombre sera réinitialisé après que tu appelles cette méthode dans un programme.

Annexe C. Méthodes de l'objet buzzer

Méthode	Ce qu'elle fait...
on(sound, *, duration=-1)	Le paramètre **sound** peut être réglé sur une note spécifique en utilisant le nom d'une note (comme la chaîne 'C3'-'G9'), le nombre d'une note MIDI (comme la chaîne '48'-'127') ou une fréquence (comme un nombre entier). Le paramètre **duration** est utilisé pour définir la durée pendant laquelle une note doit être jouée et peut être réglé en ms (millisecondes). Si le paramètre **duration** n'est pas défini, le buzzer continuera de jouer jusqu'à ce la méthode **off** soit exécutée. **bzr.on('50',duration=1000)** # Il joue la note MIDI 50 pendant 1 s. **bzr.on('C4',duration=1000)** # Il joue la note C4 pendant 1 s. **bzr.on(440)**
off()	Eteint le buzzer.

Annexe D. Méthodes de l'objet lightsensor

Méthode	Ce qu'elle fait...
get_value()	Trouve la valeur du capteur de lumière intégré dans la carte (0-4095).

Annexe E. Notes MIDI pour l'objet buzzer

		Note											
		Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
Octave	3	48	49	50	51	52	53	54	55	56	57	58	59
	4	60	61	62	63	64	65	66	67	68	69	70	71
	5	72	73	74	75	76	77	78	79	80	81	82	83
	6	84	85	86	87	88	89	90	91	92	93	94	95
	7	96	97	98	99	100	101	102	103	104	105	106	107
	8	108	109	110	111	112	113	114	115	116	117	118	119
	9	120	121	122	123	124	125	126	127				

2. Les bases de la programmation
Programmer des voitures

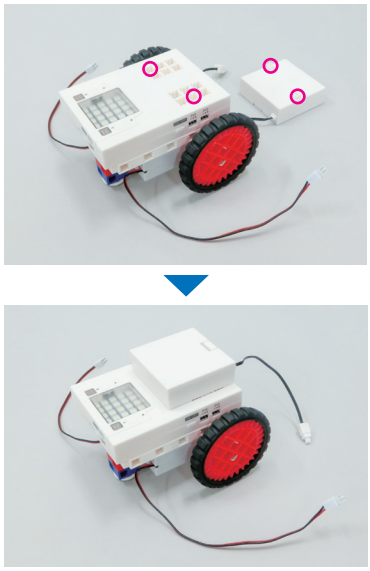
Notions abordées

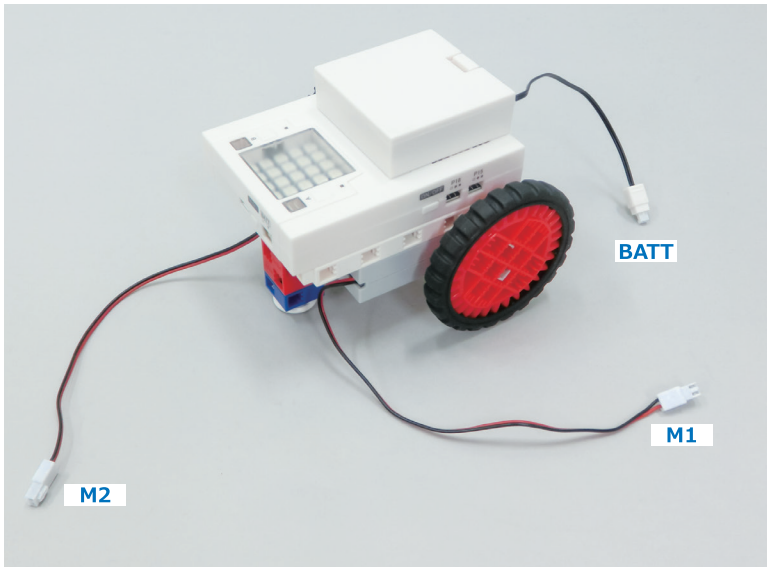
- Programmer des moteurs CC
- Programmer des voitures
- Photoréflecteurs IR
- Voiture anticollision
- Voiture antichute
- Voiture de circuit

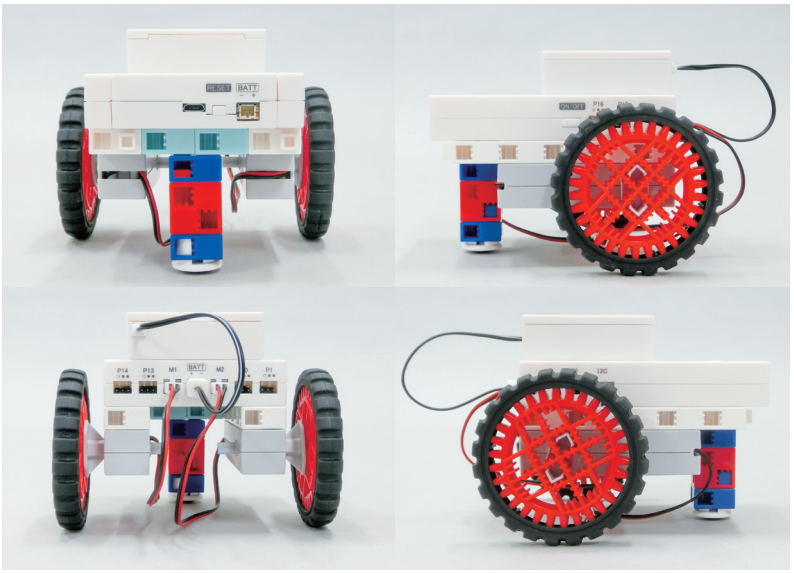
1. Programmer des moteurs CC

1.1 Construire une voiture

⑩



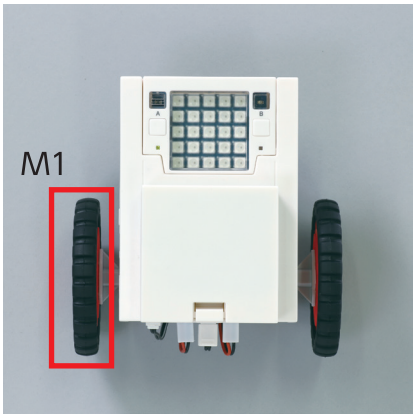




1.2 Tester les moteurs CC

Fais tourner les moteurs CC pour voir comment la programmation les affecte. Nous nous concentrerons sur la roue du moteur branché sur le port **M1**.

★ Veille à ce que la batterie soit branchée sur le port de l'extension quand tu mets en route tes moteurs CC !



Pour programmer les éléments connectés à l'extension, tu dois te servir de la bibliothèque ArtecRobo 2.0 (pyatcrobo2). Cette bibliothèque contient des classes qui te permettent de contrôler des éléments électro--niques variés que tu peux connecter à l'extension. Une **classe** est le code source utilisé pour définir un objet. Tu peux créer un objet à partir d'une classe comme suit :

```
objet = Classe(paramètre)
```

L'objet **display** que nous avons utilisé dans la partie **1. Les bases de la programmation** est un objet basé sur la classe StuduinoBitDisplay. Voici un programme que tu peux utiliser pour créer un objet qui permet de contrôler des moteurs CC :

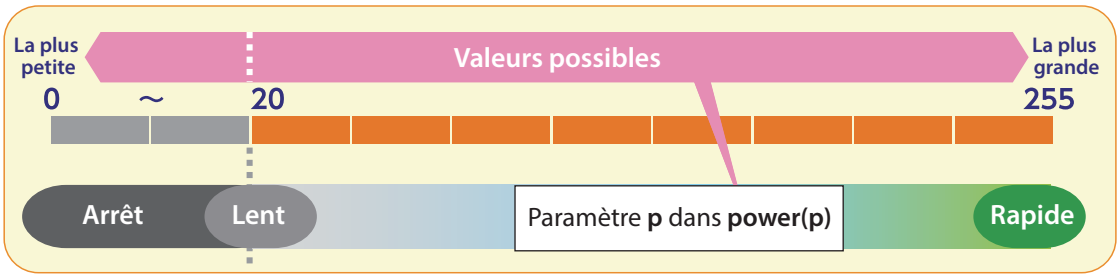
```
1 from pyatcrobo2.parts import DCMotor
2
3 dcm1 = DCMotor('M1')
```

La ligne 1, **from pyatcrobo2.parts import DCMotor**, montre que tu utilises la classe **DCMotor** de la bibliothèque ArtecRobo 2.0. Réfère-toi à l'**annexe A** (p.66) pour en savoir plus sur la classe **DCMotor**. La ligne 3, **dcm1=DCMotor('M1')**, crée un objet appelé **dcm1**.

Tu peux utiliser les propriétés et les méthodes de ce nouvel objet pour contrôler le moteur CC branché sur le port M1 de l'extension. Tu peux utiliser la programmation pour contrôler la puissance à laquelle tourne un moteur et le sens dans lequel il tourne. Si tu oublies de paramétrer la puissance ou le sens, le moteur ne tournera pas !

1.2.1 Régler la vitesse d'un moteur CC

Tu peux contrôler la puissance à laquelle tourne un moteur CC en utilisant la méthode **power(p)** avec un objet DC Motor. Le paramètre **p** de la méthode power peut être réglé à n'importe quelle puissance entre **0** (la plus lente) et **255** (la plus rapide).



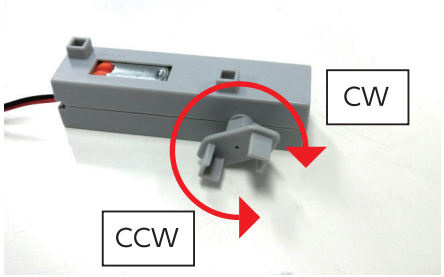

```
1 from pyatcrobo2.parts import DCMotor
2
3 dcm1 = DCMotor('M1')
4 dcm1.power(255)
5 dcm1.cw()
```

La ligne 4 de ce programme utilise la méthode **power** pour régler la puissance de l'objet DCMotor que nous avons créé à la ligne 3 à **255**. La ligne 5 règle le sens dans lequel ce moteur CC tourne, comme nous l'expliquerons ensuite.

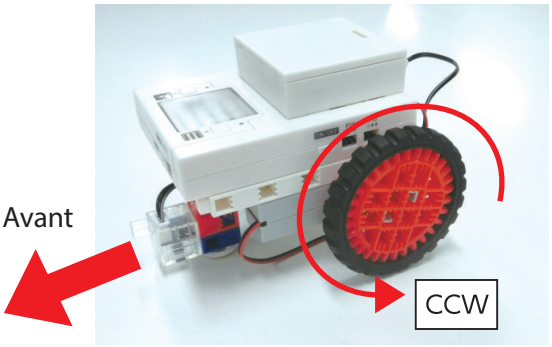
1.2.2 Régler le sens de rotation d'un moteur CC

Tu peux utiliser les méthodes **cw()** et **ccw()** avec un objet DCMotor pour contrôler le sens dans lequel tourne un moteur CC.

'cw' dans la méthode **cw()** est un acronyme pour clockwise (*dans le sens des aiguilles d'une montre*). Cette méthode fait donc tourner le moteur dans le sens des aiguilles d'une montre. Au contraire, ccw est un acronyme pour counter-clockwise. La méthode **ccw()** fait donc tourner le moteur dans le sens contraire des aiguilles d'une montre.



Quand on fait tourner les moteurs de la voiture dans le sens contraire des aiguilles d'une montre, la voiture avance, comme montré ci-dessous.

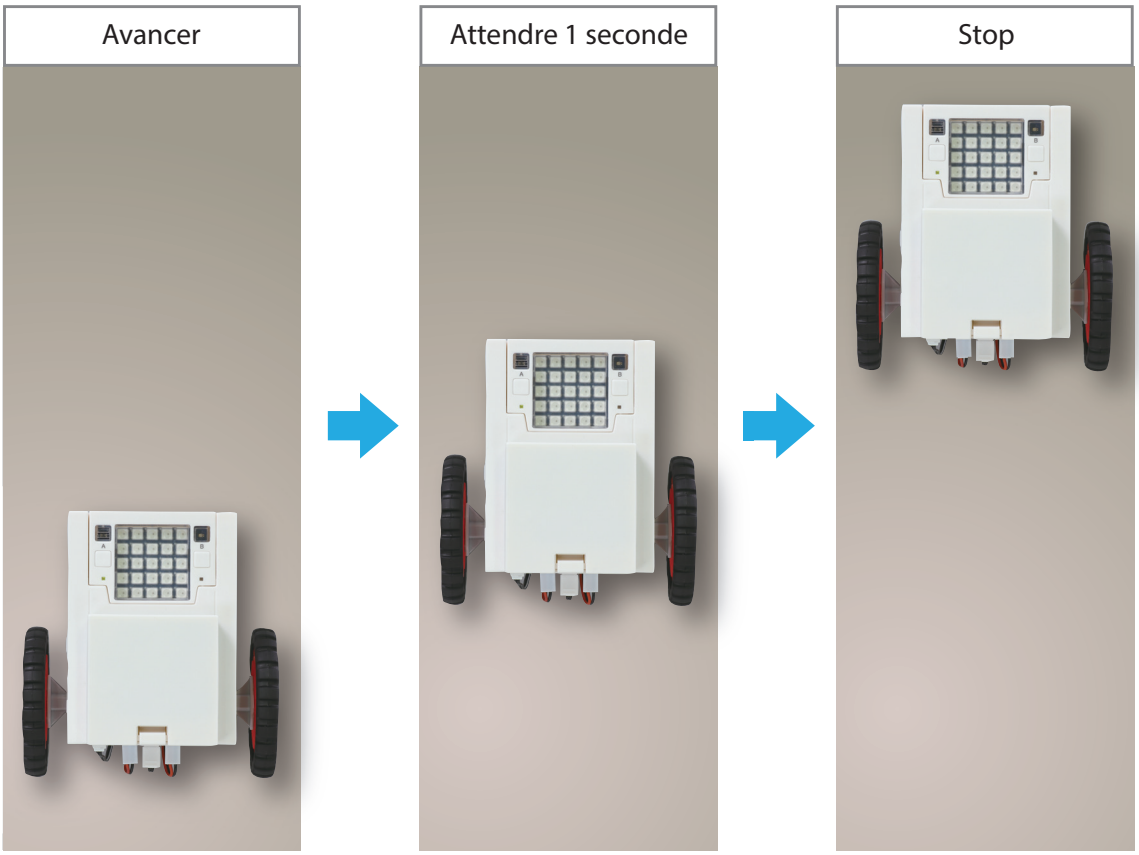


1.2.3 Arrêter un moteur CC

Tu peux utiliser les méthodes **stop()** et **brake()** avec un objet DCMotor pour arrêter un moteur. La méthode **stop()** éteint l'alimentation du moteur CC, ce qui le fait ralentir avant de s'arrêter complètement. La méthode **brake()** place de la charge électrique sur le moteur, le faisant court-circuiter et s'arrêter rapidemment et précisément.

1.3 Programmer un moteur CC

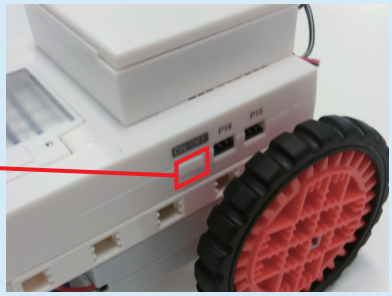
Programme ta voiture à avancer pendant seulement 1 seconde.



```
1  from pyatcrobo2.parts import DCMotor
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  dcm1.power(255)
7  dcm2.power(255)
8  dcm1.ccw()
9  dcm2.ccw()
10 time.sleep(1)
11 dcm1.brake()
12 dcm2.brake()
```

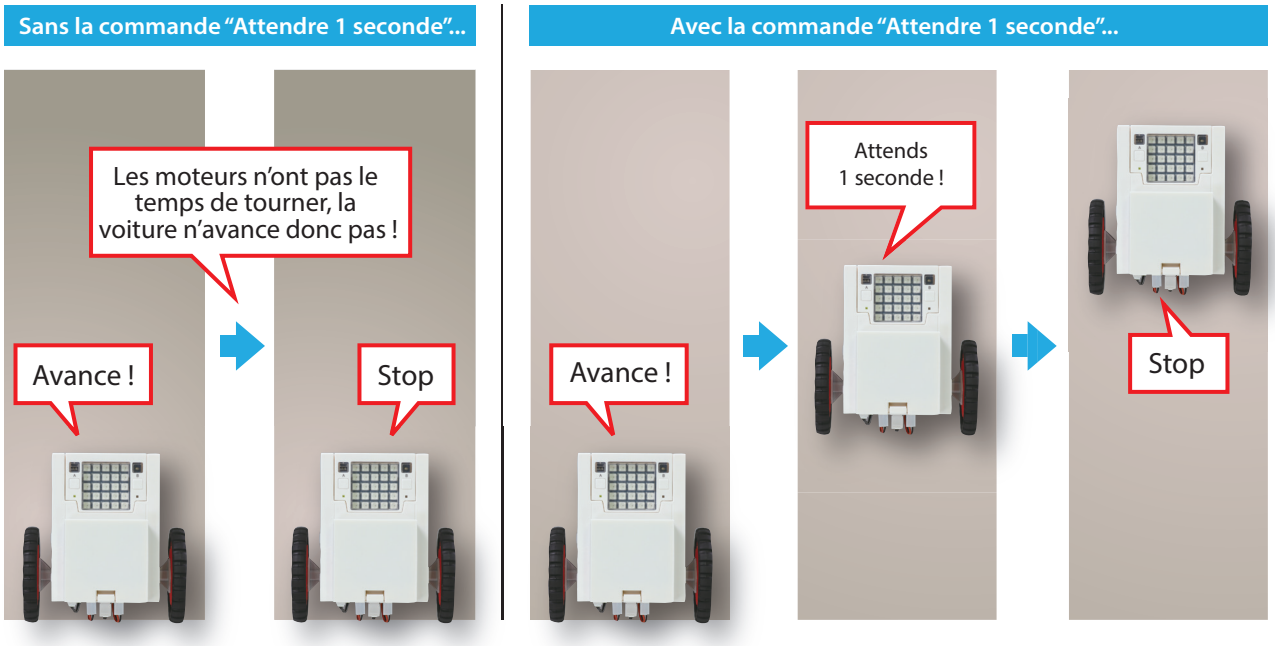
Quand les deux moteurs CC tournent leurs roues dans le même sens à la même vitesse, la voiture avance. Les lignes 4 à 9 de ce programme règlent la vitesse des moteurs M1 et M2 à 255 et les font tourner dans le sens contraire des aiguilles d'une montre. La ligne 10 met en pause le programme pendant 1 seconde, laissant les moteurs continuer à tourner comme avant. Les lignes 11 et 12 arrêtent les deux moteurs CC.

Après le transfert du programme, si tu veux débrancher le câble USB de la voiture avant qu'elle ne démarre, maintiens appuyé le bouton power sur le côté de l'extension pendant 2 secondes pour allumer l'alimentation.




```
1  from pyatcrobo2.parts import DCMotor
2
3  dcm1 = DCMotor('M1')
4  dcm2 = DCMotor('M2')
5  dcm1.power(255)
6  dcm2.power(255)
7  dcm1.ccw()
8  dcm2.ccw()
9  dcm1.brake()
10 dcm2.brake()
```

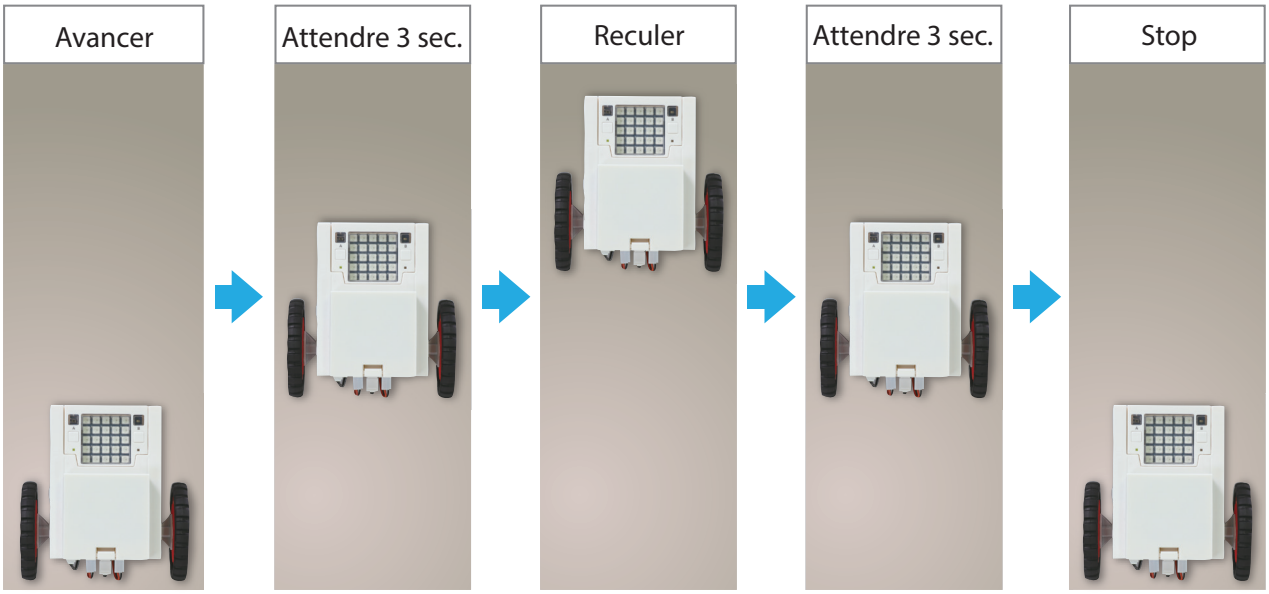
Dans cette version du programme, la ligne 10, **time.sleep(1)**, a été retirée. Comme montré sur l'image de la page suivante, ce programme ne fait pas vraiment rouler la voiture parce que **le programme exécute ses commandes très vite**. Dès qu'il envoie la commande aux moteurs d'avancer, il envoie immédiatement la commande suivante pour les arrêter. Les moteurs s'arrêtent donc aussitôt après.

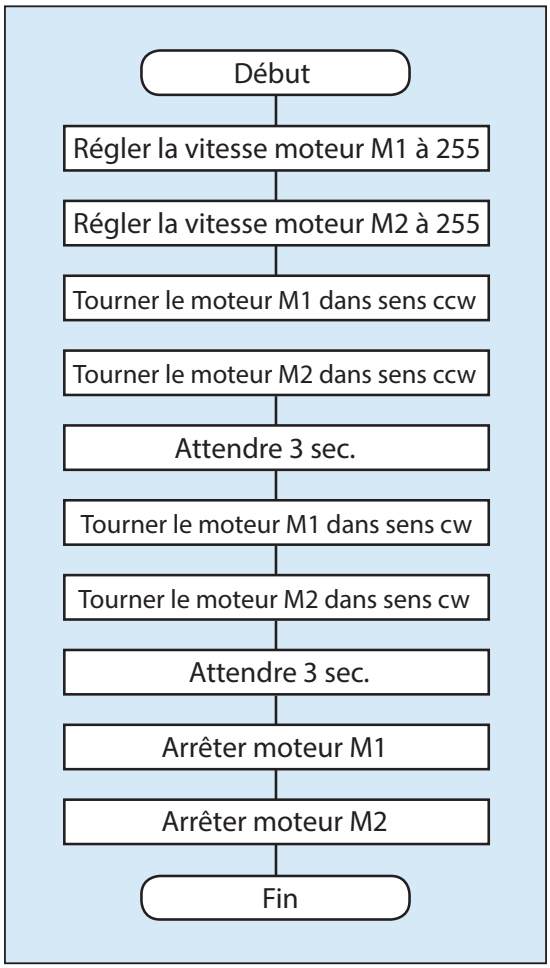


Si ta voiture n’avance pas, ajuste les paramètres des méthodes **power** de ton programme.

Exercice de programmation

Dessine un organigramme qui reproduit la séquence de conduite montrée ci-dessous, puis essaie d'en écrire le programme.

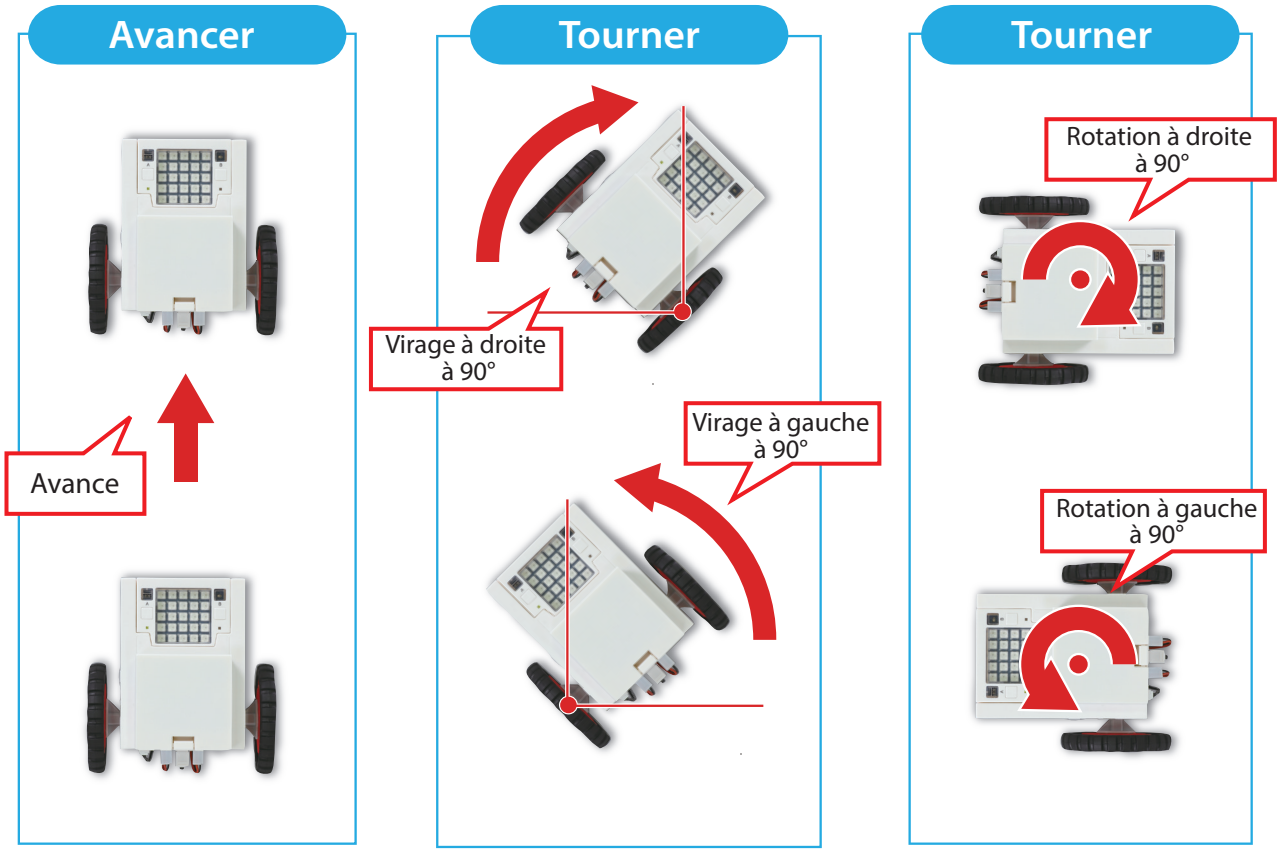





```
1  from pyatcrobo2.parts import DCMotor
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  dcm1.power(255)
7  dcm2.power(255)
8  dcm1.ccw()
9  dcm2.ccw()
10 time.sleep(3)
11 dcm1.cw()
12 dcm2.cw()
13 time.sleep(3)
14 dcm1.brake()
15 dcm2.brake()
```

2. Programmer des voitures

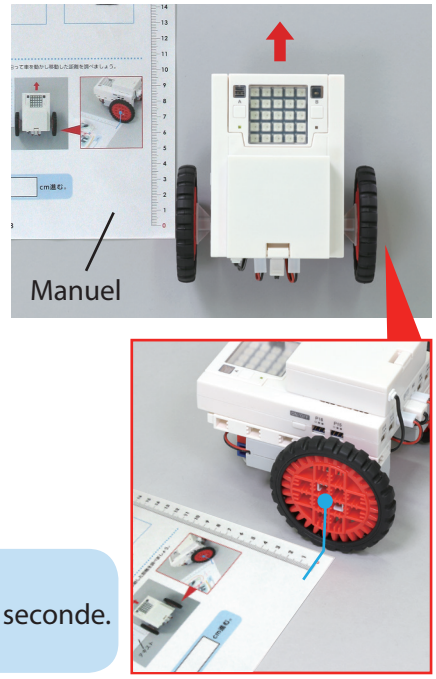
Mesurons la distance parcourue par ta voiture quand elle roule pendant 1 seconde. Nous mesurerons ensuite la durée pendant laquelle devront tourner les moteurs CC pour que ta voiture tourne exactement de 90°. ★ La distance parcourue et le temps nécessaire à un virage peuvent varier selon les moteurs et en fonction du chargement des piles.



2.1 Avancer

Programme ta voiture à avancer pendant 1 seconde. Utilise la règle imprimée sur cette page pour mesurer la distance qu'elle parcourt.

```
1  from pyatcrobo2.parts import DCMotor
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  dcm1.power(255)
7  dcm2.power(255)
8  dcm1.ccw()
9  dcm2.ccw()
10 time.sleep(1)
11 dcm1.brake()
12 dcm2.brake()
```



La voiture parcourt ☐ cm en 1 seconde.

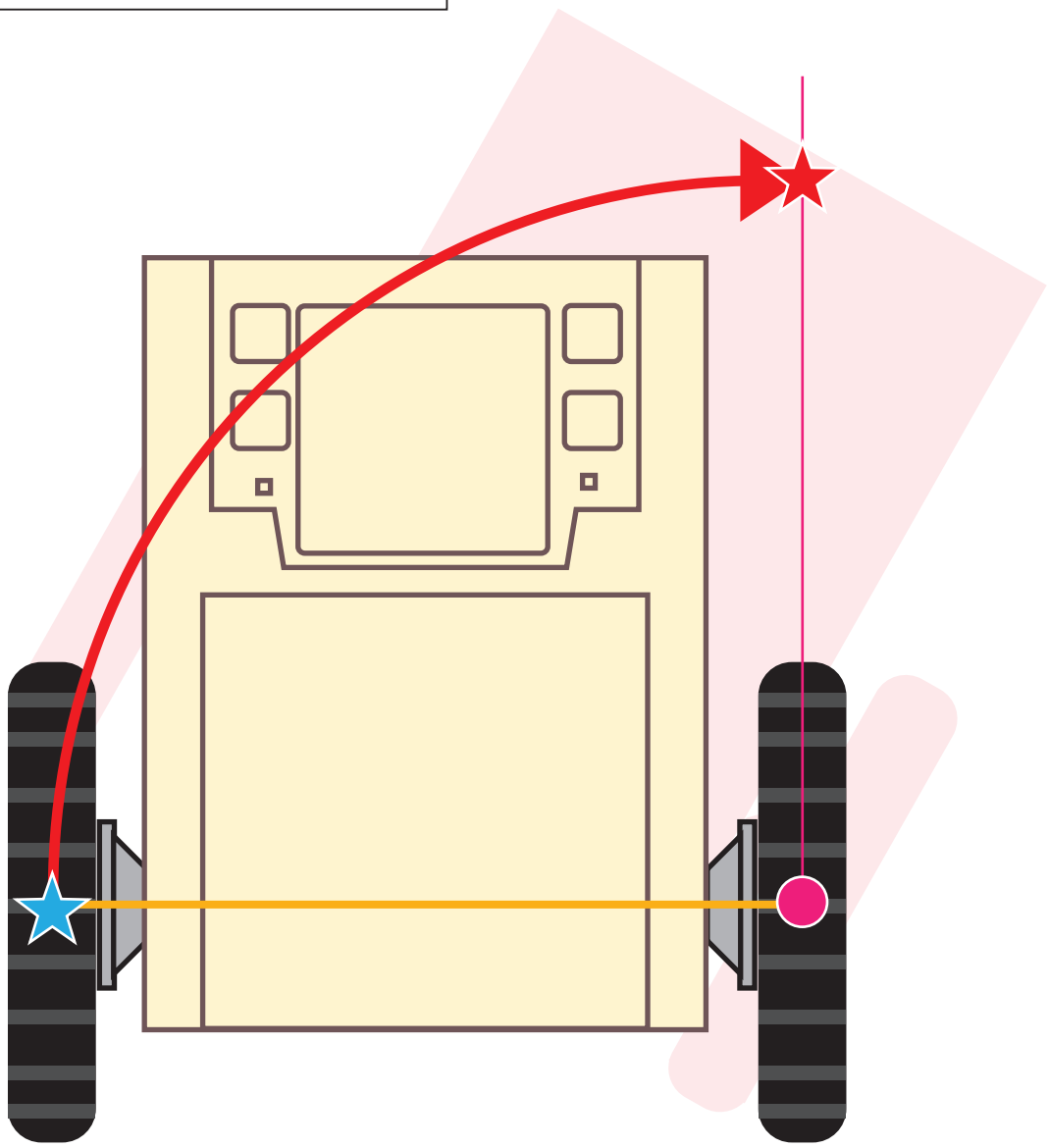
2.2 Tourner à droite

En faisant seulement tourner le moteur gauche (M1) dans le sens contraire des aiguilles d'une montre, ta voiture tournera à droite. Positionne ta voiture sur le dessin ci-dessous et teste-la pour voir combien de temps elle prend pour tourner précisément à 90°.

```
1 from pyatcrobo2.parts import DCMotor
2 import time
3
4 dcm1 = DCMotor('M1')
5 dcm1.power(255)
6 dcm1.ccw()
7 time.sleep(    )
8 dcm1.brake()
```

Reproduis le programme ci-contre et ajuste le paramètre de la méthode **time.sleep** à la ligne 7 pour tester différentes durées.

La voiture prend ☐ secondes pour tourner à 90° à droite.



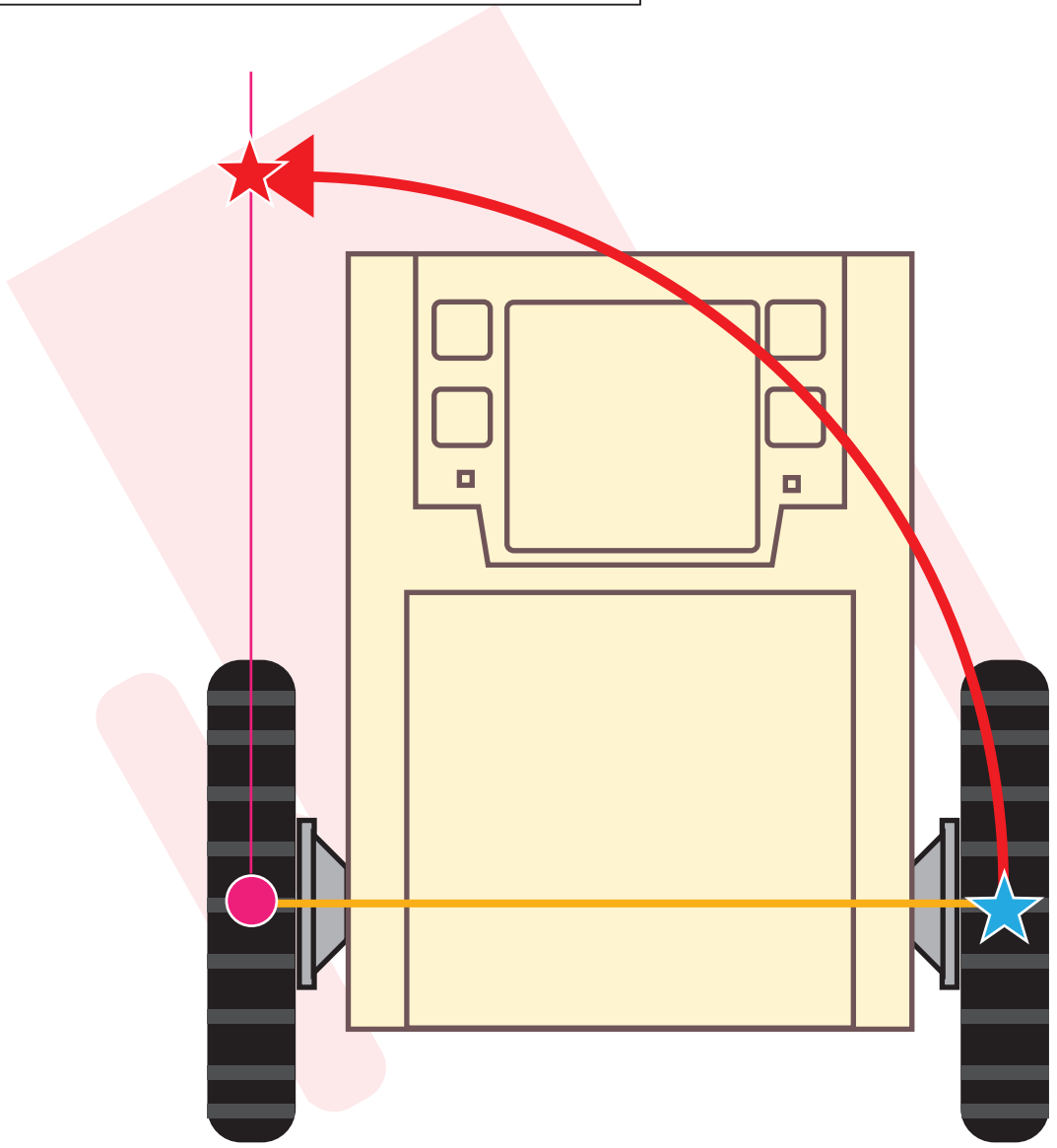
2.3 Tourner à gauche

En faisant seulement tourner le moteur droit (M2) dans le sens contraire des aiguilles d'une montre, ta voiture tournera à gauche. Positionne ta voiture sur le dessin ci-dessous et teste-la pour voir combien de temps elle prend pour tourner précisément à 90°.

```
1 from pyatcrobo2.parts import DCMotor
2 import time
3
4 dcm2= DCMotor('M2')
5 dcm2.power(255)
6 dcm2.ccw()
7 time.sleep(    )
8 dcm2.brake()
```

Reproduis le programme ci-contre et ajuste le paramètre de la méthode **time.sleep** à la ligne 7 pour tester différentes durées.

La voiture prend ☐ secondes pour tourner à 90° à gauche.



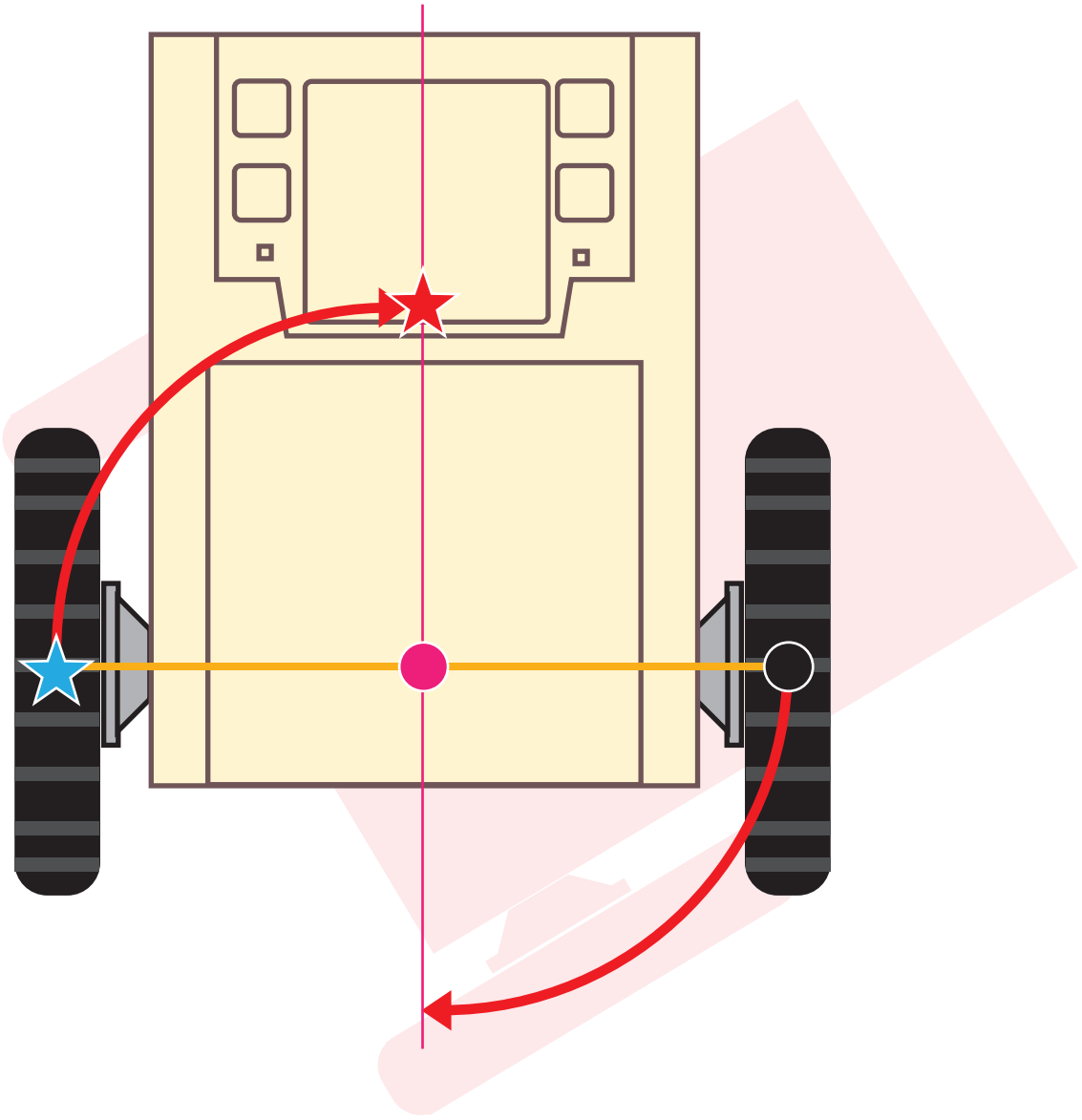
2.4 Faire une rotation à droite

En faisant tourner le moteur gauche (M1) dans le sens contraire des aiguilles d'une montre et le moteur droit (M2) dans le sens des aiguilles d'une montre, ta voiture tournera à droite. Positionne ta voiture sur le dessin ci-dessous et teste-la pour voir combien de temps elle prend pour faire une rotation de 90°.

```
1  from pyatcrobo2.parts import DCMotor
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  dcm1.power(255)
7  dcm2.power(255)
8  dcm1.ccw()
9  dcm2.cw()
10 time.sleep(  )
11 dcm1.brake()
12 dcm2.brake()
```

Reproduis le programme ci-contre et ajuste le paramètre de la méthode **time.sleep** à la ligne 10 pour tester différentes durées.

La voiture prend ☐ secondes pour tourner à 90° à droite.



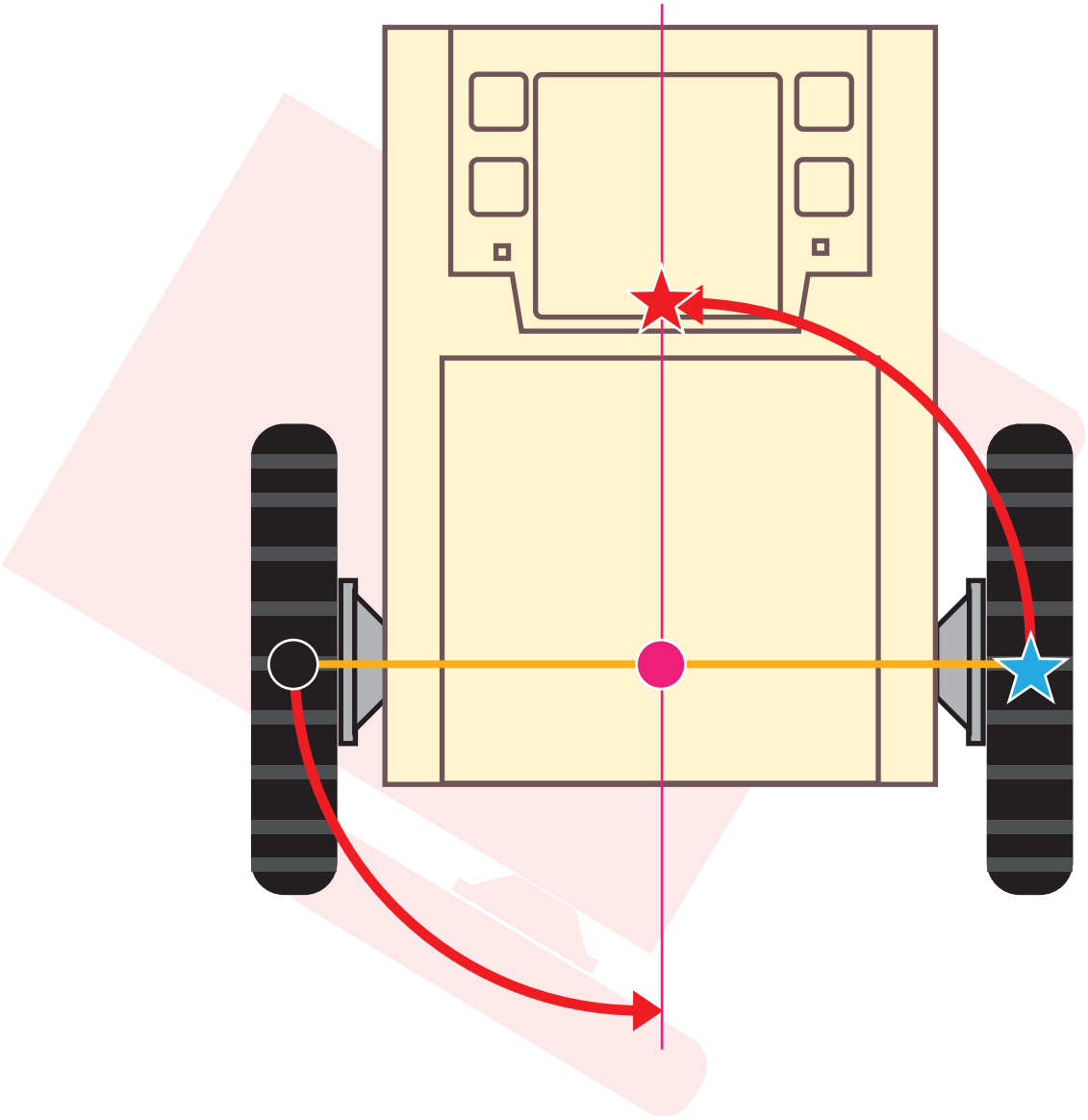
2.5 Faire une rotation à gauche

En faisant tourner le moteur droit (M2) dans le sens contraire des aiguilles d'une montre et le moteur gauche (M1) dans le sens des aiguilles d'une montre, ta voiture tournera à gauche. Positionne ta voiture sur le dessin ci-dessous et teste-la pour voir combien de temps elle prend pour faire une rotation de 90°.

```
1  from pyatcrobo2.parts import DCMotor
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  dcm1.power(255)
7  dcm2.power(255)
8  dcm1.cw()
9  dcm2.ccw()
10 time.sleep(   )
11 dcm1.brake()
12 dcm2.brake()
```

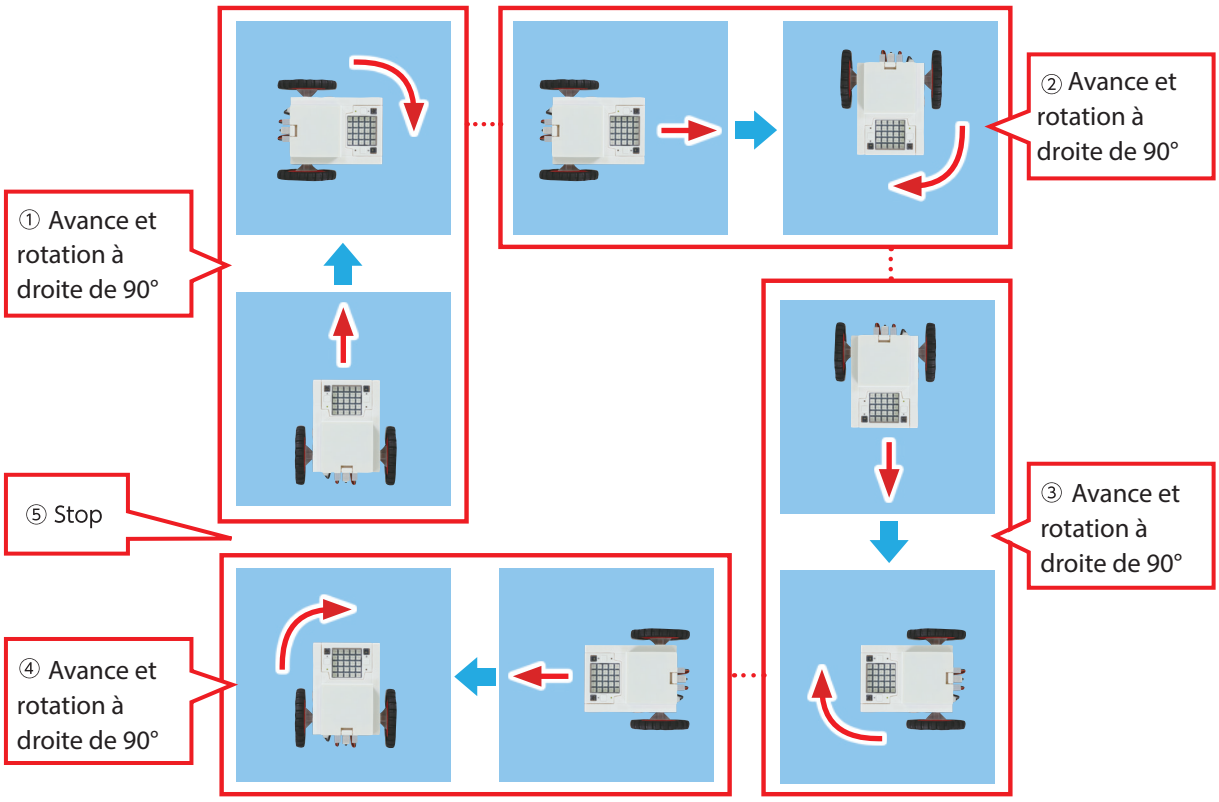
Reproduis le programme ci-contre et ajuste le paramètre de la méthode **time.sleep** à la ligne 10 pour tester différentes durées.

La voiture prend ☐ secondes pour tourner à 90° à gauche.



Exercice de programmation

Programme ta voiture à rouler dans un cercle en répétant les programmes servant à avancer et à faire une rotation à droite.

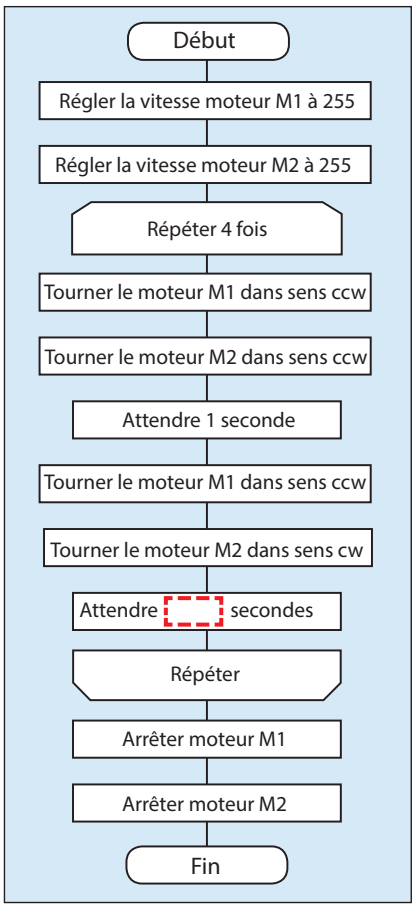


① Faire un organigramme

Ce programme répète l'action d'avancer et des rotations à droite 4 fois pour réaliser un circuit complet. Quand tu élabores un programme sous forme d'organigramme, tu peux utiliser des segments comme ceux ci-dessous pour faire une boucle. Dessine un organigramme pour élaborer ce programme.

Répéter O fois	Début de la boucle
Répéter	Fin de la boucle

★Utilise la durée trouvée dans la partie **2.4** pour les rotations de ta voiture.



② Programmer une boucle

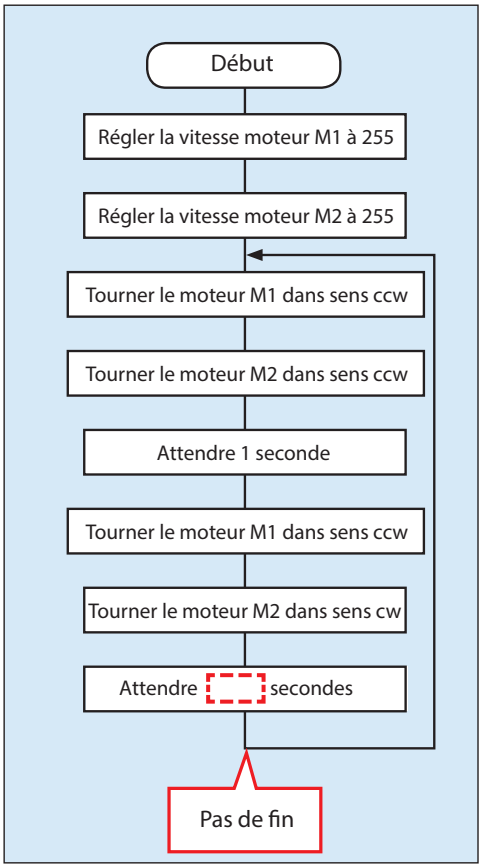
Tu peux écrire un programme en te basant sur un organigramme. Si l'on veut qu'un programme répète les mêmes actions un certain nombre de fois, on peut le faire facilement en utilisant des instructions **for**, comme suit :

```
1  from pyatcrobo2.parts import DCMotor
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  dcm1.power(255)
7  dcm2.power(255)
8  for i in range(4):
9      dcm1.ccw()
10     dcm2.ccw()
11     time.sleep(1)
12     dcm1.ccw()
13     dcm2.cw()
14     time.sleep(    )
15 dcm1.brake()
16 dcm2.brake()
```

③ Programmer une boucle infinie

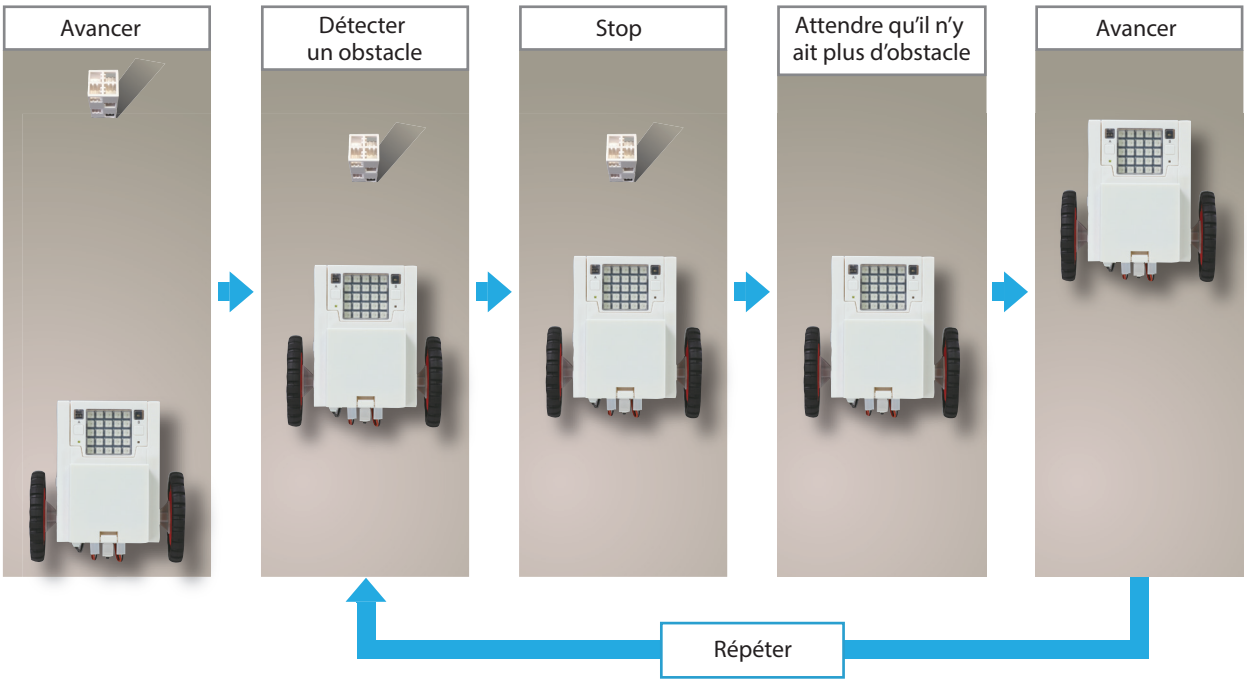
Si tu veux répéter les mêmes actions indéfiniment, tu peux utiliser une instruction **while**. Remplace l'instruction **for** du programme que tu viens de faire par une instruction **while**. En retirant le segment "Fin", le programme continuera à tourner en boucle. Dans un organigramme, on le montre en utilisant une flèche, comme montré ci-contre.

★Tu n'as plus besoin d'arrêter les moteurs, tu peux donc supprimer les lignes 15 et 16 (**dcm1.brake()** et **dcm2.brake()**) du programme ci-dessus.



3. Faire une voiture anticollision

Élabore une voiture qui utilise des photoréflecteurs IR pour détecter et éviter des obstacles sur son chemin !



Systèmes d'aide à la conduite

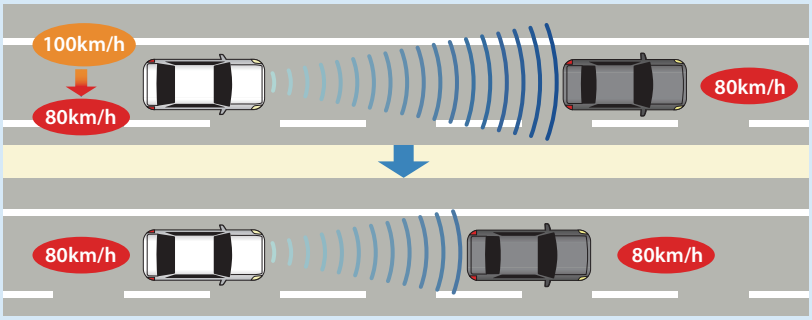
Les voitures modernes sont équipées de plusieurs systèmes basés sur des capteurs pour aider les gens à conduire en toute sécurité.

Système de freinage automatique (ABS)

Ces systèmes actionnent les freins et arrêtent la voiture quand ils détectent qu'elle va entrer en collision dans une voiture, un piéton ou un obstacle.

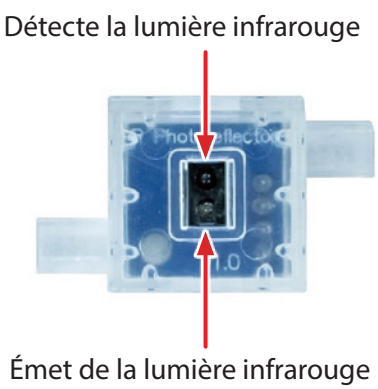
Système automatique de régulation de la vitesse

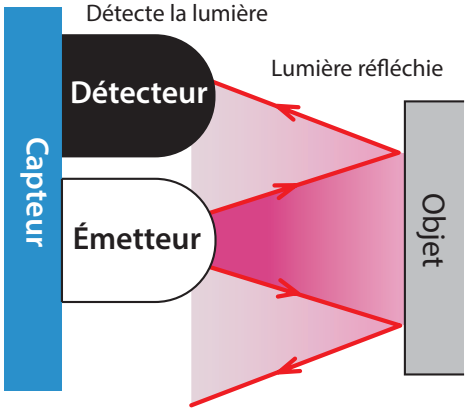
Ces systèmes ajustent la vitesse de la voiture en fonction de la distance avec la voiture devant elle. De quoi faciliter les longs trajets et le passage des bouchons !



3.1 Qu'est-ce qu'un photoréflecteur IR ?

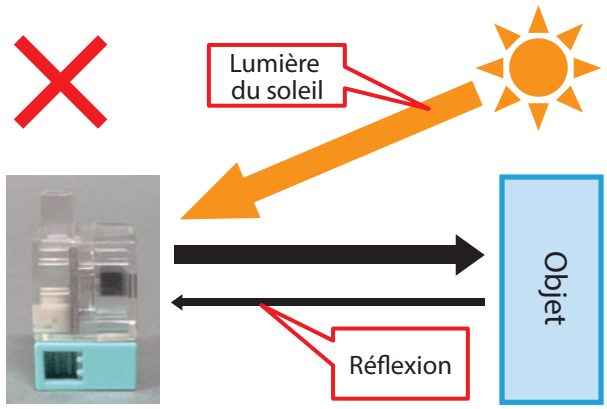
Les photoréflecteurs IR sont des capteurs qui détectent la **lumière infrarouge (IR) refléchie**. Tu observeras sur la face du capteur deux petits ronds. Le rond transparent émet des rayons de lumière infrarouge, tandis que le rond noir détecte la lumière infrarouge. Les rayons infrarouges libérés par l'émetteur rebondissent sur les objets devant le capteur et se réfléchissent pour revenir au détecteur. La force des rayons IR réfléchis que le détecteur réceptionne peut être utilisée pour déterminer si l'objet est proche du capteur.



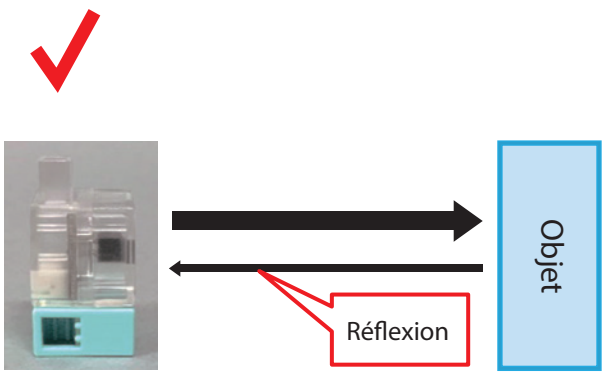


Utiliser des photoréflecteurs IR

La lumière du soleil contient beaucoup de lumière infrarouge ! Veille donc à ce que ton détecteur IR ne soit pas exposé directement à la lumière du soleil, autrement il ne fonctionnera pas correctement.

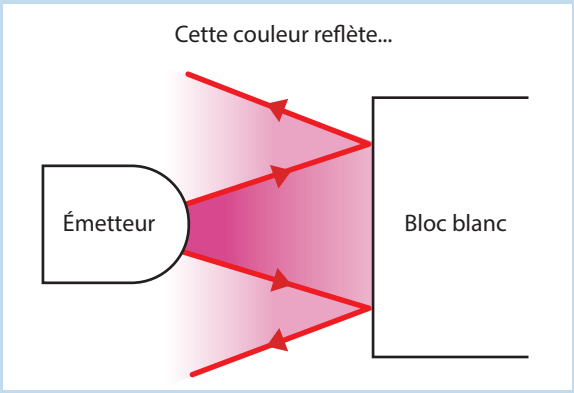


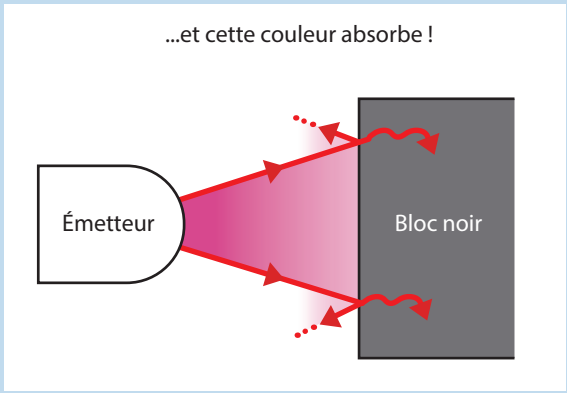
Le capteur détecte les rayons infrarouges du soleil !



Maintenant le capteur peut détecter les rayons infrarouges réfléchis par les objets.

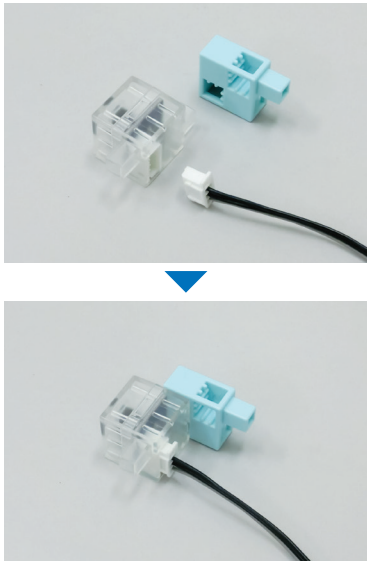
Les couleurs sombres comme le noir ont tendance à absorber plus la lumière qu'elles ne la reflètent. Le capteur réceptionnera donc des signaux plus faibles pour les objets sombres.



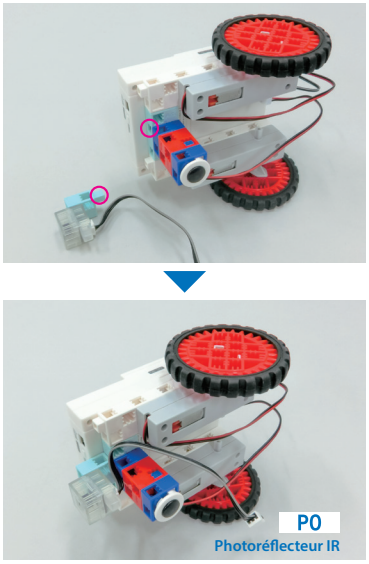


3.2 Construire une voiture anticollision

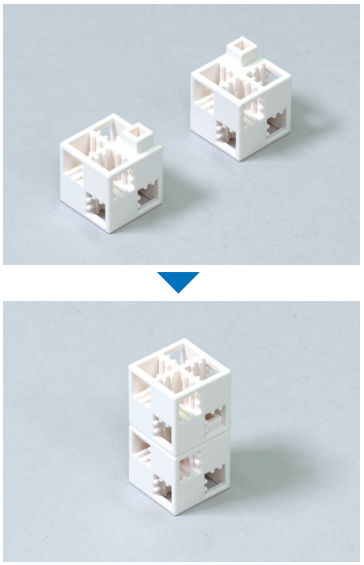
①

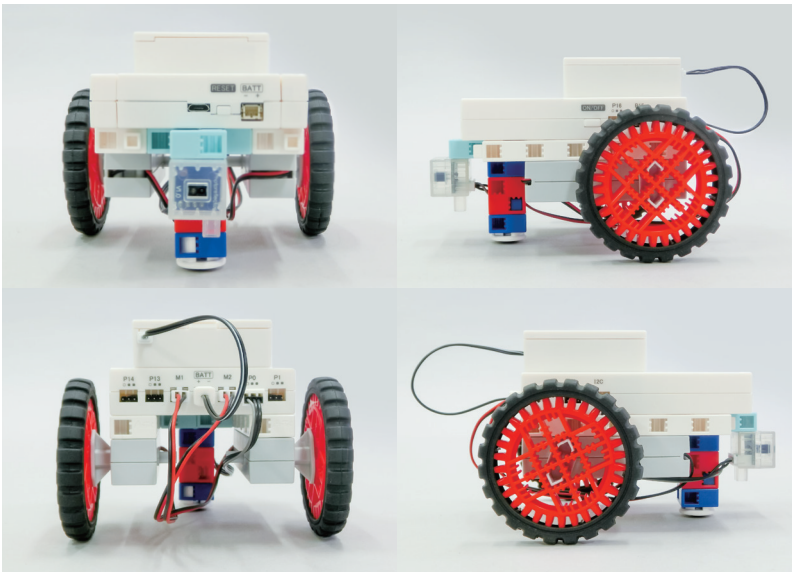


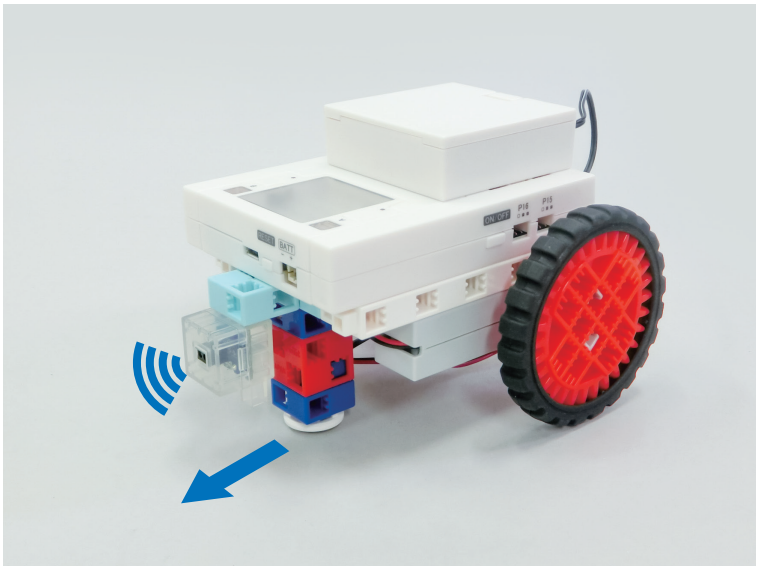
②



③





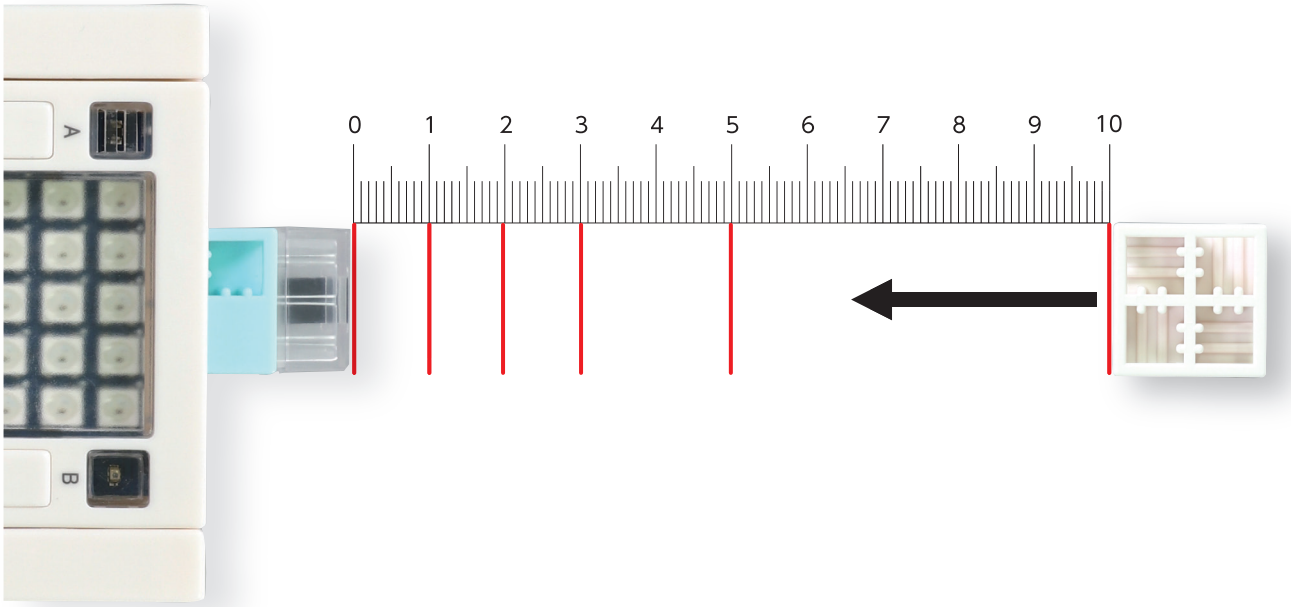


3.3 Tester les valeurs de ton photoréflecteur IR

Nous voulons que ta voiture anticollision soit capable de détecter un obstacle devant elle et de s'arrêter automatiquement pour l'éviter. Le photoréflecteur IR est l'élément qui permettra à ta voiture de détecter de tels objets. Écris et transfère le programme ci-dessous, puis consulte le tableau des capteurs pour observer les valeurs de ton photoréflecteur IR.

```
1 from pystubit.board import display
2 from pyatcrobo2.parts import IRPhotoReflector
3
4 irp = IRPhotoReflector('P0')
5 while True:
6     display.scroll(str(int(irp.get_value())))
```

Ce programme utilise la classe **IRPhotoreflector** pour nous permettre de programmer le photoréflecteur. Réfère-toi à l'**annexe B** (p.66) pour en savoir plus sur la classe **IRPhotoreflector**. Positionne ta voiture de façon à l'aligner sur le dessin ci-dessous. Place ton obstacle (un cube blanc) à 10 cm, 5 cm, 3 cm, 2 cm, 1 cm et 0 cm de distance de ton photoréflecteur IR tout en observant le panneau LED qui affichera les valeurs !

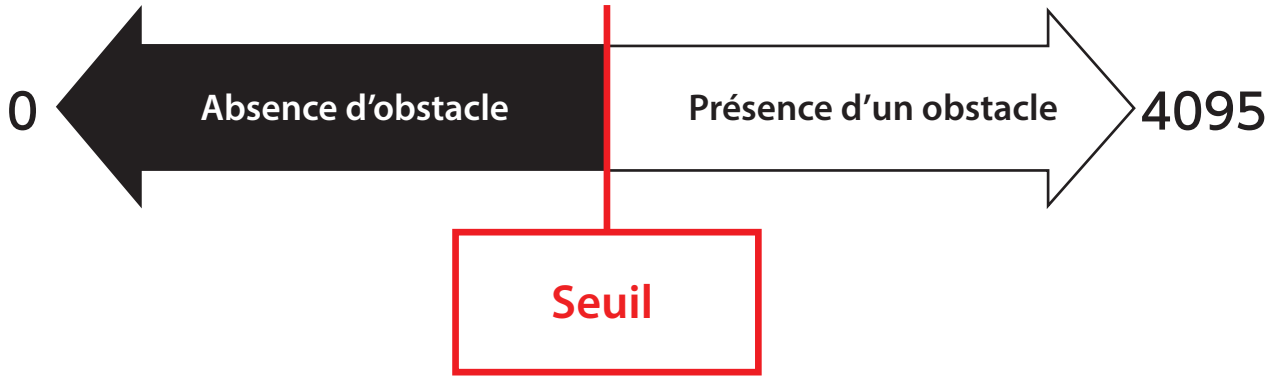


Distance	0 cm	1 cm	2 cm	3 cm	5 cm	10 cm
Valeur						

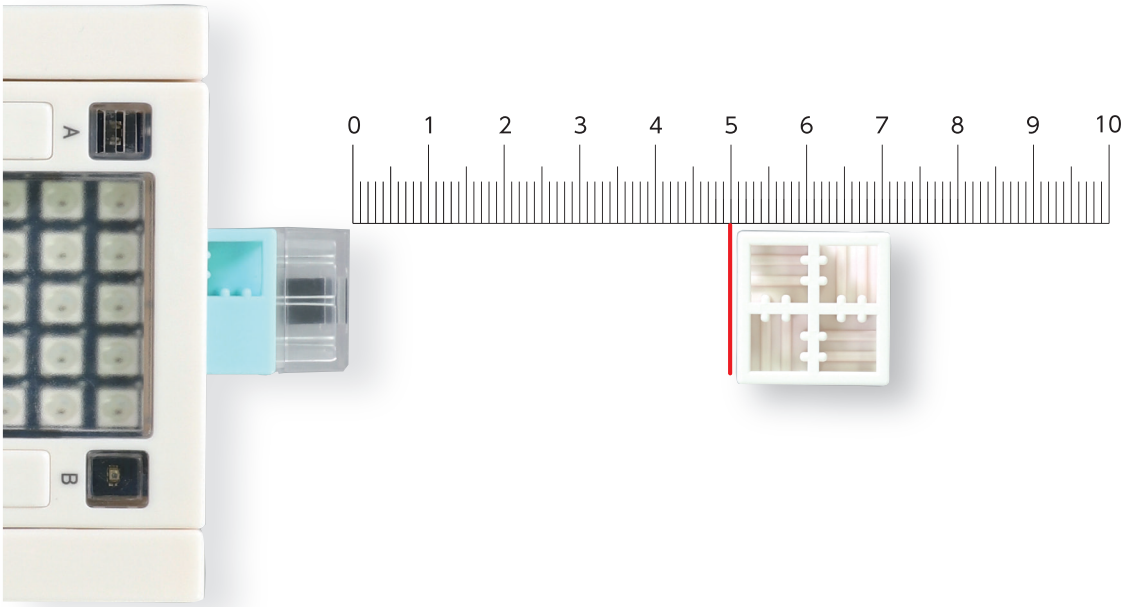
★ Il peut y avoir des différences de valeurs entre différents photoréflecteurs IR alors même qu'ils se trouvent à la même distance d'un objet.

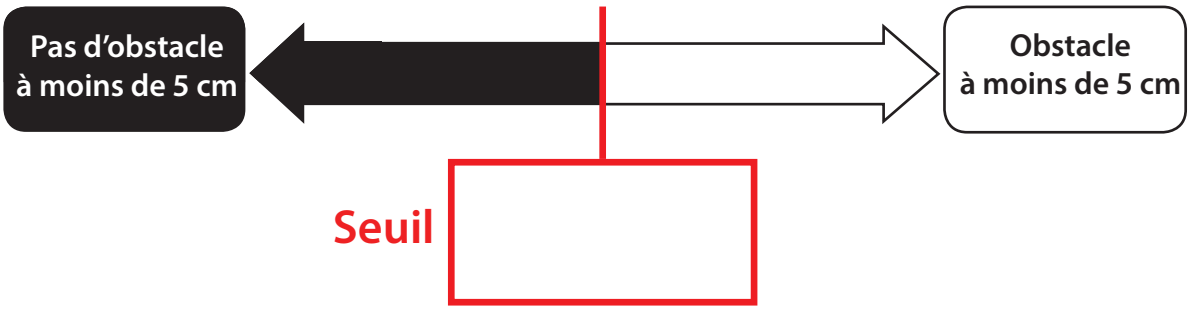
3.4 Trouver un seuil

Pour que ton photoréflecteur IR t'informe de la présence ou de l'absence d'un objet devant lui, il a besoin d'une valeur qui aura été fixée pour marquer la limite entre ces deux états et qu'il pourra utiliser pour différencier ces deux états. Cette valeur est ce qu'on appelle un **seuil**.



Détermine un seuil qui permettra à ta voiture d'**indiquer la présence d'un obstacle à moins de 5 cm devant lui**.

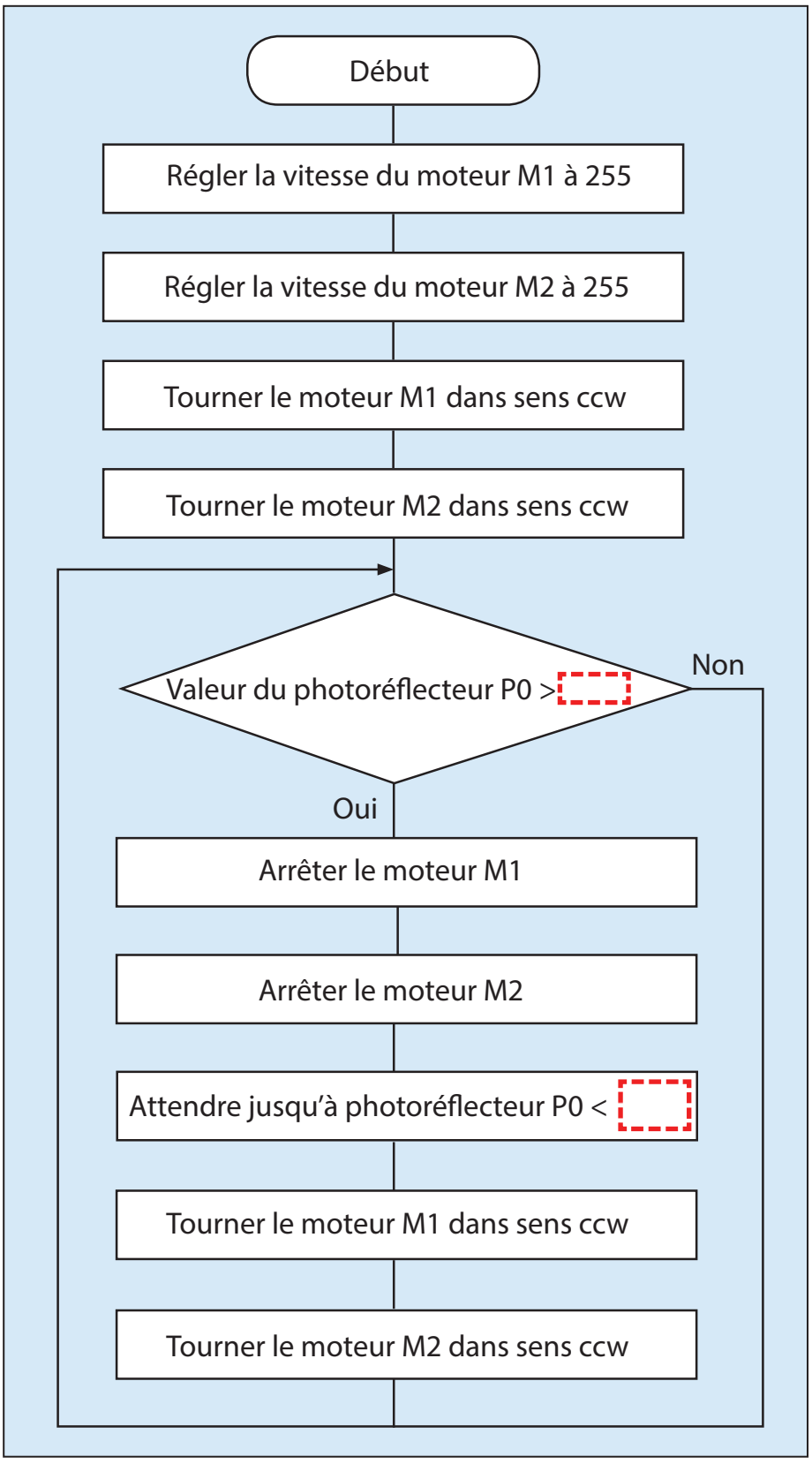




3.5 Créer un organigramme

Dessine un organigramme pour faire le programme d'une voiture anticollision. Tu peux inscrire la valeur de seuil que tu as choisi dans le carré rouge du segment de la condition. Pour que ta voiture attende que l'obstacle soit retiré, tu peux utiliser un segment comme celui-ci :

Attendre jusqu'à ce que la valeur du photoréflecteur IR <



3.6 Programmer une voiture anticollision

Tu peux programmer ce segment de l'organigramme de la partie **3.5** en utilisant une instruction **while**.

Attendre jusqu'à ce que la valeur du photoréflecteur IR < [ ]

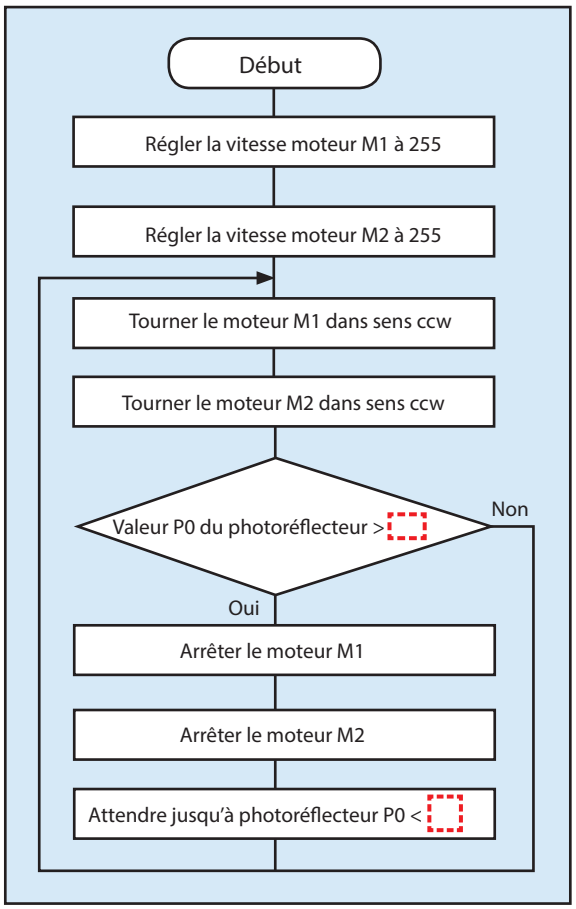
while (instruction conditionnelle):
  pass

Les instructions **pass** envoient la commande de ne rien faire. Dans le programme ci-dessous, tant que l'instruction conditionnelle est vraie, le programme continuera d'exécuter **pass**. Les autres séquences du programme ne s'exécuteront donc pas jusqu'à ce que l'instruction conditionnelle ne soit plus vraie.

```
1  from pyatcrobo2.parts import DCMotor, IRPhotoReflector
2
3  dcm1 = DCMotor('M1')
4  dcm2 = DCMotor('M2')
5  irp = IRPhotoReflector('P0')
6
7  dcm1.power(255)
8  dcm2.power(255)
9  dcm1.ccw()
10 dcm2.ccw()
11 while True:
12     if (irp.get_value() > [ ]):
13         dcm1.brake()
14         dcm2.brake()
15         while (irp.get_value() > [ ]):
16             pass
17         dcm1.ccw()
18         dcm2.ccw()
```

3.6.1 Une voiture anticollision alternative

Tu peux aussi faire fonctionner une voiture anticollision en utilisant un programme comme celui-ci :

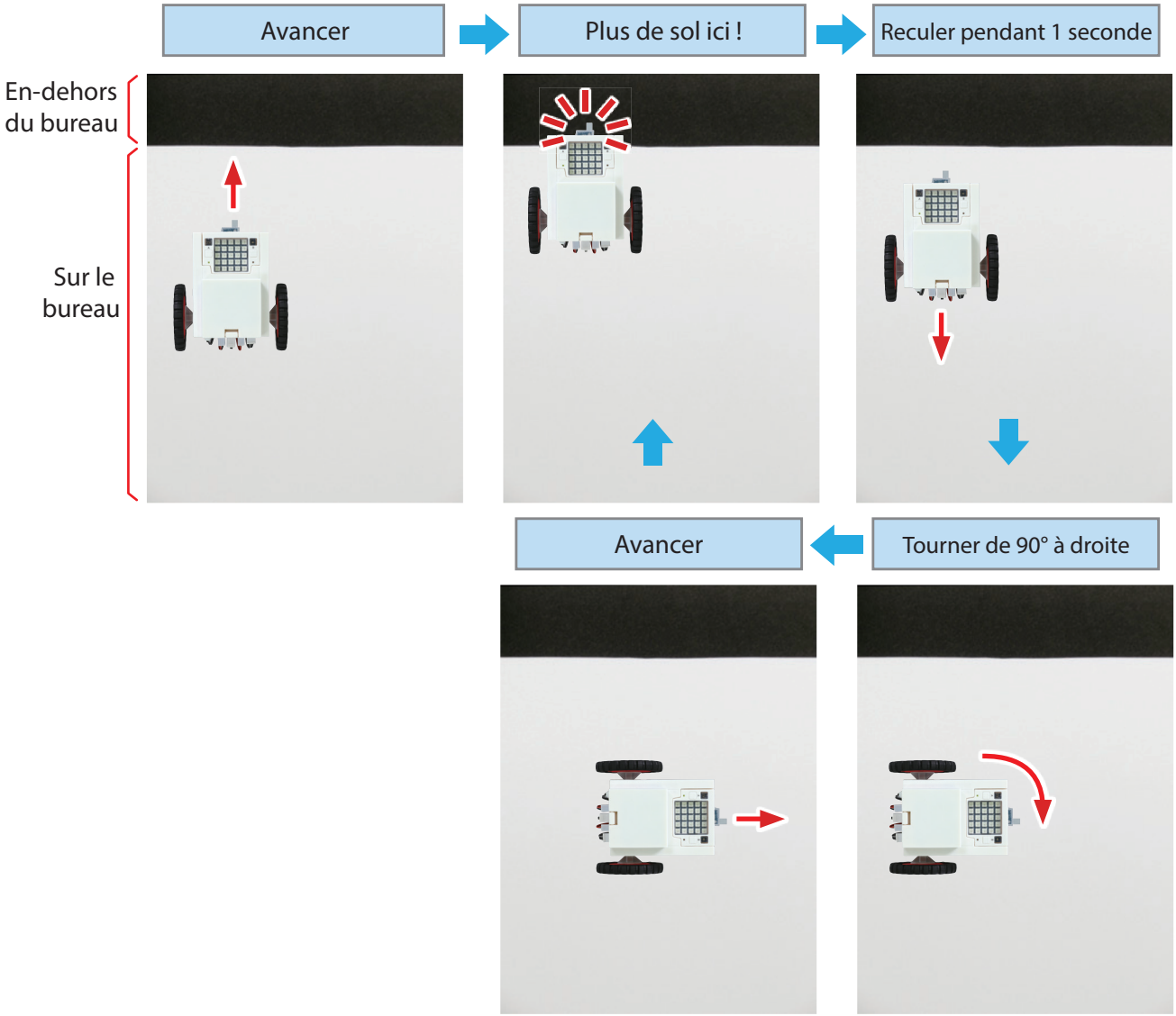



```
1  from pyatcrobo2.parts import DCMotor, IRPhotoReflector
2
3  dcm1 = DCMotor('M1')
4  dcm2 = DCMotor('M2')
5  irp = IRPhotoReflector('P0')
6
7  dcm1.power(255)
8  dcm2.power(255)
9  while True:
10   dcm1.ccw()
11   dcm2.ccw()
12   if (irp.get_value() >      ):
13     dcm1.brake()
14     dcm2.brake()
15     while (irp.get_value() >      ):
16       pass
```

Comme tu peux le voir, il y a plusieurs façons d'écrire un programme qui marche ! Tu peux écrire ton programme de la façon que tu veux, tant que tu obtiens à la fin le résultat escompté. Pense toi-même au problème pour parvenir à un programme qui marche !

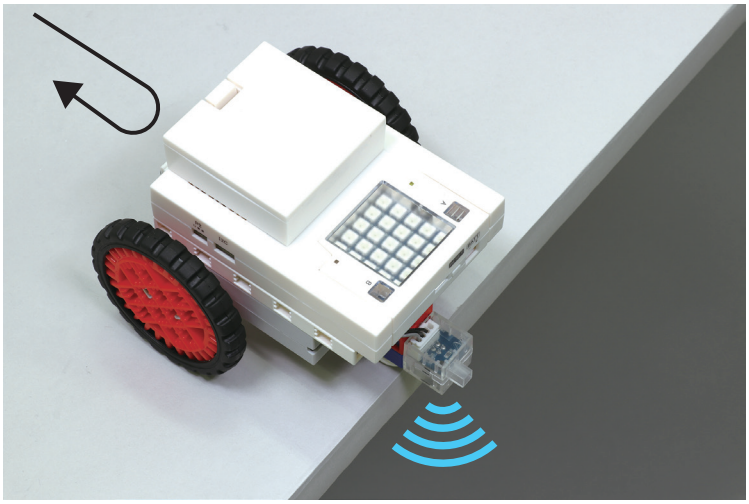
4. Faire une voiture antichute

En utilisant le photoréflecteur IR un peu différemment, on peut faire arrêter la voiture automatiquement d'elle-même pour éviter qu'elle ne chute lorsqu'elle détecte le vide en-dessous d'elle.

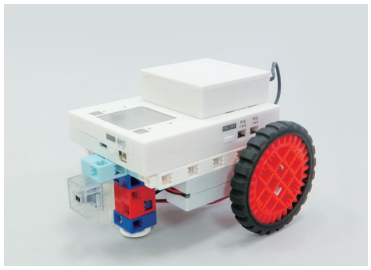


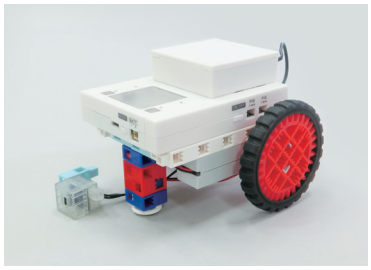
4.1 Construire une voiture antichute

Cette fois, nous voulons que la voiture détecte s'il y a du sol devant lui. Nous devrons donc positionner le photoréflecteur face au sol.

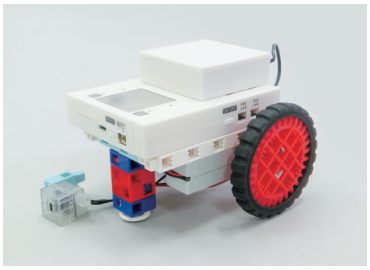


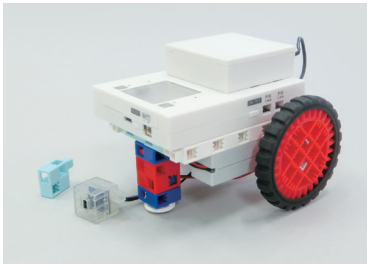
①



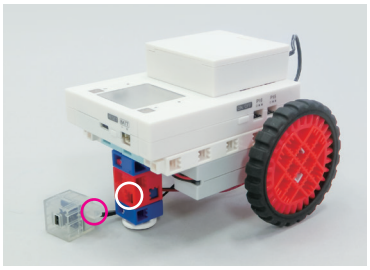


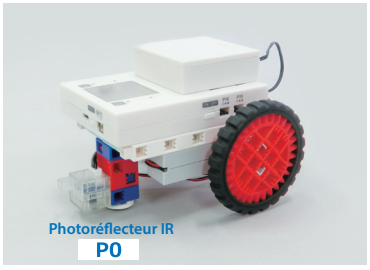
②

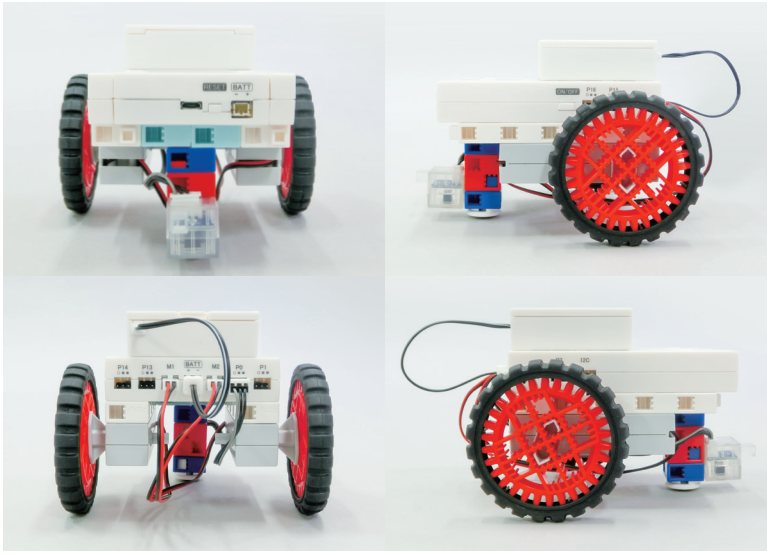


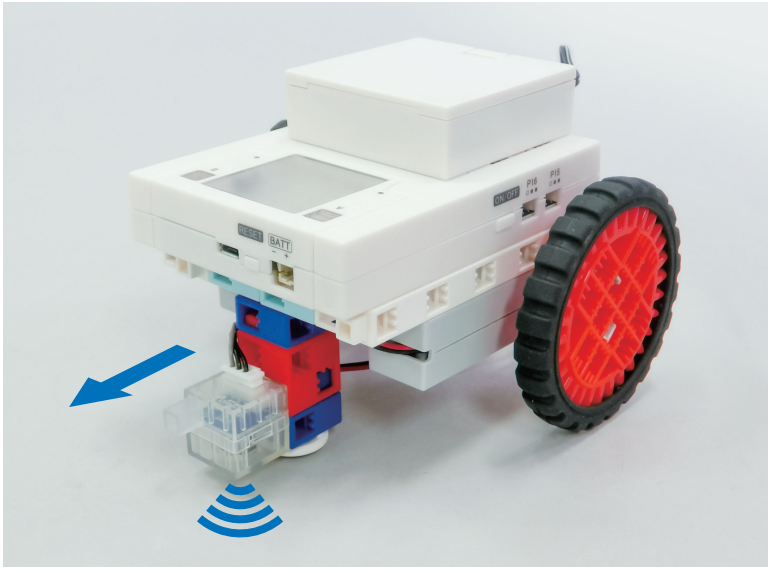


③



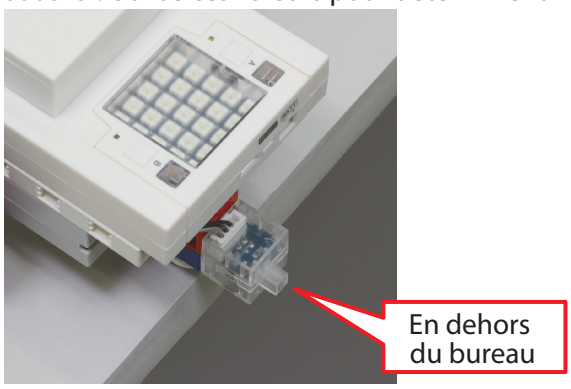


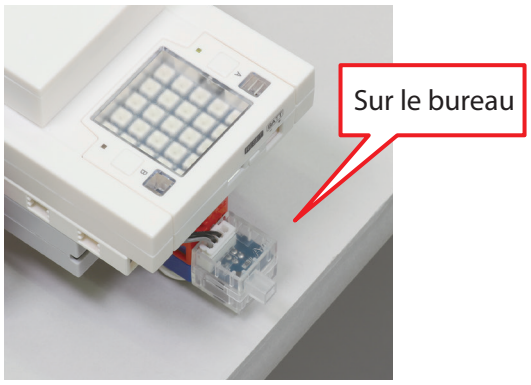


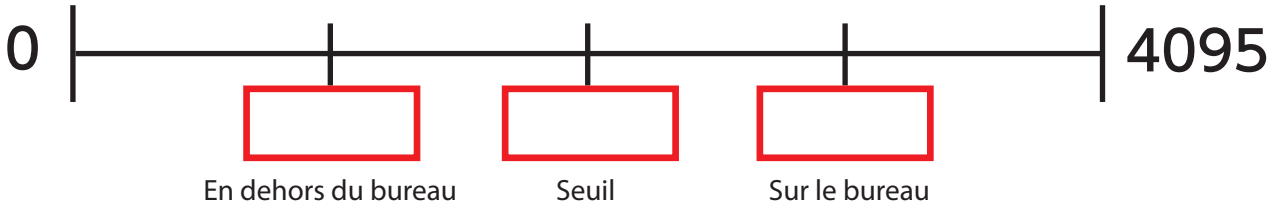


4.2 Trouver un seuil

Avec le même programme qu'en 3.3, tu peux comparer les valeurs du photoréflecteur IR avec et sans le sol sous lui. Utilise ces valeurs pour déterminer un seuil.



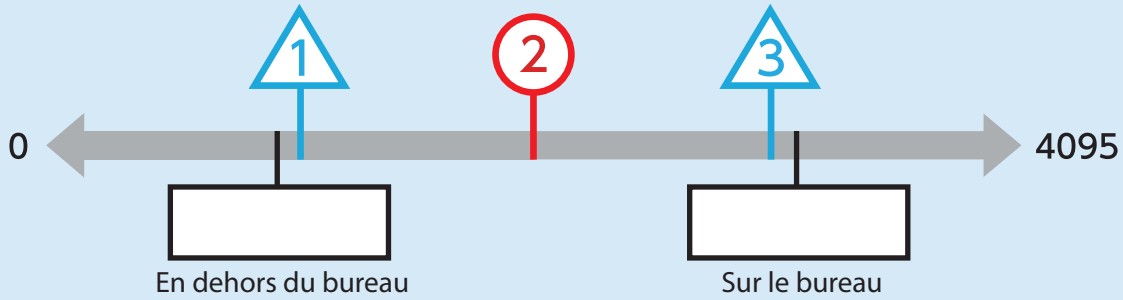




En dehors du bureau | Seuil | Sur le bureau

Comment déterminer un seuil ?

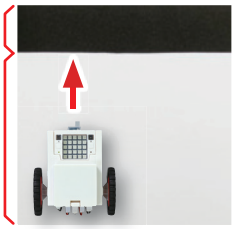
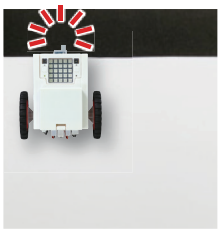
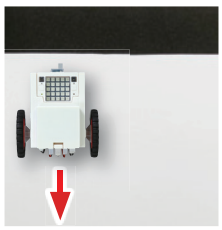
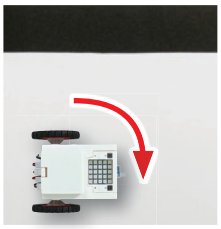
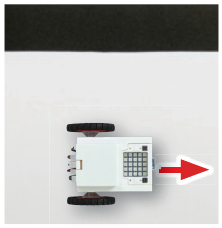
Si tu détermines une valeur de seuil plus proche d'un bout de l'échelle que de l'autre, comme les points 1 et 3, les petites variations de la valeur du capteur pourraient avoir comme conséquence que ton programme détectera des objets qui ne sont pas là ! Les valeurs du capteur peuvent changer facilement à cause de changements minimes de leur environnement, c'est pourquoi il est préférable de choisir une valeur qui soit le juste milieu des valeurs séparant les deux états, comme le point 2.



En dehors du bureau | Sur le bureau

4.3 Planifier les actions de ta voiture

Trace un chemin que ta voiture antichute suivra pour savoir quelles actions programmer.

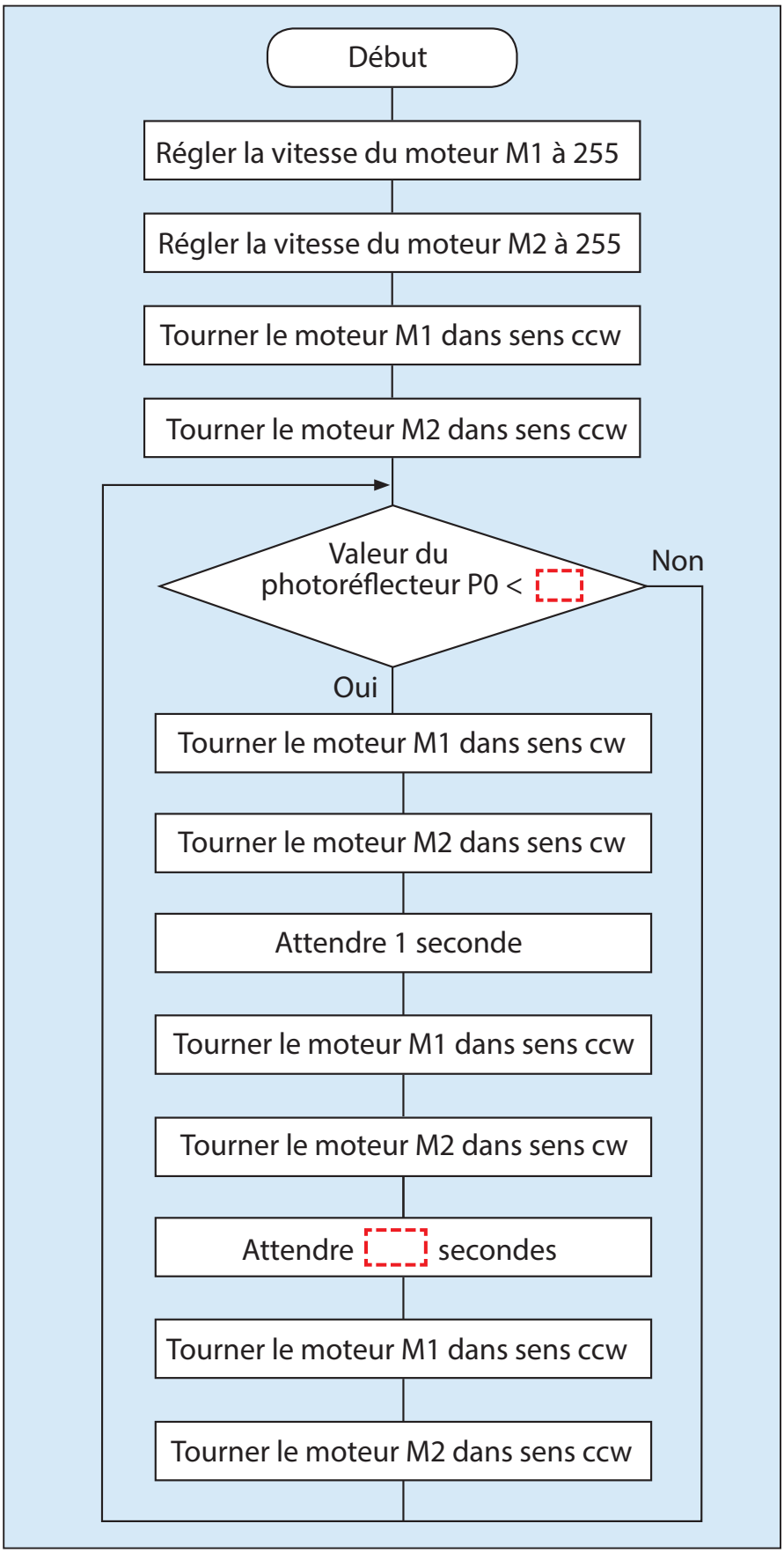
Chemin	Actions	Programme
①	Avancer En dehors du bureau Sur le bureau 	Régler le moteur M1 à une vitesse de 255 Régler le moteur M2 à une vitesse de 255 Faire tourner le moteur M1 en sens ccw Faire tourner le moteur M2 en sens ccw
②	Pas de sol ici ! 	Photoréflecteur IR P0 < Seuil
③	Reculer pendant 1 seconde 	Faire tourner le moteur M1 en sens cw Faire tourner le moteur M2 en sens cw Attendre 1 seconde
④	Tourner de 90° à droite 	Faire tourner le moteur M1 en sens ccw Faire tourner le moteur M2 en sens cw Attendre ☐ secondes
⑤	Avancer 	Faire tourner le moteur M1 en sens ccw Faire tourner le moteur M2 en sens ccw

Répéter ② - ⑤ indéfiniment

★ Utilise le temps trouvé dans la partie **4.2** pour la durée de rotation de ta voiture à 90°.

4.4 Faire un organigramme

Dessine un organigramme pour les séquences d'actions décrites dans la partie **4.3**.



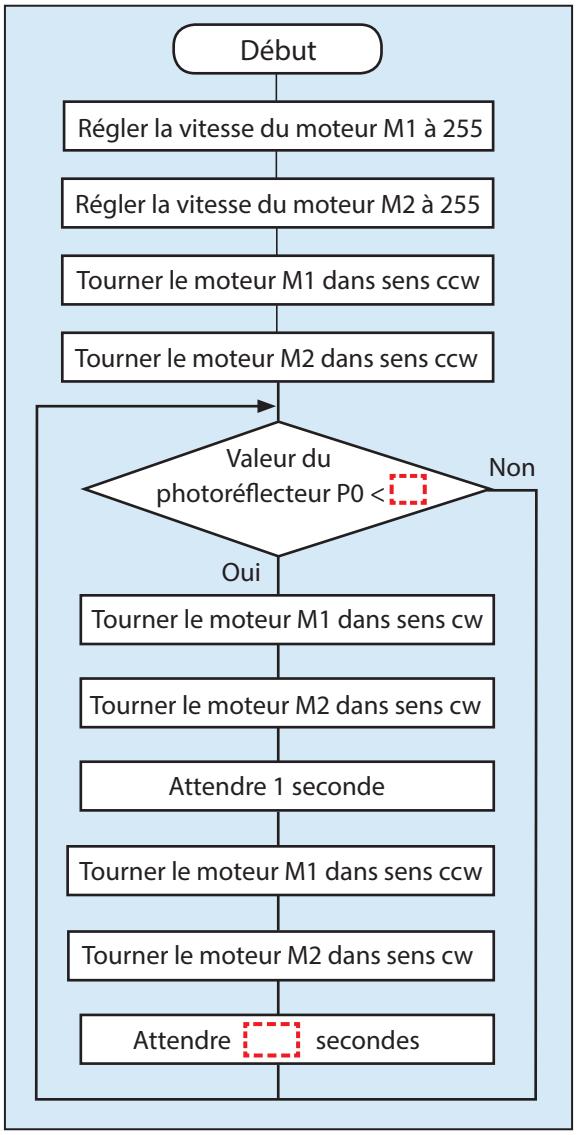
4.5 Programmer une voiture antichute

Utilise l'organigramme de la partie **4.4** pour élaborer ton programme.

```
1  from pyatcrobo2.parts import DCMotor, IRPhotoReflector
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  irp = IRPhotoReflector('P0')
7
8  dcm1.power(255)
9  dcm2.power(255)
10 dcm1.ccw()
11 dcm2.ccw()
12 while True:
13     if (irp.get_value() < [    ]):
14         dcm1.cw()
15         dcm2.cw()
16         time.sleep(1)
17         dcm1.ccw()
18         dcm2.cw()
19         time.sleep( [    ] )
20         dcm1.ccw()
21         dcm2.ccw()
```

4.5.1 Une voiture antichute alternative

Tu peux aussi faire fonctionner une voiture antichute avec ce programme-ci :

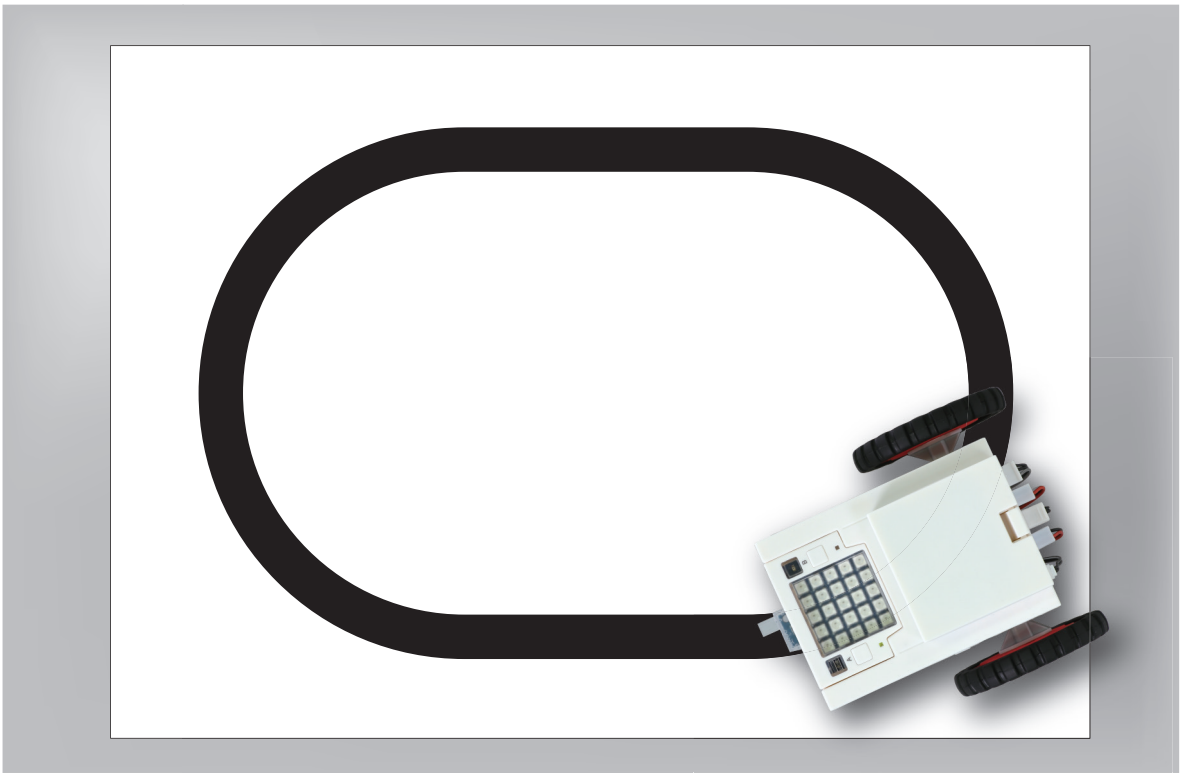



```
1  from pyatcrobo2.parts import DCMotor, IRPhotoReflector
2  import time
3
4  dcm1 = DCMotor('M1')
5  dcm2 = DCMotor('M2')
6  irp = IRPhotoReflector('P0')
7
8  dcm1.power(255)
9  dcm2.power(255)
10 while True:
11     dcm1.ccw()
12     dcm2.ccw()
13     if (irp.get_value() < ⬚ ):
14         dcm1.cw()
15         dcm2.cw()
16         time.sleep(1)
17         dcm1.ccw()
18         dcm2.cw()
19         time.sleep( ⬚ )
```

Comme tu peux le voir, il n'y a pas qu'une seule façon d'écrire un programme qui marche ! Tu peux faire n'importe quel programme, tant que tu parviens aux résultats escomptés. Réfléchis toi-même au problème pour mettre au point un programme qui fonctionne.

5. Faire une voiture de circuit

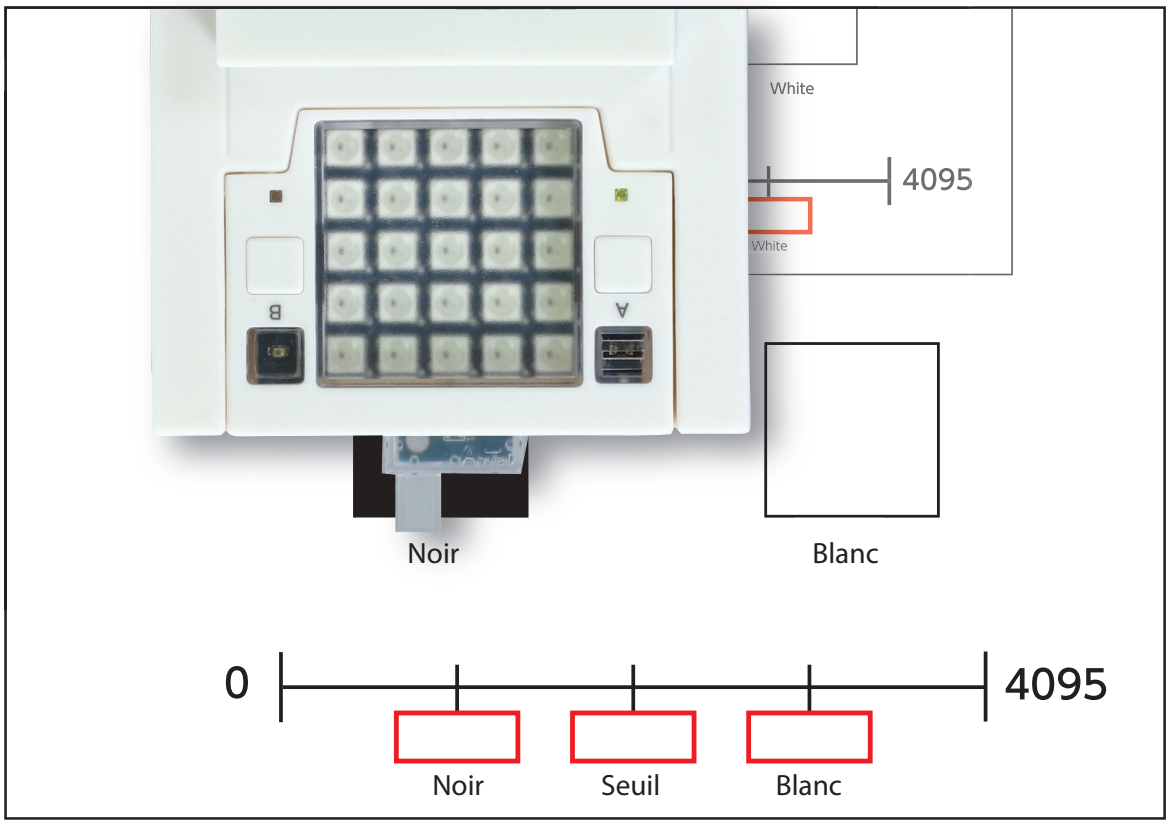
Les rayons infrarouges des photoréflecteurs IR détectent la lumière réfléchie avec une intensité différente selon les couleurs. Un photoréflecteur IR peut donc détecter une ligne noire sur fond blanc. Une voiture de circuit utilise son capteur pour détecter une ligne noire et rouler le long de celle-ci.



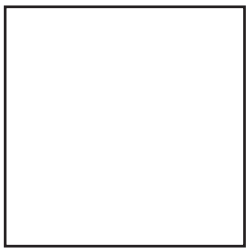
Tu peux utiliser ta voiture de la partie **4. Faire une voiture antichute** pour cette partie.

5.1 Trouver un seuil

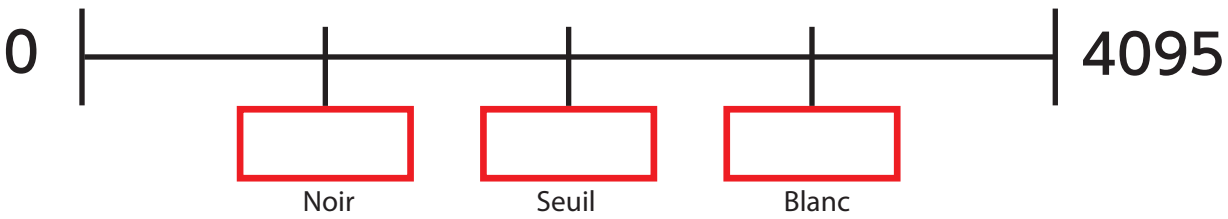
Avec le même programme qu'en **3.3**, tu peux comparer les valeurs du photoréflecteur IR quand il se trouve sur une ligne noire et quand il ne s'y trouve pas. Utilise ces valeurs pour déterminer un seuil.



Noir

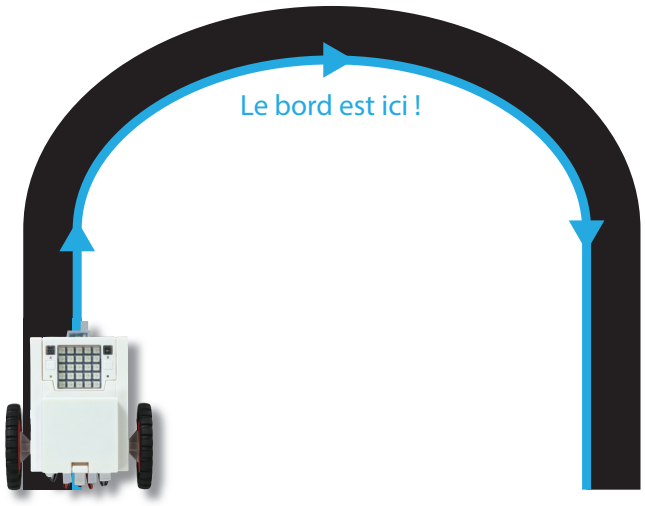


Blanc

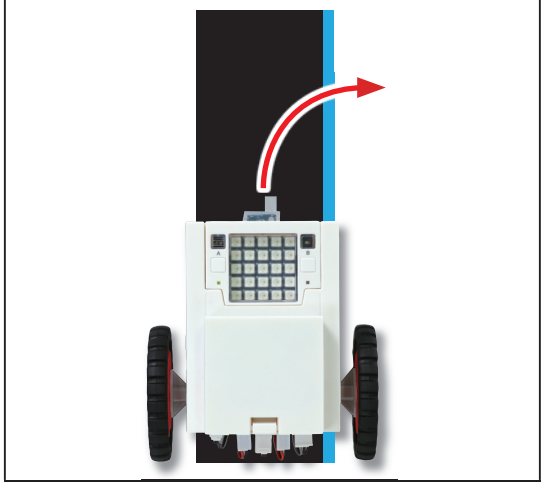
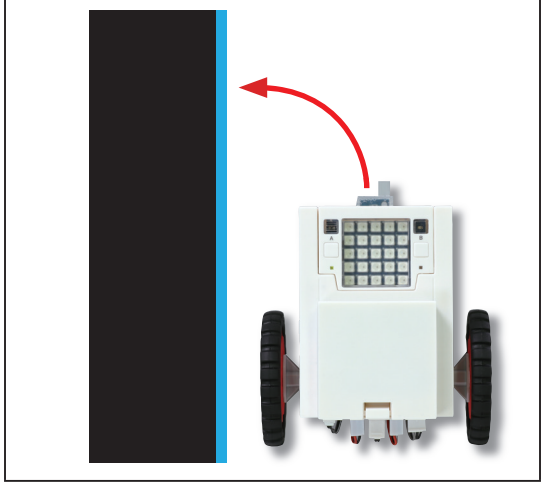


5.2 Planifier les actions de ta voiture

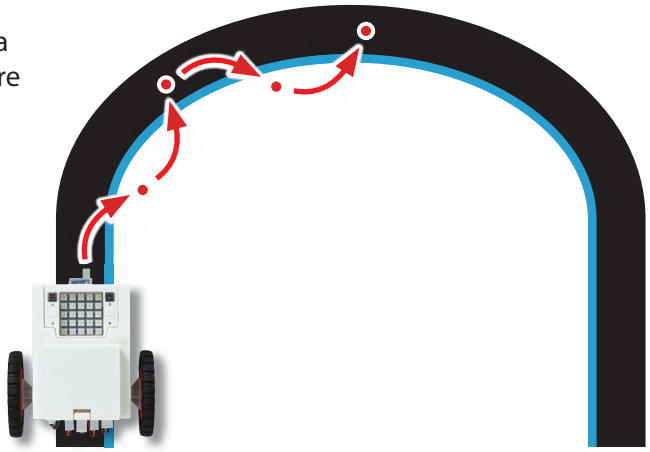
Pour que la voiture suive la ligne noire, l'un des moyens est de lui faire suivre le bord de la ligne (qui est la frontière entre la ligne et le papier blanc) au lieu de la ligne elle-même.



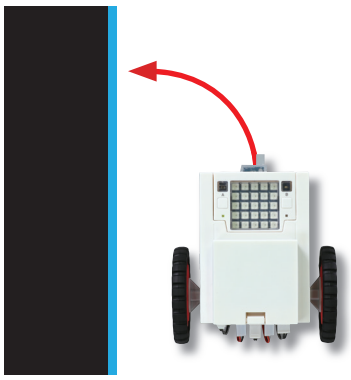
Pour que ta voiture roule le long du bord, les moteurs CC devront faire plusieurs choses selon que ta voiture se trouve sur la ligne noire ou sur l'espace blanc.

Sur la ligne (noire)	En dehors de la ligne (blanc)
	
Tourner à droite pour se rapprocher du bord.	**Tourner à gauche** pour se rapprocher du bord.

En répétant en permanence ces deux actions, ta voiture roulera le long du bord en évoluant entre la ligne noire et l'espace blanc.

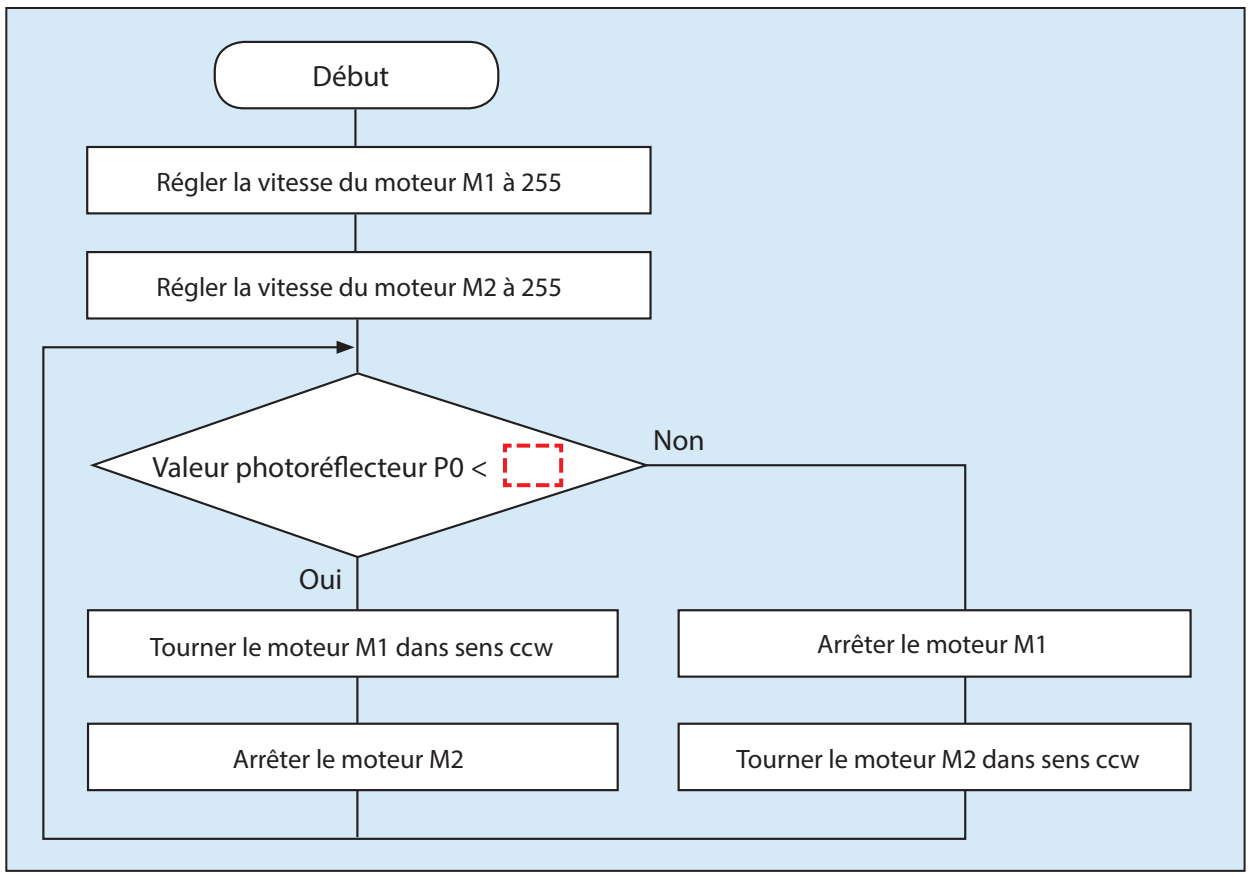


Nous pouvons résumer ces mouvements comme suit :

	Sur la ligne (noire)	En dehors de la ligne (blanc)
Image		
Condition	Photoréflecteur IR P0 < Seuil	La condition (ci-contre) n'est pas vraie
Actions	Tourner à droite	Tourner à gauche
Programme	Tourner moteur M1 dans le sens ccw Arrêter le moteur M2	Arrêter le moteur M1 Tourner moteur M2 dans le sens ccw

5.3 Créer un organigramme

Dessine un organigramme pour la séquence d'actions que tu viens de planifier. N'oublie pas ! Cette fois, tu vas devoir écrire deux commandes distinctes pour répondre aux deux résultats possibles de la condition.



5.4 Programmer une voiture de circuit

Écris un programme en te basant sur ton organigramme. Une instruction **if-else** te permettra de séparer les commandes pour les cas où la condition est remplie ou non !

```
from pyatcrobo2.parts import DCMotor, IRPhotoReflector
import time

dcm1 = DCMotor('M1')
dcm2 = DCMotor('M2')
irp = IRPhotoReflector('P0')

dcm1.power(255)
dcm2.power(255)
while True:
    if (irp.get_value() < 2048 ):
        dcm1.ccw()
        dcm2.brake()
    else:
        dcm1.brake()
        dcm2.ccw()
```

5.4.1 Une voiture de circuit alternative

Tu peux aussi faire fonctionner une voiture de circuit en utilisant deux instructions **if** séparées au lieu d'une instruction **if-else**. Tu obtiendras les mêmes résultats !

```
from pyatcrobo2.parts import DCMotor, IRPhotoReflector
import time

dcm1 = DCMotor('M1')
dcm2 = DCMotor('M2')
irp = IRPhotoReflector('P0')

dcm1.power(255)
dcm2.power(255)
while True:
    if (irp.get_value() < 2048 ):
        dcm1.ccw()
        dcm2.brake()
    if (irp.get_value() >= 2048):
        dcm1.brake()
        dcm2.ccw()
```

Annexes

Annexe A. Méthodes de la classe DCMotor

Méthode	Ce qu'elle fait...
__init__(pin)	Il s'agit d'un constructeur. Utilise-le pour créer un objet à partir de cette classe. Le paramètre **pin** peut être réglé soit sur M1, soit sur M2, en spécifiant si le moteur est branché sur les ports M1 ou M2 de l'extension.
cw()	Fait tourner un moteur CC dans le sens cw (dans le sens des aiguilles d'une montre).
ccw()	Fait tourner un moteur CC dans le sens ccw (dans le sens contraire des aiguilles d'une montre).
stop()	Arrête d'alimenter le moteur CC, ce qui permet de l'arrêter en roue libre.
brake()	Arrête la rotation du moteur CC.
power(p)	Règle la puissance dans le paramètre **p** sur un nombre entre 0 et 255 pour ajuster la vitesse du moteur CC.

Annexe B. Méthodes de la classe IRPhotoreflector

Méthode	Ce qu'elle fait...
__init__(pin)	Il s'agit d'un constructeur. Utilise-le pour créer un objet à partir de cette classe. Le paramètre **pin** peut être réglé sur P0, P1 ou P2, en spécifiant si le capteur est branché sur les ports P0, P1 ou P2 de l'extension.
get_value()	Trouve la valeur actuelle du photoréflecteur IR.

Notes

Ecole Algora

Apprendre à coder en s'amusant

PLUS DE 36 ROBOTS DIFFÉRENTS
UN CURSUS COMPLET
POUR APPRENDRE À PROGRAMMER

2 Cursus
6-9 ans
9-14 ans

Inscris-toi sur www.algora.school

Apprendre à programmer des robots pour comprendre le monde d'aujourd'hui et de demain.

Les machines programmées, de plus en plus intelligentes, font partie intégrante de notre vie de tous les jours. Elles nous accompagnent, nous entourent et ont envahi tous les domaines de notre vie quotidienne. Maîtriser le monde, ce n'est pas les utiliser, mais avant tout comprendre comment elles fonctionnent.

Comment fonctionnent-elles ?
Selon quelle logique ? Selon quels algorithmes ?
Comment sont conçus les programmes qui leur dictent leurs actions et réactions ?

C'est ce que vous apprendrez tout au long de ces livrets d'apprentissage. Et pas seulement "en théorie" : vous allez vous-même concevoir et programmer vos propres robots : des actions simples aux plus complexes, vous apprendrez à programmer des robots amusants et originaux que vous aurez conçus vous-même. Une seule limite : votre créativité !

École Robots permet à tous de s'initier à la programmation en s'amusant, un enjeu majeur, aujourd'hui et demain.

Ecole Robots
Apprendre à coder avec les robots éducatifs
CURSUS EDUCATION NATIONALE

Pour en savoir plus : www.ecolerobots.com

Réf : LIV-EN-C-PY1 (v.2)