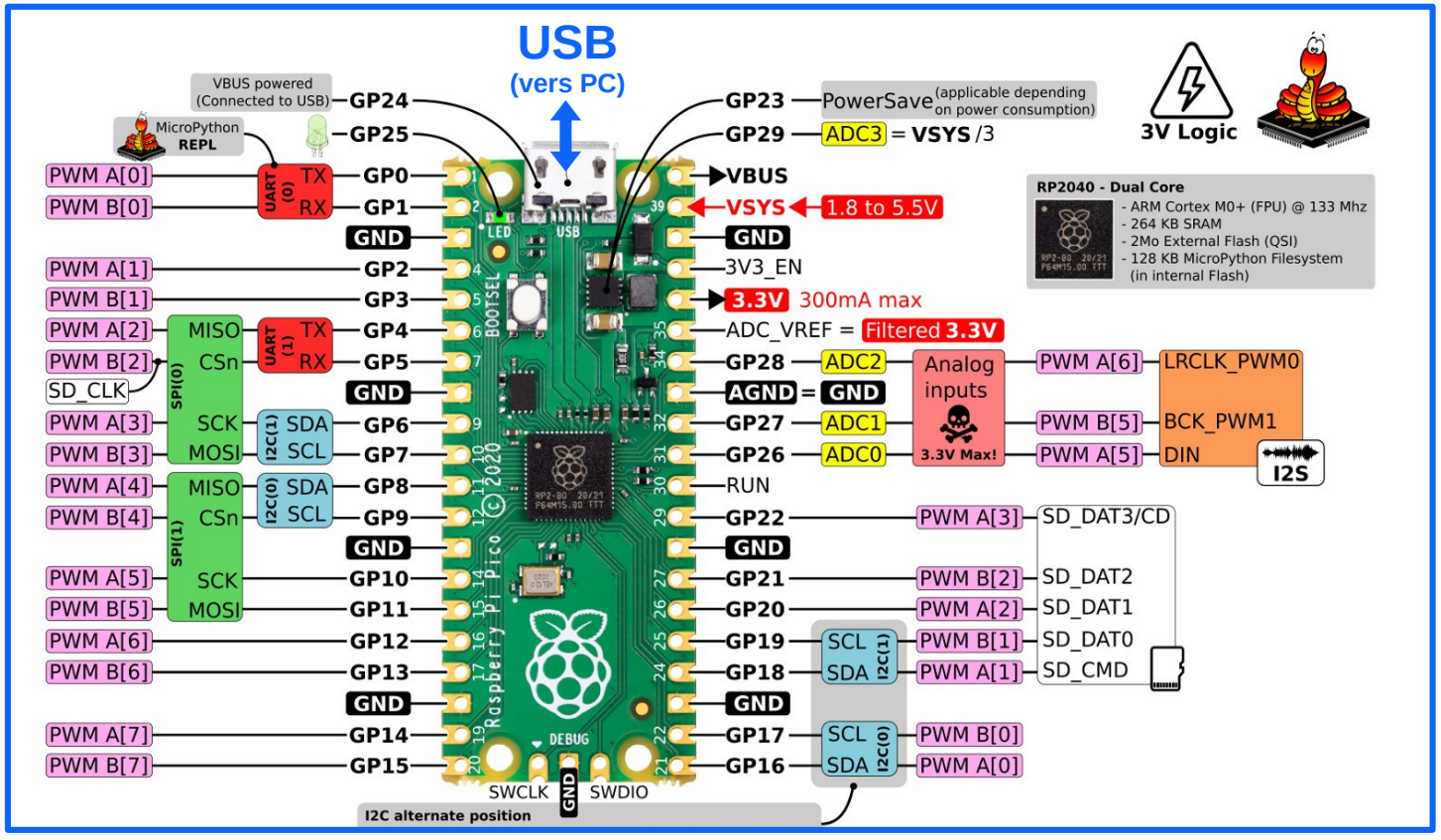
Je découvre la carte Raspberry PI PICO

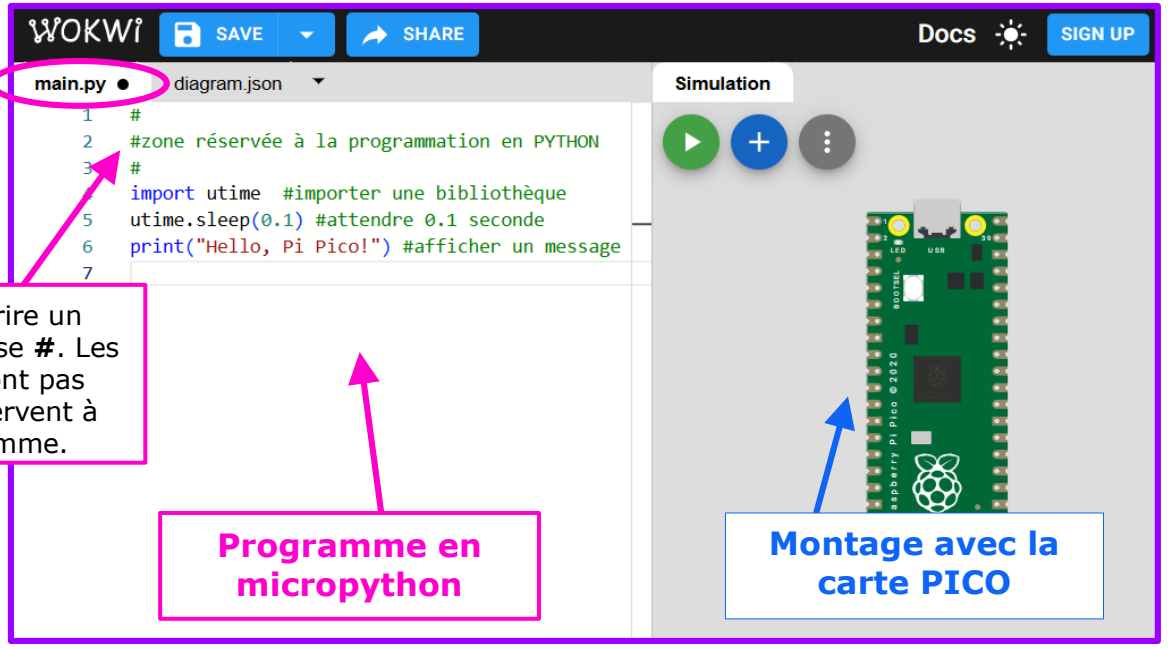
caractéristiques des **entrées/ sorties** de la **carte PICO**.



La **carte PICO** possède 26 broches **(Pin)** paramétrables comme **entrée (IN) ou sortie (OUT)**.

Je découvre l'environnement de simulation WOKWI

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



En python pour écrire un **commentaire** on utilise **#**. Les commentaires ne sont pas indispensables. Ils servent à expliquer le programme.

Programme en micropython

Montage avec la carte PICO

le **MicroPython** est une version simplifiée 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 cliquant sur ▶ on démarre la simulation

Je programme une DEL : allumer et éteindre avec des «arrêts programme»

La DEL utilise une sortie numérique

DEL n'est pas une instruction : c'est un **nom de variable** choisi par le programmeur et qui désigne ici la **sortie GP16**.

!! En python l'**indentation** (c'est à dire le retrait par rapport à la marge) doit être respectée

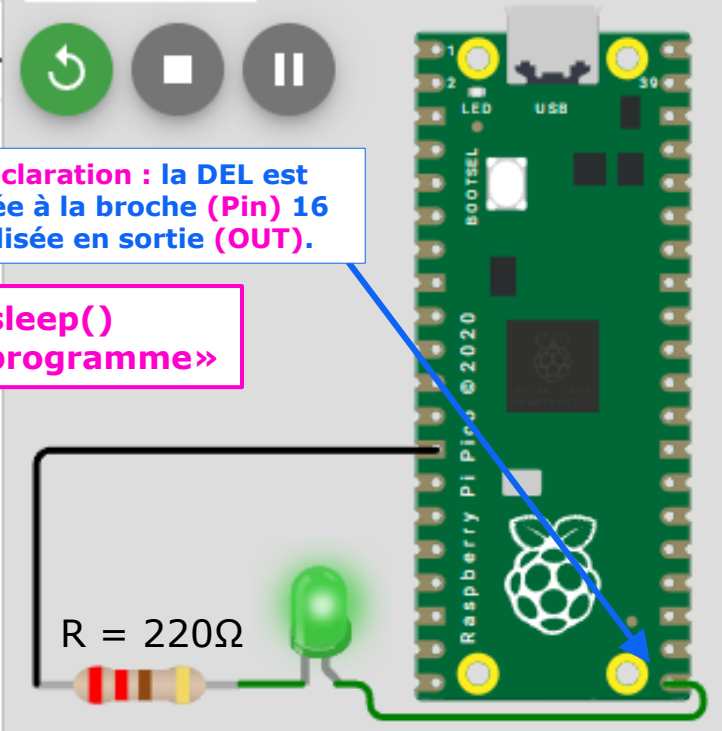
Déclaration : la DEL est reliée à la broche (Pin) 16 utilisée en sortie (OUT).

utime.sleep() «Arrêt programme»

main.py ● diagram.json ●

```
1   # Programme pour faire clignoter
2   # la DEL branchée en GP16
3   from machine import Pin
4   import utime
5   DEL=Pin(16, Pin.OUT)
6   while True:
7       DEL.value(1)
8       utime.sleep(1)
9       DEL.value(0)
10      utime.sleep(1)
```

Simulation

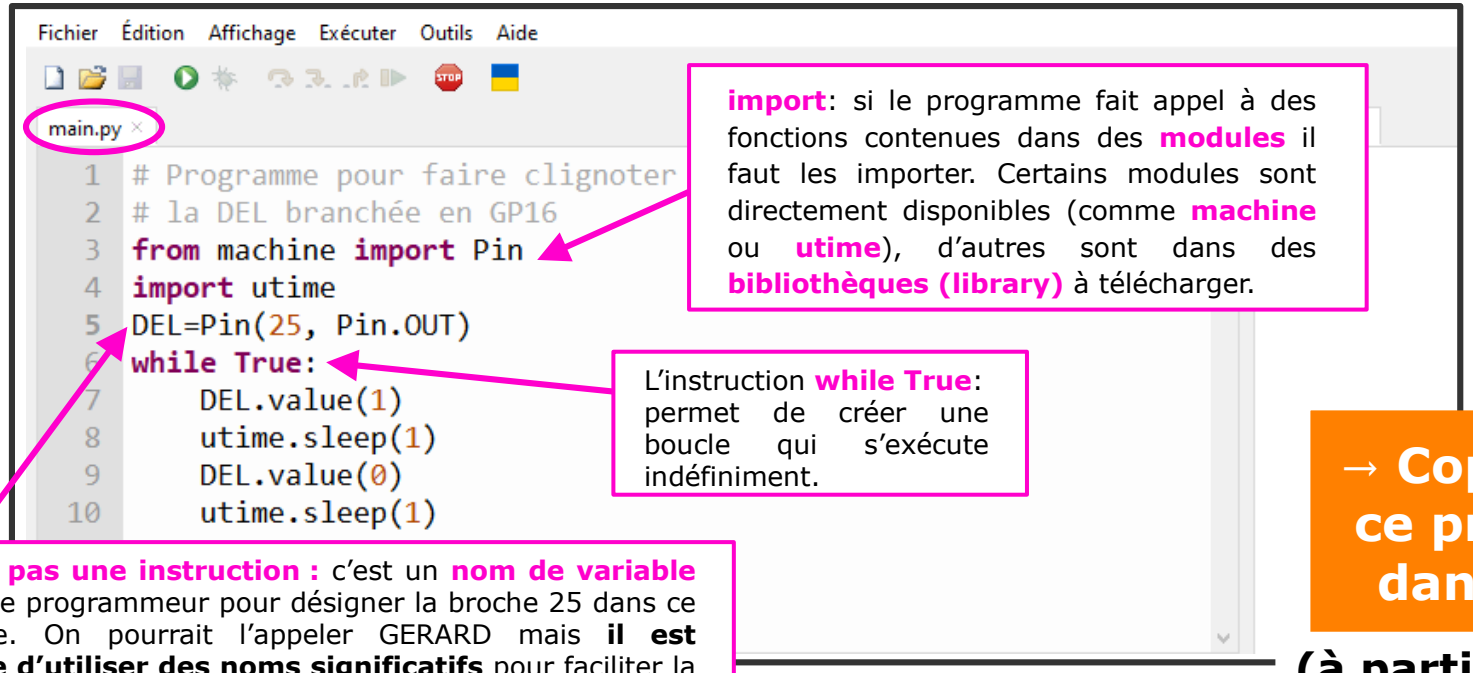


DEL.value(1) = Mettre GP16 à 1 Pour «allumer» un composant branché sur une sortie numérique on envoie un **«1» (signal «HAUT»)**: **.value(1)** . Pour l'«éteindre», on envoie un **«0» (signal «BAS»)**: **.value(0)** . On pourrait aussi utiliser les instructions **.high()** et **.low()** ou encore **.on()** et **.off()**

→ Créer et tester au point ce programme dans WOKWI

Je découvre l'environnement de développement Thonny

→ Aller à... **https://thonny.org/** et télécharger l'**IDE Thonny** (on peut télécharger une **version portable** qui ne nécessite pas d'installation)



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.

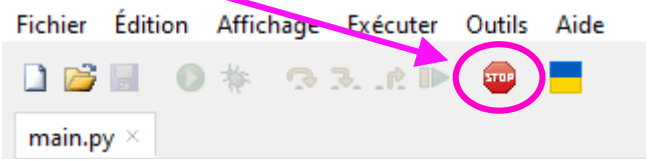
DEL n'est pas une instruction : c'est un **nom de variable** choisi par le programmeur pour désigner la broche 25 dans ce programme. On pourrait l'appeler GERARD mais **il est préférable d'utiliser des noms significatifs** pour faciliter la compréhension du programme.

→ Copier/Coller ce programme dans Thonny

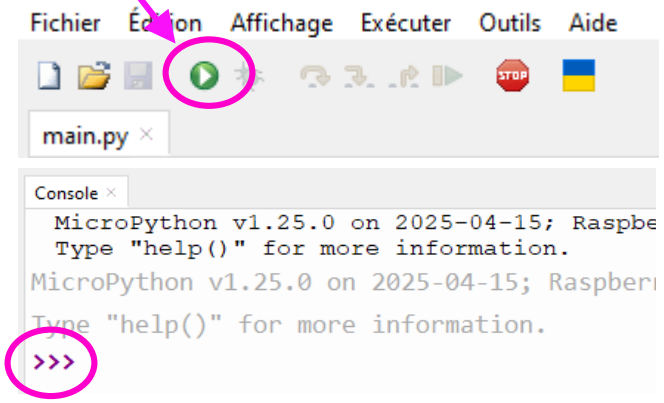
(à partir de WOKWI)

J'expérimente avec Thonny & la carte PICO

1. **brancher la carte PICO** sur un port USB
2. Dans Thonny **cliquer sur STOP**



3. **Si la flèche verte apparaît, Thonny est prêt** à exécuter un programme, la carte est reconnue



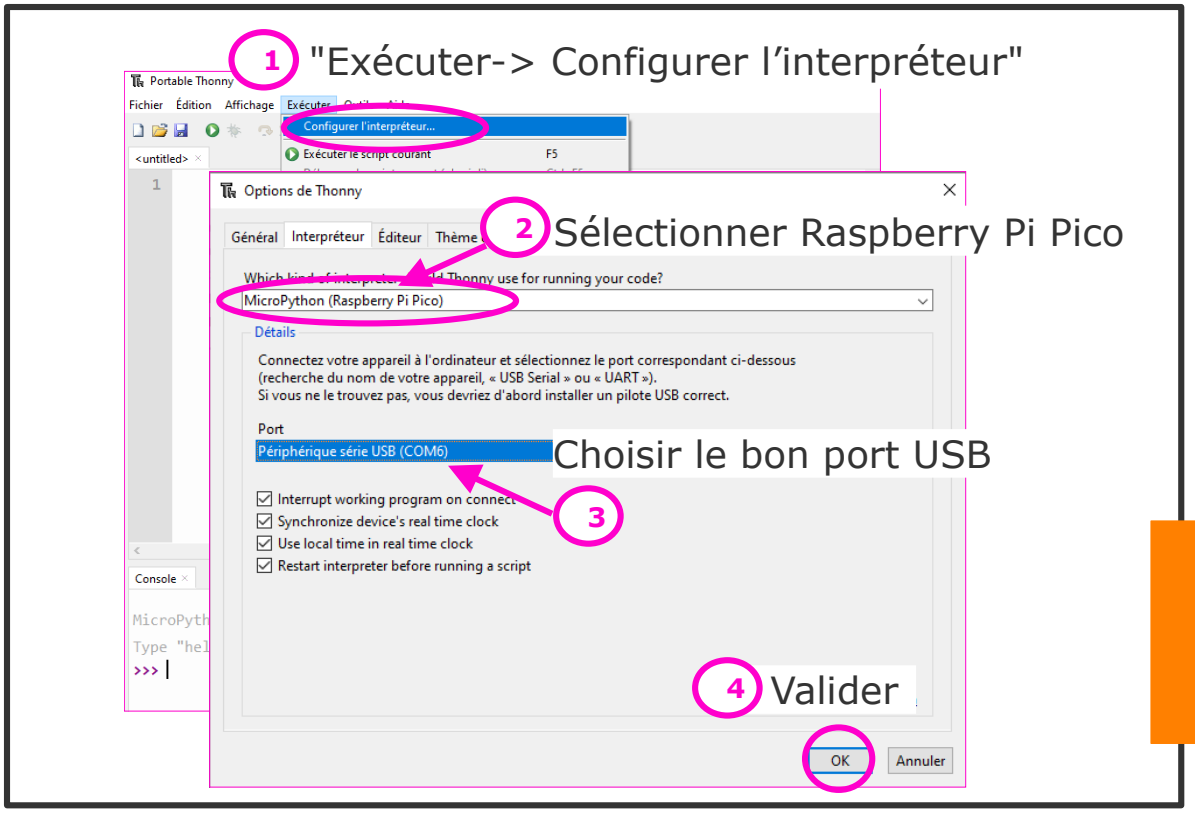
Si la carte est reconnue:

→ Exécuter le programme

! Si la carte n'est pas reconnue: voir la diapo suivante

Je résouds les problèmes d'installation...

Si la carte n'est pas reconnue:



Si la carte est reconnue:

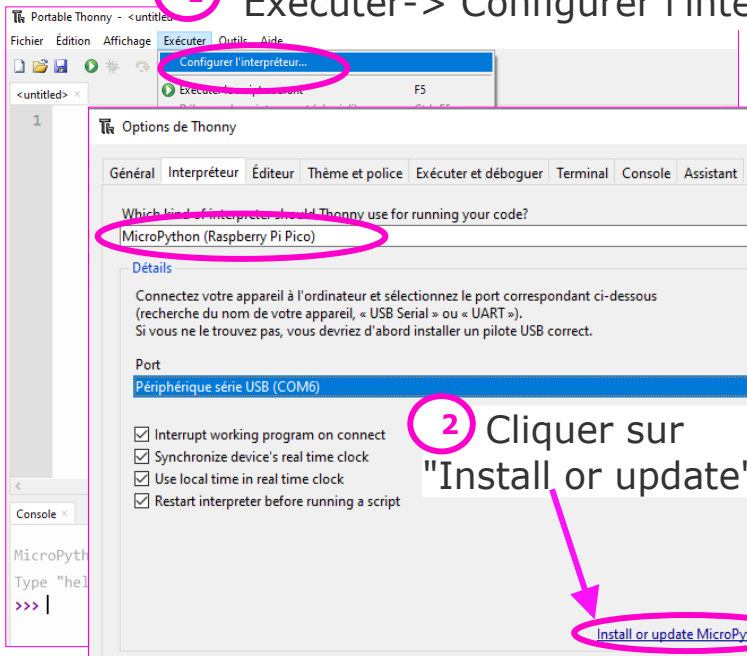
→ Exécuter le programme

! Si la carte n'est toujours pas reconnue: voir la diapo suivante

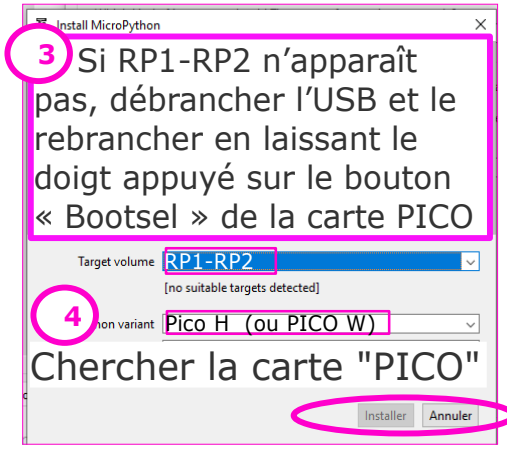
Je résouds les problèmes d'installation...

Si la carte n'est toujours pas reconnue:

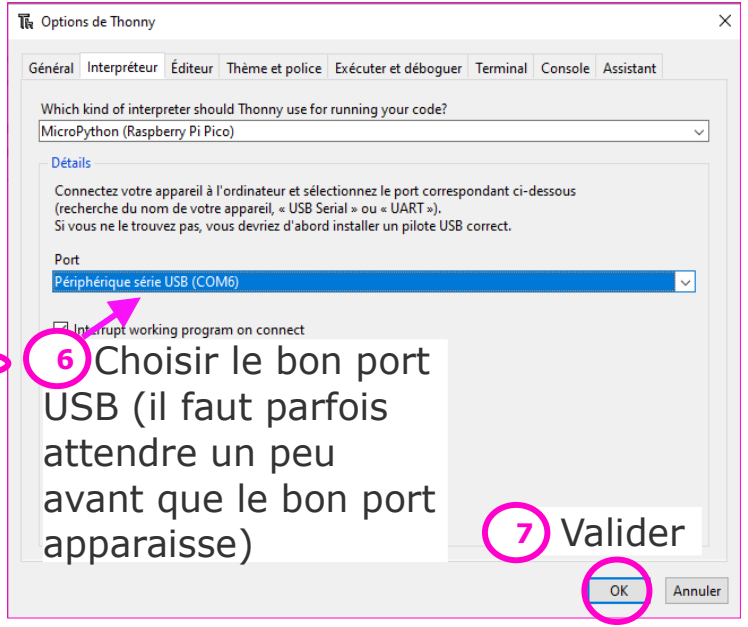
1 "Exécuter-> Configurer l'interpréteur"



2 Cliquer sur "Install or update"



5 Cliquer sur "Installer" puis "Fermer"



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

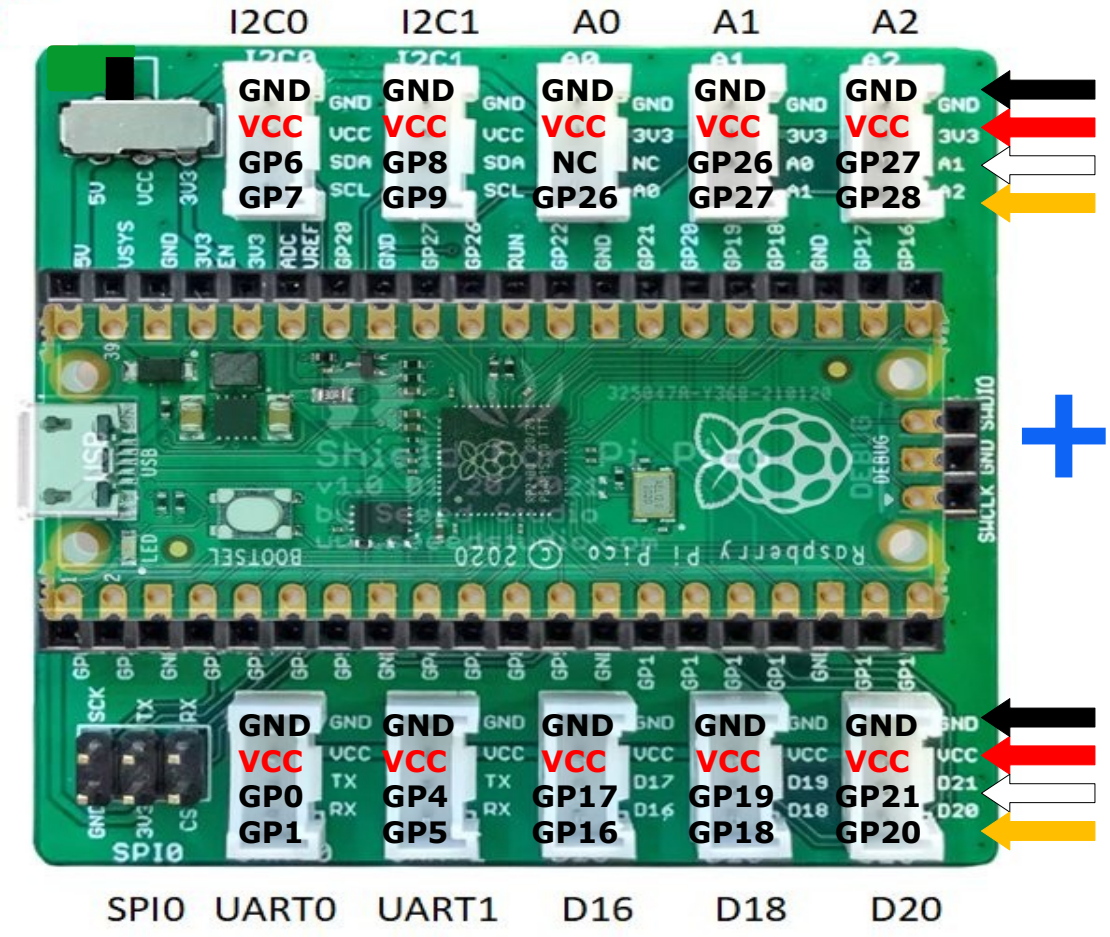
7 Valider

Si la carte est enfin reconnue:

→ Exécuter le programme

Je découvre le matériel

Le shield Grove:

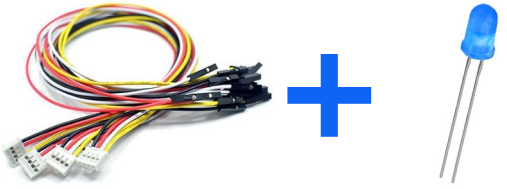


Les composants :

câbles Grove/Grove + composants Grove

On peut utiliser **10** entrées/sorties de la carte **PICO**

câbles Grove/contacts F + composants



On peut utiliser **17** entrées/sorties de la carte **PICO**

câbles contacts M/F + composants

On peut utiliser **toutes** les entrées/sorties de la carte **PICO**

Je programme une DEL : allumer et éteindre avec des CONDITIONS et des «arrêts programme»

Avec IF... ELSE et une variable

```
1   # Programme pour faire clignoter
2   # la DEL avec une IF... ELSE
3   from machine import Pin
4   import utime
5   DEL=Pin(18, Pin.OUT) #DEL branchée en GP18
6   variable = 0         #déclarer une variable
7   while True:
8       if variable == 0: #TEST: SI variable est égal à 0
9           variable = 1
10      else:             #SINON:
11          variable = 0
12      DEL.value(variable) #allumer ou éteindre la DEL
13      utime.sleep(1)      #attendre 1 seconde
```

En python on n'a pas besoin de déclarer le TYPE des variables

Condition IF... ELSE

Avec une variable booléenne

```
1   # Programme pour faire clignoter
2   # la DEL avec une variable booléenne
3   from machine import Pin
4   import utime
5   DEL=Pin(20, Pin.OUT) #DEL branchée en GP20
6   etat = False         #déclarer une variable booléenne
7   while True:
8       etat = not etat #changer la valeur de la variable
9       DEL.value(etat) #allumer ou éteindre la DEL
10      utime.sleep(1)  #attendre 1 seconde
```

Une variable booléenne peut prendre 2 valeurs : True ou False (1 ou 0)

Avec l'instruction toggle

```
1   # Programme pour faire clignoter
2   # la DEL avec l'instruction toggle
3   from machine import Pin
4   import utime
5   DEL=Pin(16, Pin.OUT) #DEL branchée en GP16
6   while True:
7       DEL.toggle()   #allumer ou éteindre la DEL
8       utime.sleep(1) #attendre 1 seconde
```

utime.sleep()
«Arrêt programme»

Avec une boucle FOR

```
1   # Programme pour faire clignoter
2   # la DEL avec une boucle FOR
3   from machine import Pin
4   import utime
5   DEL=Pin(16, Pin.OUT) #DEL branchée en GP16
6   while True:
7       for i in range(2): #POUR i parcourant 2 valeurs (de 0 à 1)
8           DEL.value(i)   #allumer ou éteindre la DEL
9           utime.sleep(1) #attendre 1 secondes
```

Boucle FOR

→ Simuler puis expérimenter

Je programme une DEL : allumer et éteindre SANS «arrêts programme»

Avec une variable pour compter les «BOUCLES»

```
1  # Programme pour faire clignoter la DEL en
2  # comptant les boucles SANS arrêt programme
3  from machine import Pin
4  import utime
5  DEL=Pin(16, Pin.OUT) #DEL branchée en GP16
6  compteur = 0  #déclarer une variable qui permettra de compter les boucles
7  while True:   #boucle qui s'exécute indéfiniment
8      compteur += 1      #à chaque boucle on incrémente la variable
9      if compteur == 1:#SI 1ère boucle: allumer la DEL
10         DEL.value(1)
11     if compteur == 5:
12         print("Pendant ce temps le programme peut")
13     if compteur == 10:#SI 10ème boucle: éteindre la DEL
14         DEL.value(0)
15     if compteur == 15:
16         print("exécuter d'autres instructions.")
17     if compteur == 20:#SI 20ème boucle: remettre le compteur à 0
18         compteur = 0
19     utime.sleep(0.1) #utile pour que la simulation fonctionne correctement
```

Une variable sert de compteur

!! En python attention à l'indentation
(c'est à dire le retrait par rapport à la marge)

Arrêt programme utile uniquement pour que la simulation **WOKWI** fonctionne correctement (pas indispensable en expérimentation)

En utilisant une fonctionde utime pour mesurer le temps

```
1  # Programme pour faire clignoter la DEL en
2  # mesurant le temps qui passe SANS arrêt programme
3  from machine import Pin
4  import utime
5  DEL=Pin(16, Pin.OUT) #DEL branchée en GP16
6  tempsAVANT = 0  #déclarer une variable pour stocker le temps présent
7  while True:     #boucle qui s'exécute indéfiniment
8      temps = utime.ticks_ms() #stocker la valeur courante du temps en ms
9      if (temps - tempsAVANT) > 1000: #TEST: SI l'intervalle de temps est > 1000ms
10         tempsAVANT = temps   #placer la valeur actuelle du temps dans tempsAVANT
11         DEL.toggle()         #allumer ou éteindre la DEL
12     utime.sleep(0.1) #utile pour que la simulation fonctionne correctement
```

Mesure du temps

Entre le moment où on allume le DEL et le moment où on l'éteint, il n'y a pas d'arrêt programme (utime.sleep): le programme n'est pas arrêté, il peut donc exécuter d'autres instructions en parallèle

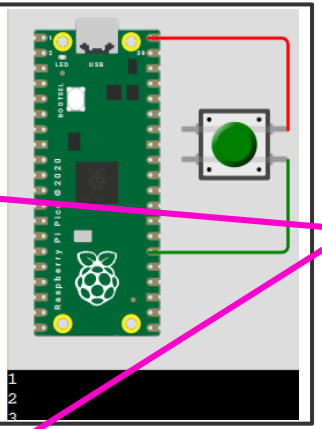
→ Simuler puis expérimenter

Je programme un bouton poussoir (Pushbutton) :

Le bouton poussoir utilise une entrée numérique

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

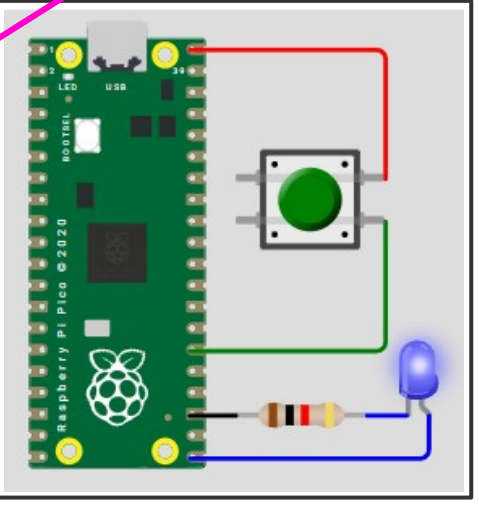
```
1   #programmer le Bouton Poussoir
2   #et compter les appuis sur BP
3   from machine import Pin
4   import utime
5   Bouton = Pin(20, Pin.IN, Pin.PULL_DOWN) #branché en GP20, forcé à 0
6   compteur = 0    #déclarer une variable qui servira de compteur
7   while True:
8       if Bouton.value() == 1: #TEST: SI appui sur BP
9           compteur += 1       #incrémenter compteur
10          print(compteur)     #afficher le contenu de compteur
11          utime.sleep(0.25)   #attendre 250ms
```



On utilise PULL_DOWN pour «forcer une entrée à 0», afin d'éviter une incertitude en entrée. On peut aussi utiliser PULL_UP pour «forcer une entrée à 1».

Exemple 2: éteindre ou allumer une DEL avec un BP

```
1   #programmer le Bouton Poussoir
2   #en interrupteur marche/arrêt
3   from machine import Pin
4   import utime
5   Bouton = Pin(20, Pin.IN, Pin.PULL_DOWN) #branché en GP20, forcé à 0
6   DEL = Pin(16, Pin.OUT)
7   marche_arret = False    #déclarer une variable booléenne
8   while True:
9       if Bouton.value(): #TEST: SI appui sur BP
10          marche_arret = not marche_arret #inverser la valeur
11          utime.sleep(0.1) #attendre 100ms pour éviter rebonds
12      if marche_arret: #TEST: SI marche_arrêt = True
13          DEL.value(1)
14      else:            #SINON
15          DEL.value(0)
16      utime.sleep(0.1)
```



→ Simuler puis expérimenter

Je programme un capteur ultrasons (HC SR04) :

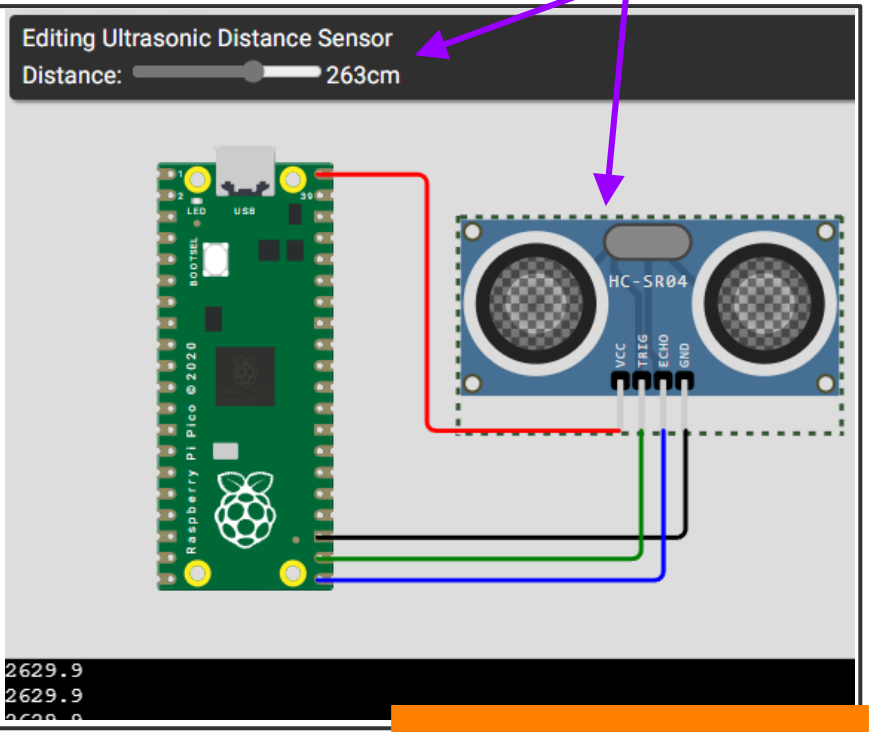
Exemple : détecter un obstacle, calculer et afficher sa distance

```
1  #programmer le capteur ultra-sons
2  #pour mesurer et afficher la distance d'un obstacle
3  from machine import Pin
4  import utime
5
6  def fonction_detection(broche_trig,broche_echo): #définir une fonction
7      trigger = Pin(broche_trig, Pin.OUT) #déclarer la broche trigger
8      trigger.value(1)            #émettre une impulsion (ultrason)
9      utime.sleep_us(2)           #attendre 2 microsecondes
10     trigger.value(0)            #arrêter d'émettre
11     echo = Pin(broche_echo, Pin.IN, Pin.PULL_DOWN) #déclarer la broche echo
12     duree = machine.time_pulse_us(echo,1) #mesurer la durée mise par le signal pour revenir(echo)
13     distance_en_mm = duree * 0.34 / 2  #calculer la distance parcourue
14     return(distance_en_mm)
15
16 while True:
17     distance = fonction_detection(17,16) #appeler la fonction, capteur US branché en 17 et 16
18     print(distance) #afficher la distance mesurée
19     utime.sleep(1)  #attendre 1s
```

Le capteur US HC-SR04 utilise 1 broche pour le trigger et 1 autre broche pour l'echo

Fonction

Cliquer sur le capteur permet de modifier la distance de l'obstacle virtuel



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
```

!! Le capteur US Grove utilise 1 SEULE ET MÊME BROCHE pour le trigger et pour l'echo. Dans le programme broche_trig et broche_echo auront donc la même valeur

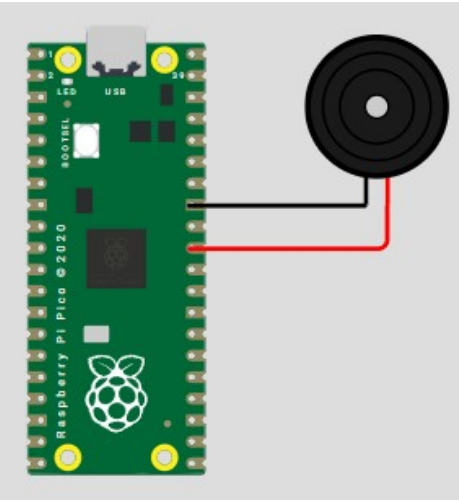
→ Simuler puis expérimenter

Je programme un buzzer :

Le buzzer utilise une sortie PWM

Exemple 1: jouer une succession de notes

```
#programmer le BUZZER
#pour émettre une série de notes
from machine import Pin, PWM
import utime
Buzzer = PWM(Pin(26)) #buzzer branché en GP26
while True:
    Buzzer.freq(440)        #jouer la note LA
    Buzzer.duty_u16(2000)   #régler le volume
    utime.sleep(1)
    Buzzer.freq(262)        #jouer la note DO
    Buzzer.duty_u16(1000)   #régler le volume
    utime.sleep(1)
    Buzzer.duty_u16(0)      #volume à 0: plus de son
    utime.sleep(1)
```



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 2: produire une succession de bips plus ou moins aigus et rapides en fonction de la distance d'un obstacle mesurée par un capteur à ultrasons (extrait du programme)

```
while True:
    distance = fonction_detection(trig1,echo1) #appeler la fonction de détection par ultrasons
    Buzzer.freq(int((4100-distance)/10)) #jouer une note inv proportionnelle à la distance
    Buzzer.duty_u16(1000)                #régler le volume
    utime.sleep_ms(100)                  #attendre 100ms (durée de la note)
    Buzzer.duty_u16(0)                   #volume à 0: plus de son
    utime.sleep_ms(int(distance/10))     #attendre une durée proportionnelle à la distance
```

→ Simuler puis expérimenter

Je programme une interruption avec un bouton poussoir

Le bouton poussoir utilise une entrée numérique

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

```
1  #programmer une INTERRUPTION
2  from machine import Pin, PWM
3  import utime
4  DEL = Pin(16, Pin.OUT)                    #DEL branchée en GP16
5  Bouton = Pin(20, Pin.IN, Pin.PULL_DOWN) #BP branché en GP20, forcé à 0
6
7  interruption = 0
8
9  def PRIORITE(Bouton):    #fonction qui sera exécutée en cas d'interruption
10     global interruption
11     interruption = 1
12
13 Bouton.irq(trigger=Pin.IRQ_RISING, handler=PRIORITE)#interruption associée à Bouton
14         #IRQ_RISING : interruption déclenchée si GP20 passe à 1
15
16 while True:
17     if interruption == 1:
18         DEL.value(0)
19         utime.sleep(2)
20         interruption = 0
21     DEL.value(1)
22     utime.sleep(0.25)
```

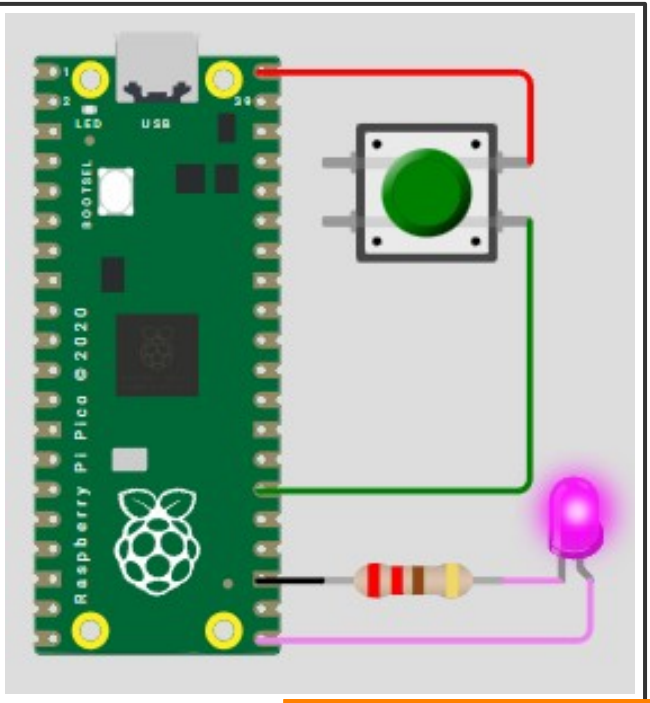
Déclaration: «Bouton» sur l'entrée 20 forcé à 0

Fonction exécutée en cas d'interruption

Paramétrage de l'interruption associée à «Bouton»

En cas d'interruption : la DEL s'éteint

Programme normal: la DEL est toujours allumée (boucle infinie)



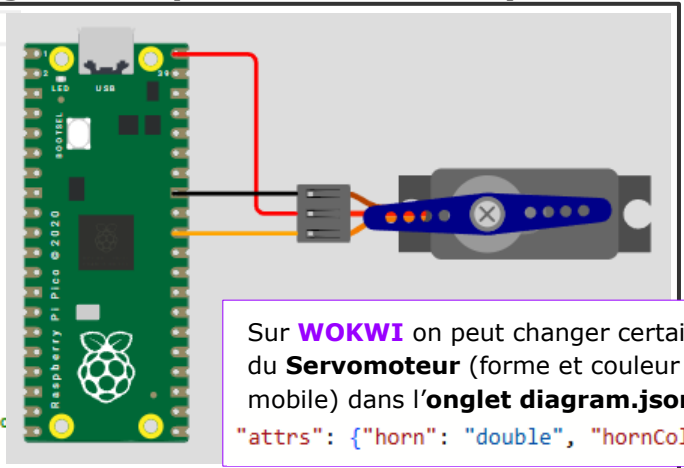
→ Simuler puis expérimenter

Je programme un Servo-moteur (angulaire):

Le Servo utilise une sortie PWM

Exemple 1: Positionner le servo en utilisant duty_u16 (largeur d'impulsion sur 16 bits)

```
#programmer un servomoteur avec une
#largeur d'impulsion sur 16 bits (duty_u16)
from machine import Pin, PWM
import utime
servomoteur = PWM(Pin(26))     #servomoteur branché en GP26
servomoteur.freq(50)  #fréquence du signal PWM fixée à 50Hz
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)
```



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

```
"attrs": {"horn": "double", "hornColor": "#000088" }
```

Exemple 2: Positionner le servo en utilisant duty_ns (largeur d'impulsion en nanosecondes)

```
#programmer un servomoteur avec une
#largeur d'impulsion en nanosecondes (duty_ns)
from machine import Pin, PWM
import utime
servomoteur = PWM(Pin(26))     #servomoteur branché en GP26
servomoteur.freq(50)  #fréquence du signal PWM fixée à 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)
```

Valeur théoriques : ces valeurs peuvent varier selon le moteur, à adapter par une série d'essais.

→ Simuler puis expérimenter

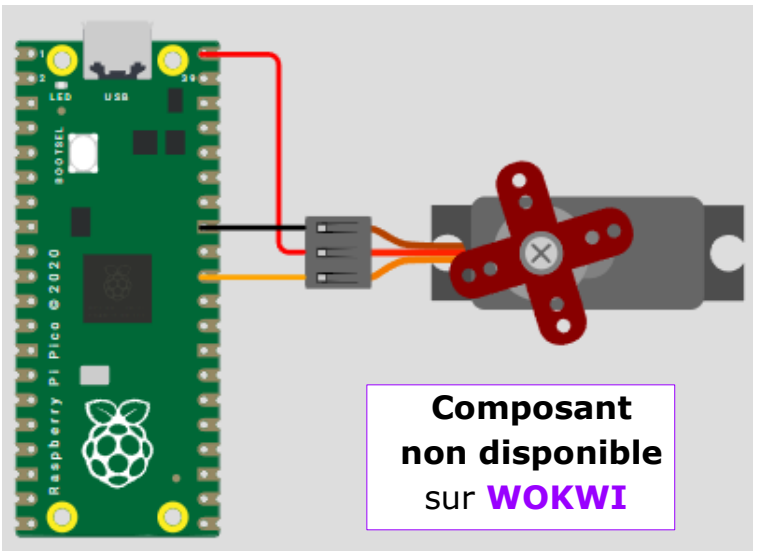
Je programme un Servo-moteur à Rotation continue :

Le Servo-R utilise une sortie PWM

On utilise **les mêmes branchements** que le montage du Servo moteur
On utilise aussi **les mêmes instructions** dans le programme, mais **les effets sont différents :** le servo moteur étant «débridé», au lieu de s'arrêter à une position angulaire, il tourne en continu, dans un sens ou dans l'autre et à une vitesse plus ou moins élevée.

```
1   #programmer un servomoteur
2   #à rotation continue
3   from machine import Pin, ADC, PWM
4   import utime
5   servomoteur = PWM(Pin(26))
6   servomoteur.freq(50)
7   while True:
8     servomoteur.duty_ns(1000000) #moteur à vitesse maximale dans le sens anti-horaire
9     utime.sleep(1)
10    servomoteur.duty_ns(1300000) #moteur à vitesse réduite dans le sens anti-horaire
11    utime.sleep(1)
12    servomoteur.duty_ns(1500000) #moteur à l'arrêt
13    utime.sleep(1)
14    servomoteur.duty_ns(1700000) #moteur à vitesse réduite dans le sens horaire
15    utime.sleep(1)
16    servomoteur.duty_ns(2000000) #moteur à vitesse maximale dans le sens horaire
17    utime.sleep(1)
```

Valeur théoriques : ces valeurs peuvent varier selon le moteur, à adapter par une série d'essais.



Composant non disponible sur WOKWI

Principe du **PWM** (Pulse Width Modulation) : 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 **durée** (ici en nanosecondes) et à une certaine **fréquence** (ici 50 Hz). 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 correspond à une tension variable comprise entre 0V et 3,3V.**

→ expérimenter

Je programme un moteur à courant continu (CC) :

Exemple1: programme test pour 1 moteur en **«Tout ou Rien»**:

Sorties numériques

```
#Programme test de fonctionnement d'un moteur
#commandé en "Tout ou Rien"
from machine import Pin
import utime
moteur1_plus  = Pin(3, Pin.OUT)
moteur1_moins = Pin(2, Pin.OUT)
while True:
    #aller en avant
    moteur1_plus(1)
    moteur1_moins(0)
    utime.sleep(1)
    #aller en arrière
    moteur1_plus(0)
    moteur1_moins(1)
    utime.sleep(1)
    #stopper
    moteur1_plus(0)
    moteur1_moins(0)
    utime.sleep(1)
```

Le moteur fonctionne en **«Tout ou Rien»** :
il est arrêté ou il est en marche.
On ne peut pas régler la vitesse.

Composants non disponibles sur **WOKWI**

Exemple2: programme test pour 1 moteur avec **vitesse réglable**

Sorties PWM

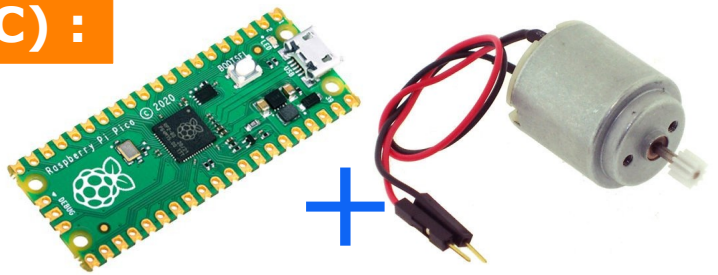
```
#Programme test de fonctionnement d'un moteur
#commandé avec vitesse variable
from machine import Pin, PWM
import utime
moteur1_plus = PWM(Pin(3))
moteur1_moins = PWM(Pin(2))
moteur1_plus.freq(50)
moteur1_moins.freq(50)
while True:
    #aller en avant à 20% de la vitesse max
    moteur1_plus.duty_u16(int(20*655.35))
    moteur1_moins.duty_u16(0)
    utime.sleep(1)
    #aller en avant à 100% de la vitesse max
    moteur1_plus.duty_u16(int(100*655.35))
    moteur1_moins.duty_u16(0)
    utime.sleep(1)
    #aller en arrière à 40% de la vitesse max
    moteur1_plus.duty_u16(0)
    moteur1_moins.duty_u16(int(40*655.35))
    utime.sleep(1)
    #stopper
    moteur1_plus.duty_u16(0)
    moteur1_moins.duty_u16(0)
    utime.sleep(5)
```

20 %

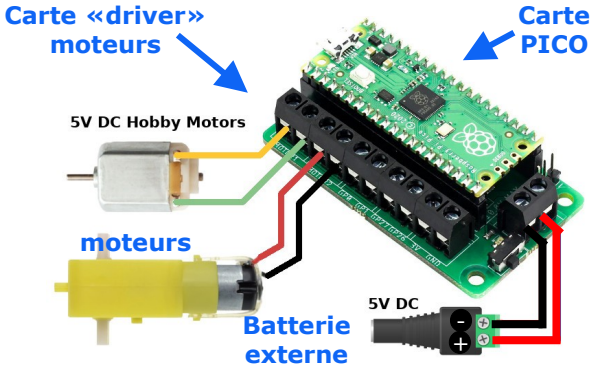
100 %

40 %

On peut régler la vitesse :
En pourcentage de la largeur maximale de l'impulsion PWM, ici donnée en valeur numérique 16 bits, c'est à dire comprise entre 0 et 65535



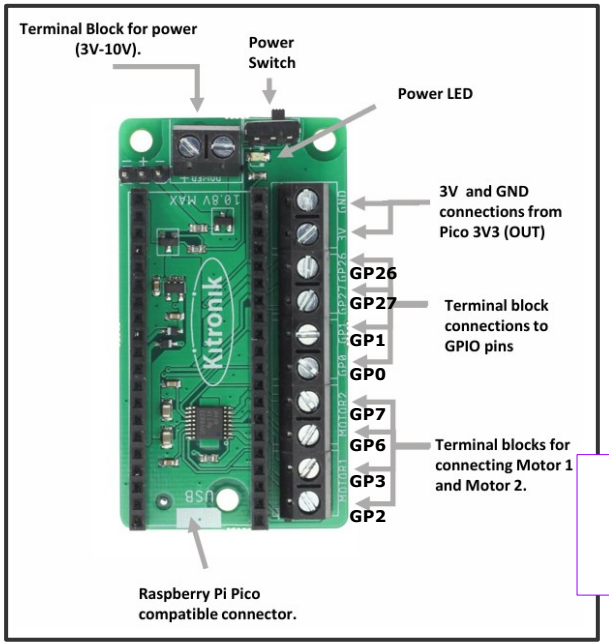
ATTENTION : sauf pour un moteur CC miniature, la **carte PICO** n'a **pas assez de puissance** pour alimenter un moteur CC.
Pour utiliser des moteurs CC, il faut associer la **carte PICO avec 1 autre carte + 1 batterie externe** dédiées à l'alimentation des moteurs.



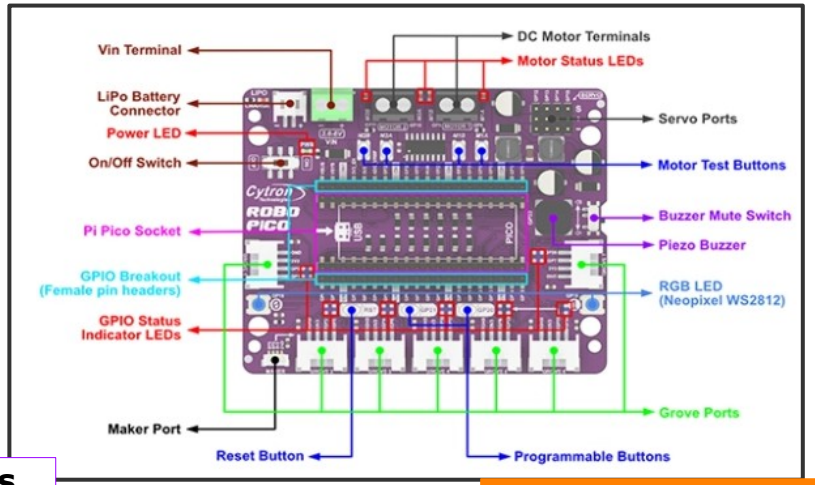
→ expérimenter

Je programme un robot :

Avec PICO + la carte «Kitronic motor driver»
Carte capable de piloter 2 moteurs CC, jusqu'à 1,5A.Permet de contrôler la vitesse et le sens de rotation des 2 moteurs indépendamment. Utilise les broches GP3(+) et GP2(-) pour commander le moteur 1 et GP6(+) et GP7(-) pour le moteur 2.
Équipée également d'un accès à 4 entrées/sorties de la PICO (GP0, GP1, GP26 et GP27), d'un interrupteur et d'accès au GND et au +VCC



Avec PICO + la carte «CYTRON Robo Pico»
Carte capable de piloter 2 moteurs CC 1A. Permet de contrôler la vitesse et le sens de rotation des 2 moteurs indépendamment. Utilise les broches GP8(+) et GP9(-) pour commander le moteur 1 et GP10(+) et GP11(-) pour le moteur 2. **Équipée également** de 4 ports pour servomoteurs (GP12, GP13, GP14 et GP15), 2 LED RVB (GP18), 2 BP programmables (GP20 et GP21), 1 buzzer (GP22) et 7 connecteurs pour modules Grove.

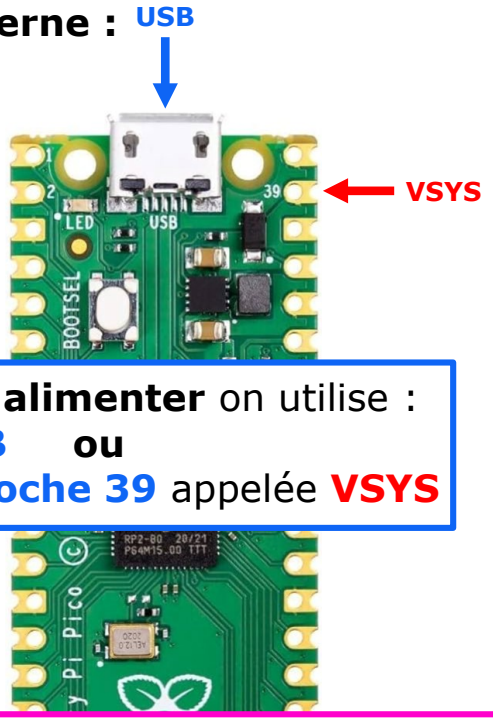


Composants non disponibles sur **WOKWI**

→ expérimenter

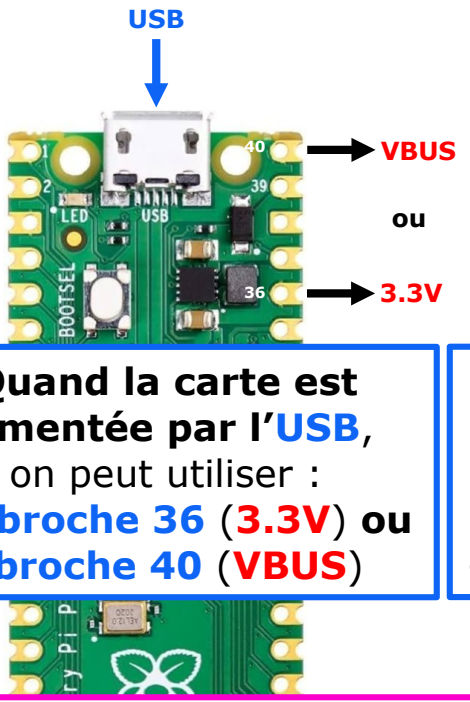
J'utilise la carte PICO avec une batterie externe

On peut alimenter la carte PICO à partir de l'ordinateur ou avec une batterie externe :

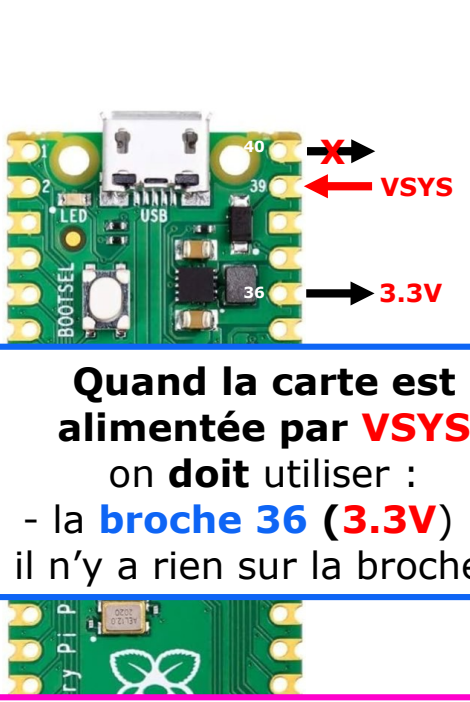


Pour alimenter on utilise :
- l'**USB** **ou**
- la **broche 39** appelée **VSYS**

Pour alimenter les composants externes (capteurs, moteurs…) il y a 2 cas :



Quand la carte est alimentée par l'USB,
on peut utiliser :
- la **broche 36** (**3.3V**) **ou**
- la **broche 40** (**VBUS**)



Quand la carte est alimentée par VSYS
on **doit** utiliser :
- la **broche 36** (**3.3V**)
- il n'y a rien sur la broche 40

ATTENTION : pour que la **carte PICO** exécute mon programme quand elle n'est plus connectée avec l'USB de l'ordinateur, il faut **enregistrer le programme dans la carte sous le nom** :
main.py

Je débogue (ou debug) mon programme

Quand la **carte PICO est branchée à l'USB** de l'ordinateur, la fonction **print()** permet d'**afficher dans la console** un repère ou le contenu d'une variable, pour repérer, comprendre et déboguer le fonctionnement d'un programme

Instructions dans le programme → **Résultats dans la console**

```
variable = 3.1416
print(variable) #afficher le contenu d'une variable
print(type(variable)) #afficher le type d'une variable
print(int(variable)) #afficher la partie entière d'une variable
print("Hello world !") #afficher un texte
variable = 95
print("Hello", variable, "!") #afficher plusieurs éléments
print("Hello\n", variable) #\n pour un retour ligne
print(bin(variable)) #afficher la valeur en binaire
print(hex(variable)) #afficher la valeur en hexadécimal
```

```
3.1416
<class 'float'>
3
Hello world !
Hello 95 !
Hello
 95
0b1011111
0x5f
```

Quand la **carte PICO n'est plus branchée à l'USB** de l'ordinateur, on peut aussi avoir besoin d'accéder à certaines données, pour déboguer le programme par exemple.
À défaut de pouvoir les afficher, on peut les **écrire dans un fichier**, qui sera consultable après, en utilisant les fonctions **open()**, **close()**, **read()** et **write()**

Instructions dans le programme → **Résultats dans le fichier**

```
variable = 95
monfichier=open("test.txt", 'w')#ouverture en mode écriture (avec effacement)
monfichier.write(str(variable))
monfichier.write('\n') #saut de ligne
monfichier.close()

monfichier=open("test.txt", 'a')#ouverture en mode écriture (sans effacement)
monfichier.write('Hello')
monfichier.write(" Bonjour\n")
monfichier.write("Hello " + str(95) + " Bonjour !")
monfichier.close()

monfichier=open("test.txt",'r')#ouverture en mode lecture
contenu_de_mon_fichier=monfichier.read()
monfichier.close()
print(contenu_de_mon_fichier)
```

```
95
Hello Bonjour
Hello 95 Bonjour !
```