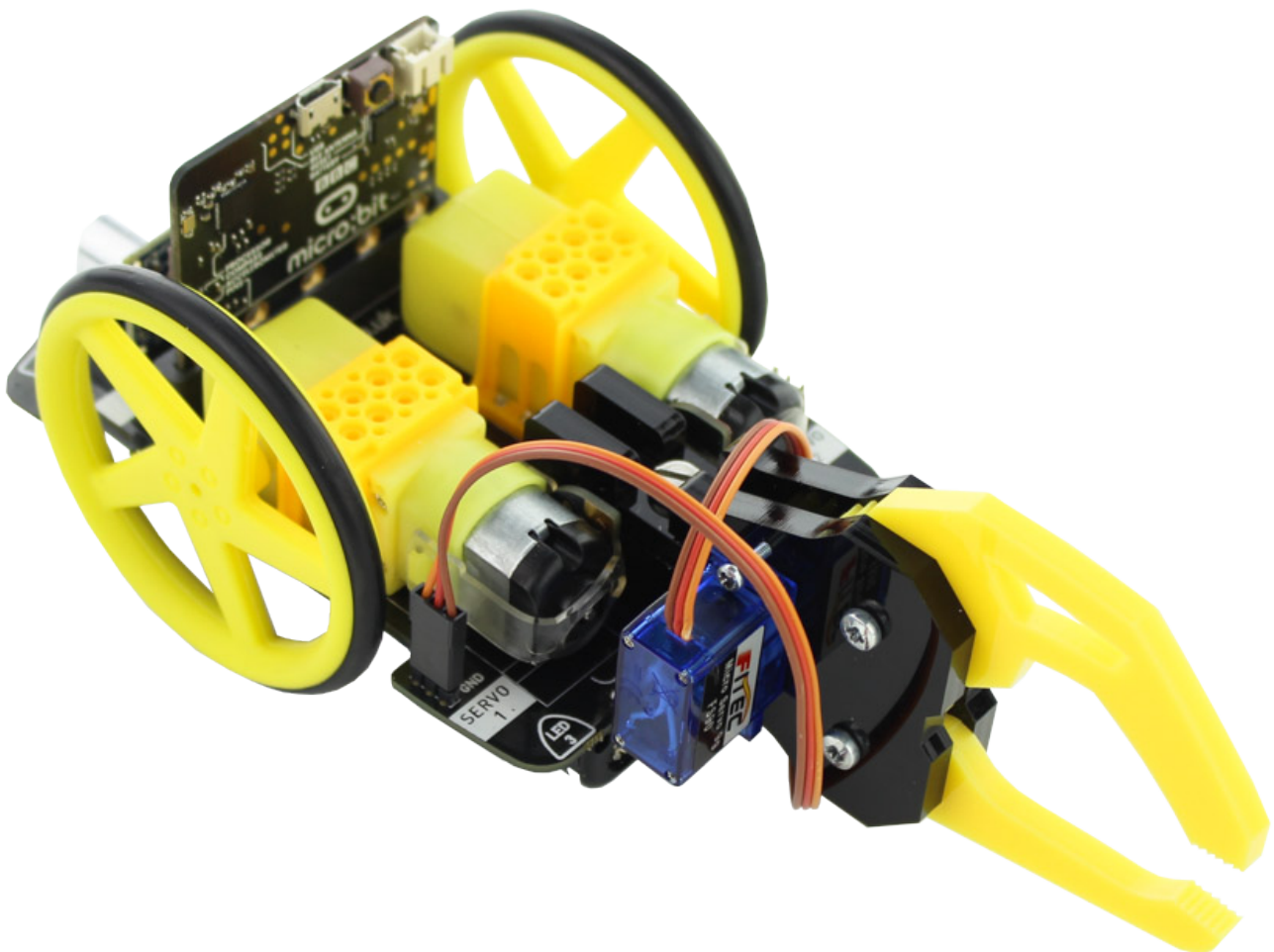




:MOVE MOTOR **KLAW** **MICROPYTHON TUTORIAL**

KITRONIK RESOURCES



LEARN HOW TO USE THE :MOVE MOTOR KLAW AND CONTROL A SERVO USING MICROPYTHON

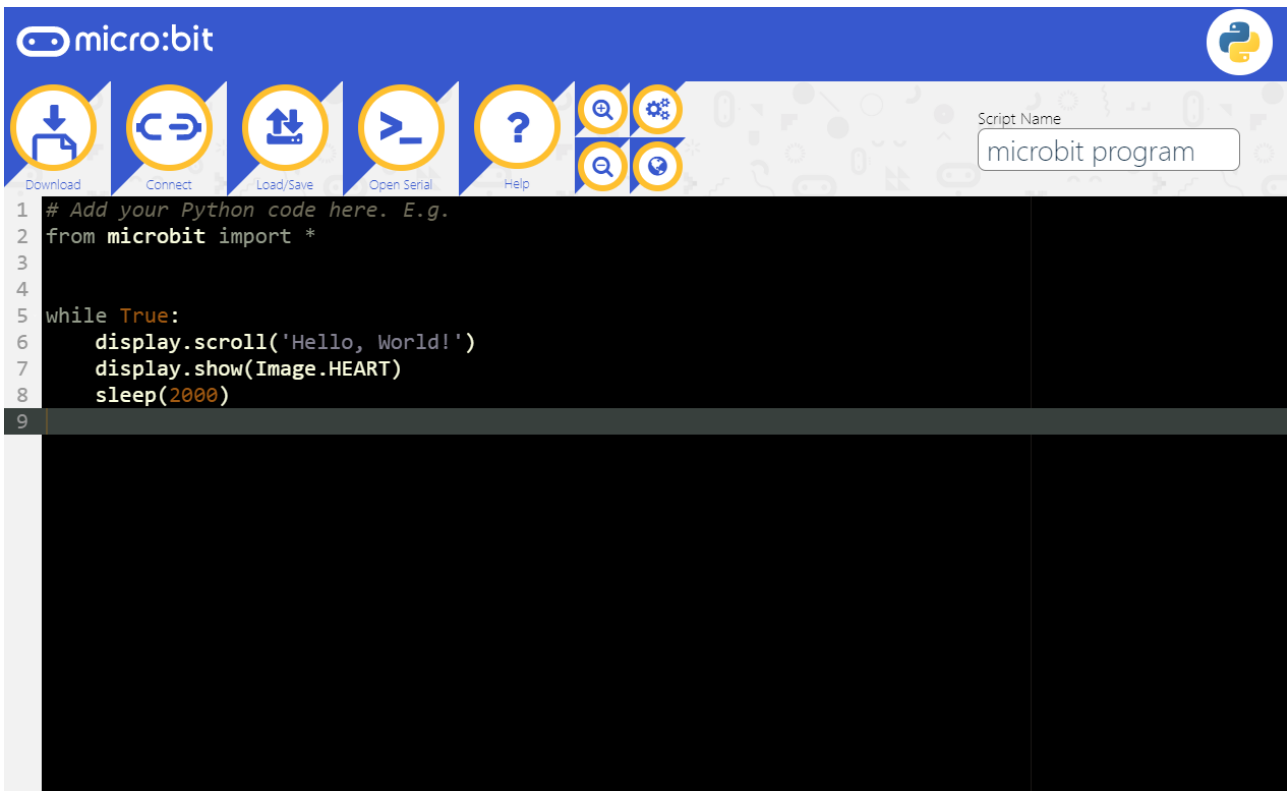
INTRODUCTION

This tutorial will show you how to control the Klaw attachment for the :MOVE Motor using Python. Within this tutorial, we will be using the micro:bit Python editor found at:

<https://python.microbit.org>

(Note: There are other editors available if you have a preferred tool to use).

WHEN A NEW PROJECT IS FIRST STARTED IT WILL APPEAR SIMILAR TO THIS:



Remove all the code apart from the “microbit import”.

Before coding, make sure the Klaw is assembled and attached to the :MOVE Motor. The instructions for this can be found on the product page: kitronik.co.uk/5696.

TECHNICAL NOTE: The :MOVE Motor Klaw uses a FiTec SG90 180 degree servo (kitronik.co.uk/2565). Its pulse width range is 700µs to 2300µs, with the frequency set to 50Hz.

THE TUTORIAL

STEP 1: There are only two pins on the :MOVE Motor which are dedicated to driving servos; these are P15 and P16. The signal on these pins need to be a PWM, which is produced with an analog write command.

First, create a class called “klaw” and then add a constructor “__init__” function inside. The function needs to have parameters for the micro:bit pin, servo frequency, servo pulse width range and the maximum drive angle for the servo. Then, set the parameter values to variables.

```
2 from microbit import *
3 class klaw:
4     def __init__(self, pin, freq=50, min_us=700, max_us=2300, angle=180):
5         self.pin = pin
6         self.freq = freq
7         self.min_us = min_us
8         self.max_us = max_us
9         self.angle = angle
```

STEP 2: There will be two functions created for controlling the servo: one to set the servo control pulse width in microseconds, the other to set the angle on the servo (for more information on servo control works, see <https://kitronik.co.uk/blogs/resources/servos-brief-guide>).

For use with both functions, we will need to create a variable for the microseconds (μ s) and set the analogue period for the pin which will have the PWM output. (**NOTE:** The “set_analog_period” function requires the period to be in milliseconds (ms), hence the multiplication by 1000).

```
2 from microbit import *
3 class klaw:
4     def __init__(self, pin, freq=50, min_us=700, max_us=2300, angle=180):
5         self.pin = pin
6         self.freq = freq
7         self.min_us = min_us
8         self.max_us = max_us
9         self.angle = angle
10        self.us = 0
11        self.analog_period = 0
12        analog_period = round((1/self.freq) * 1000)
13        self.pin.set_analog_period(analog_period)
```

STEP 3: Now to create the functions to drive the servo. Let's start with the pulse width in microseconds.

Our function will be called "write_us" and will have the parameter "us" passed into it.

We need to check if the value for microseconds ("us") is within the minimum and maximum values set in the constructor. Python has some built in functions which are very useful for this, returning the min or max value from the arguments passed in. The combined function implementation here will make sure the passed in "us" value can never be outside the allowed range, altering the value to either min or max, depending on which boundary it crossed.

The analogue output needs be calculated using the following formula (for the micro:bit, the analogue write values range from 0 to 1023):

$$\text{analog_op} = \frac{(\text{microseconds} \times \text{ADC Resolution} \times \text{frequency})}{(\text{microsecond conversion factor})}$$

The "analog_op" will need to be rounded to avoid dealing with decimal numbers. Finally, write the "analog_op" value to the servo pin using the "write_analog" function.

```
14
15 def write_us(self, us):
16     self.us = min(self.max_us, max(self.min_us, us))
17     analog_op = round(self.us * 1024 * self.freq // 1000000)
18     self.pin.write_analog(analog_op)
19
```

STEP 4: Next, the function to set the angle on the servo. This function will be called “write_angle” and will have the parameter “degrees” passed into it. This function will convert the angle into microseconds and then use the previous function to actually write a value to the servo.

First, we need to make sure the number passed in is a whole number. To do this, the Python floor division operator, “//”, can be used, dividing “degrees” by 1 and then rounding down.

Next, to convert degrees to a microsecond value, this formula is used:

$$\text{microseconds} = \frac{\text{minimum pulse} + (\text{pulse range} \times \text{degrees to turn})}{\text{maximum servo angle}}$$

For example, to turn 45 degrees:

$$\text{microseconds} = 700 + ((2300-700) \times 45) / 180 = 1100\mu\text{s pulse}$$

```
19
20 def write_angle(self, degrees=None):
21     degrees = degrees // 1
22     pulse_range = self.max_us - self.min_us
23     self.us = self.min_us + (pulse_range * degrees) // self.angle
24     self.write_us(self.us)
25
```

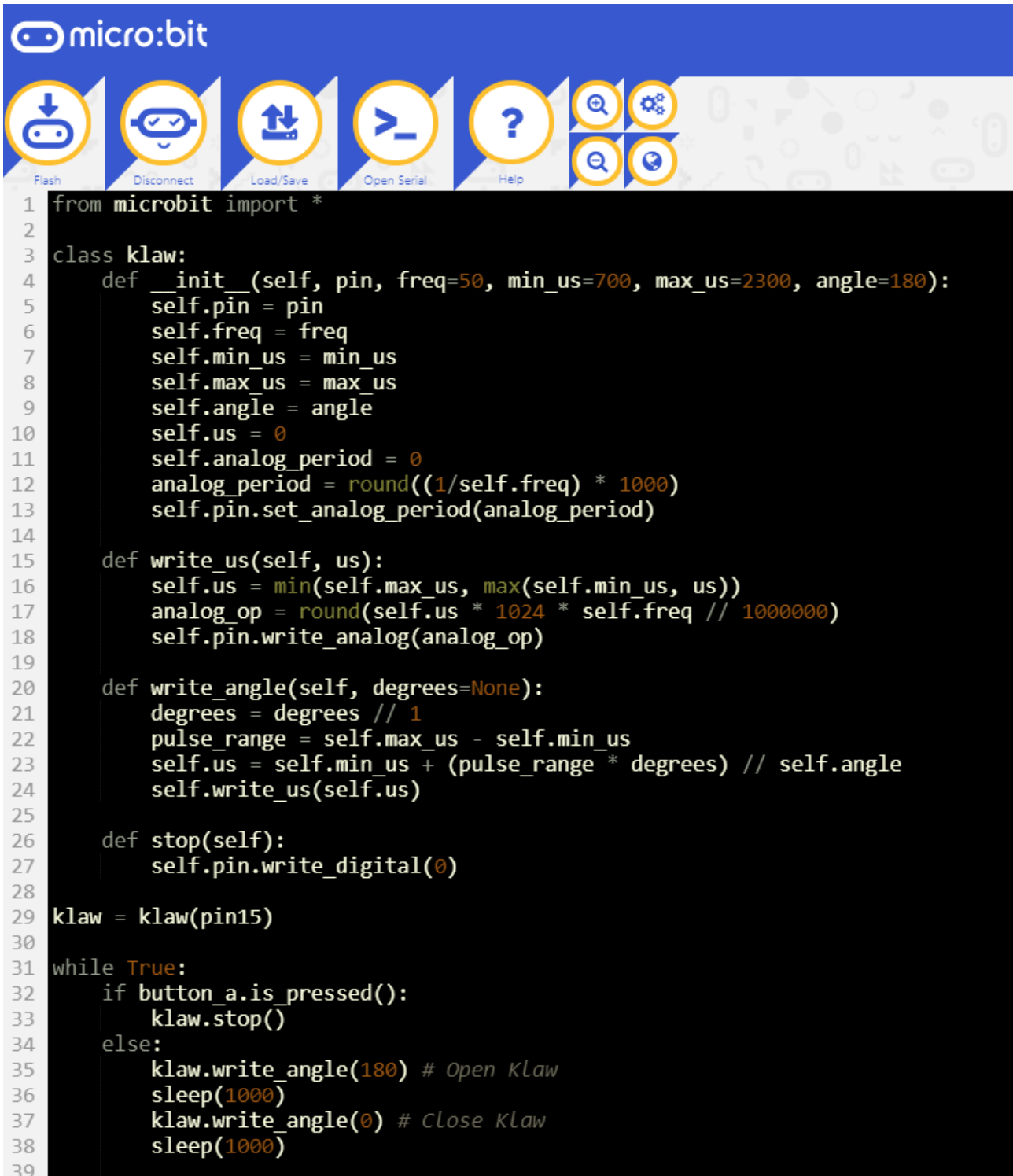
STEP 5: We can also create another function, this time to stop driving the servo. Create a function called “stop”. This function will do a digital write of “0” to the servo pin.

```
25
26     def stop(self):
27         self.pin.write_digital(0)
28
```

STEP 6: The final part of the program will make use of these functions. Create an implementation of the “klaw” class with the servo connection pin. Then, in a “while True” loop, use button A on the micro:bit to stop the Klaw opening, and by default, make the Klaw open and close every second.

```
29 klaw = klaw(pin15)
30
31 while True:
32     if button_a.is_pressed():
33         klaw.stop()
34     else:
35         klaw.write_angle(180) # Open Klaw
36         sleep(1000)
37         klaw.write_angle(0) # Close Klaw
38         sleep(1000)
39
```

FINISHED CODE



```
1 from microbit import *
2
3 class klaw:
4     def __init__(self, pin, freq=50, min_us=700, max_us=2300, angle=180):
5         self.pin = pin
6         self.freq = freq
7         self.min_us = min_us
8         self.max_us = max_us
9         self.angle = angle
10        self.us = 0
11        self.analog_period = 0
12        analog_period = round((1/self.freq) * 1000)
13        self.pin.set_analog_period(analog_period)
14
15        def write_us(self, us):
16            self.us = min(self.max_us, max(self.min_us, us))
17            analog_op = round(self.us * 1024 * self.freq // 1000000)
18            self.pin.write_analog(analog_op)
19
20        def write_angle(self, degrees=None):
21            degrees = degrees // 1
22            pulse_range = self.max_us - self.min_us
23            self.us = self.min_us + (pulse_range * degrees) // self.angle
24            self.write_us(self.us)
25
26        def stop(self):
27            self.pin.write_digital(0)
28
29 klaw = klaw(pin15)
30
31 while True:
32     if button_a.is_pressed():
33         klaw.stop()
34     else:
35         klaw.write_angle(180) # Open Klaw
36         sleep(1000)
37         klaw.write_angle(0) # Close Klaw
38         sleep(1000)
39
```

For any further queries or support, please visit the Kitronik website: www.kitronik.co.uk/5696

Or get in touch:

Telephone +44 (0) 115 970 4243

Sales email: sales@kitronik.co.uk

Tech Support email: support@kitronik.co.uk

Web: www.kitronik.co.uk



[kitronik.co.uk/twitter](https://twitter.com/kitronik)



[kitronik.co.uk/facebook](https://www.facebook.com/kitronik)



[kitronik.co.uk/youtube](https://www.youtube.com/kitronik)



[kitronik.co.uk/instagram](https://www.instagram.com/kitronik)



Designed & manufactured
in the UK by 

