# Servo Motors

## What is our GOAL for this CLASS?

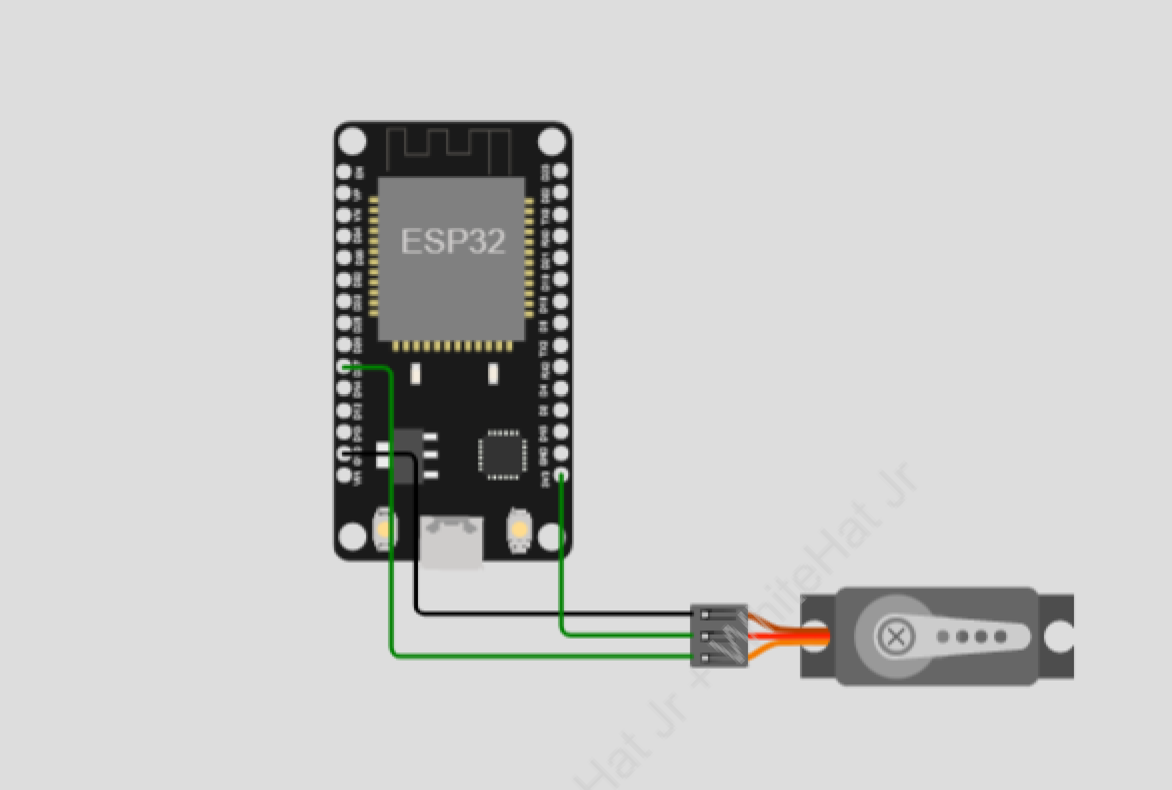
In this class, we learned how to use a servo Motor. We learned how to control the Servo Motor.

## What did we ACHIEVE in the class TODAY?

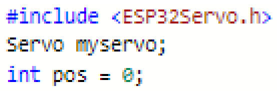
- We learned about servo Motors.
- We learned about Serov working.
- We learned about the Slide potentiometer.

## How did we DO the activities?

1. Gather the material from the IoT Simulator
   - 1 x ESP32
   - 1 x Servo
2. Do connections:
   - **Servo VCC** pin is connected to **ESP32 GPIO VIN**
   - **Servo GND** pin is connected to **ESP32 GND PIN**
   - **Servo PWM** pin is connected to the ESP32 **GPIO D27** PIN.



3. Go to the **sketch.ino**, delete the entire code, and start writing our new code.
    - Include the servo library to access the servo application.
    - Include the **ESP32 Servo** library in libraries.txt
    - Create **myservo** object
    - Define one variable, name them **pos** and assign a value.

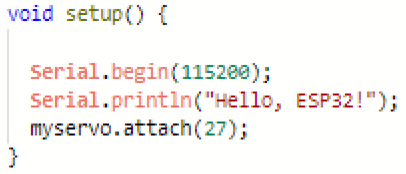


```
#include <ESP32Servo.h>
Servo myservo;
int pos = 0;
```

4. Initialize using **void setup()** function

- **Serial. begin(115200)** is used to measure the speed of data exchange. This tells the Arduino to get ready to exchange messages with the Serial Monitor at a data rate of 9600 bits per second. That's 9600 binary ones or zeros per second and is commonly called a baud rate.

- servo.attach(pin)
    - *servo*: a variable of type **Servo**
    - *pin*: the number of the pin that the servo is attached

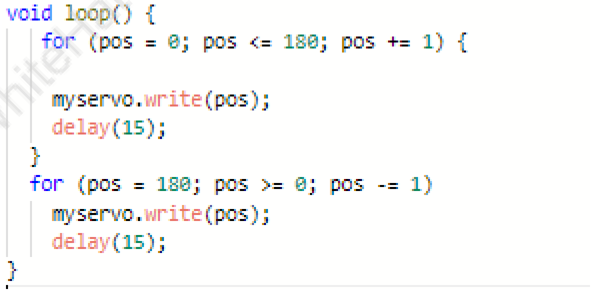


```
void setup() {

  Serial.begin(115200);
  Serial.println("Hello, ESP32!");
  myservo.attach(27);
}
```

### Execution of the main process:

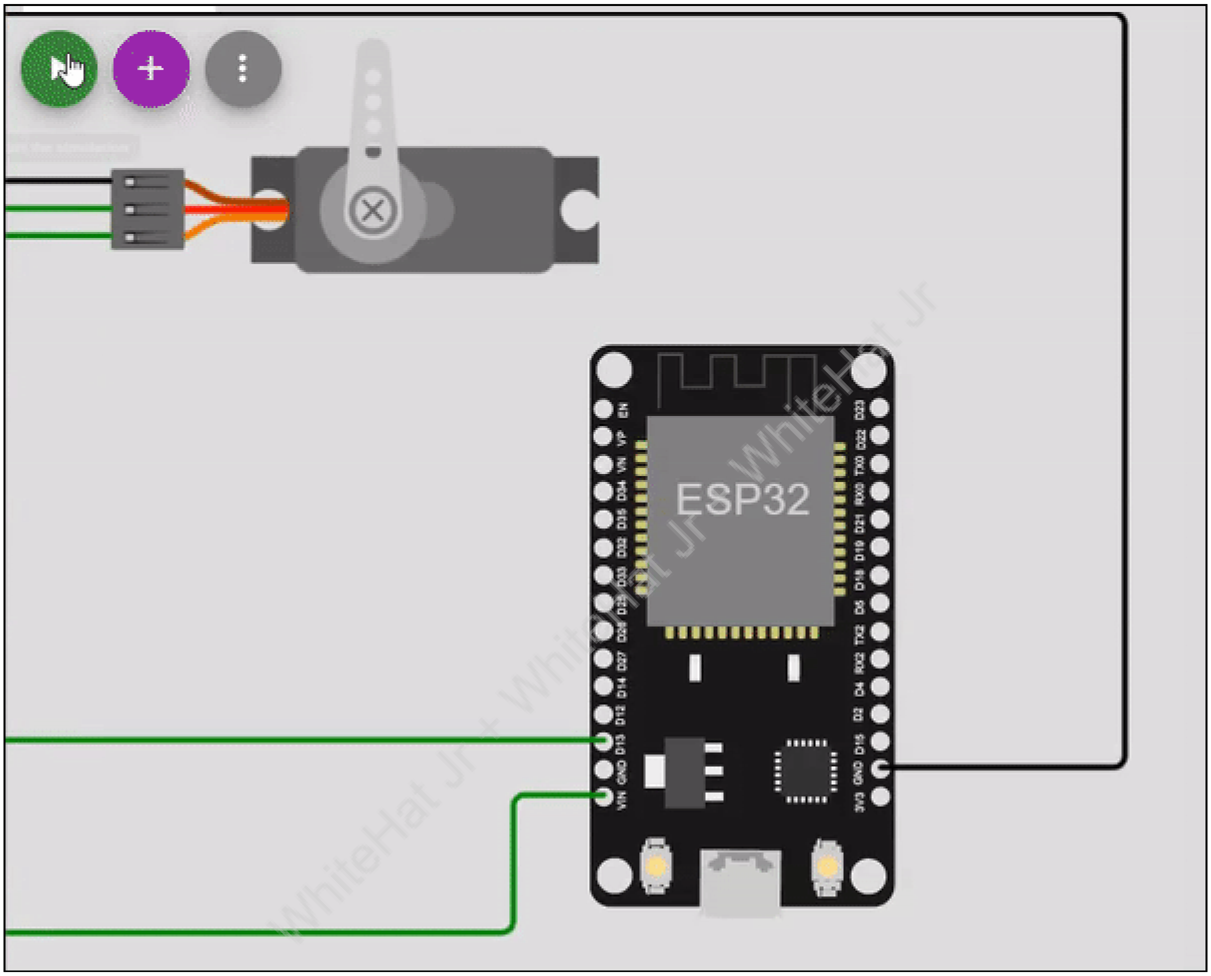
- Now in the void loop we need to write the main process which needs to be processed.
- **for loop** will help to increment & decrement the motor rotation process. The servo motor rotates between 0 to 180. We need to set a loop between 0 to 180.
- For the forward position, we need to increment the same. But this time the starting position will be 0 and it will go till 180.
- **myservo.write(pos)** tell servo to go to position in variable **‘pos’**
- Set the delay to **15 ms**
- For the backward position, we need to decrease the same. But this time the starting position will be 180 and it will go till 0.
- Set the delay to **15 ms**



```
void loop() {
   for (pos = 0; pos <= 180; pos += 1) {

    myservo.write(pos);
    delay(15);
  }
  for (pos = 180; pos >= 0; pos -= 1)
    myservo.write(pos);
    delay(15);
}
```

4. **Output:**
    - Click on the Save button and then click on the simulation button
    - Press the key and see the output on the Serial Monitor of the simulator.
    - Just press the keys and you will get the output.



5. Create a servo Security Toll Gate application.

6. Gather the material from the simulator.
    - 1  x ESP32
    - 1  x Servo:
    - 1 x Buzzer
    - 1 x PIR sensor
    - 1 x Light

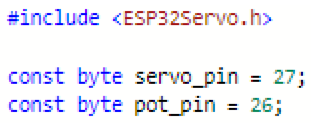
7. **Do connections:**
    - Servo **VCC pin** is connected to the ESP32 **VIN** PIN.
    - Servo **GND pin** is connected to the ESP32 **GND** PIN.

- Servo PWM pin is connected to **ESP32 GPIO D27**PIN.
- **Potentiometer SIG** is connected to **D26** PIN.
- **Potentiometer VCC pin** is connected to **ESP32 3V3**PIN.

8. Write the program: Go to the **sketch.ino**, delete the entire code, and start writing our new code.

    - Include the **ESP32 Servo** library

    - Define variables for potentiometer and motors as **pot_pin, and servo_pin** along with their data type

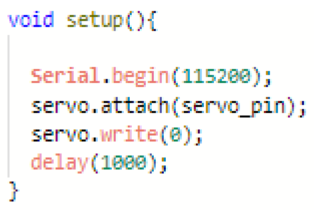


```
#include <ESP32Servo.h>

const byte servo_pin = 27;
const byte pot_pin = 26;
```

9. **Set the** password for the same
    - Initialize using **void setup()** function
    - **Serial. begin(115200)** is used to measure the speed of data exchange. This tells the Arduino to get ready to exchange messages with the Serial Monitor at a data rate of 9600 bits per second. That's **115200** binary ones or zeros per second and is commonly called a baud rate.
    - servo.attach(servo_pin):
    - **Parameters**
        - *servo*: a variable of type **Servo**
        - *pin*: the number of the pin that the servo is attached

- servo.**write()**
- Set the **delay** of 1000 ms.



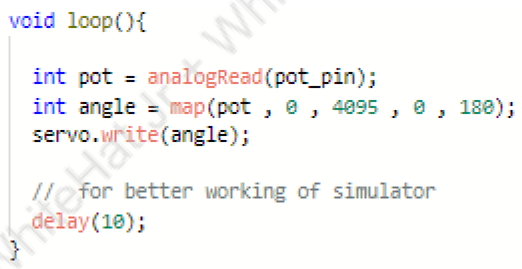
```
void setup(){

  Serial.begin(115200);
  servo.attach(servo_pin);
  servo.write(0);
  delay(1000);
}
```

We need to read the input pin that can be possible using **analogRead()**

- define a variable **pot** using data type **.**
- **analogRead():** Reads the value from the specified analog pin. ESP32 contains a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023.analogRead() read the input pin

  Syntax: analogRead(pin)
- Parameters pin: the name of the analog input pin to read ,
- define variable **angle** using data type **int.** angle is the pot value re-scaled to 0-179
- The resolution of the analog to digital converter can be done in this range (0-1023 for 10 bits or 0-4095 for 12 bits) using **map()** function.
- **map method()** will scale servo angle with the potentiometer value .
- Set the **delay** of 10 ms

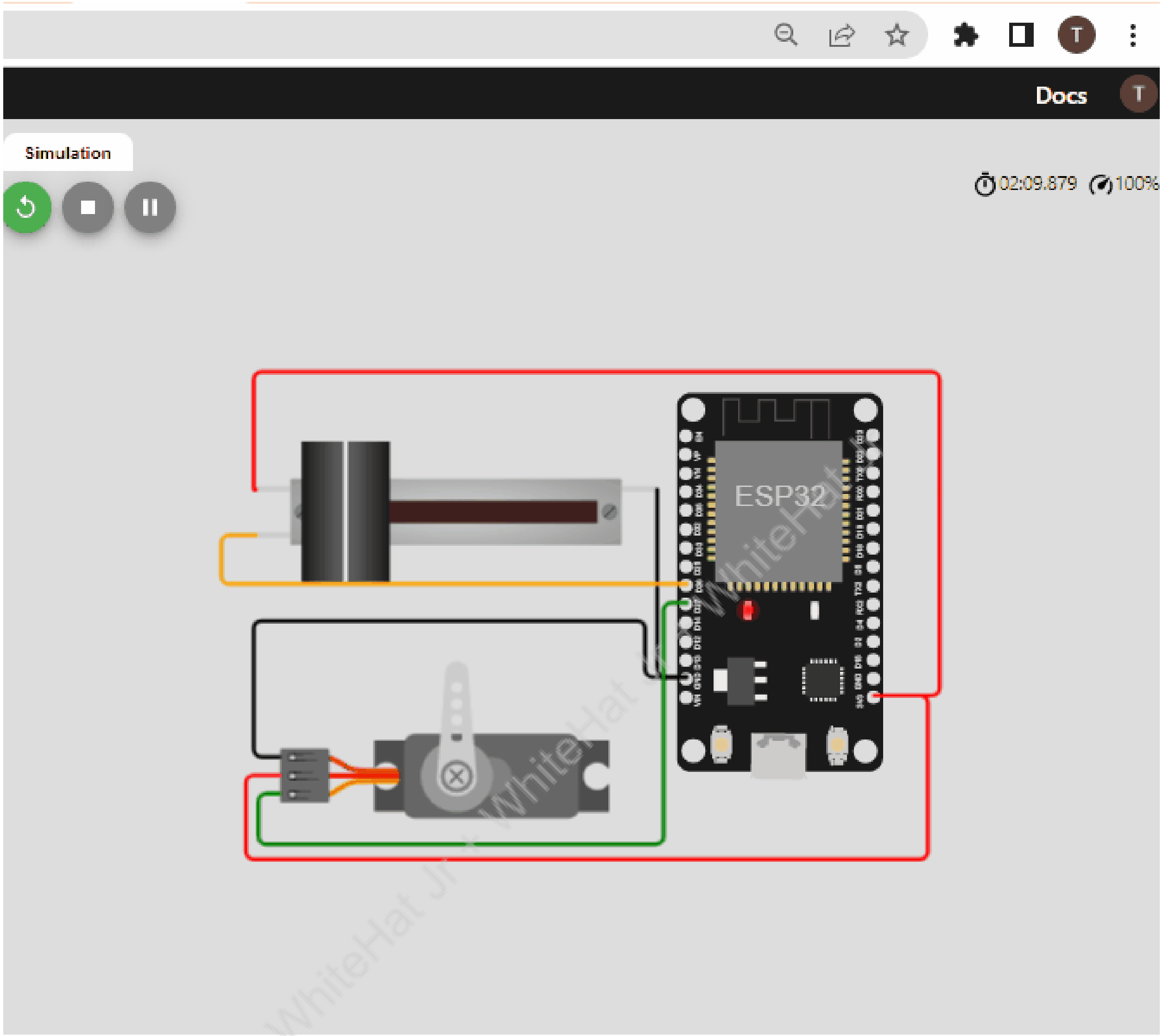


```
void loop(){

  int pot = analogRead(pot_pin);
  int angle = map(pot , 0 , 4095 , 0 , 180);
  servo.write(angle);

  //  for better working of simulator
  delay(10);
}
```

10. Output
- Click on the Save button and then click on the simulation button
- Press the key and see the output on the Serial Monitor of the simulator.
- Just press the keys and you will get the output



## What's NEXT?

In the **next class**, we will learn about **PIR sensors**

## Expand Your Knowledge

To know more about **Keypad** click here.