SOMMAIRE

L'environnement de développement
- La carte Raspberry PICO p.2
- L'environnement de simulation WOKWI p.3
https://wokwi.com/projects/new/micropython-pi-pico
- L'environnement de programmation python Thonny p.4
- Connecter Thonny et la carte PICO p.5
Référence : https://www.micropython.fr/port_pi_pico/

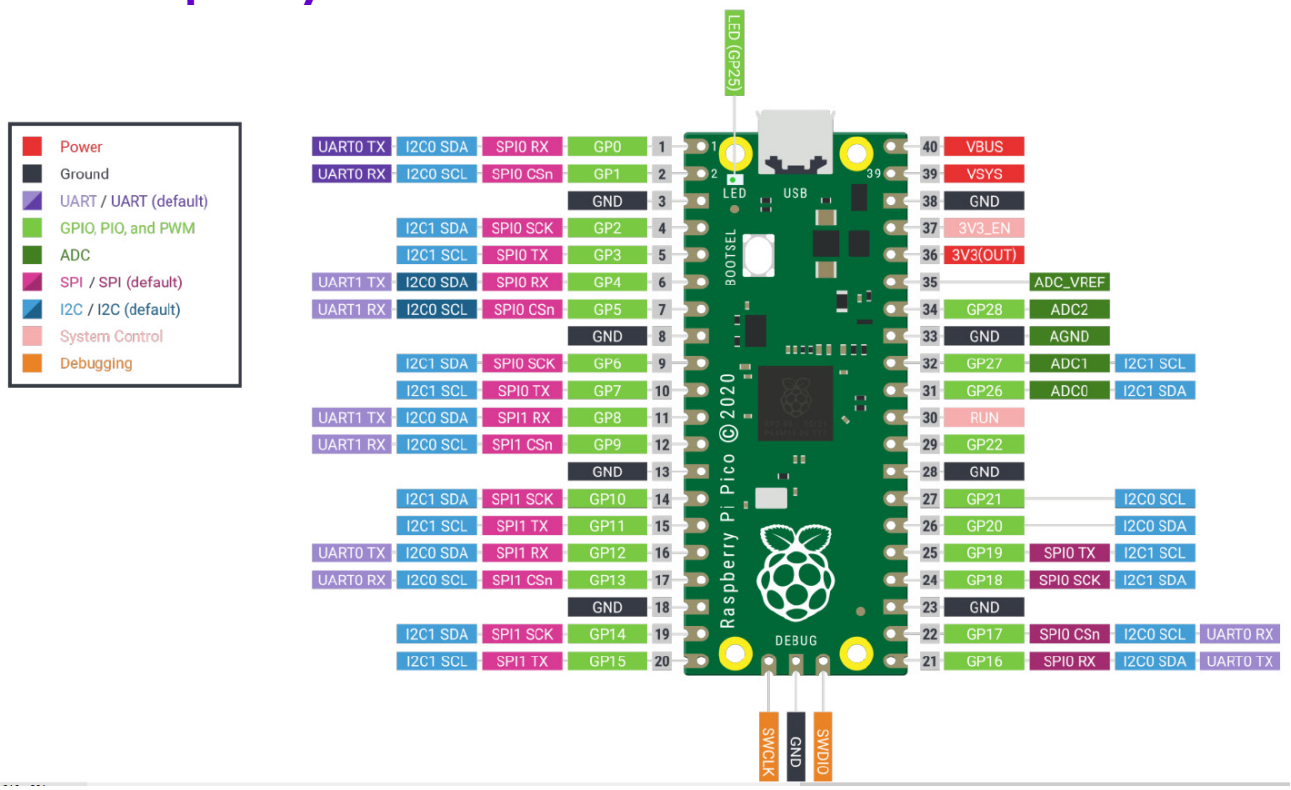
La DEL et quelques programmes de base
- DEL clignotante avec délais p.6
- DEL clignotante sans délais p.7
- Allumage progressif d'une DEL p.7
- DEL avec INTERRUPTION p.8

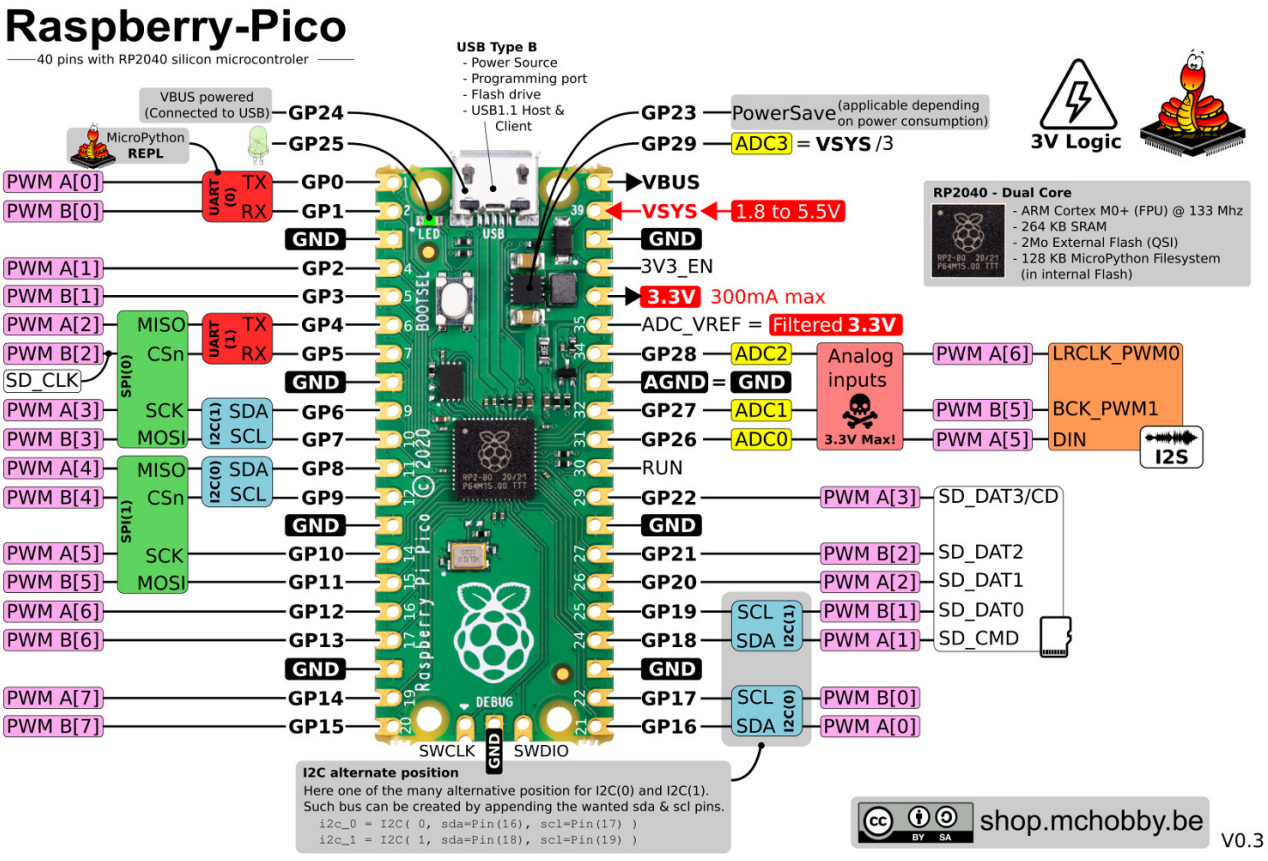
ACQUÉRIR
- Potentiomètre (permet de remplacer tout capteur analogique) p.9
- Bouton poussoir (permet de remplacer tout capteur TOR) p.10
- Interrupteur à glissière (permet de remplacer capteur TOR) p.11
- Capteur de distance à ultra-sons HC-SR04 p.12
- Capteur de mouvement PIR p.13
- Capteur INFRA-ROUGE et télécommande p.14

COMMUNIQUER et AGIR
- Buzzer p.15
- DEL RVB p.16
- Moteur p.17
- Relais miniature p.18
- Servo-moteur p.19
- Servo-moteur à rotation continue p.21

Les LIAISONS
- liaison Bluetooth avec le module HC-05 p.22

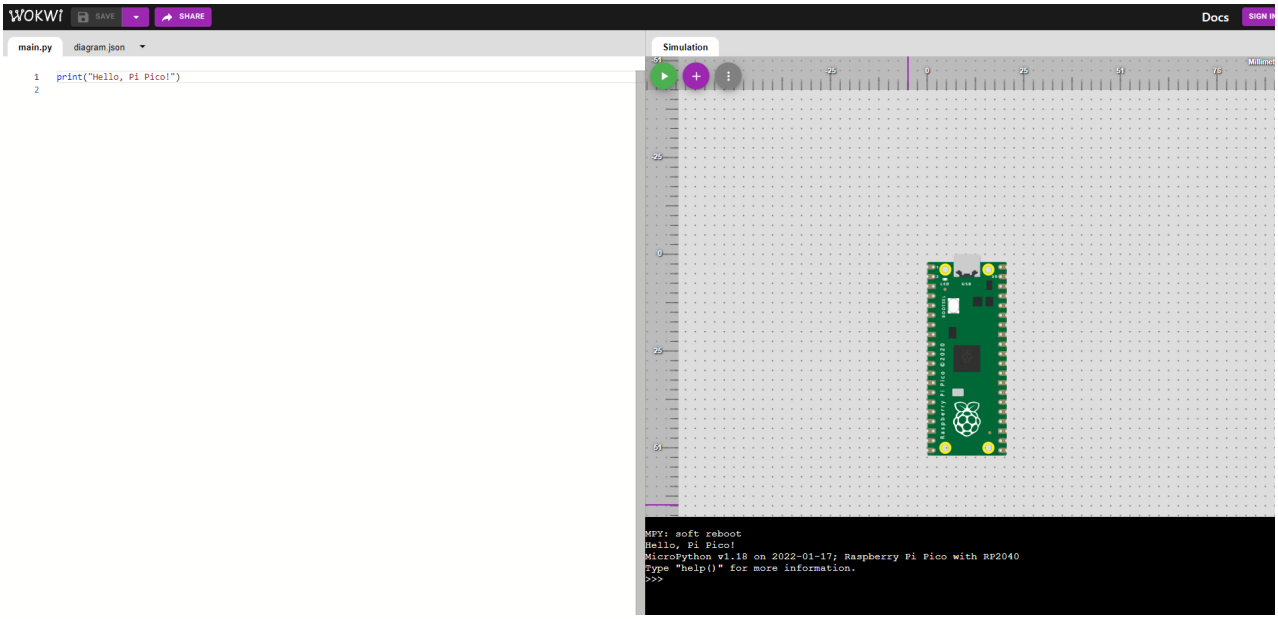
La carte Raspberry PICO





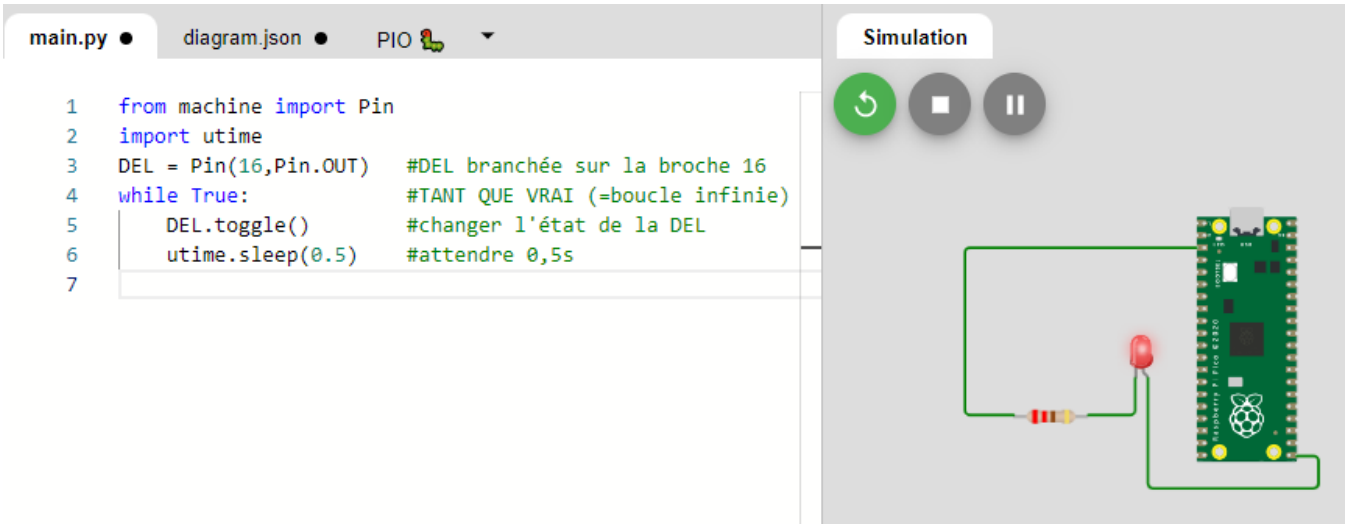
L'environnement de simulation WOKWI.com

https://wokwi.com/projects/new/micropython-pi-pico



REMARQUE: certaines des caractéristiques des composants peuvent être modifiées dans l'**onglet diagram.json**. Par exemple la résistance a par défaut la valeur de 1000ohms. Pour changer la valeur de la résistance à 220ohms il faut aller dans l'onglet diagram.json et remplacer 1000 par 220.
ATTENTION: pour un meilleur fonctionnement de la simulation, il est conseillé de toujours ajouter une ligne **utime.sleep() à la fin du programme** (sauf s'il y en a déjà au moins une dans le programme).

Exemple: programme pour faire clignoter une LED



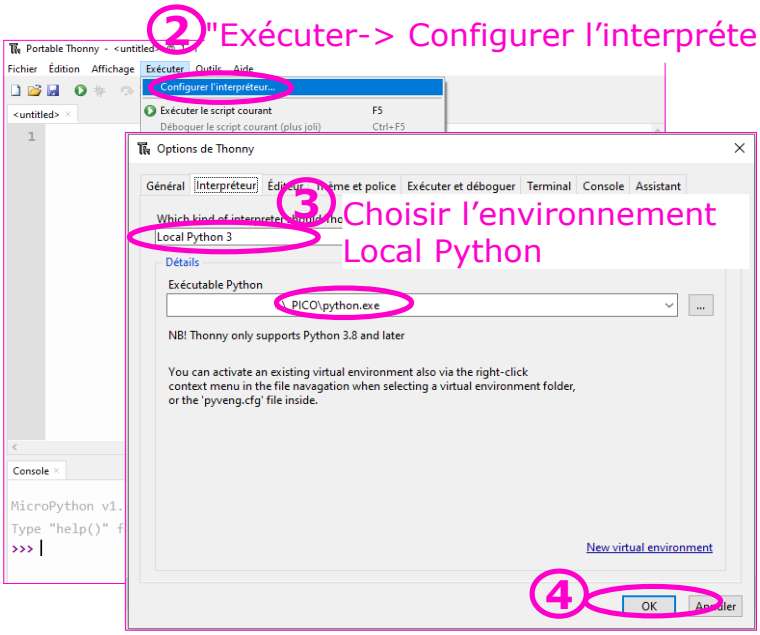
COMPATIBILITÉ avec le **SHIELD GROVE**: Le SHIELD GROVE utilise seulement quelques broches de la carte PICO:

- Entrées/Sortie numériques: D16=broche GP16, D18=broche GP18, D20=broche GP20
- Entrées Analogique : A0 = broche GP26, A1 = broche GP27, A2 = broche GP28
- Liaison I2C: I2C1 : SDA=GP6 et SCL=GP7 ; I2C0 : SDA=GP8 et SCL=GP9
- Liaison série (UART): UART0: TX=GP0 et RX=GP1 ; UART1: TX=GP4 et RX=GP5
- Sorties PWM: la plupart des sorties de la carte PICO peuvent être utilisées en PWM. **Mais sur la simulation WOKWI certaines sorties ne fonctionnent pas** en PWM (par exemple, uniquement sur la simulation, les broches 16, 18 et 20 ne fonctionnent pas en PWM)

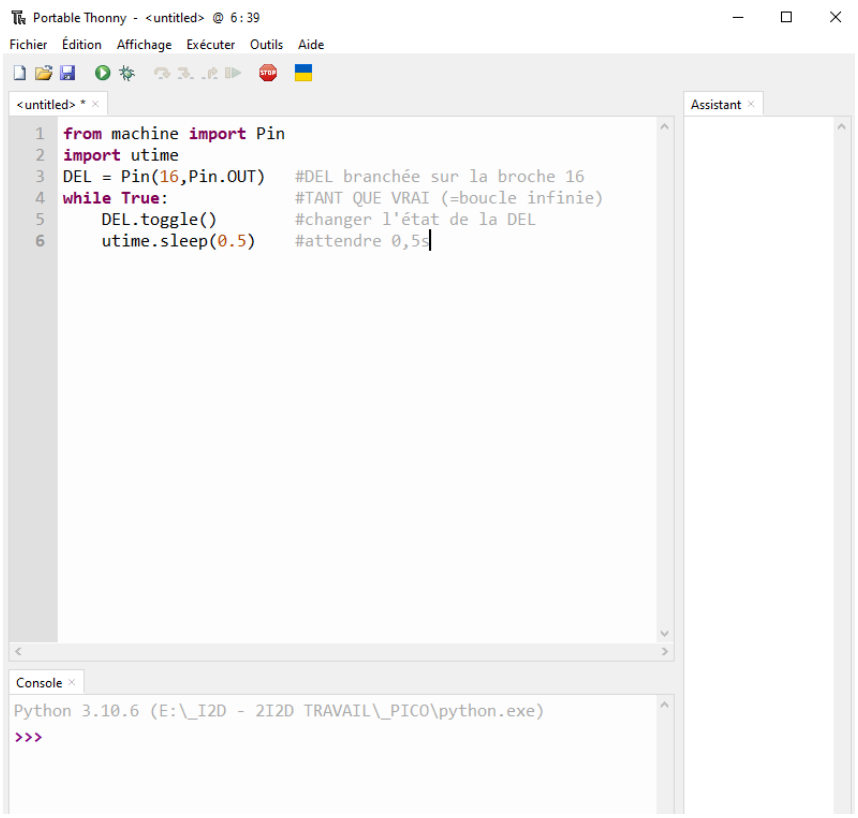
L'environnement de développement (IDE) Python THONNY

Utilisation de l'IDE THONNY

① lancer **Thonny.exe**



Exemple d'un programme PYTHON dans l'environnement Thonny



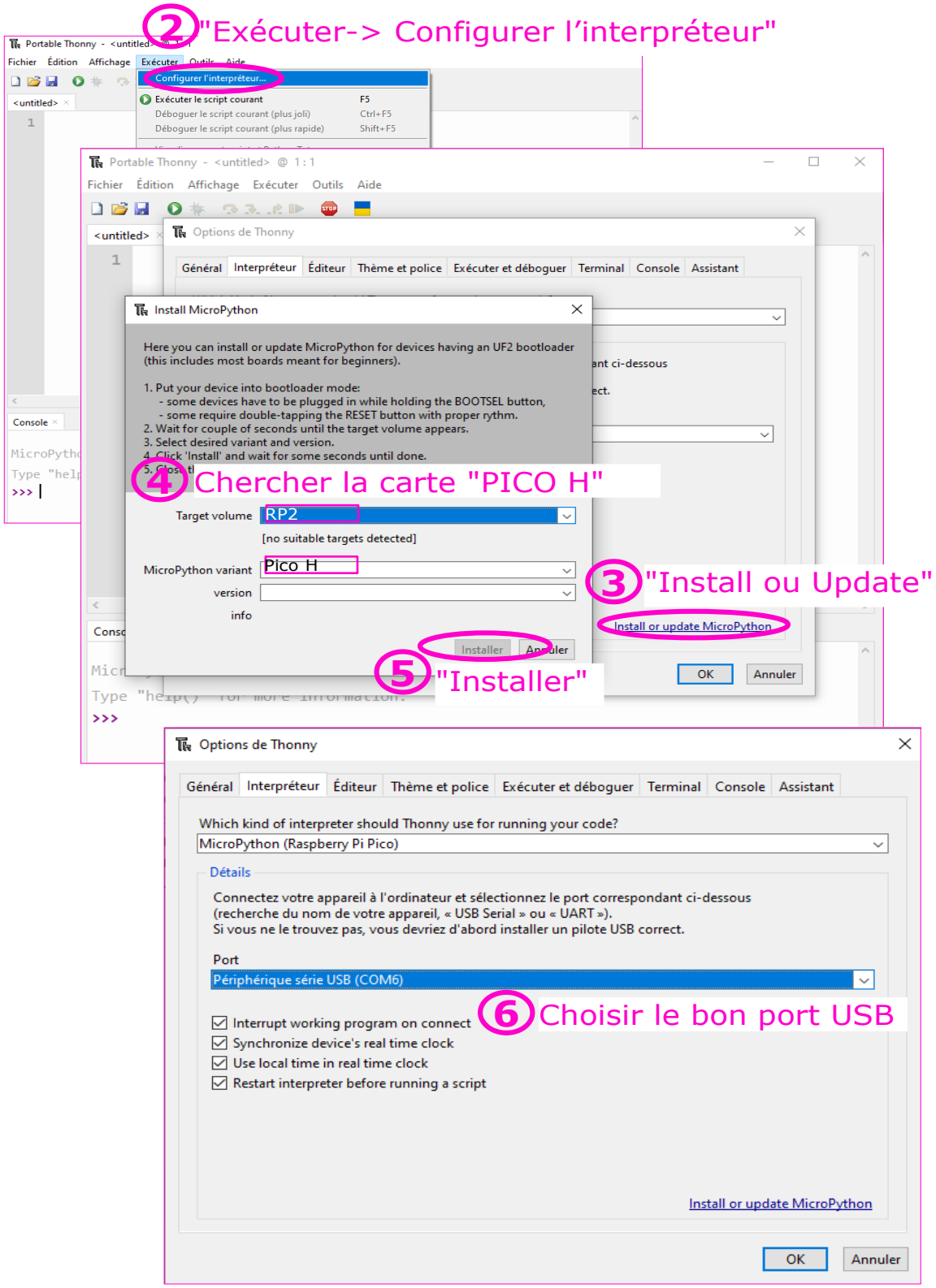
REMARQUES:

- L'instruction **while True:** permet de créer une boucle qui s'exécute indéfiniment.
- L'**indentation** (c'est à dire le retrait par rapport à la marge) doit être respectée en python, sinon le programme ne fonctionne pas.

Connecter la carte PICO avec THONNY

Utilisation de l'IDE THONNY avec la carte Raspberry PICO:

① brancher la **carte** sur un port USB et lancer **Thonny.exe**

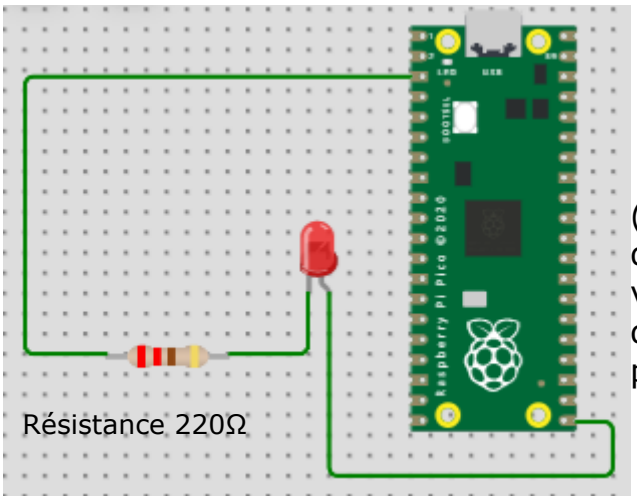


- Il ne reste plus qu'à cliquer sur **Exécuter** pour que la carte PICO exécute le programme

La Diode Electro-Luminescente DEL (ou LED) :

La Diode Electro-Luminescente (DEL), ou Light Emitting Diode (LED) doit **toujours être utilisée en série avec une résistance** afin de limiter l'intensité du courant qui la traverse. **L'anode (+)** de la DEL (patte la plus longue) se branche sur une **sortie numérique** de la carte Arduino et **la cathode (-)** (patte la plus courte, côté plat) sur un **GND** (ground).

Exemple de montage



(La résistance disponible sur WOKWI a par défaut la valeur de 1000ohms. Pour changer la valeur de la résistance à 220ohms il faut aller dans l'onglet **diagram.json** et remplacer 1000 par 220.)

Programme pour faire clignoter la DEL : avec des «délais» (utime.sleep)
Faire clignoter la DEL 10 fois avec une boucle FOR:

```
1   from machine import Pin
2   import utime
3   DEL = Pin(16,Pin.OUT)    #DEL branchée sur la broche 16
4   for i in range(10): #boucle for : sera exécutée 10 fois
5       DEL.value(1)         #allumer la DEL
6       utime.sleep(1)       #attendre 1s
7       DEL.value(0)         #éteindre la DEL
8       utime.sleep(1)       #attendre 1s
```

Faire clignoter la DEL indéfiniment avec une boucle WHILE:

```
1   from machine import Pin
2   import utime
3   DEL = Pin(16,Pin.OUT)    #DEL branchée sur la broche 16
4   while True:              #TANT QUE VRAI (=boucle infinie)
5       DEL.value(1)         #allumer la DEL
6       utime.sleep(1)       #attendre 1s
7       DEL.value(0)         #éteindre la DEL
8       utime.sleep(1)       #attendre 1s
```

La DEL (suite) : autres programmes

Programme pour faire clignoter la DEL sans «délais»

c) en comptant les «boucles»: d) en utilisant la mesure du temps:

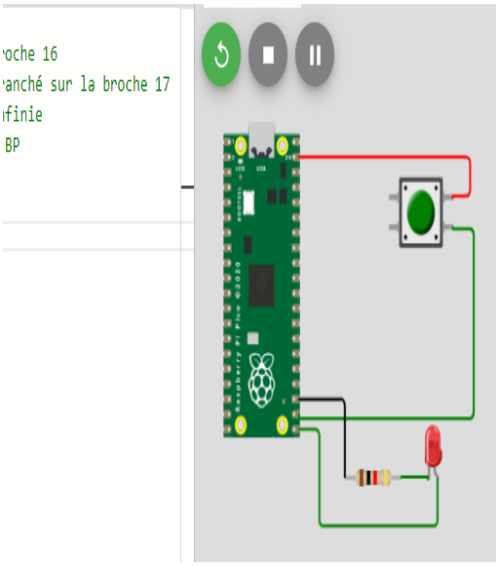
Programme pour commander l'allumage progressif de la DEL:

!! la DEL doit être branchée sur une sortie **~ (PWM)**.

La VALEUR utilisée dans analogWrite doit être comprise entre 0 et 65535 (16 bits): par exemple pour la DEL si on met 0 elle est éteinte, si on met 32768 elle est allumée à 50 % si on met 65535 elle est allumée à 100 % .

La DEL (suite) : autres programmes

Programmer une interruption avec IRQ :



Exemple d'interruption : l'appui sur le bouton poussoir interrompt l'allumage de la DEL.

```
1   from machine import Pin
2   import utime
3   DEL = Pin(16, Pin.OUT) #DEL branchée sur la broche 16
4   Bouton = Pin(17, Pin.IN, Pin.PULL_DOWN) #BP branché sur la broche 17, forcée à 0
5
6   interruption = 0
7   def PRIORITE(Bouton): #fonction qui sera appelée en cas d'interruption sur la broche 17
8       global interruption #déclarer que cette variable est globale
9       interruption = 1
10
11  Bouton.irq(trigger=Pin.IRQ_RISING, handler=PRIORITE) #déclaration d'une interruption au cas où la broche 17 passe à 0
12
13  while True: #toujours vrai = boucle infinie
14      if interruption==1:
15          DEL.value(0)
16          utime.sleep(2)
17          interruption = 0
18      DEL.value(1)
19      utime.sleep(0.25)
```

Il existe 5 modes d'interruption possibles :

- trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING : l'interruption est déclenchée quand la broche concernée change d'état, c'est à dire passe de LOW à HIGH ou bien de HIGH à LOW. Il s'agit d'un événement.

- trigger=Pin.IRQ_LOW_LEVEL : l'interruption est déclenchée quand la broche concernée est LOW. Comme il s'agit d'un état et non d'un événement, l'interruption sera déclenchée tant que la broche est LOW. Par conséquent, dès qu'elle aura terminé son exécution, elle la recommencera.

- trigger=Pin.IRQ_HIGH_LEVEL : l'interruption est déclenchée quand la broche concernée est HIGH. Comme il s'agit d'un état et non d'un événement, l'interruption sera déclenchée tant que la broche est HIGH. Par conséquent, dès qu'elle aura terminé son exécution, elle la recommencera.

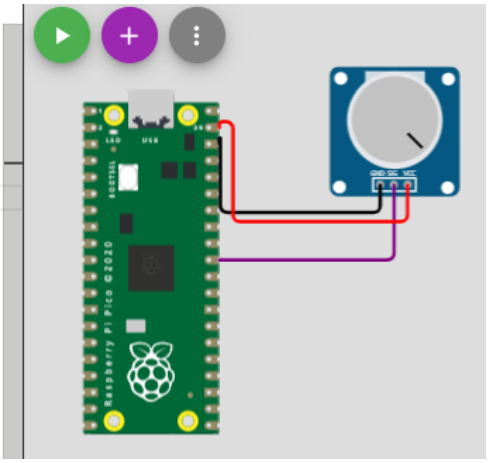
- trigger=Pin.IRQ_RISING : l'interruption est déclenchée quand la broche concernée passe de LOW à HIGH. Il s'agit d'un événement.

- trigger=Pin.IRQ_FALLING : l'interruption est déclenchée quand la broche concernée passe de HIGH à LOW. Il s'agit d'un événement.

- handler=fonction() : désigne la fonction qui sera appelée quand arrive l'interruption.

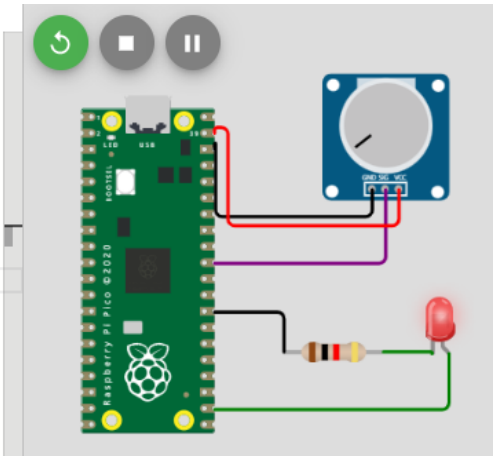
Le POTENTIOMÈTRE

```python
from machine import ADC
import utime
potentiometre = ADC(0) #le potentiomètre est branché sur la broche 26 (A0)
while True:
    valeur = potentiometre.read_u16() #lire la valeur du potentiomètre
    print(valeur)   #afficher la valeur (16 bits donc compris entre 0 et 65535)
    utime.sleep(1)
```



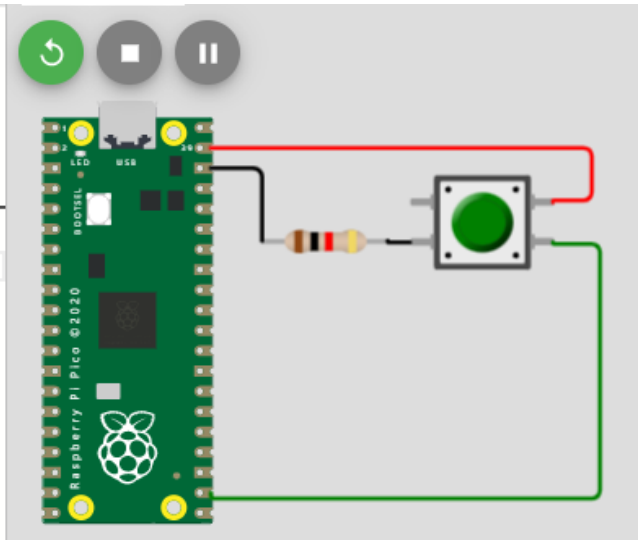
Faire varier l'intensité d'éclairage d'une DEL (par un signal PWM) avec un potentiomètre

```python
from machine import Pin, ADC, PWM
import utime
potentiometre = ADC(0) #le potentiomètre est branché sur la broche 26 (A0)
DEL = PWM(Pin(17)) #la DEL est branchée sur la broche 17
DEL.freq(500)  #fréquence du signal PWM 500Hz
while True:
    valeur = potentiometre.read_u16() #lire la valeur du potentiomètre
    print(valeur)   #afficher la valeur (16 bits donc compris entre 0 et 65535)
    DEL.duty_u16(valeur)  #allumer la DEL plus ou moins fort selon la valeur
    utime.sleep(1)  #attendre 1s
```



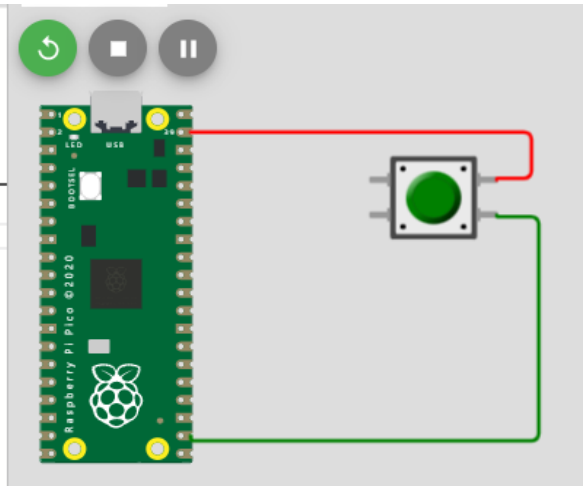
Le BOUTON POUSSOIR (Pushbutton) :

```
1  from machine import Pin
2  import utime
3  Bouton = Pin(17, Pin.IN) #BP branché sur la broche 17
4  compteur=0
5  while True:          #toujours vrai = boucle infinie
6      if Bouton.value()==1: #TEST: si appui sur BP
7          compteur+=1       #incrémenter compteur
8          print(compteur)   #afficher compteur
9          utime.sleep(0.25) #attendre 250ms
```



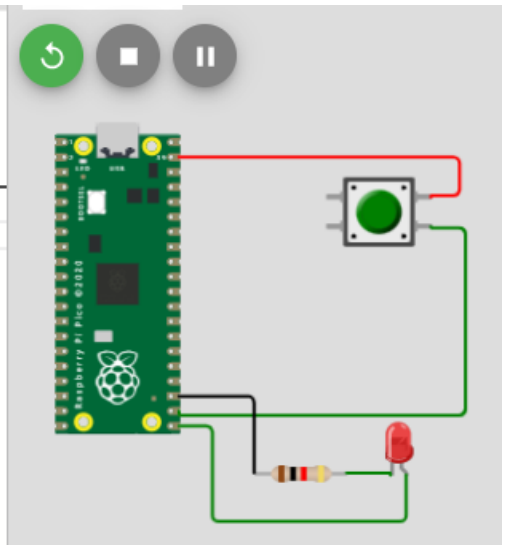
La **connexion au GND** du BP en passant par la résistance force la broche 17 à 0 quand on n'appuie pas sur le BP (sinon le programme ne fonctionne pas correctement).
On peut se passer de la connexion au GND : l'instruction PULL_DOWN permet de forcer la broche à 0. **Le même programme sans cette connexion :**

```
1  from machine import Pin
2  import utime
3  Bouton = Pin(17, Pin.IN, Pin.PULL_DOWN) #BP branché sur la broche 17
4  compteur=0
5  while True:          #toujours vrai = boucle infinie
6      if Bouton.value()==1: #TEST: si appui sur BP
7          compteur+=1       #incrémenter compteur
8          print(compteur)   #afficher compteur
9          utime.sleep(0.25) #attendre 250ms
```



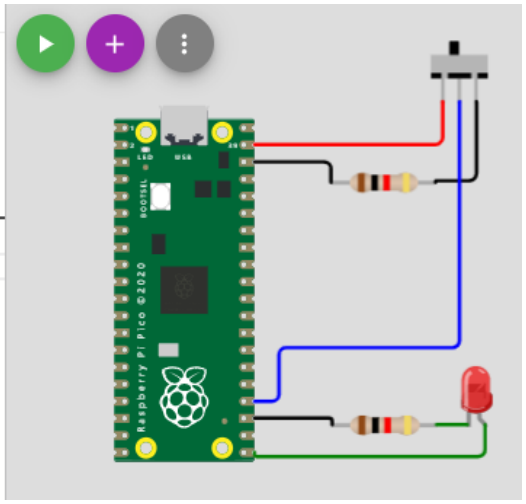
éteindre ou allumer une DEL avec un BP

```
1  from machine import Pin
2  DEL = Pin(16, Pin.OUT) #DEL branchée sur la broche 16
3  Bouton = Pin(17, Pin.IN, Pin.PULL_DOWN) #BP branché sur la broche 17
4  while True:          #toujours vrai = boucle infinie
5      if Bouton.value()==1: #TEST: si appui sur BP
6          DEL.value(1)
7      else:
8          DEL.value(0)
```

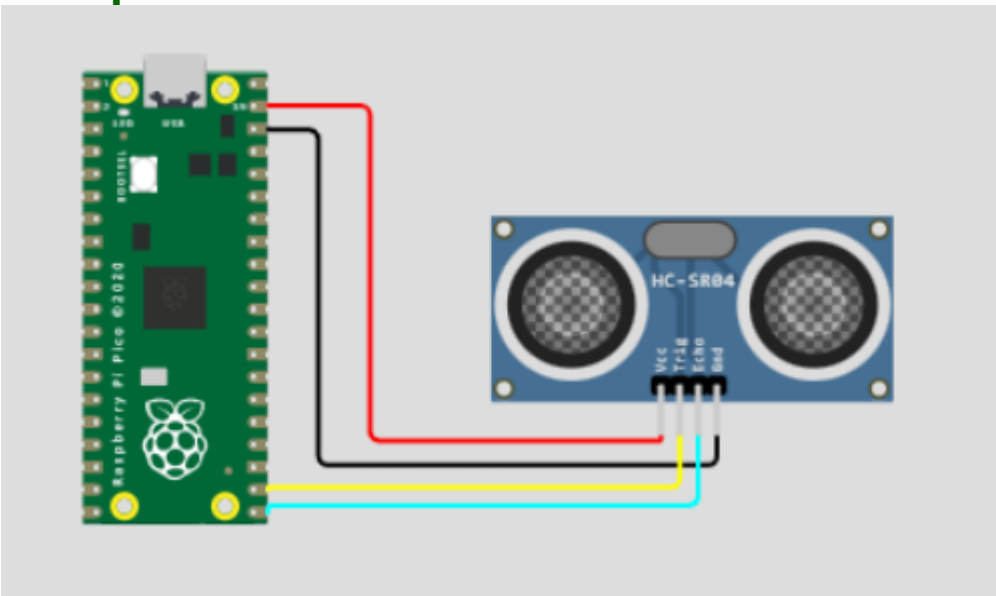


L'INTERRUPTEUR à glissière (slide switch)

```
1  from machine import Pin
2  DEL = Pin(16,Pin.OUT) #la DEL est branchée sur la broche 16
3  Interrupteur = Pin(18,Pin.IN) #l'interrupteur est branché sur la broche 18
4  while True:
5      valeur = Interrupteur.value() #LIRE la valeur de l'interrupteur
6      if valeur == 1:    #TEST : si l'interrupteur est fermé
7          DEL.value(1)   #allumer la DEL
8      else:
9          DEL.value(0)   #éteindre la DEL
```



Le capteur à Ultrasons HC-SR04



Exemple de programme : détecter un obstacle, calculer et afficher sa distance:

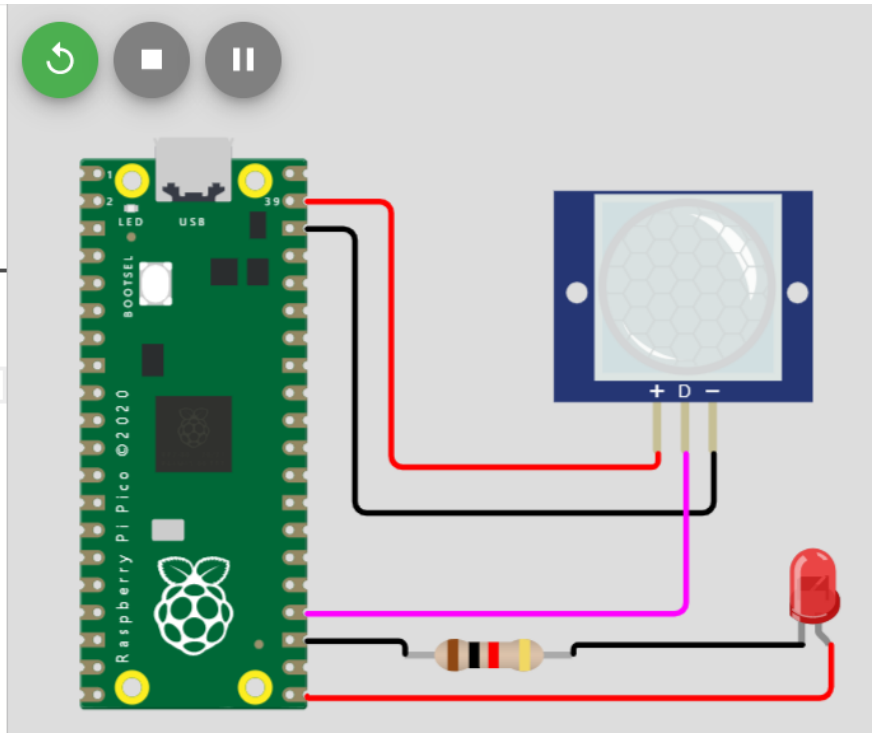
```
from machine import Pin
import utime

trig = Pin(17,Pin.OUT) #capteur à ultrasons branché entre 17 (trigger) et 16 (echo)
echo = Pin(16, Pin.IN, Pin.PULL_DOWN)

while True:                 #boucle infinie
    trig.value(1)           #émettre un ultrason
    utime.sleep_us(2)       #attendre 2 microsecondes
    trig.value(0)           #arrêter d'émettre
    duree = machine.time_pulse_us(echo,1) #mesurer la durée mise par le signal pour revenir (écho)
    distance_en_mm = duree * 0.34 / 2     #calculer la distance parcourue (/2 pour l'aller-retour)
    print(distance_en_mm)                 #afficher la valeur calculée
```

Le même programme avec une FONCTION pour récupérer la distance:

```
from machine import Pin
import utime

trig1 = Pin(17,Pin.OUT)
echo1 = Pin(16, Pin.IN, Pin.PULL_DOWN)

def detection_ultra_sons(trig,echo):    #fonction qui calcule la distance de l'obstacle
    trig.value(1)
    utime.sleep_us(2)
    trig.value(0)
    duree = machine.time_pulse_us(echo,1)
    distance_en_mm = duree * 0.34 / 2
    return(distance_en_mm)

while True:                                                 #boucle infinie
    distance_en_cm = detection_ultra_sons(trig1,echo1) / 10  #appel à la fonction pour calculer la distance
    print(distance_en_cm)                                   #affichage de la valeur calculée
```

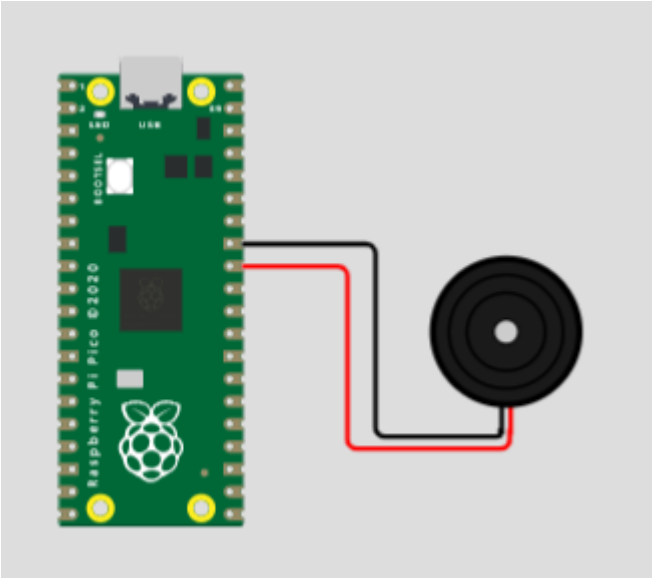
Le capteur de mouvement PIR

```
1   from machine import Pin
2   import utime
3   PIR = Pin(18,Pin.IN)
4   DEL = Pin(16,Pin.OUT)
5   while True:
6       if PIR.value() == 1:
7           DEL.value(1)
8           utime.sleep(1)
9       else:
10          DEL.value(0)
11          utime.sleep(1)
```



Le capteur INFRA-ROUGE + télécommande

Le BUZZER :



notation française	notation anglaise	Fréquence
do4	c4	262
do#4	cs4	277
ré4	d4	294
ré#4	ds4	311
mi4	e4	330
fa4	f4	349
fa#4	fs4	370
sol4	g4	392
sol#4	gs4	415
la4	a4	440
la#4	as4	466
si4	b4	494

Succession de notes :

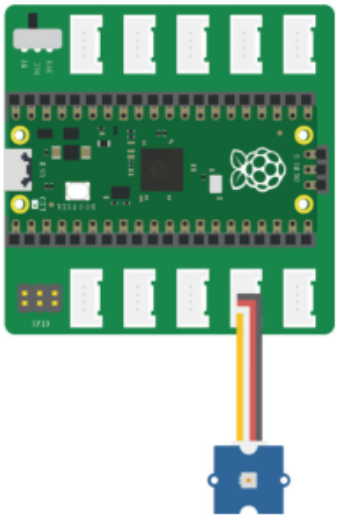
```
1   from machine import Pin, PWM
2   import utime
3   Buzzer = PWM(Pin(27)) #Buzzer branché sur la broche 27
4   while True:
5       Buzzer.freq(440)      #jouer la note LA
6       Buzzer.duty_u16(2000) #régler le volume
7       utime.sleep(1)
8       Buzzer.freq(1046)     #jouer la note DO
9       Buzzer.duty_u16(1000) #régler le volume
10      utime.sleep(1)
11      Buzzer.duty_u16(0)    #volume à 0: plus de son
12      utime.sleep(1)
```

Succession de bips plus ou moins aigus :

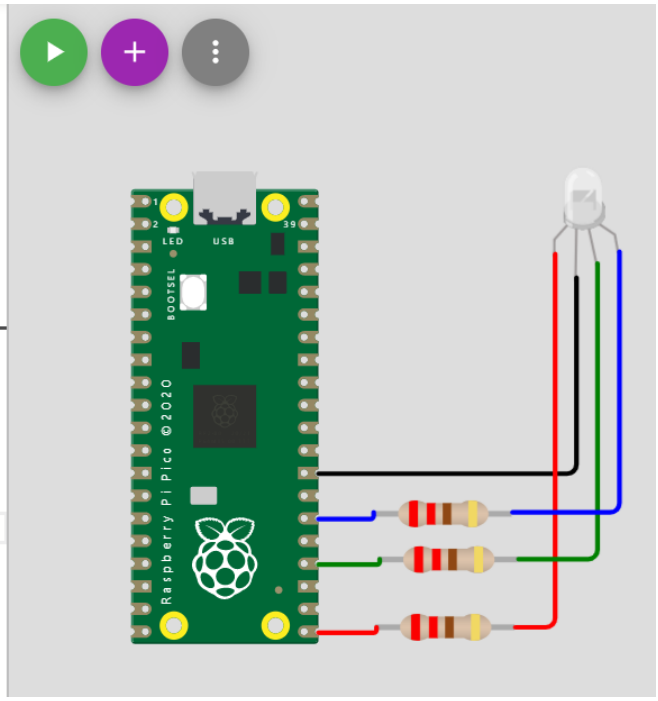
```
1   from machine import Pin, PWM
2   import utime
3   Buzzer = PWM(Pin(27))
4   obstacle = 80
5   while True:
6       if obstacle < 100:
7           Buzzer.freq(880)
8           Buzzer.duty_u16(2000)
9           utime.sleep(0.5)
10      elif obstacle < 200:
11          Buzzer.freq(440)
12          Buzzer.duty_u16(1000)
13          utime.sleep(1)
14      else:
15          Buzzer.duty_u16(0)
```

La DEL RVB

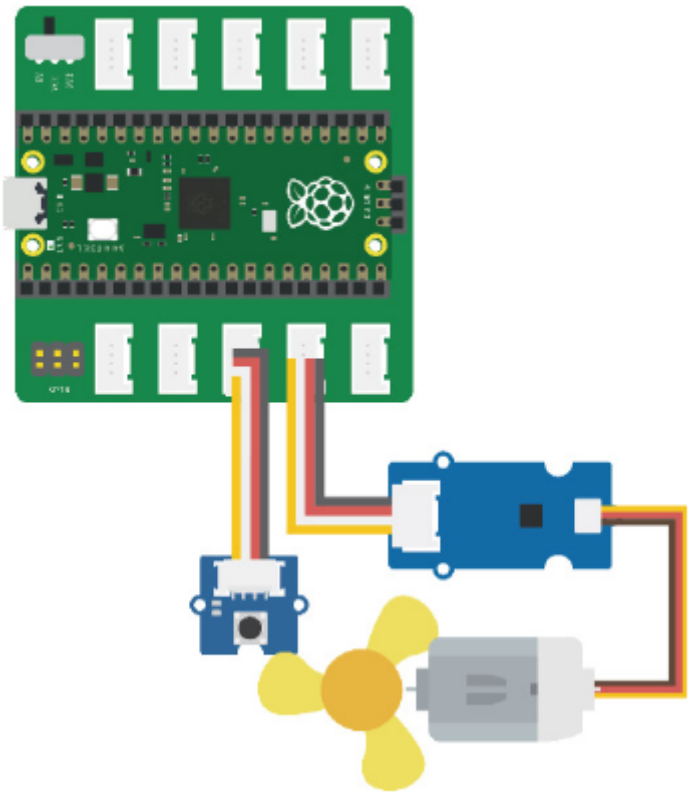
```python
from ws2812 import WS2812
import utime

BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
WHITE = (255, 255, 255)
COLORS = (BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE)

led = WS2812(18,1)#WS2812(pin_num,led_count)

while True:
    print("fills")
    for color in COLORS:
        led.pixels_fill(color)
        led.pixels_show()
        utime.sleep(0.2)
```



```python
from machine import Pin
import utime

red = Pin(16, Pin.OUT)
green = Pin(18, Pin.OUT)
blue = Pin(20, Pin.OUT)

while True:
    red.value(1)
    green.value(1)
    blue.value(1)
    utime.sleep(1)

    red.value(0)
    green.value(0)
    blue.value(0)
    utime.sleep(1)
```



Le MOTEUR (ventilateur)



Exemples de programmes :

le moteur tourne quand
on appuie sur le bouton

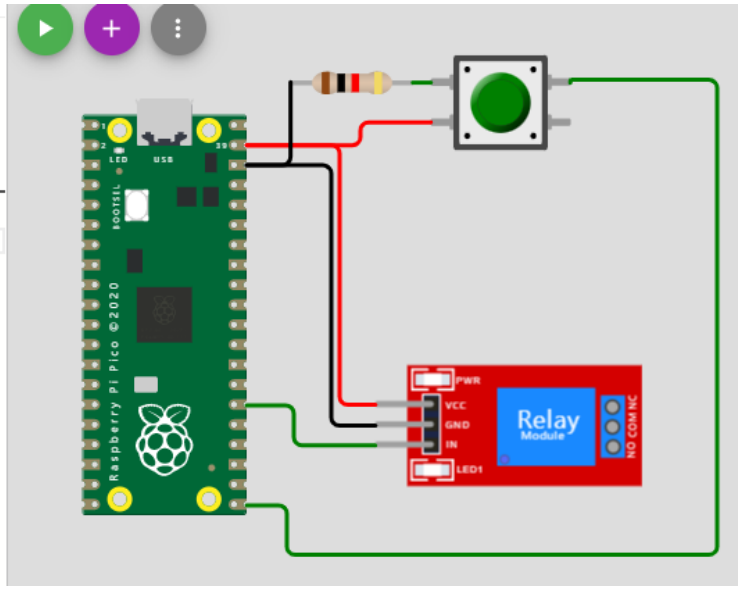
```
from machine import Pin
bouton = Pin(16,Pin.IN)
ventilateur = Pin(18,Pin.OUT)
while True:
    valeur = bouton.value()
    if valeur == 1:
        ventilateur.value(1)
    else:
        ventilateur.value(0)
```

Le moteur change d'état
à chaque appui sur le bouton

```
from machine import Pin
import utime
bouton = Pin(16,Pin.IN)
ventilateur = Pin(18,Pin.OUT)
while True:
    valeur = bouton.value()
    if valeur == 1:
        ventilateur.toggle()
        utime.sleep_ms(100)
```

Le RELAIS miniature:

```
1  from machine import Pin
2  import utime
3  bouton = Pin(16,Pin.IN) #bouton branché sur broche 16
4  relais = Pin(20,Pin.OUT)#relais branché sur broche 20
5  while True:
6      valeur = bouton.value()#lire la valeur du bouton
7      if valeur == 1:      #condition: si valeur égale 1
8          relais.toggle()  #changer l'état du relais
9          utime.sleep(1)   #attendre 1s
```



Le SERVO-MOTEUR :

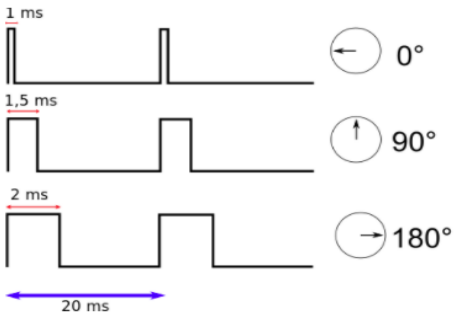
Un servomoteur est un moteur électrique utilisé pour obtenir un déplacement angulaire (donc une **position**) spécifique.

Un servomoteur se connecte avec 3 fils :

- **Noir** : masse (GND)
- **Rouge** : Vcc
- **Jaune** : signal de commande

Le signal de commande d'un servomoteur est un signal **PWM** *(Pulse Width Modulation, modulation de largeur d'impulsions, MLI en français).* Si le signal a une fréquence de 50Hz le niveau haut doit durer entre environ 1ms et 2ms:

- Le niveau haut dure 1ms : on obtient 0° de déplacement angulaire.
- Le niveau haut dure 1,5ms : on obtient 90° de déplacement angulaire.
- Le niveau haut dure 2ms : on obtient 180° de déplacement angulaire.

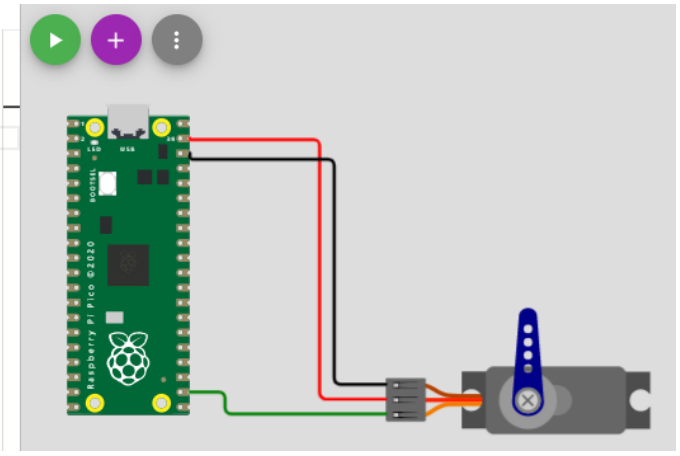


ATTENTION: sur **WOKWI** (simulateur PICO) les broches 16, 18 et 20 ne fonctionnent pas en PWM. Par contre sur le **shield Grove** on utilise les broches 16, 18 ou 20 pour le signal PWM.

Exemples de programmes pour commander un servo-moteur:

Positionner le servomoteur en utilisant duty_u16 (largeur d'impulsion sur 16 bits):

```
from machine import Pin, PWM
import utime
servomoteur = PWM(Pin(17))
servomoteur.freq(50)
while True:
    servomoteur.duty_u16(1638) #positionner le servomoteur à 0°
    utime.sleep(1)
    servomoteur.duty_u16(4750) #positionner le servomoteur à 90°
    utime.sleep(1)
    servomoteur.duty_u16(7864) #positionner le servomoteur à 180°
    utime.sleep(1)
```

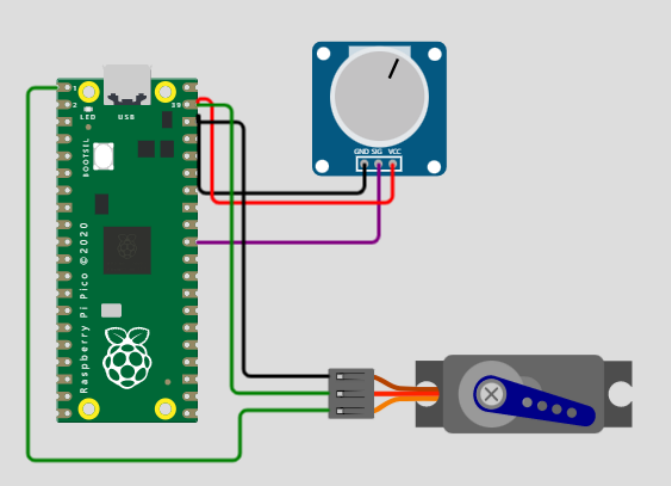


Positionner le servomoteur en utilisant duty_ns (largeur d'impulsion en nanosecondes):

```
from machine import Pin, PWM
import utime
servomoteur = PWM(Pin(17)) #le servomoteur est branché sur la broche 17
servomoteur.freq(50)       #fréquence du signal PWM: 50Hz
while True:
    servomoteur.duty_ns(500000) #positionner le servomoteur à 0°, largeur impulsion 0,5ms
    utime.sleep(1)
    servomoteur.duty_ns(1450000) #positionner le servomoteur à 90°, largeur impulsion 1,45ms
    utime.sleep(1)
    servomoteur.duty_ns(2400000) #positionner le servomoteur à 180°, largeur impulsion 2,5ms
    utime.sleep(1)
```

Le SERVO-MOTEUR (suite) :

Autre montage, commander la position d'un servomoteur à l'aide d'un potentiomètre:



En utilisant duty_u16 (largeur d'impulsion sur 16 bits):

```
from machine import Pin, ADC, PWM
import utime
potentiometre = ADC(0) #le potentiomètre est branché sur la broche 26 (A0)
servomoteur = PWM(Pin(0)) #le servomoteur est branché sur la broche 0
servomoteur.freq(50)  #fréquence 50Hz
while True:
    valeur = potentiometre.read_u16() #lire la valeur du potentiomètre
    print(valeur)   #afficher la valeur (16 bits donc compris entre 0 et 65535)
    angle = int(valeur * 180 / 65535) #convertir en angle
    print(angle)    #afficher l'angle (compris entre 0 et 180°)
    largeur =int((angle * (7864 - 1638) / 180) + 1638) #calculer la largeur d'impulsion
    print(largeur)  #afficher la largeur d'impulsion comprise entre 500us (0°) et 2400us (180°)
    servomoteur.duty_u16(largeur) #positionner le servomoteur
    utime.sleep(1)  #attendre 1s
```

En utilisant duty_ns (largeur d'impulsion en nanosecondes)

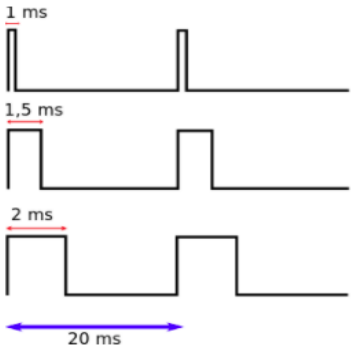
```
from machine import Pin, ADC, PWM
import utime
potentiometre = ADC(0) #le potentiomètre est branché sur la broche 26 (A0)
servomoteur = PWM(Pin(0)) #le servomoteur est branché sur la broche 0
servomoteur.freq(50)  #fréquence 50Hz
while True:
    valeur = potentiometre.read_u16() #lire la valeur du potentiomètre
    print(valeur)   #afficher la valeur (16 bits donc compris entre 0 et 65535)
    angle = int(valeur * 180 / 65535) #convertir en angle
    print(angle)    #afficher l'angle (compris entre 0 et 180°)
    largeur =int((angle * (2400 - 500) / 180) + 500) #calculer la largeur d'impulsion en ns
    print(largeur)  #afficher la largeur d'impulsion comprise entre 500us (0°) et 2400us (180°)
    servomoteur.duty_ns(largeur*1000)  #positionner le servomoteur
    utime.sleep(1)  #attendre 1s
```

Le SERVO-MOTEUR à rotation continue :

Un servomoteur à rotation continue est un moteur électrique utilisé pour obtenir un déplacement.
Le servomoteur à rotation continue se connecte avec 3 fils :

Noir : masse (GND)
Rouge : Vcc
Jaune : signal de commande

Le **signal de commande** d'un servomoteur est un **signal PWM** *(Pulse Width Modulation, modulation de largeur d'impulsions, MLI en français)* de fréquence 50Hz et dont le niveau haut doit durer entre 1ms et 2ms.



Le niveau haut dure 1ms : le moteur tourne à la vitesse maximale dans le sens anti-horaire.
Le niveau haut dure 1,5ms : le moteur est à l'arrêt.
Le niveau haut dure 2ms : le moteur tourne à la vitesse maximale dans le sens horaire.

ATTENTION: sur **WOKWI** (simulateur PICO) il n'y a pas de servo-moteur à rotation continue. Sur le **shield Grove** on peut utiliser les broches 16, 18 ou 20 pour le signal PWM.

```
1   from machine import Pin, PWM
2   import utime
3   MotG = PWM(Pin(1)) #moteur gauche branché sur broche 1
4   MotG.freq(50)
5   MotD = PWM(Pin(0)) #moteur droit branché sur broche 0
6   MotD.freq(50)
7   def avancer(): #fonction pour avancer (motD et motG montés en opposition)
8       MotD.duty_ns(2000000)
9       MotG.duty_ns(1000000)
10  def reculer(): #fonction pour reculer
11      MotD.duty_ns(1000000)
12      MotG.duty_ns(2000000)
13  def droite(): #fonction pour tourner à droite
14      MotD.duty_ns(2000000)
15      MotG.duty_ns(2000000)
16  def gauche(): #fonction pour tourner à gauche
17      MotD.duty_ns(1000000)
18      MotG.duty_ns(1000000)
19  def stop(): #fonction pour arrêter les moteurs
20      MotD.duty_ns(1500000)
21      MotG.duty_ns(1500000)
22  stop()
23  while True :
24      avancer()
25      utime.sleep(1)
26      reculer()
27      utime.sleep(1)
28      droite()
29      utime.sleep(1)
30      gauche()
31      utime.sleep(1)
```

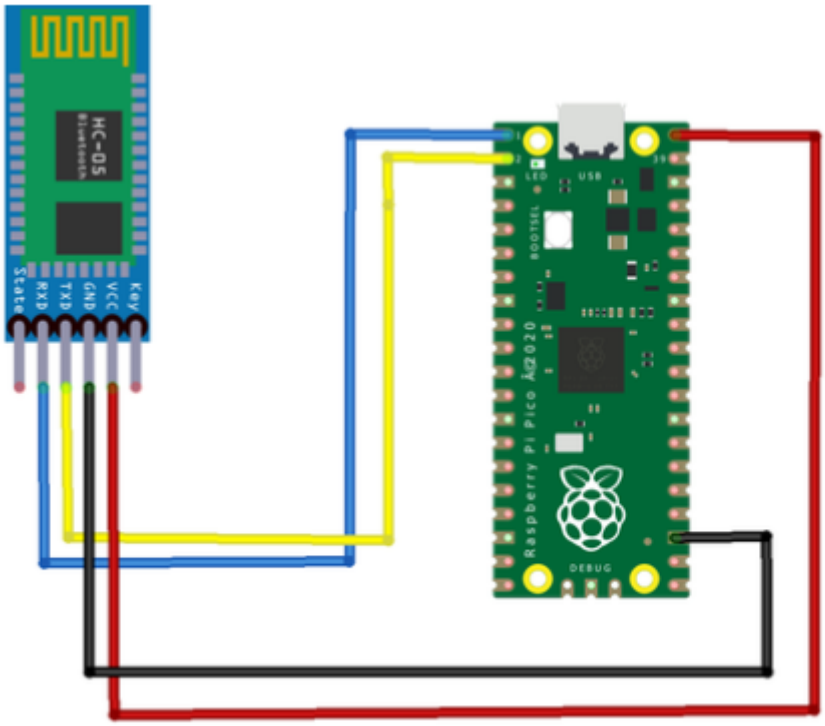
duty_ns(1500000): niveau haut 1,5ms : le moteur est à l'arrêt

duty_ns(1000000): niveau haut 1ms:le moteur est à vitesse maximale dans le sens anti-horaire

duty_ns(2000000): niveau haut 2ms: le moteur est à vitesse maximale dans le sens horaire

LIAISON BLUETOOTH AVEC LE MODULE HC-05

La carte PICO-H ne comporte pas de moyen de communication WIFI ou Bluetooth.
Un module Bluetooth HC-05 connecté sur une carte PICO permet d'établir une communication Bluetooth entre la carte et un autre appareil.
(**Ne peut pas être testé avec le simulateur WOKWI**)




```
1  from machine import Pin, UART
2  import utime
3  uart = UART(0,9600) #établir la liaison série avec le module HC-05 branché en 0 (RX) et 1 (TX)
4  while True:
5      reception = 0
6      if uart.any(): #vérifier si une information est disponible sur le port série
7      reception = uart.readline() #lire l'information disponible
8
9      uart.write('1') #écrire une information sur le port série
```

LIAISON BLUETOOTH HC-05 - SMARTPHONE

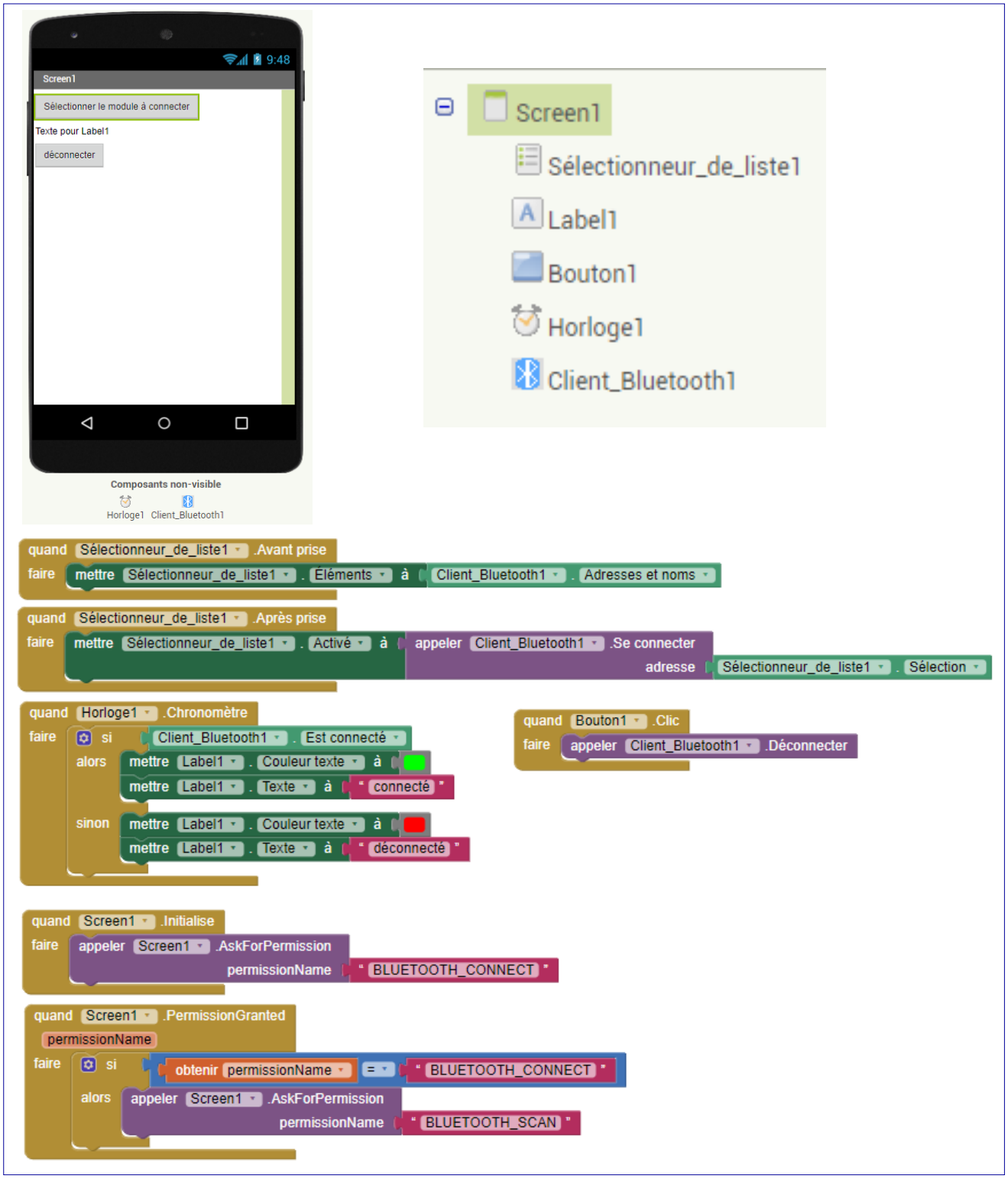
Communication bluetooth avec un smartphone

Le module HC-05 doit être appairé avec le smartphone.

Le code PIN par défaut est : 1234.

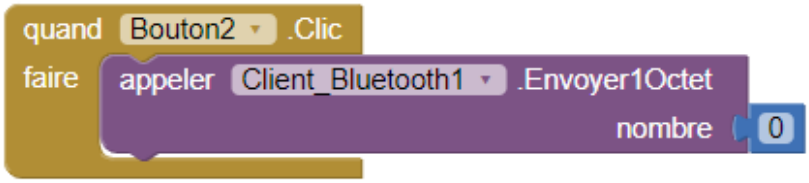
Exemples de programme développé avec l'IDE AppInventor2 :

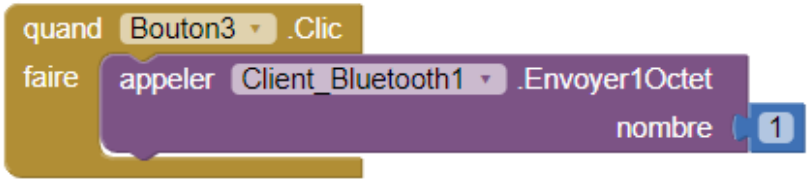
- Déclarer le Bluetooth et se connecter / déconnecter :



LIAISON BLUETOOTH HC-05 - SMARTPHONE (suite)

Envoyer un octet





Recevoir un octet