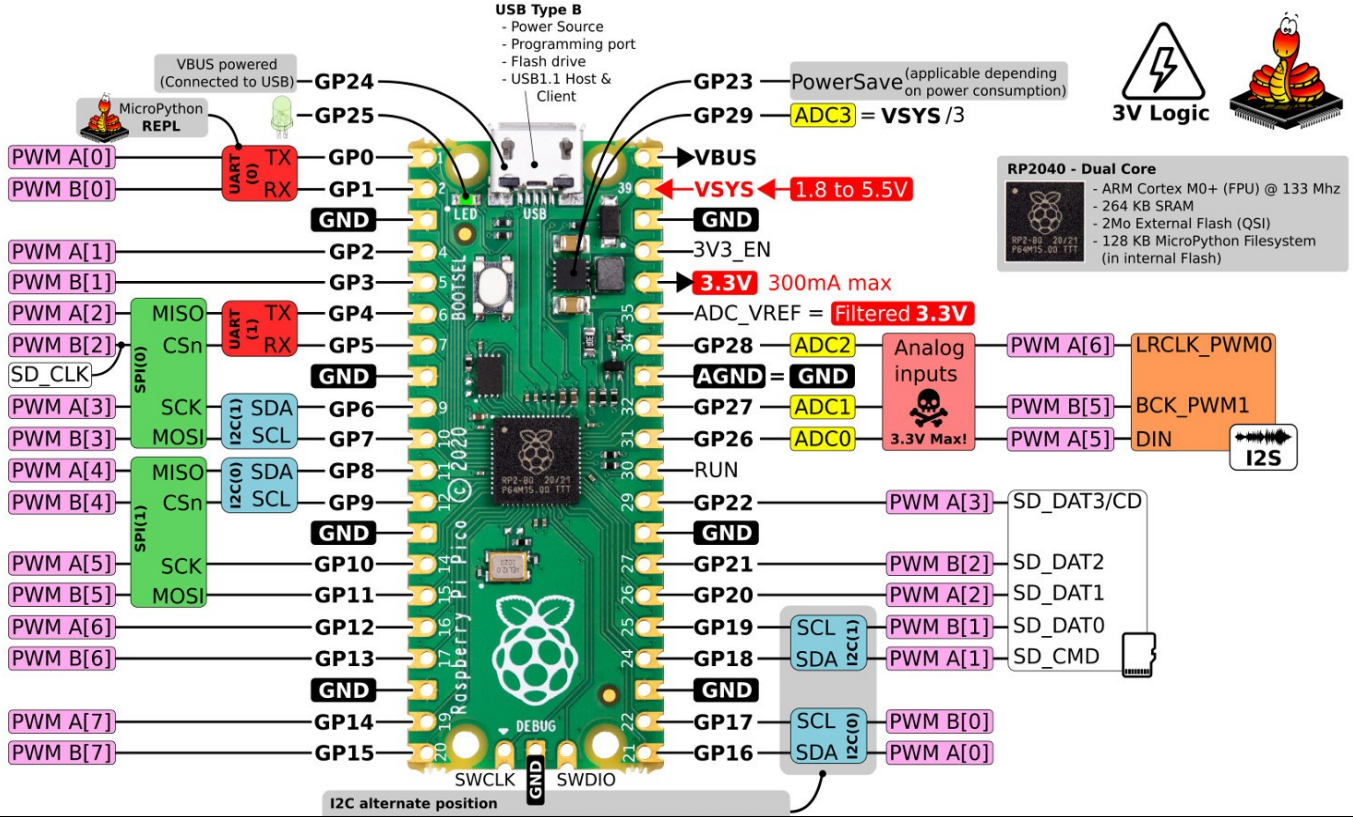
SOMMAIRE

La carte Raspberry PICO ou PICO W

Raspberry PICO

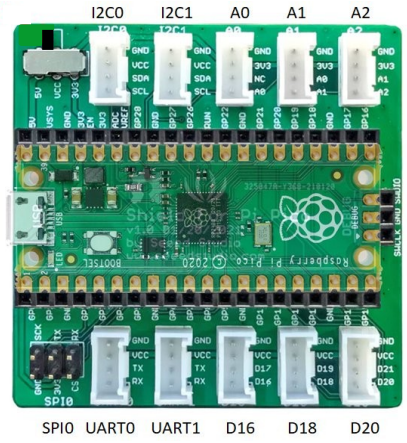
Remarques pour l'**alimentation**:
-1er cas : alimentation via **USB** : pour connecter les composants externes on peut utiliser **VBUS** (broche 40) ou **3.3V** (broche 36).
- 2ème cas : **alimentation externe** via **VSYS** : pour connecter les composants externes il faut utiliser **3.3V** (broche 36). (il n'y a rien sur VBUS).



Remarques concernant les **sorties PWM**:
-Il y a 16 sorties PWM associées par paire (A,B), numérotées de 0 à 7.
- **Certaines des sorties PWM sont disponibles à 2 endroits** sur la carte: par exemple la PWM A[0] est disponible sur la broche 0 (GP0) ou la broche 21 (GP16)

SHIELD GROVE:

- Entrées Analogique :
A0 = GP26, A1 = GP27, A2 = GP28
- Entrées / Sorties numériques :
D16= GP16, D18= GP18, D20= GP20
- Sorties PWM: utiliser les sorties numériques D16, D18 ou D20.

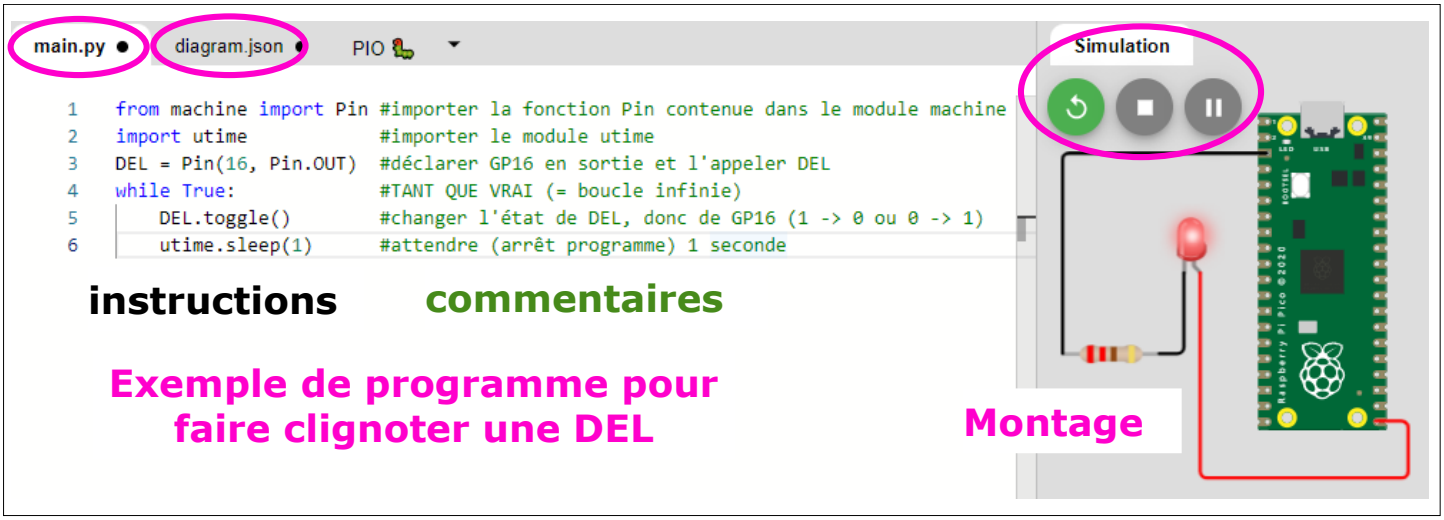


- Liaisons I2C:
I2C1 : SDA = GP6 et SCL = GP7
I2C0 : SDA = GP8 et SCL = GP9
- Liaisons série (UART):
UART0: TX = GP0 et RX = GP1
UART1: TX = GP4 et RX = GP5

L'environnement de simulation de la carte PICO : WOKWI.com

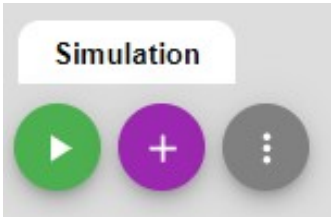
https://wokwi.com/projects/new/micropython-pi-pico

(Ce lien WOKWI permet de programmer la carte PICO en MicroPython)



RÉALISER UN MONTAGE AVEC WOKWI:

- en cliquant sur + on ajoute des **composants**.
- en cliquant sur les **pattes** les **broches** les **entrées / sorties** des composants on crée des **fils de liaison**.
- en cliquant sur un **fil** on change sa **couleur**.
- en cliquant sur ▶ on démarre la simulation



REMARQUES:

- le MicroPython est une version du langage de programmation Python adaptée aux micro-contrôleurs. Tout ce qui est décrit dans ce document concerne micropython, mais reste valable très souvent en python.
- En python pour écrire un commentaire on utilise #. Les commentaires (en vert dans WOKWI) ne sont pas indispensables. Ils servent à expliquer le programme.
- En python l'indentation (c'est à dire le retrait par rapport à la marge) doit être respectée.
- import: si le programme fait appel à des fonctions contenues dans des modules il faut les importer. Certains modules sont directement disponibles (comme machine ou utime), d'autres sont dans des bibliothèques (library) à télécharger.

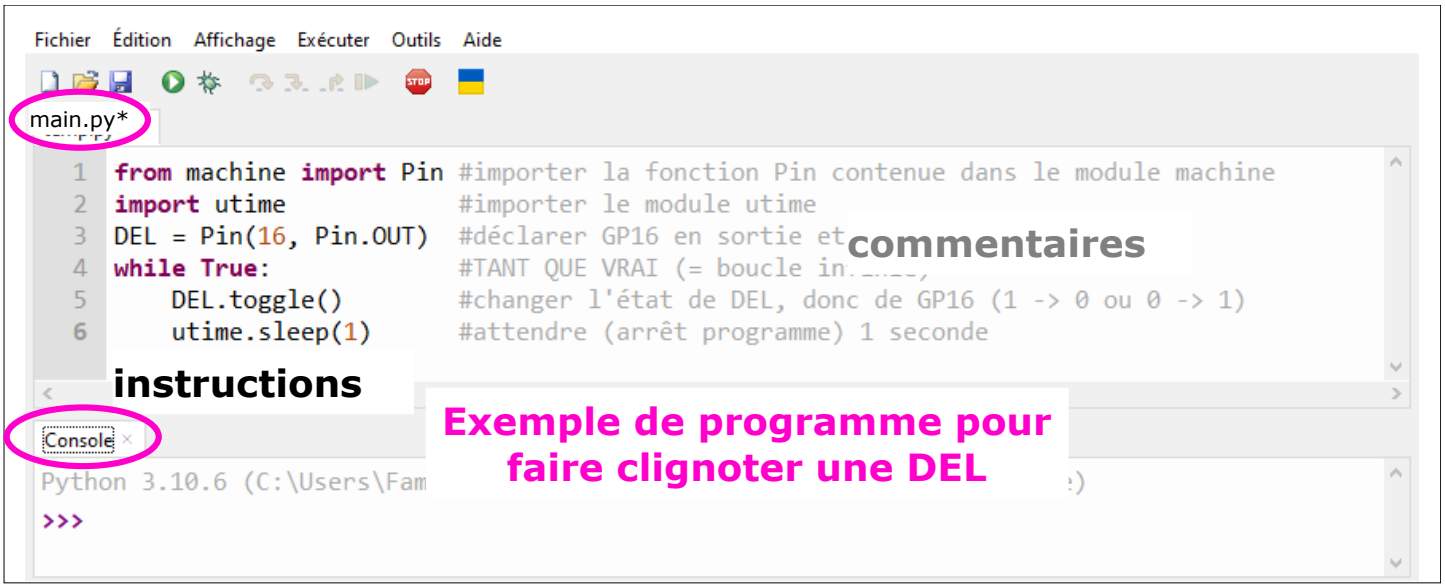
- dans WOKWI certaines des caractéristiques des composants peuvent être modifiées dans l'onglet diagram.json. Par exemple la résistance a par défaut la valeur de 1000ohms. Pour changer la valeur de la résistance à 220ohms il faut aller dans l'onglet diagram.json et remplacer 1000 par 220.
- dans WOKWI 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).
- import: Dans WOKWI pour utiliser un module particulier il faut ouvrir la bibliothèque téléchargée dans un nouvel onglet (new file) puis ajouter la ligne "import" dans le programme principal main.py.
- dans WOKWI certaines sorties ne fonctionnent pas en PWM (par exemple, sur la simulation, les GP16, GP18 et GP20 ne fonctionnent pas en PWM)

- on peut consulter la référence MicroPython et la référence WOKWI :

http://www.micropython.fr/reference/ **https://docs.wokwi.com/?utm_source=wokwi**

L'environnement de développement (IDE) THONNY

L'IDE Thonny permet de programmer en Python.
On peut l'utiliser pour programmer la carte PICO ou PICO W en MicroPython



REMARQUES:

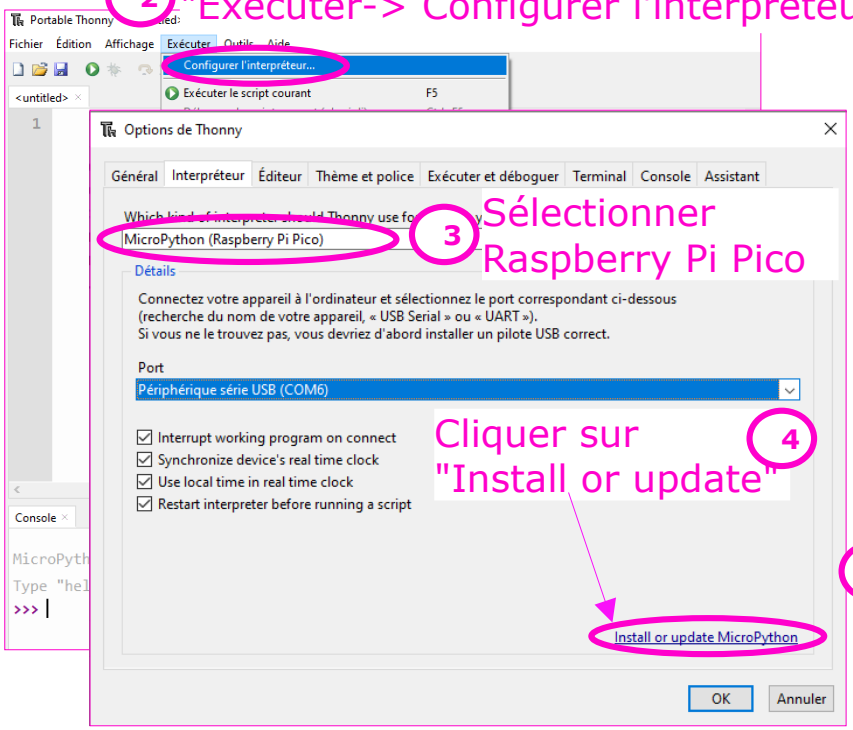
- le MicroPython est une version du langage de programmation Python adaptée aux micro-contrôleurs. Tout ce qui est décrit dans ce document concerne micropython, mais reste valable très souvent en python. - En python pour écrire un commentaire on utilise #. Les commentaires (en gris clair dans Thonny) ne sont pas indispensables. Ils servent à expliquer le programme. - En python l'indentation (c'est à dire le retrait par rapport à la marge) doit être respectée. - import: si le programme fait appel à des fonctions contenues dans des modules il faut les importer. Certains modules sont directement disponibles (comme machine ou utime), d'autres sont dans des bibliothèques (library) à télécharger. - L'instruction while True: permet de créer une boucle qui s'exécute indéfiniment.	- **dans THONNY** cliquer sur Exécuter pour exécuter un programme. Les éventuels messages d'erreur ou les résultats s'affichent dans la "**Console**". - **avec la carte PICO**, au moment d'enregistrer le programme, **Thonny** propose de l'enregistrer sur le PC ou sur la carte PICO. Pour q'un programme enregistré sur la carte s'exécute de façon autonome (une fois déconnecté de l'IDE Thonny) **il doit être enregistré sur la carte ET porter le nom main.py** - import: **dans THONNY** il faut aller dans le menu "Outils -> gérer les paquets" pour installer une bibliothèque dans l'IDE, et ajouter la ligne "import" dans le programme main.py. - import: **avec la carte PICO**, si un programme nécessite une bibliothèque (library), il faut l'ouvrir dans Thonny, puis l'enregistrer dans la mémoire de la carte PICO ("Fichiers" -> "enregistrer sous"). Dans le programme main.py il faut rajouter une ligne "import" pour importer cette bibliothèque.

- on peut consulter la référence MicroPython : **http://www.micropython.fr/reference/**
- sur le site **https://thonny.org/** on peut télécharger l'**IDE Thonny** pour l'installer. On peut aussi télécharger une **version portable** de l'**IDE Thonny** qui ne nécessite pas d'installation.

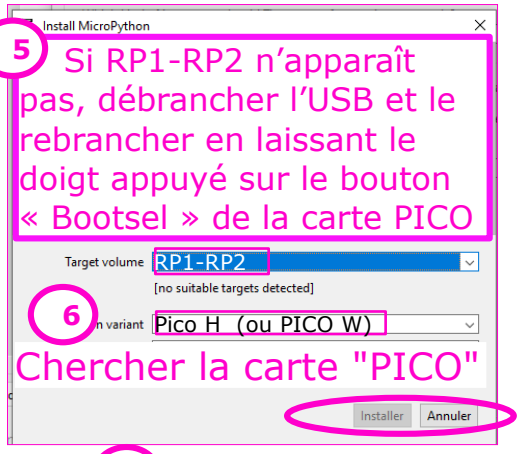
Connecter la carte PICO avec THONNY

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

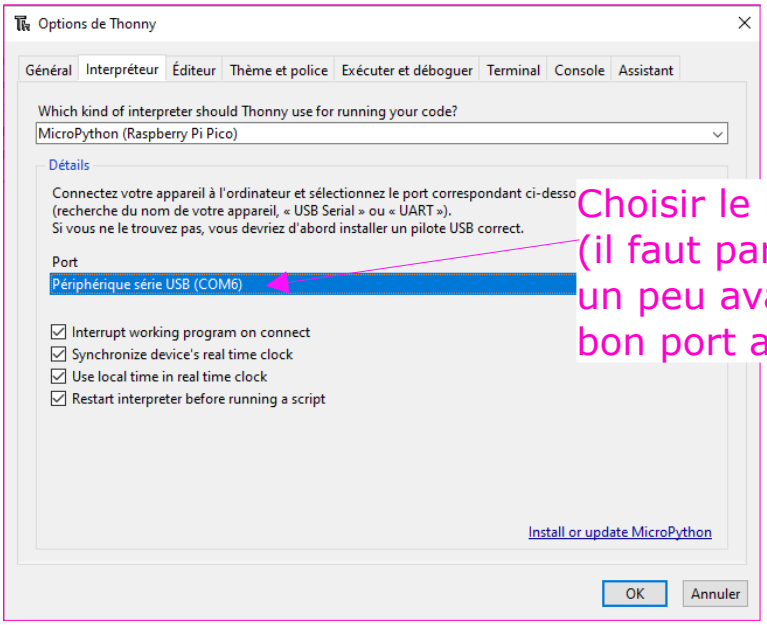
2 "Exécuter-> Configurer l'interpréteur"



REMARQUE: on peut aussi utiliser l'IDE Thonny sans connecter de carte PICO. Dans ce cas, à l'étape 3 on sélectionne simplement «Local Python» au lieu de «Raspberry Pi Pico» et on valide.



7 Cliquer sur "Installer" puis "Fermer"



8 Choisir le bon port USB (il faut parfois attendre un peu avant que le bon port apparaisse)

La Diode Electro-Luminescente DEL (ou LED):

La Diode Electro-Luminescente (DEL), ou Light Emitting Diode (LED) est branchée sur une **sortie numérique de la carte PICO.**

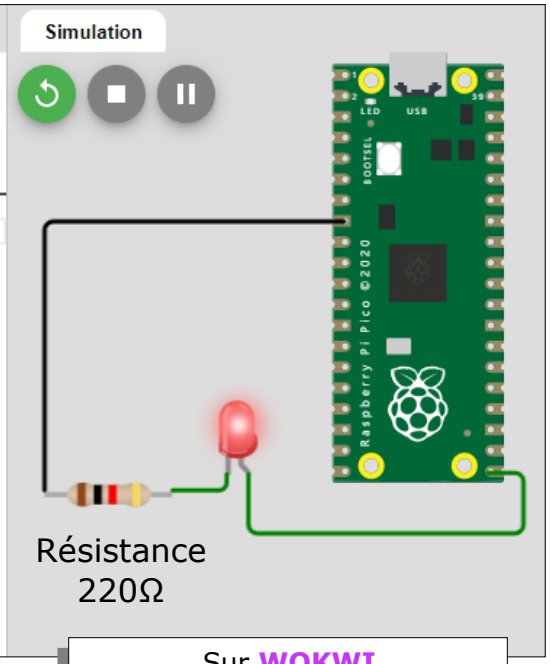
L'anode (+) de la DEL (patte la plus longue, côté arrondi) se branche sur une sortie numérique et la cathode (-) (patte la plus courte, côté plat) sur un GND (ground, le 0V).

Une DEL doit toujours être utilisée en série avec une résistance afin de limiter l'intensité du courant qui la traverse.

Exemple de montage et de programme pour allumer et éteindre une DEL :

main.py ● diagram.json ● PIO

```
1  from machine import Pin #importer la fonction Pin
2  import utime            #importer le module utime
3  DEL = Pin(16, Pin.OUT)  #déclarer que la DEL est branchée en GP16
4  DEL.value(1)    #allumer la DEL
5  utime.sleep(5)  #attendre 5 secondes
6  DEL.value(0)    #éteindre la DEL
7  utime.sleep(2)  #attendre 2 secondes
```



Une **sortie numérique** peut prendre uniquement 2 valeurs: 0 ou 1. Pour «allumer» un composant branché sur une sortie numérique on envoie en sortie un **«1» (signal «HAUT», 3,3V)**. Pour l'«éteindre», on envoie en sortie un **«0» (signal «BAS», 0V)**.

la **carte PICO** possède 26 broches paramétrables comme **entrée ou sortie numérique**. Le **shield Grove** permet d'accéder facilement à 3 d'entre elles : **D16=GP16, D18=GP18, D20=GP20**

Les **cartes PICO** ont une **DEL intégrée**, disponible en **GP25**:
- Sur une **carte PICO W** pour l'utiliser il faut déclarer: **DEL=Pin('LED', Pin.OUT)**
- Sur une **carte PICO**, pour l'utiliser il faut déclarer: **DEL=Pin(25, Pin.OUT)**

Sur **WOKWI** la résistance a par défaut la valeur de 1000Ω. Pour changer la valeur de la résistance à 220Ω il faut aller dans l'onglet **diagram.json** et Remplacer 1000 par 220.

De même, par défaut, la résistance est rouge. Pour changer sa couleur il faut aller dans l'onglet **diagram.json** et remplacer « red » par une autre couleur.

La DEL: Faire clignoter AVEC des «ARRÊTS PROGRAMME» :

Arrêt programme: à chaque fois qu'on utilise la fonction utime.sleep() du module utime le programme s'arrête et ne fait rien d'autre.

- clignoter 1 seule fois, avec **Pin.value()**:

```
1   from machine import Pin #importer la fonction Pin contenue dans le module machine
2   import utime            #importer le module utime
3   DEL = Pin(16, Pin.OUT)  #déclarer que la DEL est branchée en GP16
4   DEL.value(1)        #allumer la DEL
5   utime.sleep(2)      #attendre 2 secondes
6   DEL.value(0)        #éteindre la DEL
7   utime.sleep(2)      #attendre 2 secondes
```

- **clignoter pour toujours** avec une **boucle WHILE** (**boucle infinie**):

```
1   from machine import Pin
2   import utime
3   DEL = Pin(16, Pin.OUT) #DEL branchée en GP16
4   while True:            #TANT QUE VRAI (= boucle infinie)
5       DEL.value(1)       #allumer DEL
6       utime.sleep(1)     #attendre 1 seconde
7       DEL.value(0)       #éteindre DEL
8       utime.sleep(1)     #attendre 1 seconde
```

- **clignoter pour toujours** , programme plus **rapide** avec la fonction **toggle()**

```
1   from machine import Pin #importer la fonction Pin contenue dans le module machine
2   import utime            #importer le module utime
3   DEL = Pin(16, Pin.OUT)  #déclarer GP16 en sortie et l'appeler DEL
4   while True:             #TANT QUE VRAI (= boucle infinie)
5       DEL.toggle()        #changer l'état de DEL, donc de GP16 (1 -> 0 ou 0 -> 1)
6       utime.sleep(1)      #attendre (arrêt programme) 1 seconde
```

En micropython, l'objet Pin du module machine met à notre disposition 3 fonctions pour allumer ou éteindre une sortie numérique: Pin.value(1) et Pin.value(0) Pin.high() et Pin.low() Pin.on() et Pin.off()	En micropython la fonction toggle() permet de changer l'état d'une sortie (passer de 0 à 1 ou de 1 à 0)	L'instruction while True: permet de créer une fonction qui s'exécute indéfiniment.	En python l'indentation (c'est à dire le retrait par rapport à la marge) doit être respectée. Elle indique que les instructions appartiennent à une fonction

Pour connaître toutes les **fonctions** disponibles avec la **classe Pin**:
http://www.micropython.fr/reference/05.micropython/machine/classe_pin/

La DEL: Faire clignoter AVEC des «ARRÊTS PROGRAMME» : (suite)

- clignoter sous **condition** avec **IF… ELSE** et une **variable**:

```
from machine import Pin
import utime
DEL = Pin(16, Pin.OUT)  #DEL branchée en GP16
variable = 0            #créer une variable
while True:
    if variable == 0:   #TEST: SI variable = 0
        variable = 1
    else:               #SINON:
        variable = 0
    DEL.value(variable) #allumer la DEL
    utime.sleep(2)      #attendre 2 secondes
```

- clignoter sous condition avec **IF… ELSE** et une **variable booléenne**:

```
from machine import Pin
import utime
DEL = Pin(16, Pin.OUT)  #DEL branchée en GP16
etat = False            #déclarer une variable booléenne
while True:             #TANT QUE VRAI (= boucle infinie)
    if etat:            #SI etat est True (vrai)
        DEL.value(1)    #allumer DEL
        etat = False
    else :              #SINON (etat est False (faux))
        DEL.value(0)    #éteindre DEL
        etat = True
    utime.sleep(1)      #attendre 1 seconde
```

- clignoter 10 fois avec une **boucle FOR**:

```
from machine import Pin
import utime
DEL = Pin(16, Pin.OUT)  #DEL branchée en GP16
for i in range(10):     #POUR i de 1 à 10 (= boucle exécutée 10 fois)
    DEL.value(1)        #allumer DEL
    utime.sleep(1)      #attendre 1 seconde
    DEL.value(0)        #éteindre DEL
    utime.sleep(1)      #attendre 1 seconde
```

- **"voir" ce qui se passe** dans la boucle for avec l'instruction **print()**:

```
from machine import Pin
import utime
DEL = Pin(16, Pin.OUT)  #DEL branchée en GP16
for i in range(10):     #POUR i de 1 à 10 (= boucle exécutée 10 fois)
    print(i)            #afficher la valeur de l'indice i
    DEL.value(1)
    utime.sleep(1)
    DEL.value(0)
    utime.sleep(1)
```

En python la fonction print() permet d'afficher un message dans la console. Dans le cas d'une utilisation avec la **carte PICO**, print() correspond à l'envoi d'une chaine de caractères sur le port série.

La DEL: Faire clignoter SANS «ARRÊTS PROGRAMME» :

comme on n'utilise pas la fonction utime.sleep() le programme ne s'arrête pas et est disponible pour exécuter d'autres tâches ou instructions.

Dans WOKWI pour le bon 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 une dans le programme).

- avec une **variable** pour compter les «**BOUCLES**»:

```
1   from machine import Pin
2   import utime
3   DEL = Pin(16, Pin.OUT) #DEL branchée en GP16
4   compteur = 0           #variable qui permettra de compter les boucles
5   while True:
6       compteur += 1      #à chaque boucle on incrémente la variable
7       if compteur == 1:
8           DEL.value(1)
9       if compteur == 5:
10          DEL.value(0)
11      if compteur == 10:
12          compteur = 0
13      utime.sleep(0.1) #utile pour que la simulation fonctionne sur WOKWI
```

- en utilisant une **fonction** de la **bibliothèque utime** pour mesurer le temps:

```
1   from machine import Pin
2   import utime
3   DEL = Pin(16, Pin.OUT) #DEL branchée en GP16
4   tempsAVANT = 0
5   while True:
6       temps = utime.ticks_ms()  #stocker la valeur courante du temps"
7       if (temps - tempsAVANT) > 1000:  #TEST: intervalle de temps
8           tempsAVANT = temps           #placer valeur actuelle du temps dans tempsAVANT
9           DEL.toggle()                 #changer état de la DEL
10      utime.sleep(0.1)
```

- Pour en savoir davantage sur la fonction **print()**:
  http://www.micropython.fr/reference/03.builtin/print/

- Pour de connaître les **propriétés**, **fonctions** et **classes** disponibles avec le module **machine:** http://www.micropython.fr/reference/05.micropython/machine/00.module_machine/

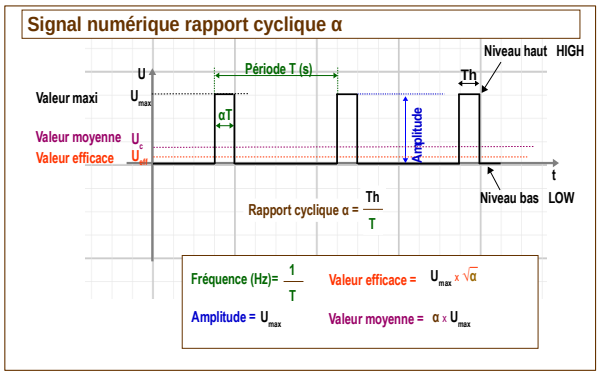
- Pour connaître les **fonctions** disponibles avec le module **utime:**
  http://www.micropython.fr/reference/04.standards/utime/00.module_time/

La DEL: commander l'allumage progressif d'une DEL:

L'anode (+) de la DEL (patte la plus longue, côté arrondi) se branche sur une sortie déclarée comme une **sortie PWM**.

Le **PWM** (**Pulse Width Modulation ou MLI en français pour Modulation de Largeur d'Impulsion)** est une technique qui permet de générer des tensions variables sur des broches numériques.

Principe du PWM: on envoie sur une sortie numérique (qui ne peut prendre que 2 valeurs: 0 ou 1) une **impulsion** (à «1» c'est à dire 3,3V) d'une certaine **largeur** (ou durée) et à une certaine **fréquence** (nombre de fois par secondes). On obtient un **signal numérique de rapport cyclique α**
(en anglais le rapport cyclique se dit **duty cycle**)



La valeur moyenne de ce signal correspond à la formule $\mathbf{V_{moyenne} = \alpha \times V_{max}}$.
«Vu de l'extérieur» ce signal est une tension variable comprise entre 0V et 3,3V.
En pratique, le PWM est utilisé pour : contrôler la luminosité d'une LED, contrôler la vitesse d'un moteur, générer des notes de musique.

Exemple de programme qui allume progressivement une DEL:

```
1  from machine import Pin, PWM
2  import utime
3  DEL = PWM(Pin(17))       #DEL branchée en GP17 (PWM B[0])
4  DEL.freq(50)             #fréquence du signal PWM à 50Hz
5  while True:
6      for i in range(65535):  #POUR de 1 à 65535
7          DEL.duty_u16(i)     #allumer DEL selon valeur de l'indice
8          utime.sleep_us(1)   #attendre 1 micro seconde
```

- La fonction PWM.freq() permet de fixer la valeur de la fréquence du signal.
- La fonction PWM.duty_u16() permet de fixer la largeur d'impulsion en valeur numérique sur 16 bits c'est à dire entre 0 et 65535.
- La fonction PWM.duty_ns() permet de fixer la largeur d'impulsion en nano-secondes.

ATTENTION: la carte PICO possède 26 broches qui peuvent être déclarées comme sortie PWM mais **en réalité il n'y a que 16 sorties PWM** (associées par paire (A,B), numérotées de 0 à 7). **Certaines des sorties PWM sont disponibles à 2 endroits** sur la carte: par exemple la PWM A[0] est disponible sur la broche 0 (GP0) ou la broche 21 (GP16). **Par exemple, on ne peut pas utiliser en même temps la GP0 et la GP16 en sortie PWM, puisque c'est la même…**

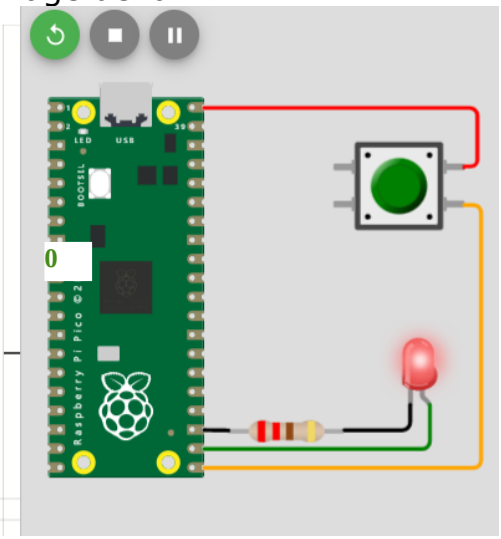
ATTENTION: sur WOKWI les ports 16, 18 et 20 ne fonctionnent pas en PWM. PAR CONTRE sur le shield Grove on utilise les broches 16, 18 ou 20 pour le signal PWM.

- Pour connaître les **fonctions** disponibles avec la **classe PWM:**
http://www.micropython.fr/reference/05.micropython/machine/classe_PWM/

La DEL : programmer une interruption avec IRQ

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

```
1  from machine import Pin, PWM
2  import utime
3  DEL = Pin(17, Pin.OUT)                  #DEL branchée en GP17
4  Bouton = Pin(16, Pin.IN, Pin.PULL_DOWN) #bouton pousoir branché en GP16, forcé à 0
5
6  interruption = 0
7
8  def PRIORITE(Bouton):       #fonction qui sera exécutée en cas d'interruption
9      global interruption
10     interruption = 1
11
12 Bouton.irq(trigger=Pin.IRQ_RISING, handler=PRIORITE) #INTERRUPTON associée à Bouton (GP16)
13             #IRQ_RISING : interruption déclenchée si GP16 passe à 1
14
15 while True:
16     if interruption == 1:
17         DEL.value(0)
18         utime.sleep(2)
19         interruption = 0
20     DEL.value(1)
21     utime.sleep(0.25)
```



Bouton.irq(trigger=Pin.IRQ_RISING, handler=PRIORITE)

Nom du port utilisé pour déclencher l'interruption

Nom de la fonction qui sera exécutée en cas d'interruption

Mode de déclenchement de l'interruption

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

la fonction IRQ appartient à la classe Pin.

En python on peut créer ses propres fonctions avec l'instruction def:

```
def NOM_FONCTION(PARAMETRES_ENTRÉE):
        instruction1      #instructions de la fonction
        instruction2 ...
return VALEUR_RENVOYEE    # facultatif
```

En python l'indentation (c'est à dire le retrait par rapport à la marge) doit être respectée. Elle indique que les instructions appartiennent à la fonction

Pour connaître toutes les **fonctions** disponibles avec la **classe Pin**:
http://www.micropython.fr/reference/05.micropython/machine/classe_pin/

Le POTENTIOMÈTRE
CAPTEUR ANALOGIQUE.

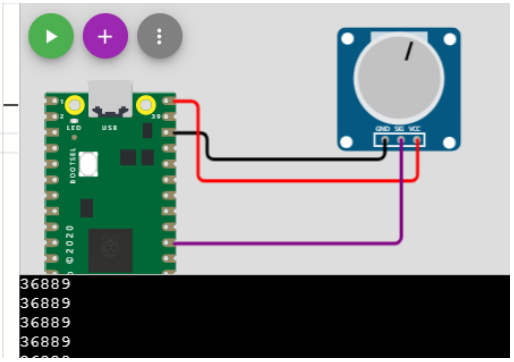
La broche «SIGnal» du potentiomètre se branche sur une **entrée analogique (ADC)**.

Dans une simulation, le POTENTIOMÈTRE peut REMPLACER n'importe quel capteur analogique, particulièrement ceux qui n'existent pas sur WOKWI.

la **carte PICO** possède 3 broches paramétrables comme **entrée analogique**: GP26, GP27 et GP28
Le **shield Grove** permet d'y accéder facilement :
A0=GP26, A1=GP27, A2=GP28

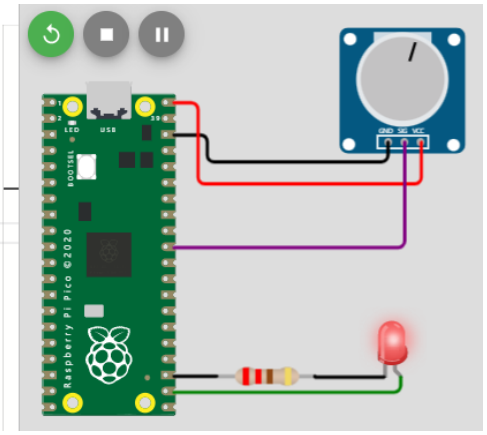
Exemple: afficher la valeur (ou position) du potentiomètre

```
1  from machine import Pin, PWM, ADC
2  import utime
3  potentiometre = ADC(0)  #potentiomètre en GP26 (entrée analogique A0)
4  while True:
5      valeur = potentiometre.read_u16()   #lire valeur du potentiomètre (sur 16 bits, entre 0 et 65535)
6      print(valeur)   #afficher la valeur du potentiomètre
7      utime.sleep(0.1)
```



Autre exemple : faire varier l'intensité d'une DEL avec un potentiomètre en affichant la valeur du potentiomètre

```
1  from machine import Pin, PWM, ADC
2  import utime
3  DEL = PWM(Pin(17))      #DEL branchée en GP17 (PWM B[0])
4  DEL.freq(50)            #fréquence du signal PWM à 50Hz
5  potentiometre = ADC(0)  #potentiomètre en GP26 (entrée analogique A0)
6  while True:
7      valeur = potentiometre.read_u16()   #lire valeur du potentiomètre (sur 16 bits, entre 0 et 65535)
8      print(valeur)   #afficher a valeur du potentiomètre
9      DEL.duty_u16(valeur) #allumer DEL selon valeur lue du potentiomètre
10     utime.sleep(0.1)
```



- Pour connaître les **propriétés** et **fonctions** disponibles avec la **classe ADC:**
http://www.micropython.fr/reference/05.micropython/machine/classe_ADC/ (par exemple la fonction **.read_u16()**)

L'INTERRUPTEUR à glissière (slide switch)

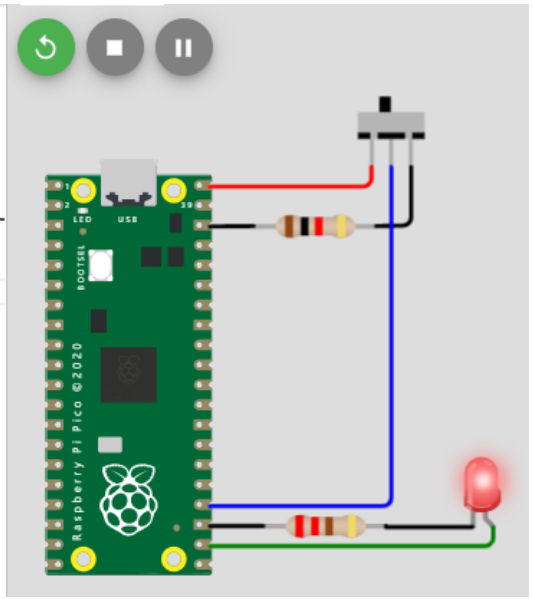
CAPTEUR TOR (LOGIQUE).

L'interrupteur se branche sur une **entrée numérique**.

Dans une simulation, l'INTERRUPTEUR peut REMPLACER n'importe quel capteur TOR, particulièrement ceux qui n'existent pas sur WOKWI.	la **carte PICO** possède 26 broches paramétrables comme **entrée ou sortie numérique**. Le **shield Grove** permet d'accéder facilement à 3 d'entre elles : **D16=GP16, D18=GP18, D20=GP20**

Exemple: éteindre ou allumer une DEL avec un interrupteur

```
1  from machine import Pin
2  import utime
3  DEL = Pin(17, Pin.OUT)  #DEL branchée en GP16
4  interrupteur = Pin(18, Pin.IN)  #interrupteur branché en GP18
5  while True:
6      valeur = interrupteur.value()   #lire valeur de l'interrupteur (1 ou 0)
7      if valeur == 1:    #TEST: si interrupteur est fermé
8          DEL.value(1)   #allumer la DEL
9      else:              #sinon:
10         DEL.value(0)   #éteindre la DEL
11     utime.sleep(0.1)
```



Le BOUTON POUSSOIR (Pushbutton) :
CAPTEUR TOR (LOGIQUE).

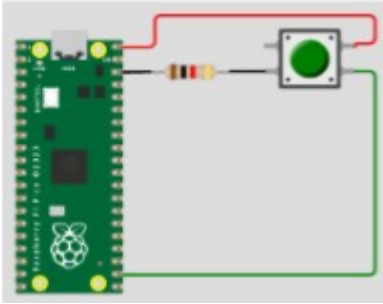
Le bouton poussoir se branche sur une **entrée numérique**.

Dans une simulation, le BOUTON POUSSOIR peut REMPLACER n'importe quel capteur TOR, particulièrement ceux qui n'existent pas sur WOKWI.

Dans le montage 1 on **connecte le BP au GND** (en passant par une résistance) **pour forcer GP17 à 0** quand on n'appuie pas sur le BP (si on ne le fait pas il y a une incertitude et le programme ne fonctionne pas correctement).
Dans le montage 2 on fait **autrement** : l'instruction **PULL_DOWN permet de forcer une entrée à 0**.

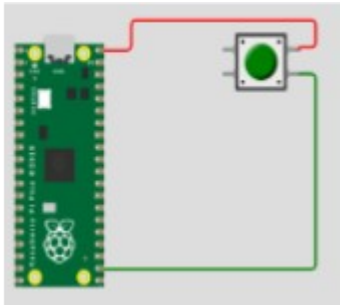
Exemple 1: compter le nombre d'appuis sur un bouton

```
1  from machine import Pin
2  import utime
3  Bouton = Pin(17, Pin.IN) #BP branché en GP17
4  compteur = 0
5  while True:
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
```



Exemple 2 avec PULL-DOWN:

```
1  from machine import Pin
2  import utime
3  Bouton = Pin(17, Pin.IN, Pin.PULL_DOWN) #BP branché en GP17, forcé à 0
4  compteur = 0
5  while True:
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
```



On pourrait aussi utiliser **PULL_UP** qui permet de forcer une entrée à 1.
PULL_UP et **PULL_DOWN** sont des propriétés de la **classe Pin**.

Pour connaître toutes les **fonctions** disponibles avec la **classe Pin**:
http://www.micropython.fr/reference/05.micropython/machine/classe_pin/

Autre exemple: éteindre ou allumer une DEL (branchée en GP21) avec un BP

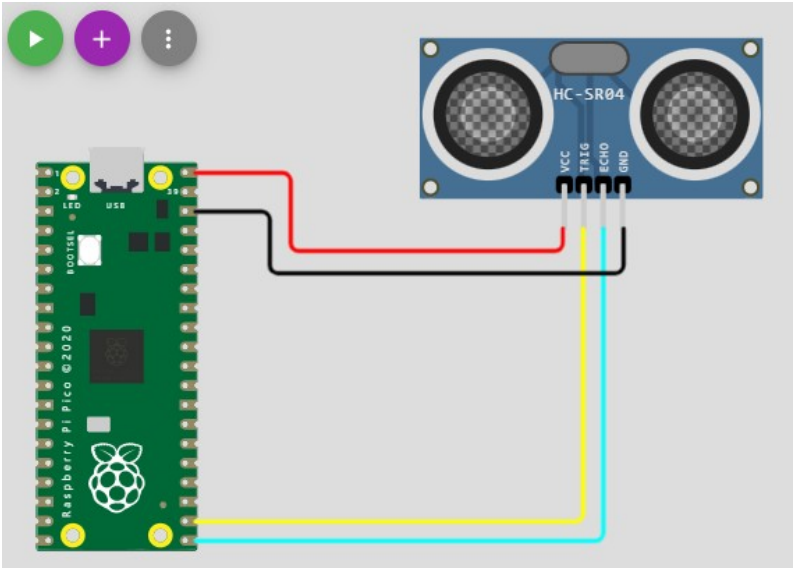
```
1  from machine import Pin
2  import utime
3  DEL = Pin(21, Pin.OUT)      #DEL branchée sur GP21
4  Bouton = Pin(17, Pin.IN, Pin.PULL_DOWN) #BP branché en GP17, forcé à 0
5  while True:
6      if Bouton.value() == 1: #TEST : si appui sur BP
7          DEL.value(1)        #allumer DEL
8      else:                   #sinon
9          DEL.value(0)        #éteindre DEL
```

Le capteur à Ultrasons HC-SR04

CAPTEUR NUMÉRIQUE.

La broche Trig du capteur à US se branche sur une **sortie numérique**.

La broche Echo du capteur à US se branche sur une **entrée numérique**.



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:

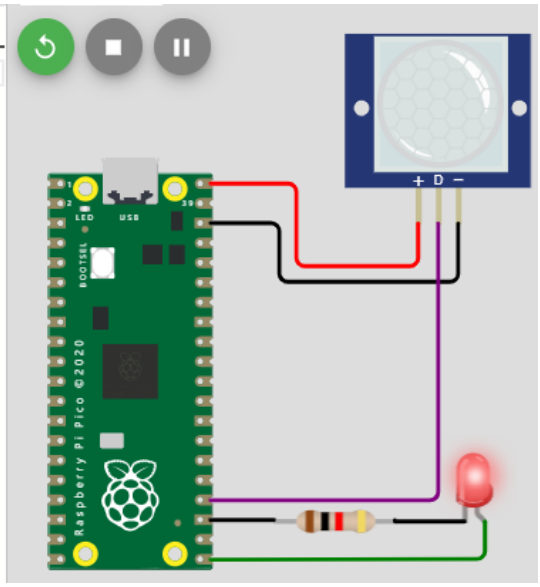
```
from machine import Pin
import utime

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

CAPTEUR TOR (LOGIQUE).

La broche D (OUT) du capteur PIR se branche sur une **entrée numérique**.

```
from machine import Pin
import utime
DEL = Pin(16, Pin.OUT)  #DEL branchée en GP16
PIR = Pin(18, Pin.IN)  #capteur PIR branché en GP18
while True:
    if PIR.value() == 1:    #TEST: si il y a un mouvement
        DEL.value(1)        #allumer la DEL
        utime.sleep(1)
    else:                   #sinon:
        DEL.value(0)        #éteindre la DEL
        utime.sleep(1)
```



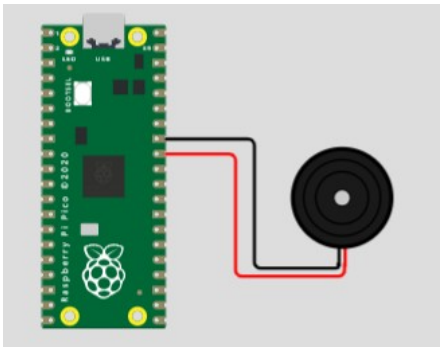
Le BUZZER :

Le buzzer se branche sur une sortie déclarée comme **sortie PWM**.

- La fonction PWM.freq() permet de fixer la valeur de la fréquence du signal.
- La fonction PWM.duty_u16() permet de fixer la largeur d'impulsion en valeur numérique sur 16 bits c'est à dire entre 0 et 65535.
- La fonction PWM.duty_ns() permet de fixer la largeur d'impulsion en nano-secondes.

ATTENTION: la carte PICO possède 26 broches qui peuvent être déclarées comme sortie PWM mais **en réalité il n'y a que 16 sorties PWM** (associées par paire (A,B), numérotées de 0 à 7). **Certaines des sorties PWM sont disponibles à 2 endroits** sur la carte: par exemple la PWM A[0] est disponible sur la broche 0 (GP0) ou la broche 21 (GP16). **Par exemple, on ne peut pas utiliser en même temps la GP0 et la GP16 en sortie PWM, puisque c'est la même...**

ATTENTION: sur WOKWI les ports 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.



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

Exemple: jouer une 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)
```

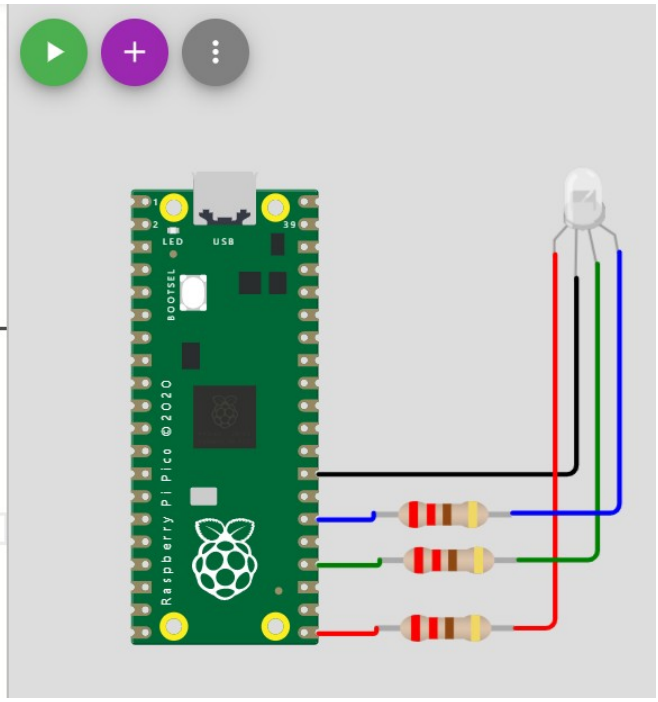
Autre exemple: produire une succession de bips plus ou moins aigus :

```
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 (ou DEL RGB)

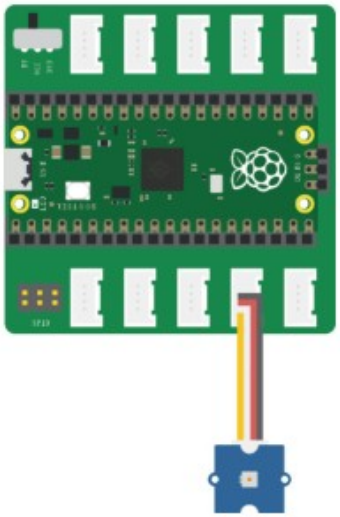
La DEL RVB se branche sur **3 sorties numériques**.

```
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)
```



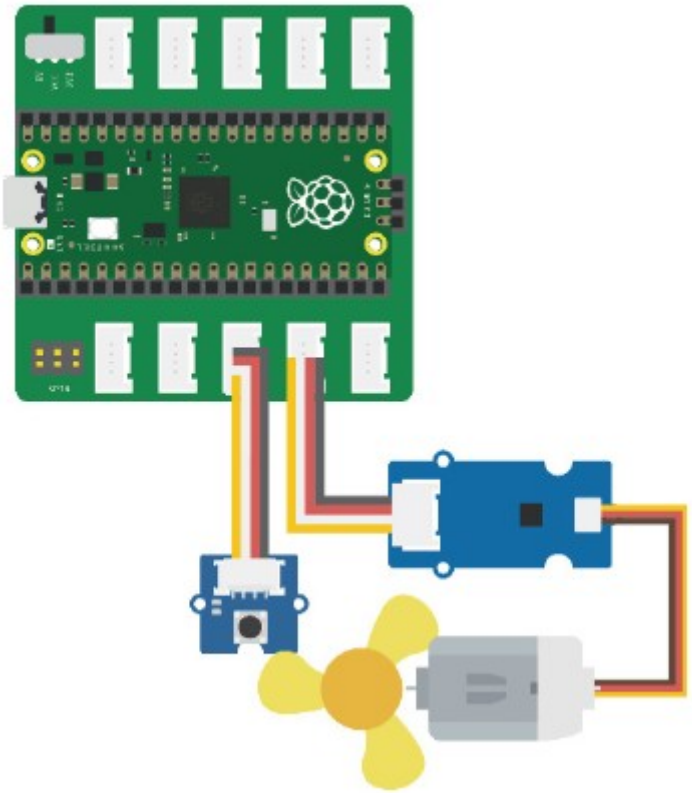
Montage et programme pour la DEL RVB Grove en utilisant la **bibliothèque** ws2812

```
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)
```



Le MOTEUR GROVE (ventilateur)

Le moteur grove se branche sur une **sortie numérique**.
Montage du moteur grove en D18, avec un bouton poussoir en D16:



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)
```

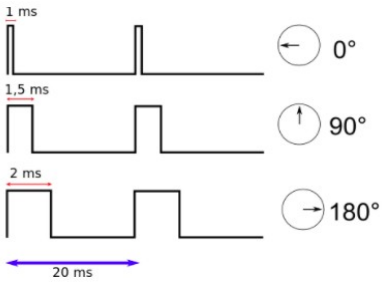
Le RELAIS miniature:

Le relais miniature se branche sur une **sortie numérique**.

Le SERVO-MOTEUR :

Le «signal» (fil jaune ou orange) du servomoteur se branche sur une **sortie PWM**.

Un **servomoteur** est un **moteur** électrique utilisé **pour obtenir un déplacement angulaire** (donc une position) spécifique. Le signal de commande d'un servomoteur est un **signal PWM**.
Si le signal a une **fréquence** de **50Hz** (=**période 20ms**) le niveau haut doit durer entre environ 1ms et 2ms:
- Le niveau haut dure 1ms = 0° de déplacement angulaire.
- Le niveau haut dure 1,5ms = 90° de déplacement angulaire.
- Le niveau haut dure 2ms = 180° de déplacement angulaire.



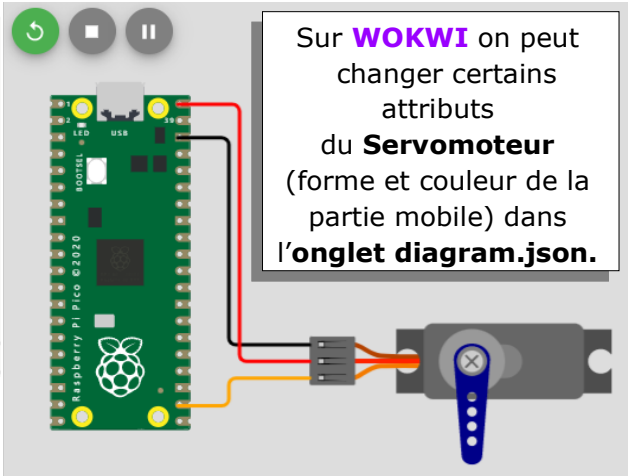
- La fonction PWM.freq() permet de fixer la valeur de la fréquence du signal.
- La fonction PWM.duty_u16() permet de fixer la largeur d'impulsion en valeur numérique sur 16 bits c'est à dire entre 0 et 65535.
- La fonction PWM.duty_ns() permet de fixer la largeur d'impulsion en nano-secondes.

ATTENTION: la carte PICO possède 26 broches qui peuvent être déclarées comme sortie PWM mais **en réalité il n'y a que 16 sorties PWM** (associées par paire (A,B), numérotées de 0 à 7). **Certaines des sorties PWM sont disponibles à 2 endroits** sur la carte: par exemple la PWM A[0] est disponible sur la broche 0 (GP0) ou la broche 21 (GP16). **Par exemple, on ne peut pas utiliser en même temps la GP0 et la GP16 en sortie PWM, puisque c'est la même…**

ATTENTION: sur WOKWI les ports 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.

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

```
1   from machine import Pin, PWM
2   import utime
3   servomoteur = PWM(Pin(17)) #servomoteur branché en GP17
4   servomoteur.freq(50)       #fréquence du signal PWM
5   while True:
6       servomoteur.duty_u16(1638) #positionner le servomoteur à 0°
7       utime.sleep(1)
8       servomoteur.duty_u16(4750) #positionner le servomoteur à 90°
9       utime.sleep(1)
10      servomoteur.duty_u16(7864) #positionner le servomoteur à 180°
11      utime.sleep(1)
```



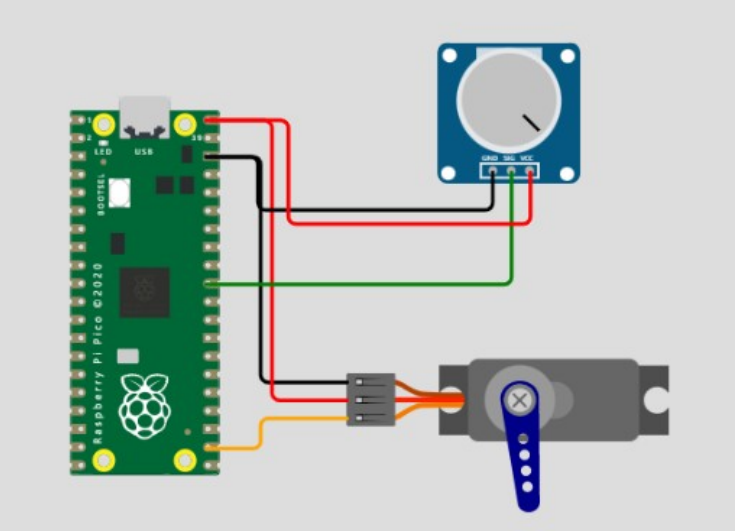
Sur **WOKWI** on peut changer certains attributs du **Servomoteur** (forme et couleur de la partie mobile) dans l'**onglet diagram.json.**

Autre exemple: Positionner avec duty_ns (largeur d'impulsion en nanosecondes)

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

Le SERVO-MOTEUR (suite):

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



Exemple: En utilisant duty_u16:

```
from machine import Pin, ADC, PWM
import utime
potentiometre = ADC(0)     #potentiomètre branché en GP26 (A0)
servomoteur = PWM(Pin(17)) #servomoteur branché en GP17
servomoteur.freq(50)       #fréquence du signal PWM
while True:
    valeur = potentiometre.read_u16() #lire la valeur du potentiomètre (CAN en entrée sur 16 bits)
    print(valeur)                     #afficher la valeur du potentiomètre (nombre entier entre 0 et 65535)
    angle = int(valeur * 180 / 65535) #convertir la valeur du potentiomètre en degrés
    print(angle)                      #afficher l'angle (degrés)
    largeur = int((angle * (7684 -1638) / 180) + 1638) #calculer la lageur d'impulsion
    print(largeur)                    #afficher la largeur calculée (nombre entier entre 1638 et 7684)
    servomoteur.duty_u16(largeur)     #positionner le servomoteur
    utime.sleep(1)
```

Autre exemple: En utilisant duty_ns

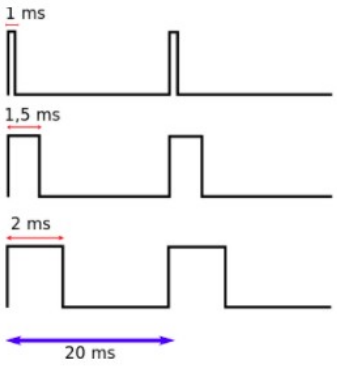
```
from machine import Pin, ADC, PWM
import utime
potentiometre = ADC(0)     #potentiomètre branché en GP26 (A0)
servomoteur = PWM(Pin(17)) #servomoteur branché en GP17
servomoteur.freq(50)       #fréquence du signal PWM
while True:
    valeur = potentiometre.read_u16() #lire la valeur du potentiomètre (CAN en entrée sur 16 bits)
    print(valeur)                     #afficher la valeur du potentiomètre (nombre entier entre 0 et 65535)
    angle = int(valeur * 180 / 65535) #convertir la valeur du potentiomètre en degrés
    print(angle)                      #afficher l'angle (compris entre 0 et 180 degrés)
    largeur = int((angle * (2400 - 500) / 180) + 500) #calculer la largeur d'impulsion en microsecondes
    largeur = largeur * 1000          #convertir la largeur en nanosecondes
    print(largeur)                    #afficher la largeur calculée (nombre entier entre 500000ns et 2400000ns)
    servomoteur.duty_ns(largeur)      #positionner le servomoteur
    utime.sleep(1)
```

- Pour connaître les **fonctions** disponibles avec la **classe PWM:**
  http://www.micropython.fr/reference/05.micropython/machine/classe_PWM/
- Pour **changer les attributs** du servomoteur **WOKWI** voir https://docs.wokwi.com/parts/wokwi-servo

Le SERVO-MOTEUR à rotation continue :

Le «signal» (fil jaune ou orange) du servomoteur se branche sur une **sortie PWM**.

Un servomoteur à rotation continue est un servomoteur utilisé pour obtenir un **déplacement** (**tourne de façon continue**). Le signal de commande d'un servomoteur est un **signal PWM**.
Si le signal a une **fréquence** de **50Hz** (=**période 20ms**) le niveau haut doit durer entre environ 1ms et 2ms:
- Le niveau haut dure 1ms = le moteur tourne à la vitesse maximale dans le sens anti-horaire. (1ms = 1000000ns)
- Le niveau haut dure 1,5ms = le moteur est à l'arrêt. (1,5ms = 1500000ns)
- Le niveau haut dure 2ms : le moteur tourne à la vitesse maximale dans le sens horaire. (2ms =2000000ns)



- La fonction PWM.freq() permet de fixer la valeur de la fréquence du signal.
- La fonction PWM.duty_u16() permet de fixer la largeur d'impulsion en valeur numérique sur 16 bits c'est à dire entre 0 et 65535.
- La fonction PWM.duty_ns() permet de fixer la largeur d'impulsion en nano-secondes.

ATTENTION: la carte PICO possède 26 broches qui peuvent être déclarées comme sortie PWM mais **en réalité il n'y a que 16 sorties PWM** (associées par paire (A,B), numérotées de 0 à 7). **Certaines des sorties PWM sont disponibles à 2 endroits** sur la carte: par exemple la PWM A[0] est disponible sur la broche 0 (GP0) ou la broche 21 (GP16). **Par exemple, on ne peut pas utiliser en même temps la GP0 et la GP16 en sortie PWM, puisque c'est la même...**

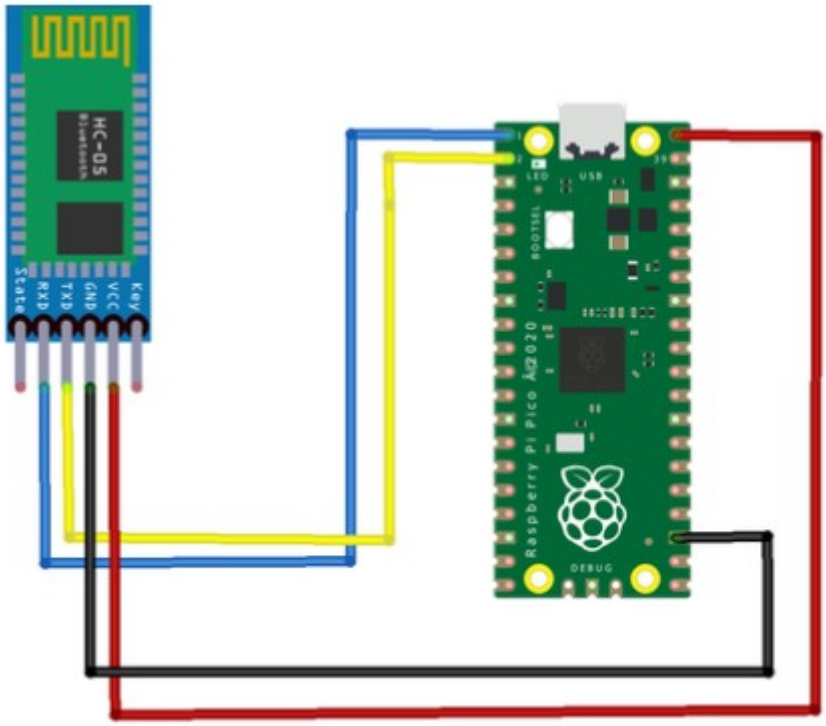
ATTENTION: sur WOKWI IL N'Y A PAS de servo-moteur à rotation continue. Sur le SHIELD GROVE on utilise les broches 16, 18 ou 20 pour le signal PWM.

Exemple: programme qui commande les 2 servomoteurs à rotation continue d'un robot (MotG = moteur gauche et MotD =moteur droit):

```
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)
```

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**)



Recevoir une information

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

Envoyer une information

```
from machine import Pin, UART
import utime
uart = UART(0,9600) #établir la liaison série avec le module HC-05 branché en 0 (Rx) et 1 (Tx)

obstacle = 10
while True:
    if obstacle < 10:
        uart.write('A')  #envoyer 'A'
        utime.sleep(0.5) #! il faut une pause suffisamment longue
    else:
        uart.write('B')  #envoyer 'B'
        utime.sleep(0.5) #! il faut une pause suffisamment longue
```

LIAISON BLUETOOTH HC-05 - SMARTPHONE

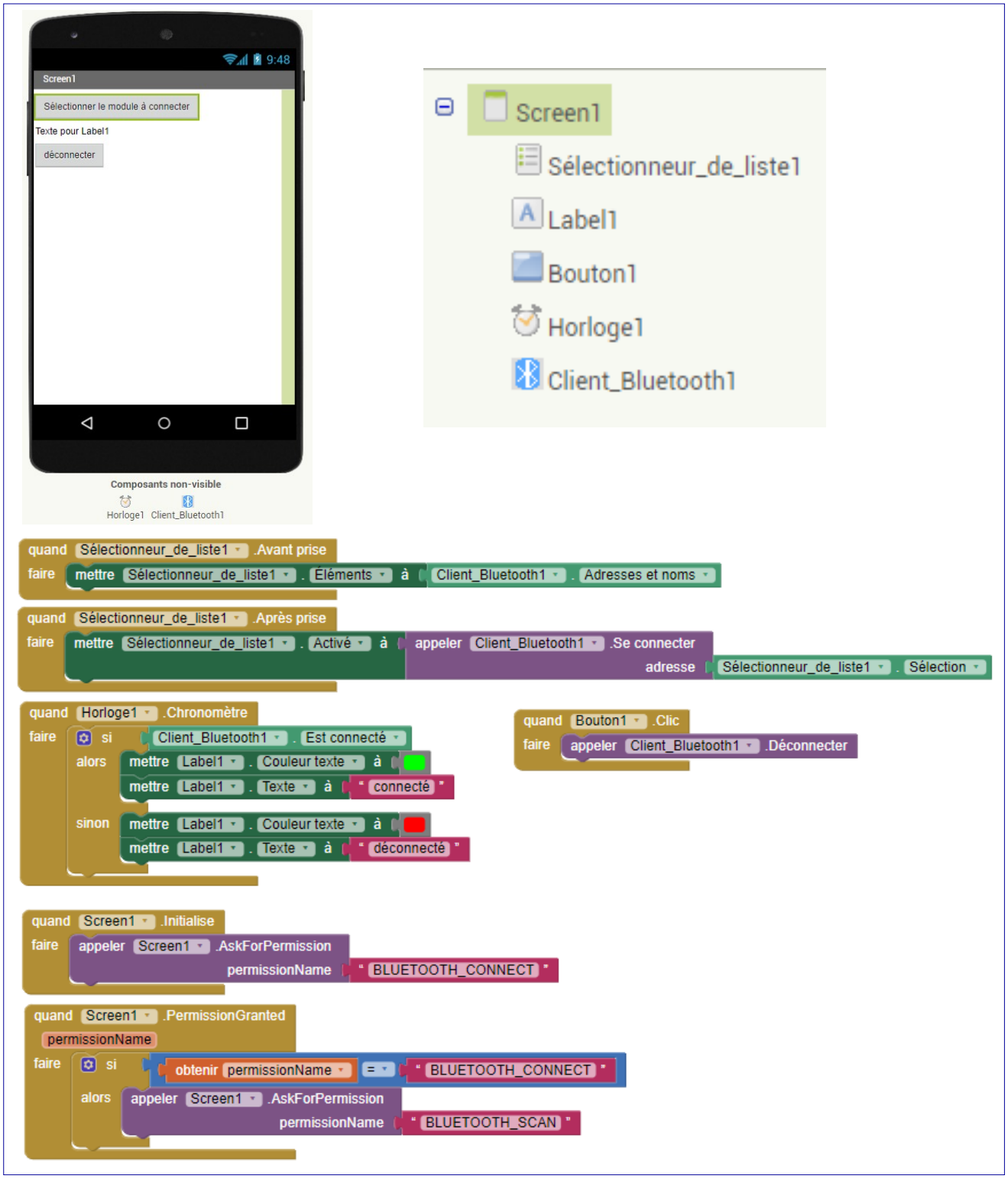
Communication bluetooth avec un smartphone

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

Le code PIN par défaut est : 1234.

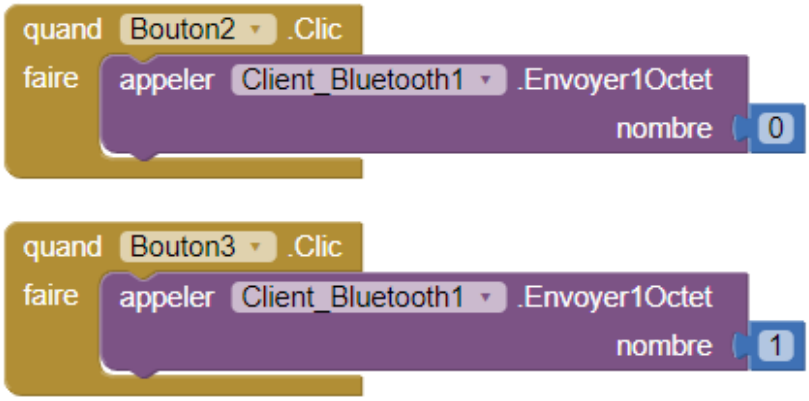
Exemples de programme développé avec l'IDE AppInventor2 :

- Déclarer le Bluetooth et se connecter / déconnecter :



LIAISON BLUETOOTH HC-05 - SMARTPHONE (suite)

Envoyer un octet



Recevoir un octet