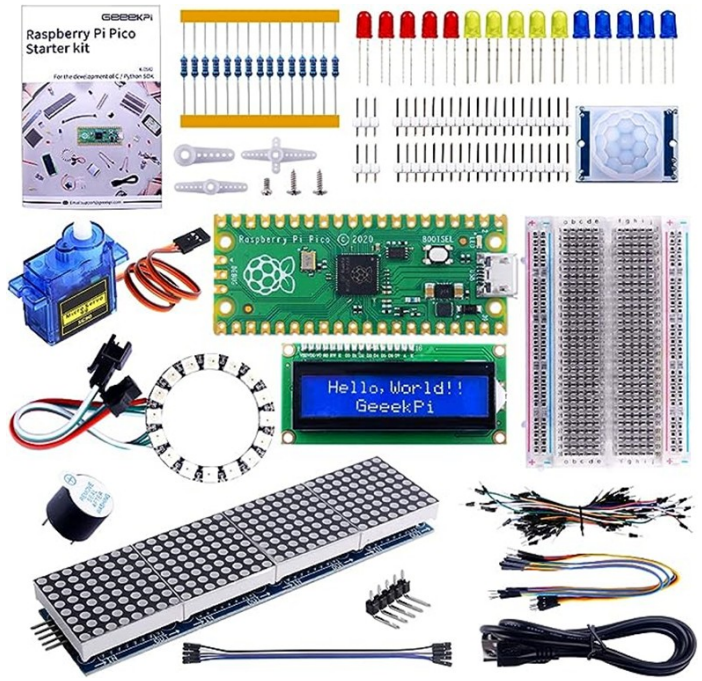
Atelier N°15 La Mini Serre
Le servo moteur

Atelier Raspi

Logo du Raspberry Pico

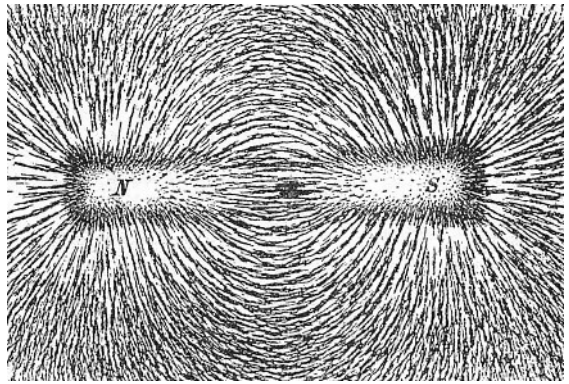
Logo du MicroPython



L’atelier a pour valeurs, le partage, l’aide, la formation, le faire et construire ensemble à partir de l’expérience des participants

Atelier N°15 La Mini Serre
Le servo moteur

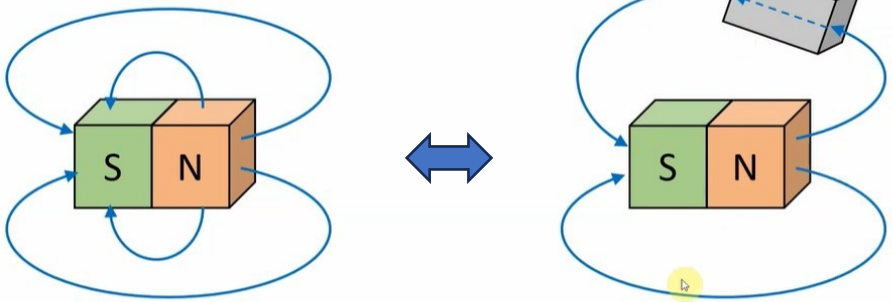
A15E1 Le champ magnétique



Superposition de:
limaille de fer,
papier,
aimant

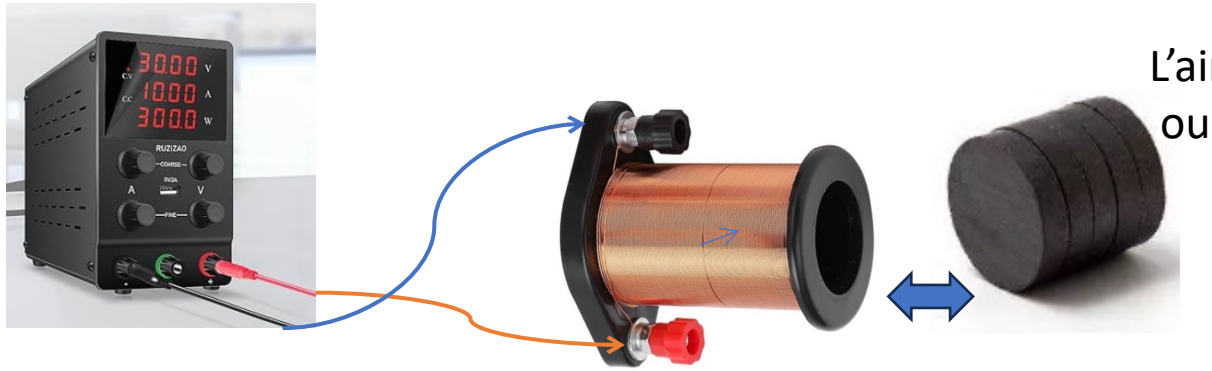
Visualisation du champ magnétique crée par l'aimant

Le fer doux guide les lignes du champs magnétique et est attiré par l'aimant

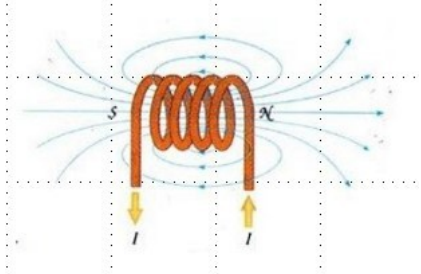


Des lignes de champ forme un spectre magnétique autour de l'aimant.
Elles se déplacent du nord au sud à l'extérieur de l'aimant.
Elles sortent et entrent à angle droit.

Un matériaux magnétique placé dans le champ magnétique va dévier les lignes de champ. Elles prennent le trajet qui présente le moins d'opposition à leur passage (réluctance).



L'aimant est attiré
ou repoussé par la bobine



Visualisation du champ magnétique crée par la bobine

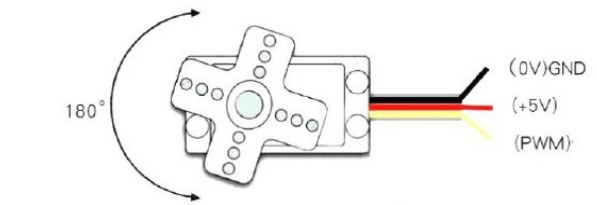
C'est la circulation du courant dans la bobine qui crée le champ magnétique

Servo-moteurs

- Un **servomoteur** (vient du latin *servus* qui signifie « esclave ») est un moteur capable de maintenir une opposition à un effort statique et dont la position est vérifiée en continu et corrigée en fonction de la mesure. C'est donc un système **asservi.**

C’est un ensemble mécanique et électronique comprenant :

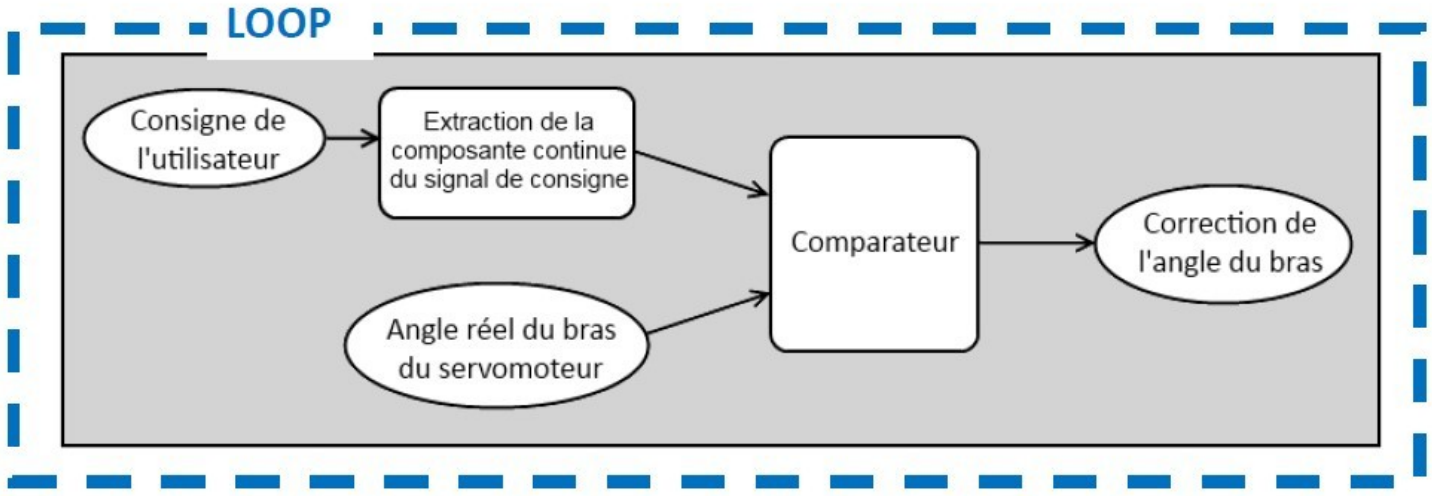
- un moteur à courant continu
- un réducteur en sortie de ce moteur diminuant la vitesse mais augmentant le couple ;
- un potentiomètre (faisant fonction de diviseur résistif) qui génère une tension variable, proportionnelle à l'angle de l'axe de sortie ;
- un dispositif électronique d'asservissement ;
- un axe dépassant hors du boîtier avec différents bras ou roues de fixation.

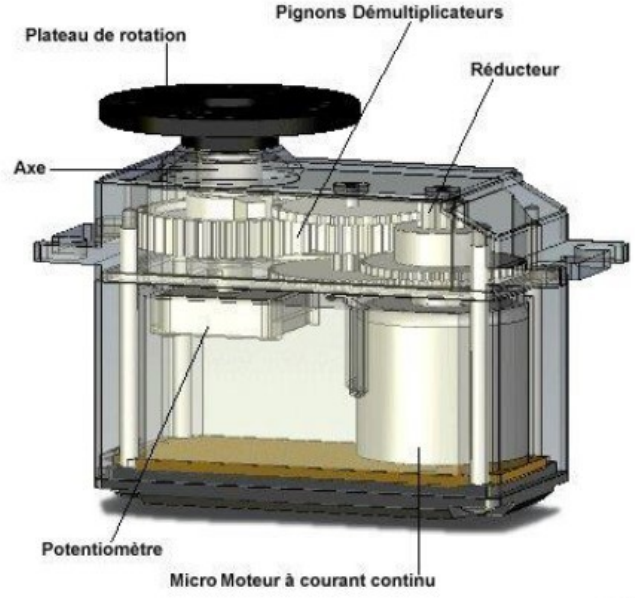


Les servos peuvent actionner les parties mobiles d ’un robot, drone, etc

Servo-moteurs : Asservissement ?

- C'est un moyen de gérer corriger une commande en fonction d'une consigne et d'un capteur de position
- Par exemple pour un servomoteur : on lui envoie un signal de commande qui défini l'angle désiré (consigne), et le moteur va corriger sans angle de départ pour tendre vers la consigne de l'utilisateur. Et il réalise cette opération en boucle.
- Pour pouvoir réaliser la correction de l'ange du bras, le servo utilise une électronique d'asservissement. Cette électronique est constituée d'un comparateur qui compare la position du bras du servo à la consigne.
- La position du bras est obtenue grâce à un potentiomètre couplé à l'axe du moteur.
- Après une rapide comparaison entre la consigne et valeur réelle de position du bras, le servomoteur (du moins son électronique de commande) va appliquer une correction si le bras n'est pas orienté à l'angle imposé par la consigne.



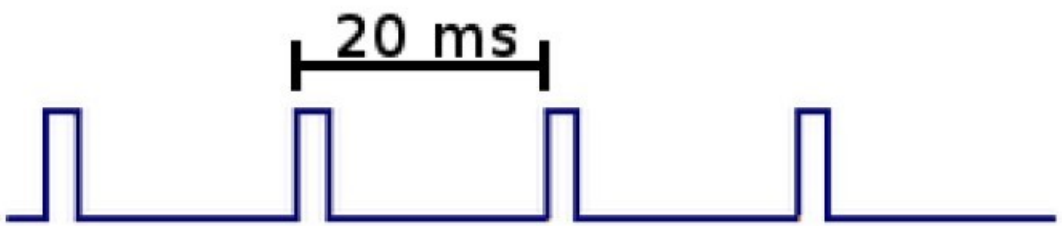


Servo-moteurs : Signal de commande (1/2)

La consigne envoyée au servomoteur est un signal électronique de type PWM. Il dispose cependant de deux caractéristiques indispensables pour que le servo puisse fonctionner : la fréquence fixe et la durée de l'état haut

LA FRÉQUENCE FIXE

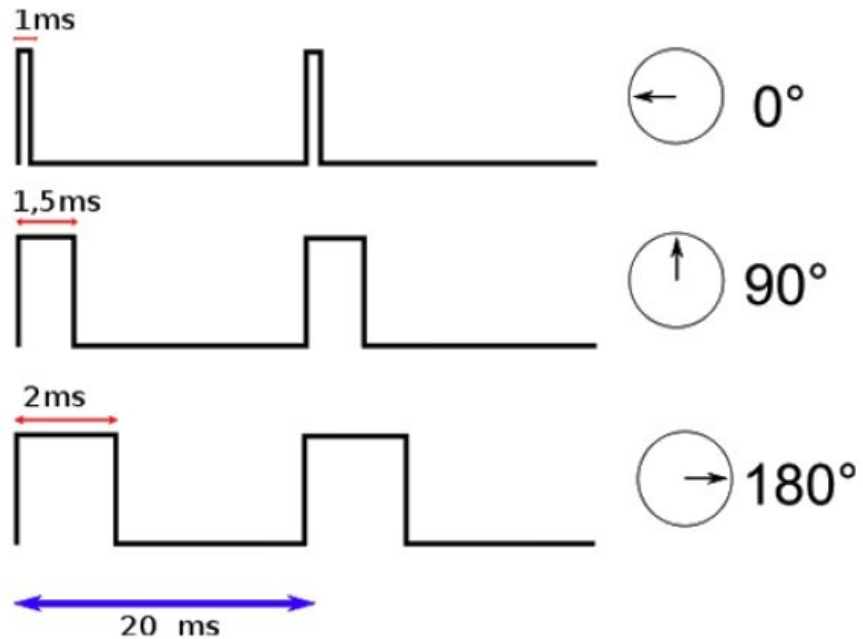
- Le signal que nous allons devoir générer doit avoir une fréquence de 50 Hz. Autrement dit, le temps séparant deux fronts montants est de 20 ms.
- La fonction analogWrite() de Arduino ne pourra donc pas utiliser cette fonction car on ne peut pas régler la fréquence



Servo-moteurs : Signal de commande (2/2)

LA DURÉE DE L'ÉTAT HAUT

- Cette durée indique au servomoteur l'angle précis qui est souhaité par l'utilisateur.
- Un signal ayant une durée d'état HAUT de 1ms donnera un angle à 0°, le même signal avec une durée d'état HAUT de 2ms donnera un angle au maximum de ce que peut admettre le servomoteur.



A3E6 Les fonctions du module PWM

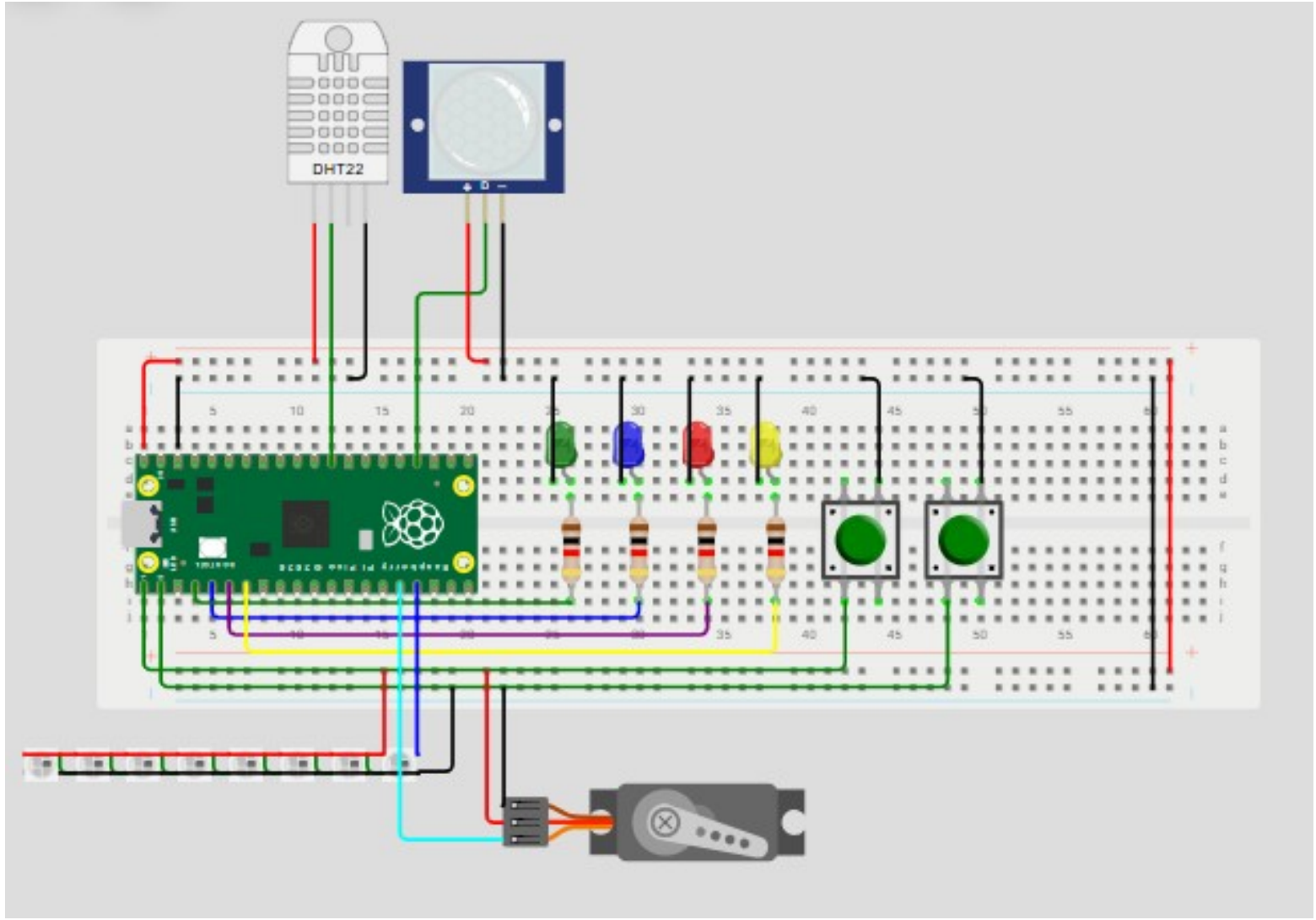
Fonctions utilisées:

servo **= PWM(Pin(12))** # servo est un objet, initialise le Pin GPIO 12 en sortie PWM
servo.**freq(50)** # Initialise la fréquence des signaux carrés à 50Hz soit 20ms
servo**.duty_u16(int(newDuty))** # duty_u16 est le rapport cyclique entre le temps du signal carré haut sur sa période, new duty est une variable, le max est à 65535.

servo**.deinit()** # réinitialise le PWM

servoX**.duty_ns(**pulseDX) #commande le rapport cyclique du PWM en nanosecondes suivant la variable **pulseDX** de l'objet **servoX**

A14E7 Le Montage

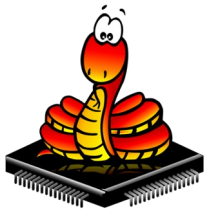


Composant	GPIO
Led Verte	2
Led Bleue	3
Led Rouge	4
Led Jaune	5
DHT11/ DHT22	22
PIR	18
Ruban Neopixel	13
BP1	0
BP2	1
Servo	**12**

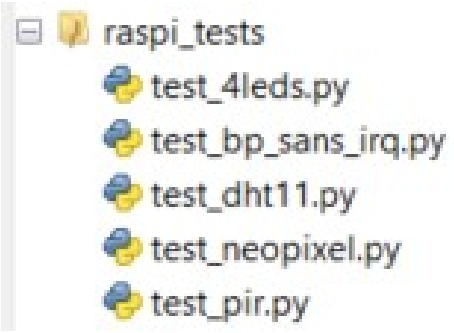
Code couleur des câbles :

- Vcc : Rouge / Orange
- Gnd : Noir / Marron
- Données : Vert / Bleu / Jaune / Violet

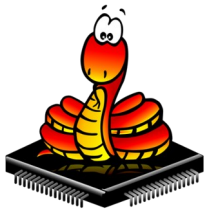
A14E8 Bibliothèque de code : les tests

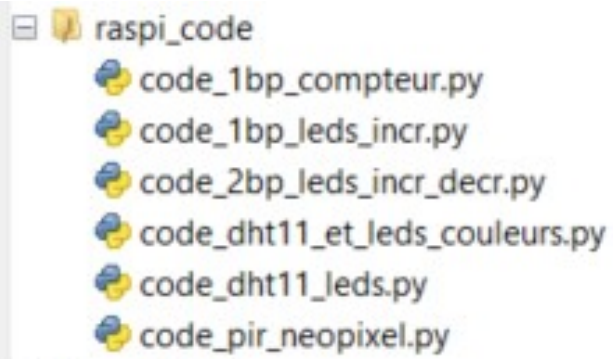


Tests	**Résultat attendu**
test_4leds	Les leds s'allument les unes après les autres
test_dht11	Affiche la température dans la console
test_pir	Affiche un message dans la console
test_neopixel	Allume le ruban de differentes couleurs
test_bp_sans_irq	Indique l'état du bouton dans la console : appuyé ou relaché



A14E9 Bibliothèque de code : les codes plus complexes





Code	**Résultat attendu**
code_dht11_leds	Affiche la température sur les leds
code_dht11_et_leds_couleur	Affiche la température en utilisant la couleur des leds
code_pir_neopixel	Allume le ruban quand une présence est détectée
code_1bp_compteur	Incrémente un compteur à chaque appui sur le bouton
code_1bp_leds_incr	Allume une led supplémentaire à chaque appui sur BP1
code_2bp_leds_incr_decr	Allume une led à chaque appui sur BP1, éteint une led à chaque appui sur BP2

A14E10 Le programme de test du Servo Moteur

```python
from machine import Pin, PWM
from time import sleep

servo = PWM(Pin(12))   # fil jaune sur GPIO 12
servo.freq(50)

while True:
    servo.duty_u16(2000)  # gauche - angle 0 degrés
    sleep(1)

    servo.duty_u16(5000)  # milieu - angle 90 degrés
    sleep(1)

    servo.duty_u16(8000)  # droite - angle 180 degrés
    sleep(1)
```

✅ **50 Hz = le bon rythme**

Les servos classiques (SG90, MG90...) sont faits pour :

> ❤️‍🔥 **1 message toutes les 20 ms**

Donc :

```python
servo.freq(50)
```

```
0° -------- 90° -------- 180°
2000       5000        8000
```

Quand on écrit :

```python
servo.duty_u16(5000)
```

👉 on demande au servo d'aller à **90 degrés.**

🔢 **Pourquoi 5000 = 90° ?**

Parce que :

- 2000 ≈ 0°
- 8000 ≈ 180°

Le milieu, c'est :

```yaml
(2000 + 8000) ÷ 2 = 5000
```

A14E11 Servo Moteur et Bouton Poussoir

Chaque appui sur le bouton poussoir 1 augmente l'angle du servo moteur

```
from machine import Pin, PWM
import time

#Declaration du bouton
bouton = Pin(0, machine.Pin.IN, Pin.PULL_UP)

#Declaration du servo
servo = PWM(Pin(12))
servo.freq(50)
servo.duty_u16(2000)

#Declaration d'une variable compteur
compteur = 2000

while True:
    #On regarde l'etat du bouton
    if bouton.value() == 0:
        compteur = compteur + 1000
        print("Compteur = ",compteur)

    #On verifie et on corrige si besoin
    #les valeurs du compteur
    if (compteur > 8000):
        compteur = 8000

    #On envoie la valeur au servo
    servo.duty_u16(compteur)

    #On attends un peu (pour éviter les rebonds du BP)
    time.sleep(0.3)

```

A14E12 Servo Moteur et 2 Boutons Poussoirs

Chaque appui sur le bouton poussoir 1 augmente l'angle du servo moteur
Chaque appui sur le bouton poussoir 2 diminue l'angle du servo moteur

```
from machine import Pin, PWM
import time

#Declaration du bouton
bouton = Pin(0, machine.Pin.IN, Pin.PULL_UP)
bouton2 = Pin(1, machine.Pin.IN, Pin.PULL_UP)

#Declaration du servo
servo = PWM(Pin(12))
servo.freq(50)
servo.duty_u16(2000)

#Declaration d'une variable compteur
compteur = 2000

while True:
    #On regarde l'etat du bouton 1
    if bouton.value() == 0:
        compteur = compteur + 1000
        print("Compteur = ",compteur)

    #On regarde l'etat du bouton 2
    if bouton2.value() == 0:
        compteur = compteur - 1000
        print("Compteur = ",compteur)

    #On verifie et on corrige si besoin
    #les valeurs du compteur
    if (compteur > 8000):
        compteur = 8000
    if (compteur < 2000):
        compteur = 2000

    #On envoie la valeur au servo
    servo.duty_u16(compteur)

    #On attends un peu (pour éviter les rebonds du BP)
    time.sleep(0.3)

```

A14E13 Servo Moteur, PIR et LEDs

<u>Propositions de tests pour aller plus loin :</u>

Ecrire un test avec les fonctionnalités suivantes :
- le PIR détecte une présence, la fenêtre de la serre s'ouvre
- le PIT ne détecte personne, la fenêtre de la serre se ferme

Ecrire un test, qui ouvre la fenetre en fonction de la température intérieure

Ecrire un test qui indique l'angle d'ouverture du servo sur les leds

Prog 1

```python
from machine import Pin, PWM
from time import sleep

servo = PWM(Pin(12))   # fil jaune sur GPIO 12
servo.freq(50)

while True:
    servo.duty_u16(2000)  # gauche - angle 0 degrés
    sleep(1)

    servo.duty_u16(5000)  # milieu - angle 90 degrés
    sleep(1)

    servo.duty_u16(8000)  # droite - angle 180 degrés
    sleep(1)
```

Prog 2

```python
from machine import Pin, PWM
import time

#Declaration du bouton
bouton = Pin(0, machine.Pin.IN, Pin.PULL_UP)

#Declaration du servo
servo = PWM(Pin(12))
servo.freq(50)
servo.duty_u16(2000)

#Declaration d'une variable compteur
compteur = 2000

while True:
    #On regarde l'etat du bouton
    if bouton.value() == 0:
        compteur = compteur + 1000
        print("Compteur = ",compteur)

    #On verifie et on corrige si besoin
    #les valeurs du compteur
    if (compteur > 8000):
        compteur = 8000

    #On envoie la valeur au servo
    servo.duty_u16(compteur)

    #On attends un peu (pour éviter les rebonds du BP)
    time.sleep(0.3)

```

Prog 3

```python
from machine import Pin, PWM
import time

#Declaration du bouton
bouton = Pin(0, machine.Pin.IN, Pin.PULL_UP)
bouton2 = Pin(1, machine.Pin.IN, Pin.PULL_UP)

#Declaration du servo
servo = PWM(Pin(12))
servo.freq(50)
servo.duty_u16(2000)

#Declaration d'une variable compteur
compteur = 2000

while True:
    #On regarde l'etat du bouton 1
    if bouton.value() == 0:
        compteur = compteur + 1000
        print("Compteur = ",compteur)

    #On regarde l'etat du bouton 2
    if bouton2.value() == 0:
        compteur = compteur - 1000
        print("Compteur = ",compteur)

    #On verifie et on corrige si besoin
    #les valeurs du compteur
    if (compteur > 8000):
        compteur = 8000
    if (compteur < 2000):
        compteur = 2000

    #On envoie la valeur au servo
    servo.duty_u16(compteur)

    #On attends un peu (pour éviter les rebonds du BP)
    time.sleep(0.3)

```