SMARTSERVO

GETTING STARTED WITH THE SMART SERVO

SMARTSERVO PROJECT

GETTING STARTED WITH THE SMART SERVO

Version 1.0 | Published: June 17, 2025 | Author: Judson Wagner, Wagner Labs LLC

⚠ IMPORTANT DISTRIBUTION REQUIREMENTS

This cover page must be included with any distribution, reproduction, or posting of this guide. Any printed copies, electronic distributions, or online postings must include this complete cover page to maintain proper attribution and licensing terms.

Commercial Use & Smart Servo Requirement

For commercial use, bulk educational licensing, or use outside of traditional educational settings, please contact Wagner Labs LLC.

Hardware Requirement: This guide requires **Smart Servo devices** to complete the projects and activities described. Smart Servos are available through the Smart Servo Store and authorized distributors.

About the Smart Servo Project

The Smart Servo Project empowers inclusive innovation by providing accessible tools for creating assistive technologies and engaging STEM education. Our mission is to bridge technology and compassion through community-driven maker education.

Support our mission by purchasing Smart Servos and sharing our resources with your educational community.

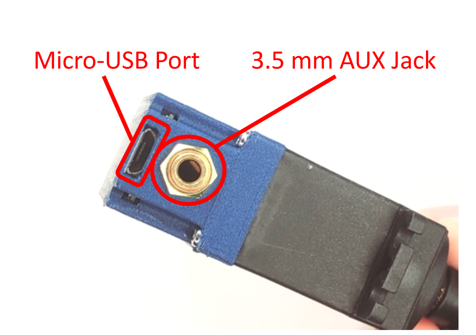
Contact Information:
Judson Wagner | Wagner Labs LLC
Email: **Judson@WagnerLabs.net**
Website: **WagnerLabs.net/SmartServo**
Smart Servo Store: **WagnerLabs..Store**

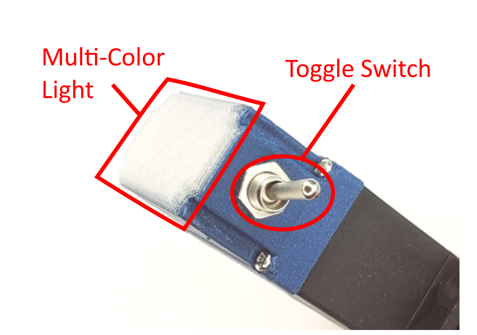
SMARTSERVO

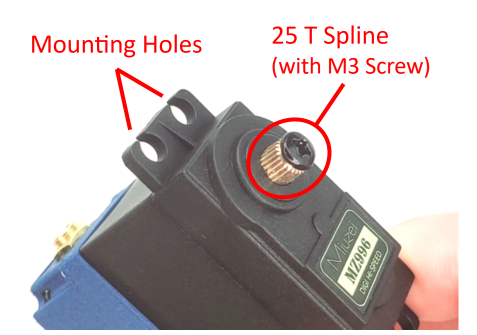
GETTING STARTED

This guide will show you how to get started with using your Smart Servo.

PART 1: Main Parts of the Smart Servo

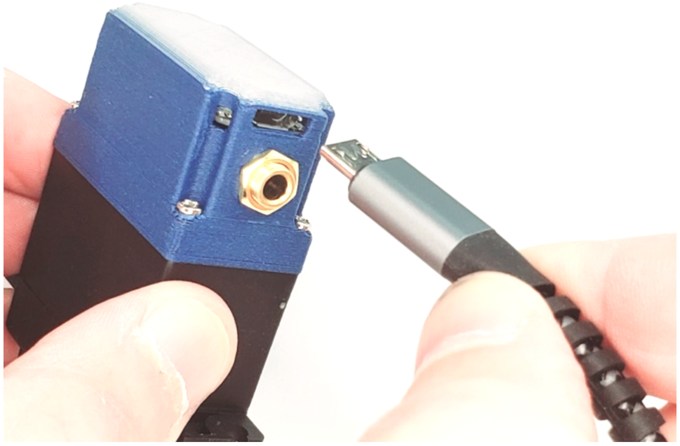






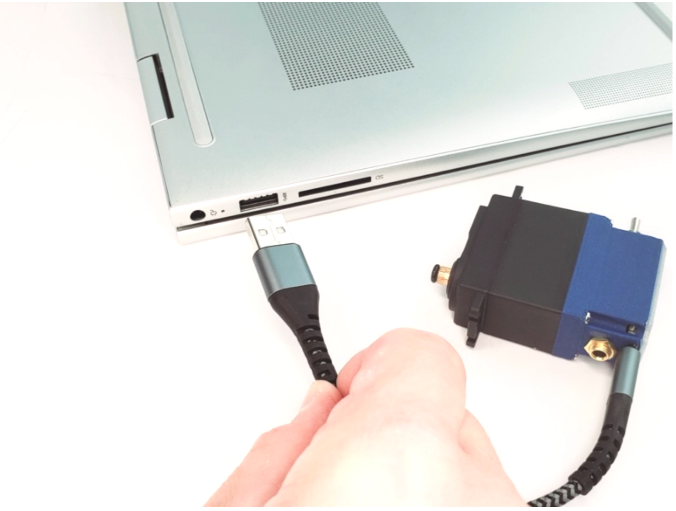
PART 2: Adding Power to the Smart Servo (3 Options)

First - Connect USB cable to Micro-USB port. You can use the Included cable.

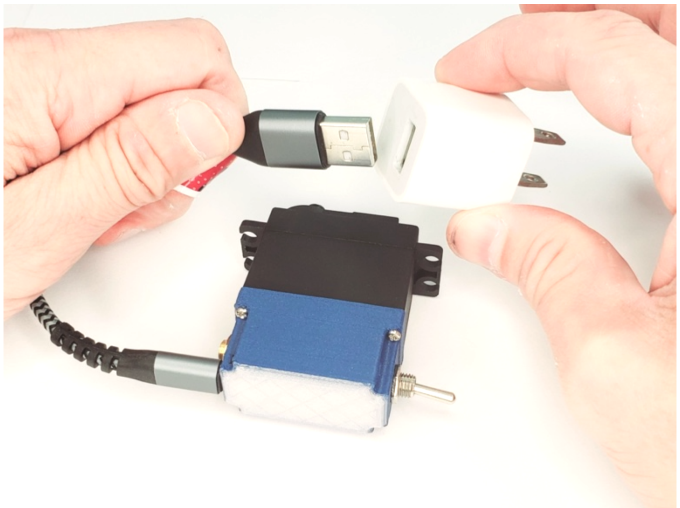


Option 1 - Connect other end of USB cable to rechargeable power bank, aka phone charger. (recommended for assistive technology projects)

Option 2 - Connect other end to laptop USB port. (recommended for re-programming & testing)

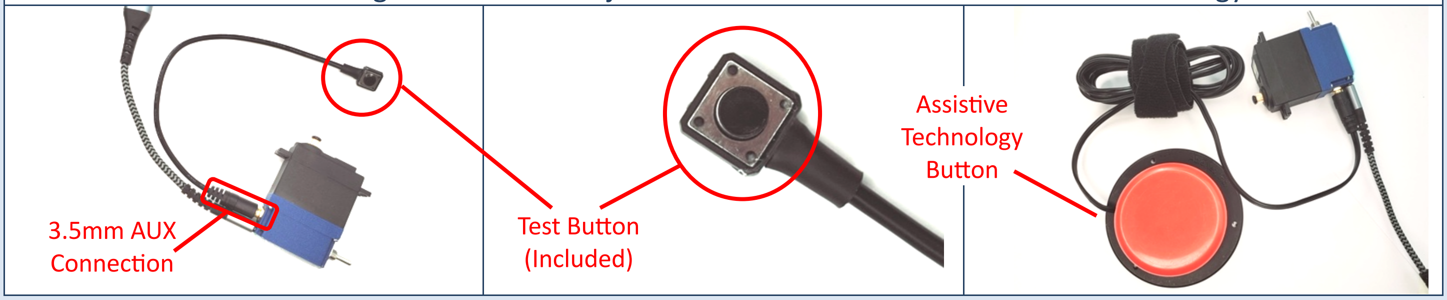


Option 3 - Connect to standard USB wall charger. (recommended for projects near walls)



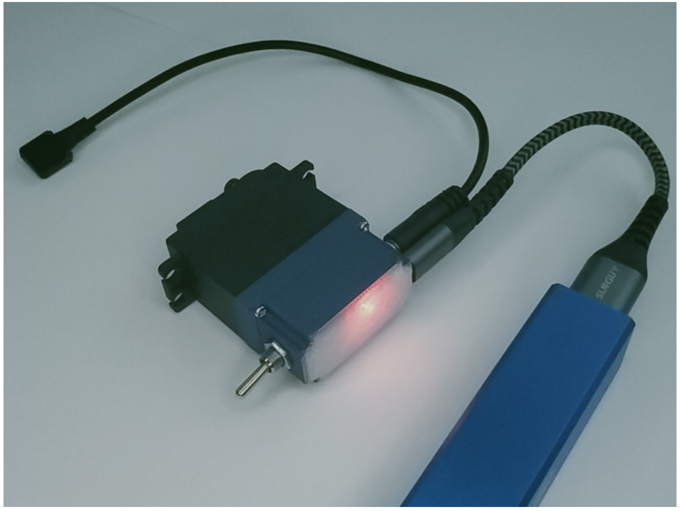
PART 3: Connecting Buttons to the Smart Servo (2 Common Types)

Buttons are connected using the 3.5mm AUX jack. This is a standard for all assistive technology buttons.

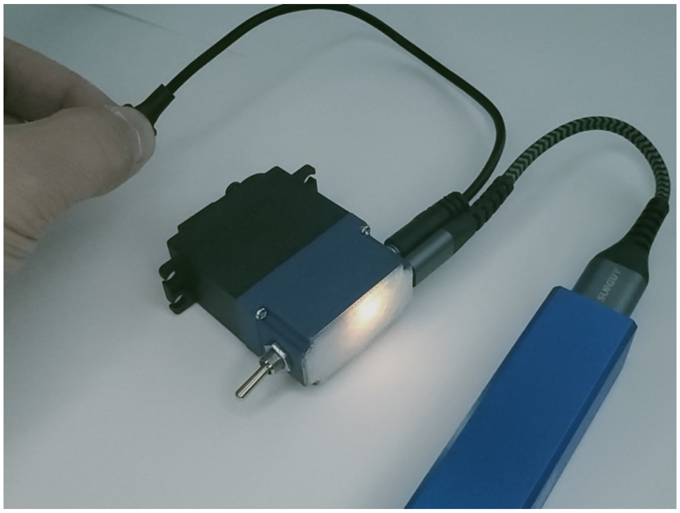


PART 4: Light Indicators on the Smart Servo (for base code – can be changed later)

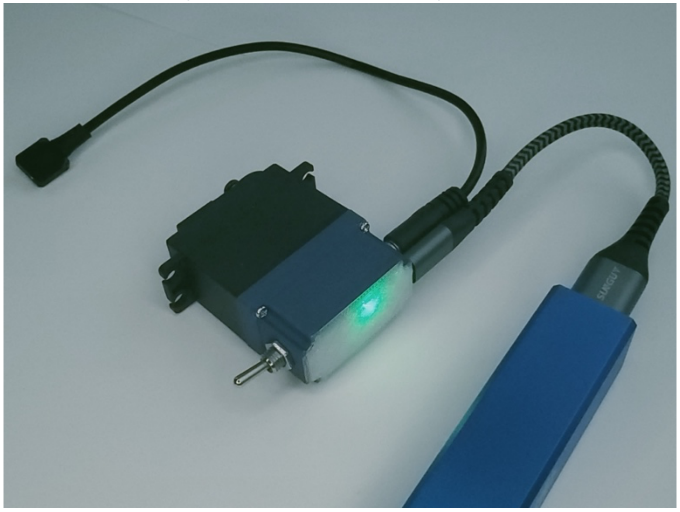
Red – When the light has a red pulse, the toggle switch has put the Smart Servo into Safe Mode. It will not operate even when a button is pushed.



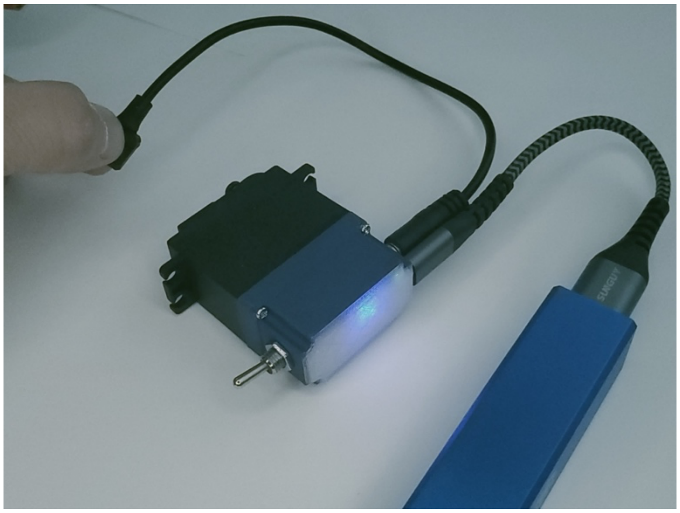
Yellow – A button can be tested when the Smart Servo is in Safe Mode by pressing and holding the button. Yellow indicates that the button works.



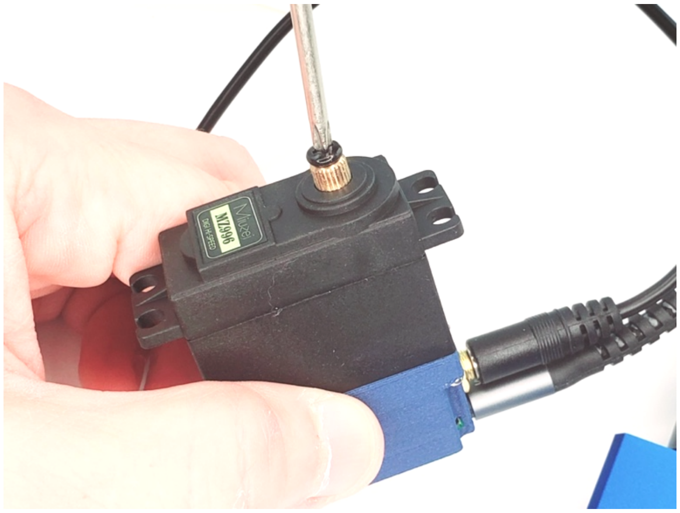
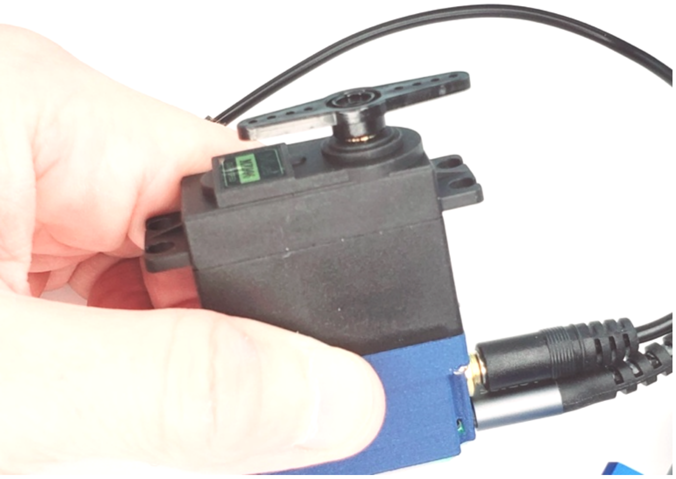
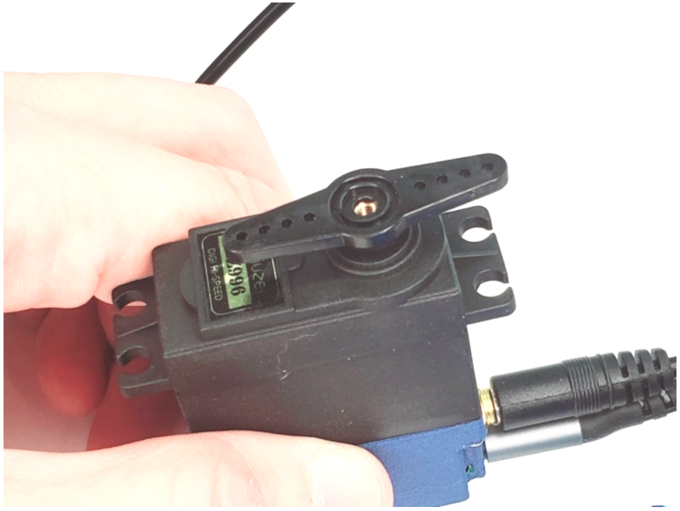
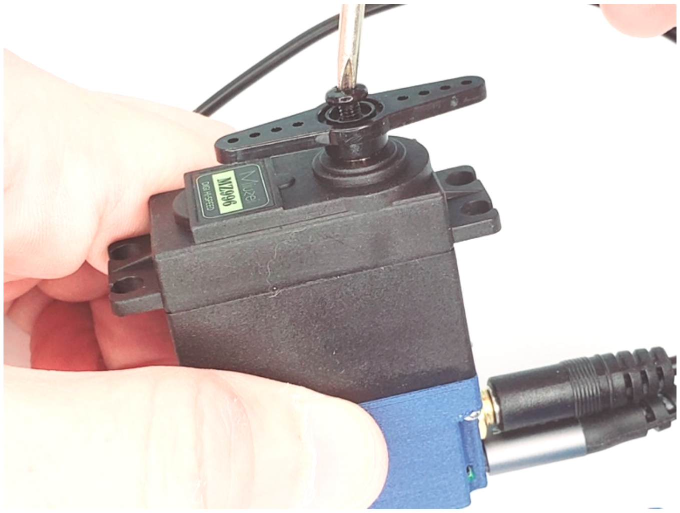
Green – Switching the toggle will arm the servo and the light will turn green. This means the Smart Servo will respond to a button input.



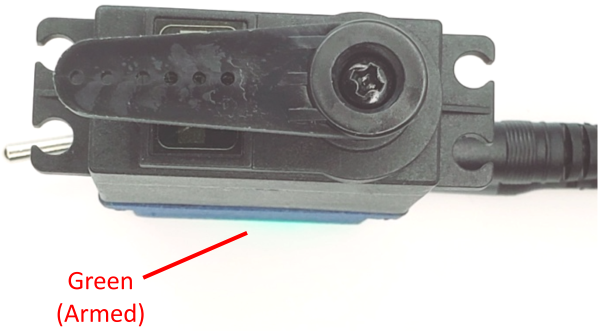
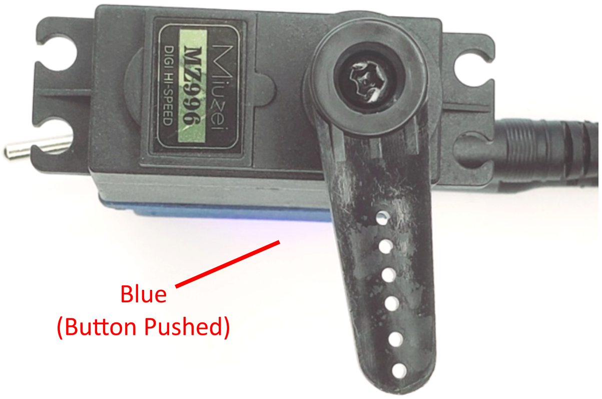
Blue – Pressing a button when the Smart Servo is armed will show a blue light.



PART 5: Connecting Servo Horns to the Smart Servo (Included)

Remove the M3 screw with a small (size #1) philips - head screwdriver. 	Align one of the included servo horns onto the 25T metal spline. The horn may shift to align with the teeth. 
Press the horn down far enough for the screw threads to reach the spline threads. 	Secure the horn with the M3 screw. 

PART 6: Check the Rotation Angle of the Smart Servo

With the Smart Servo Armed, note the initial position of the servo horn.  	Press the button and note how far the horn rotates. It should be just over 90 degrees in the counterclockwise direction.  Remember that this can be re-programmed later for specific projects later.

ALTERING ONBOARD CODE

This guide will show you how to alter the code on your Smart Servo.

1: Choosing a Text Editor

The code that comes pre-installed on the Smart Servo is written in Circuit Python. This Python coding can be read and edited by nearly anything that con open a text file. We recommend using a text editing program that will recognize the Python code and colorize it for readability. Here are three text editors we recommend:

CODE PAD

- Best option for Chromebooks
- Free Chrome Extension
- Install guide at tinyurl.com/CodePadInstall

SUBLIME TEXT

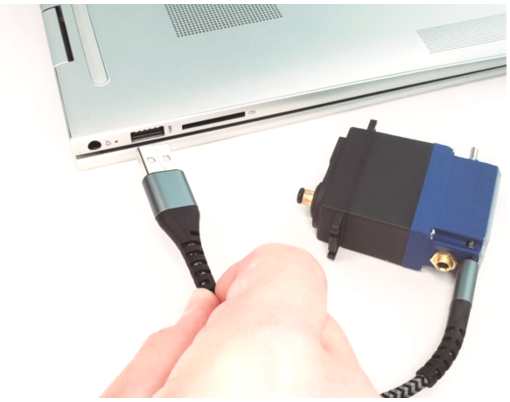
- Works on Mac & PC (Used in guides below)
- Free install
- Download at https://www.sublimetext.com/

CIRCUIT PYTHON CODE

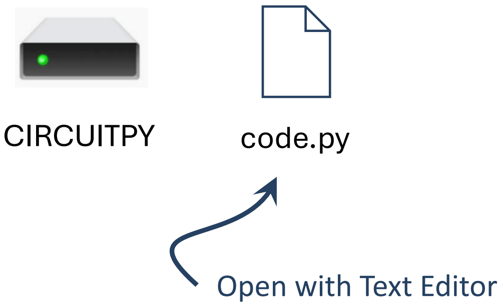
- Web-based
- Includes Serial Monitor
- Access at https://code.circuitpython.org/

2: Accessing the Programming that Came with Your Smart Servo

Step 1: Connect one end of the Micro-USB cable to the Smart Servo and the other end to the USB port of the computer you want to use to program.



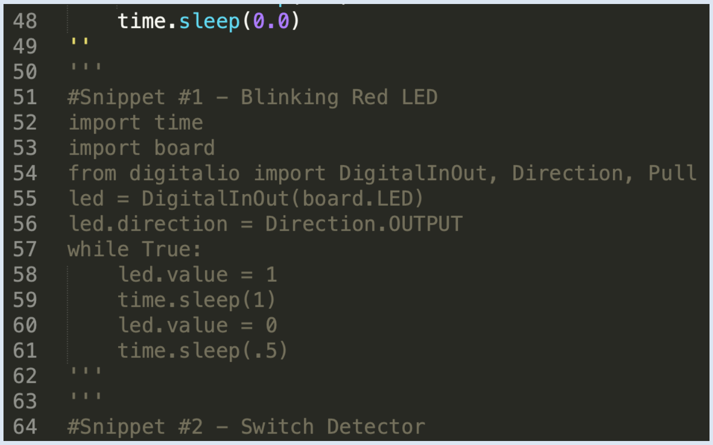
Step 2: Like an external drive, the Smart Servo will appear with the name **CIRCUITPY**. Open this drive and find the file **code.py**. Open this file with your text editor.



3: Starting with ‘snips’ of code

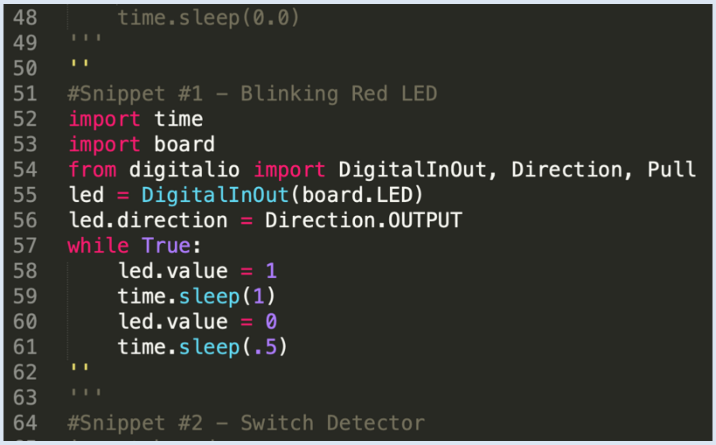
The base code that is used in the Getting Started Guide was developed and tested in Adapted Physical Education equipment created for students with physical disabilities. This default code can be found in the first lines of the code.py file. This code can be edited, saved, and tested.

To start working with “snips” of code, we’re going to use three single apostrophes written as ’’’ before and after the base code. This will gray-out the code and cause the Smart Servo to ignore this code when it runs. To start working with a snip, simple remove one of the apostrophes before and after the snip (’’’→’’)



```
48      time.sleep(0.0)
49  ''
50  '''
51  #Snippet #1 - Blinking Red LED
52  import time
53  import board
54  from digitalio import DigitalInOut, Direction, Pull
55  led = DigitalInOut(board.LED)
56  led.direction = Direction.OUTPUT
57  while True:
58      led.value = 1
59      time.sleep(1)
60      led.value = 0
61      time.sleep(.5)
62  '''
63  '''
64  #Snippet #2 - Switch Detector
```

Smart Servo ignores all code after line 49.

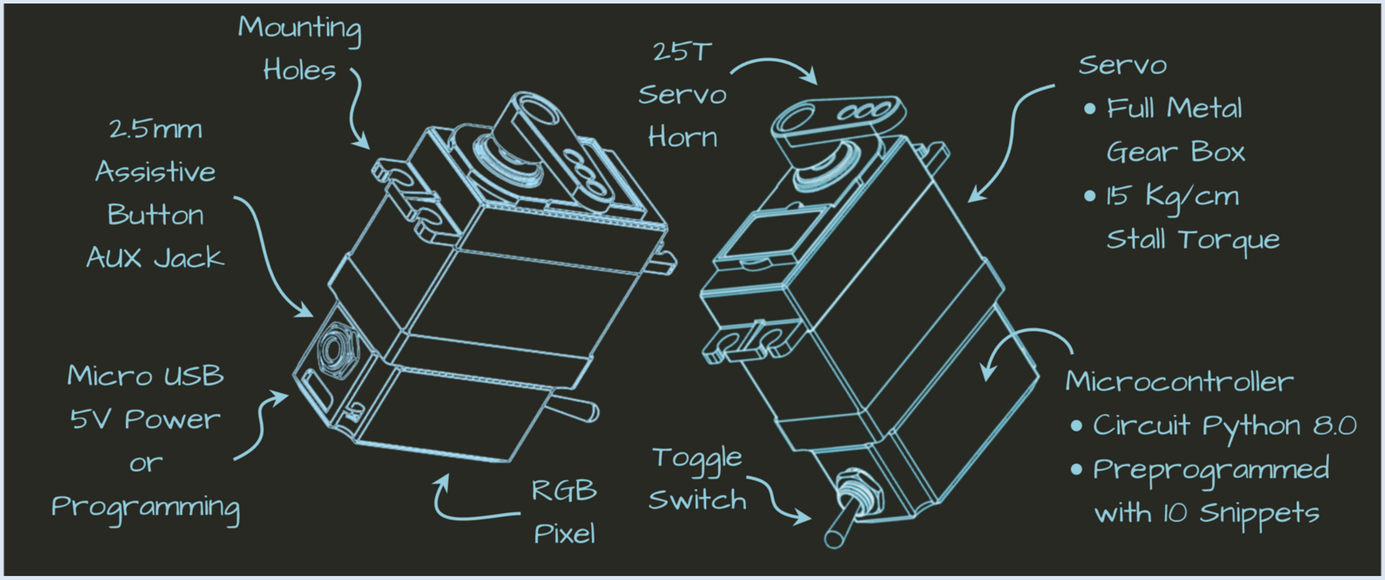


```
48      time.sleep(0.0)
49  '''
50  ''
51  #Snippet #1 - Blinking Red LED
52  import time
53  import board
54  from digitalio import DigitalInOut, Direction, Pull
55  led = DigitalInOut(board.LED)
56  led.direction = Direction.OUTPUT
57  while True:
58      led.value = 1
59      time.sleep(1)
60      led.value = 0
61      time.sleep(.5)
62  ''
63  '''
64  #Snippet #2 - Switch Detector
```

Smart Servo ignores all code except lines 50 to 62.

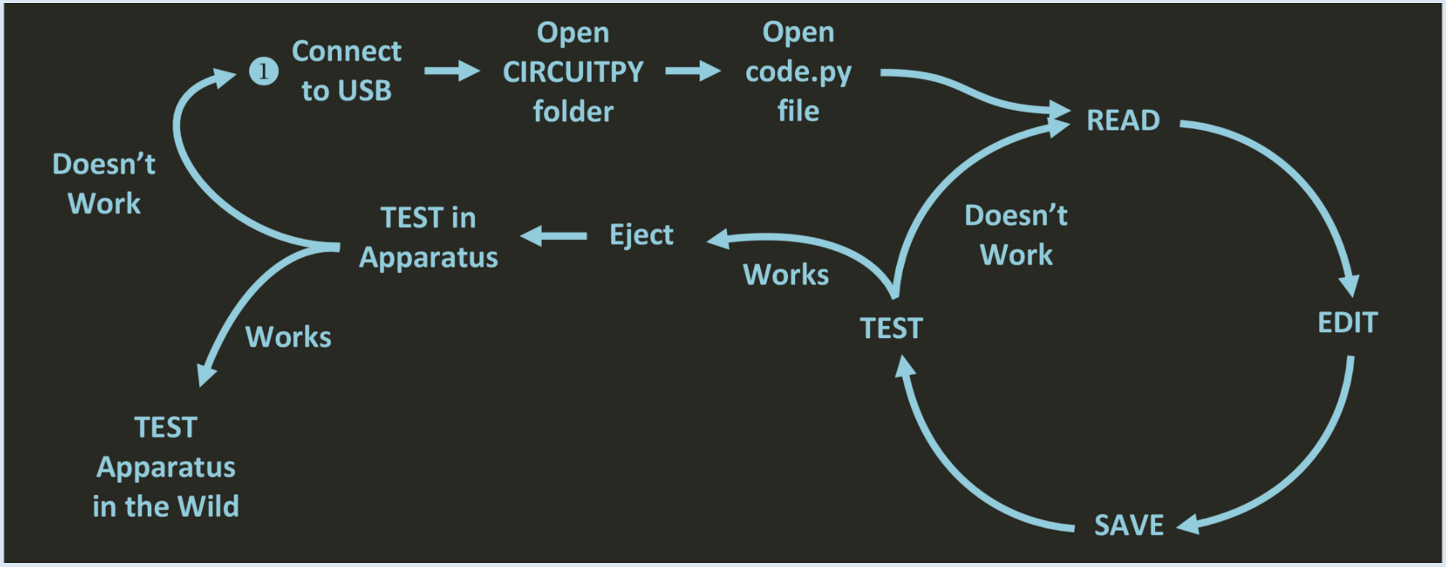
4: Parts of the Smart Servo

As some of the code refers to specific components of the Smart Servo, the diagram below gives us a chance to review some of the main features.



5: Feedback Loop

Because the Smart Servo's micro controller does all of the code compiling in real time, it's easy to establish a rapid feedback loop. This graphics below summarizes this loop starting with j where we connected the Smart Servo to a computer.



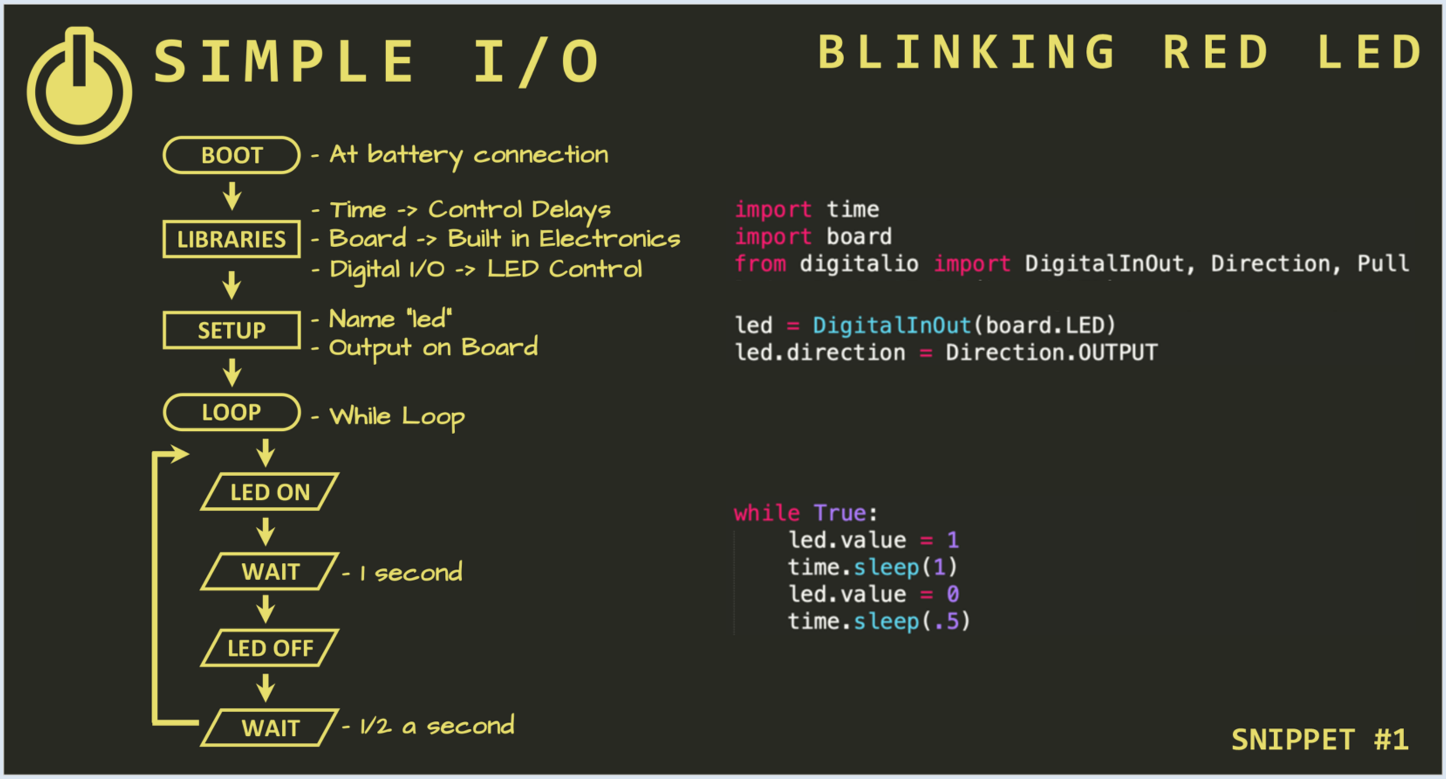
6: Work through each Snippet.

It is recommended that beginners start by working through each snippet on the Smart Servo. Each snippet introduces a new input or output or shows how different strategies can be used to make the Smart Servo more sophisticated in what it can do. With each snippet, try to mess with some of the code to see what it does. If something stops working, undo the step that made it stop working. If things get really out of whack, replace the files on the Smart Servo with the ones found here:

tinyurl.com/SmartServoSnips

Below, you'll find the snips with some reminders and references. Also included is a flow chart to illustrate the logic behind each snippet.

This code controls a tiny red LED light inside of the Smart Servo.



SIMPLE I/O

BLINKING RED LED

BOOT - At battery connection

LIBRARIES
- Time -> Control Delays
- Board -> Built in Electronics
- Digital I/O -> LED Control

SETUP
- Name "led"
- Output on Board

LOOP - While Loop

LED ON

WAIT - 1 second

LED OFF

WAIT - 1/2 a second

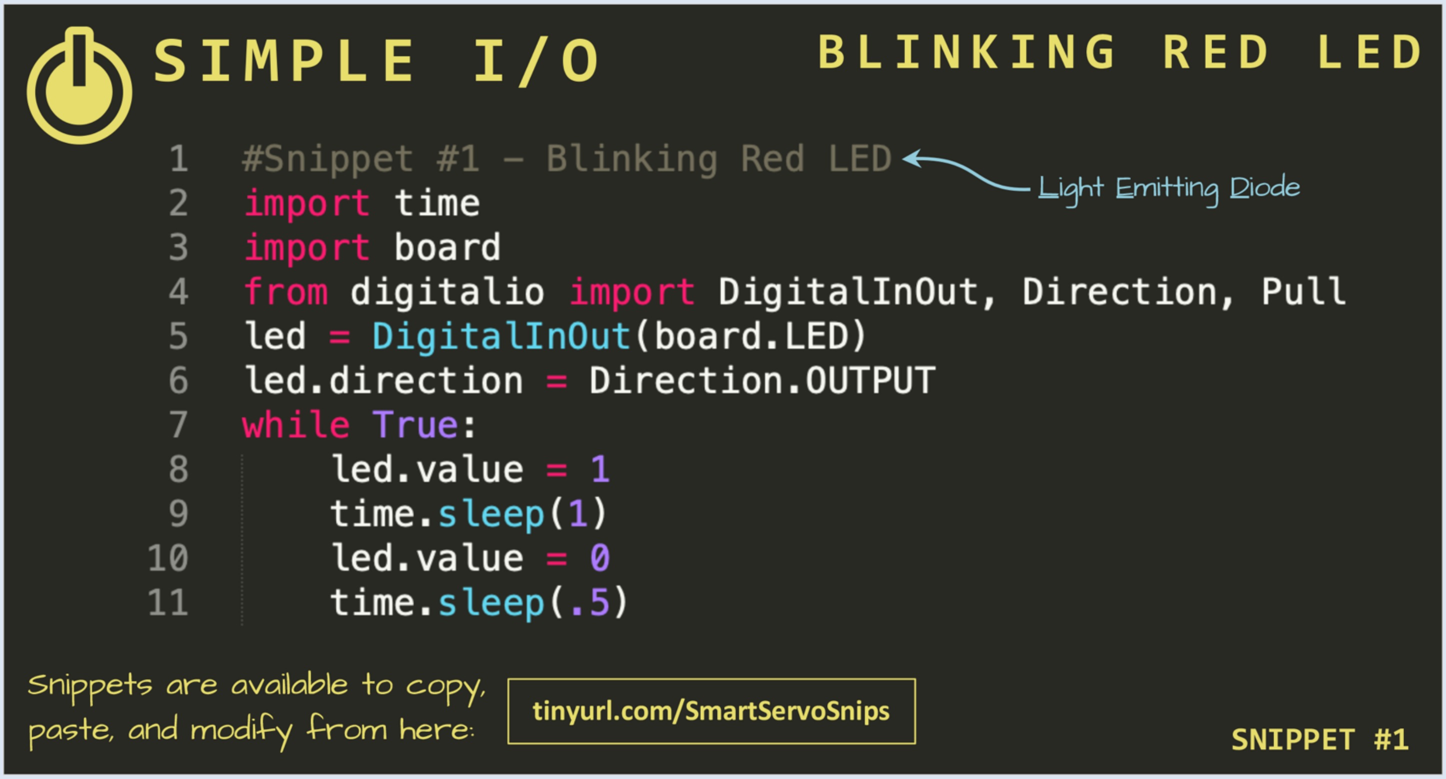
```
import time
import board
from digitalio import DigitalInOut, Direction, Pull

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT



while True:
    led.value = 1
    time.sleep(1)
    led.value = 0
    time.sleep(.5)
```

SNIPPET #1



SIMPLE I/O

BLINKING RED LED

```
#Snippet #1 - Blinking Red LED
import time
import board
from digitalio import DigitalInOut, Direction, Pull
led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
while True:
    led.value = 1
    time.sleep(1)
    led.value = 0
    time.sleep(.5)
```

Light Emitting Diode

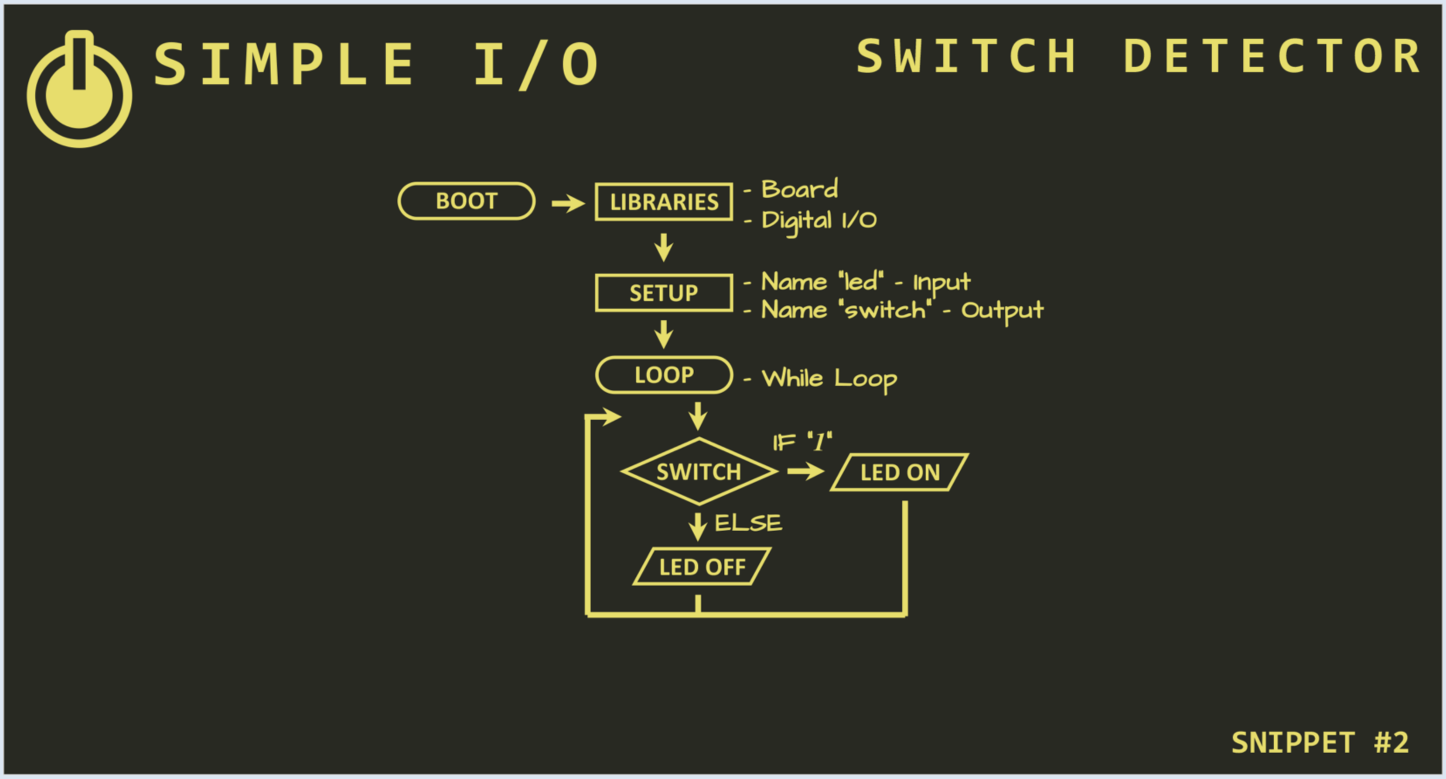
Snippets are available to copy, paste, and modify from here:

tinyurl.com/SmartServoSnips

SNIPPET #1

Try changing the led.value to control how the light blinks.

This code uses the LED to let you know the position of the toggle switch



SIMPLE I/O

SWITCH DETECTOR

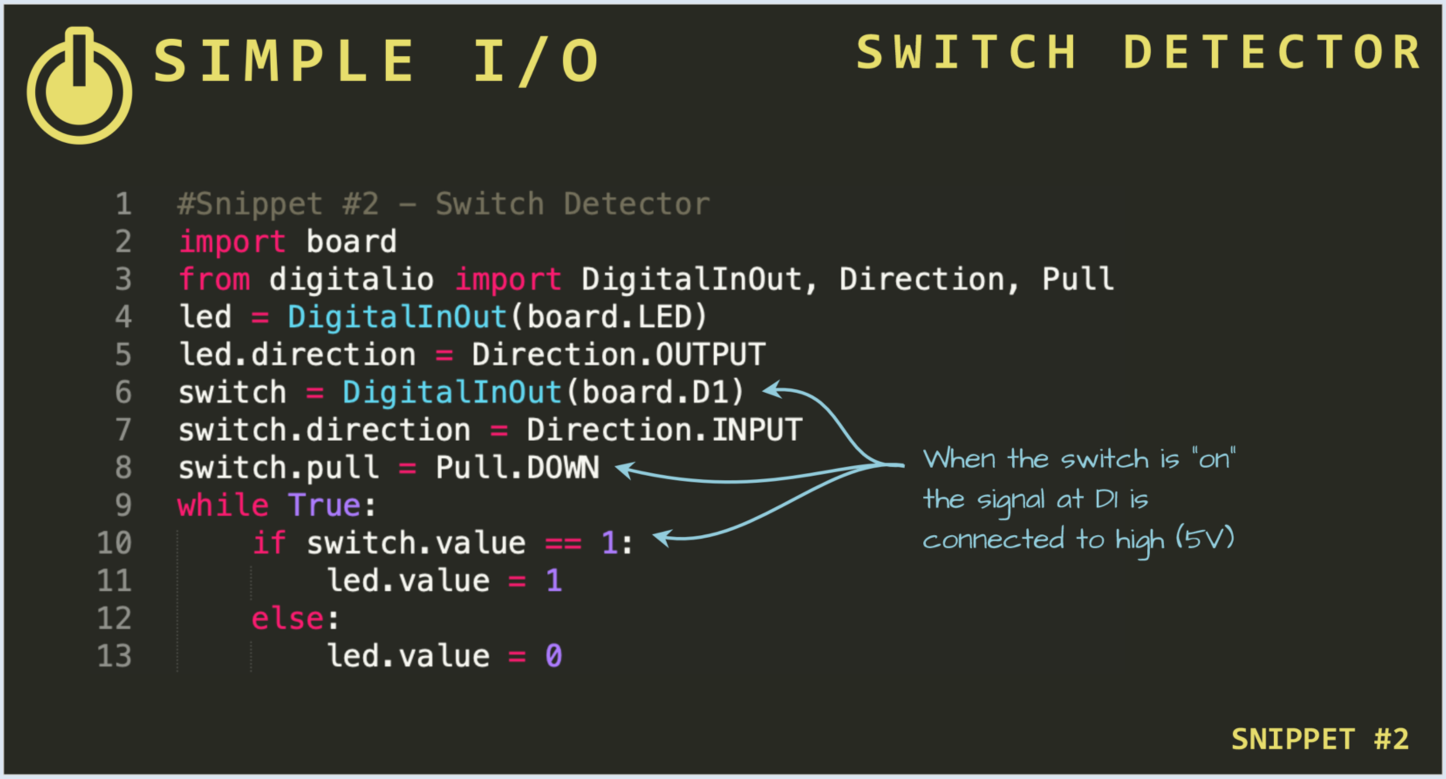
BOOT → LIBRARIES
- Board
- Digital I/O

SETUP
- Name "led" - Input
- Name "switch" - Output

LOOP
- While Loop

SWITCH
IF "1" → LED ON
ELSE → LED OFF

SNIPPET #2



SIMPLE I/O

SWITCH DETECTOR

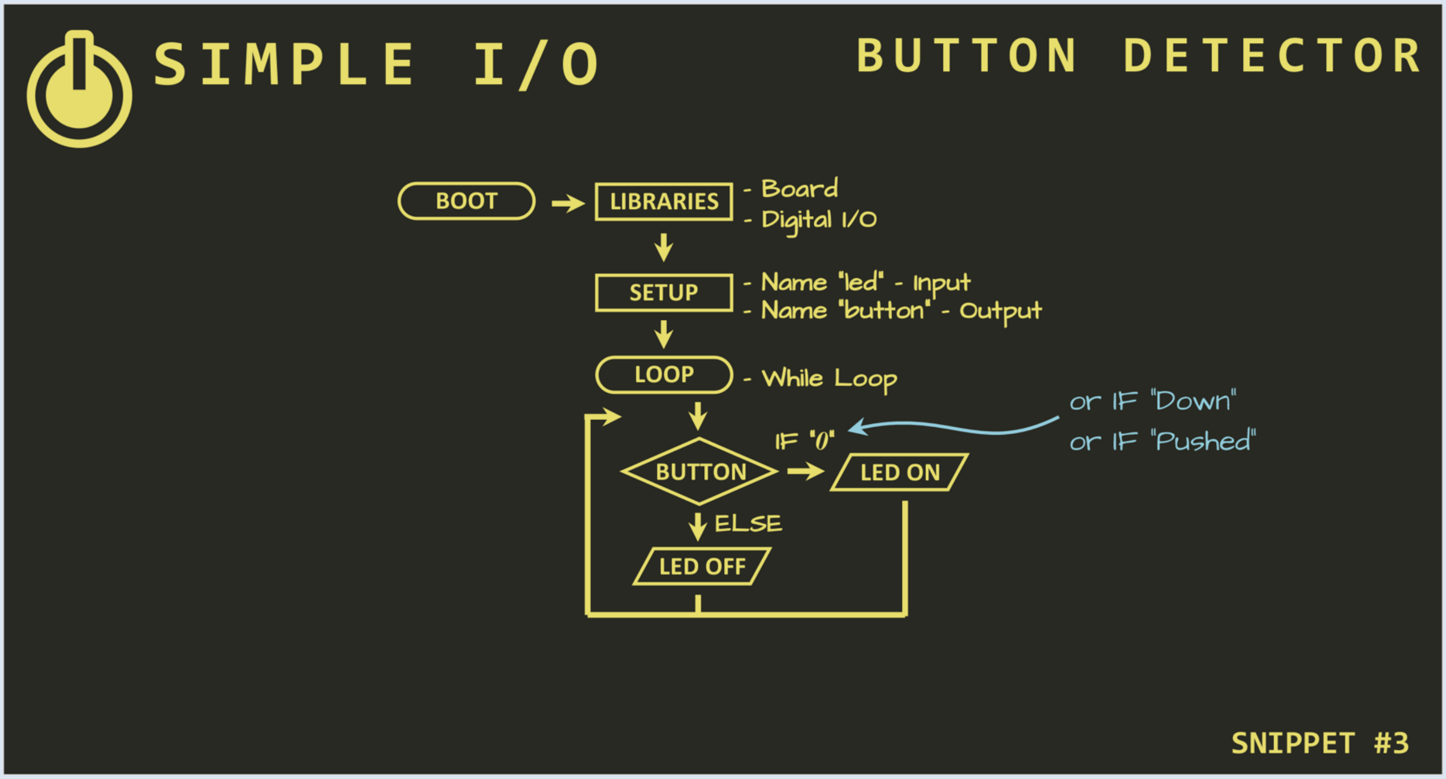
```
#Snippet #2 - Switch Detector
import board
from digitalio import DigitalInOut, Direction, Pull
led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
switch = DigitalInOut(board.D1)
switch.direction = Direction.INPUT
switch.pull = Pull.DOWN
while True:
    if switch.value == 1:
        led.value = 1
    else:
        led.value = 0
```

When the switch is "on" the signal at D1 is connected to high (5V)

SNIPPET #2

See if you can control when the LED light goes on.

Plug in your test button to see how it can be used as another input device.



SIMPLE I/O BUTTON DETECTOR

BOOT → LIBRARIES - Board - Digital I/O

SETUP - Name "led" - Input - Name "button" - Output

LOOP - While Loop

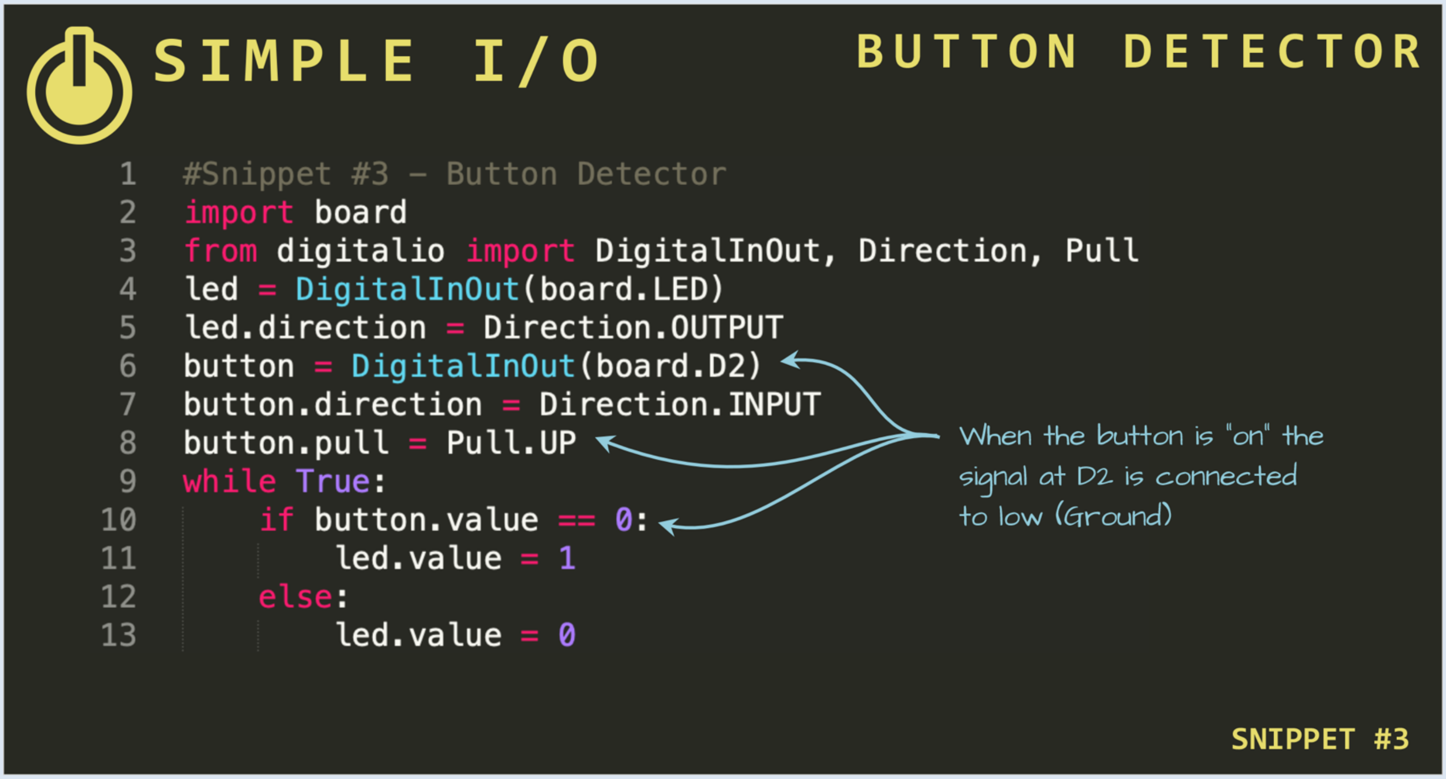
BUTTON → IF "0" → LED ON

or IF "Down"
or IF "Pushed"

ELSE

LED OFF

SNIPPET #3



SIMPLE I/O BUTTON DETECTOR

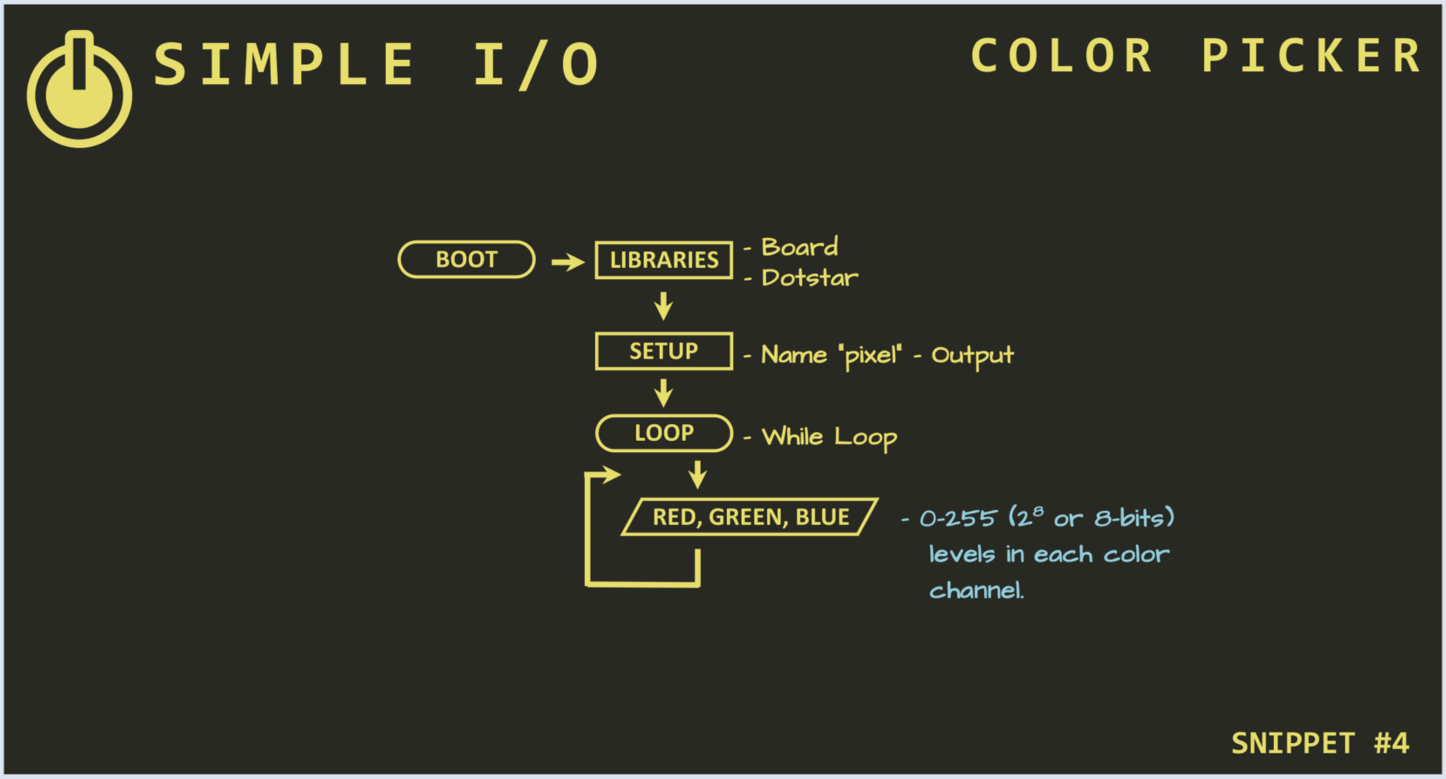
```
#Snippet #3 – Button Detector
import board
from digitalio import DigitalInOut, Direction, Pull
led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
button = DigitalInOut(board.D2)
button.direction = Direction.INPUT
button.pull = Pull.UP
while True:
    if button.value == 0:
        led.value = 1
    else:
        led.value = 0
```

When the button is "on" the signal at D2 is connected to low (Ground)

SNIPPET #3

See if you can use the button to turn off the LED instead of turning it on.

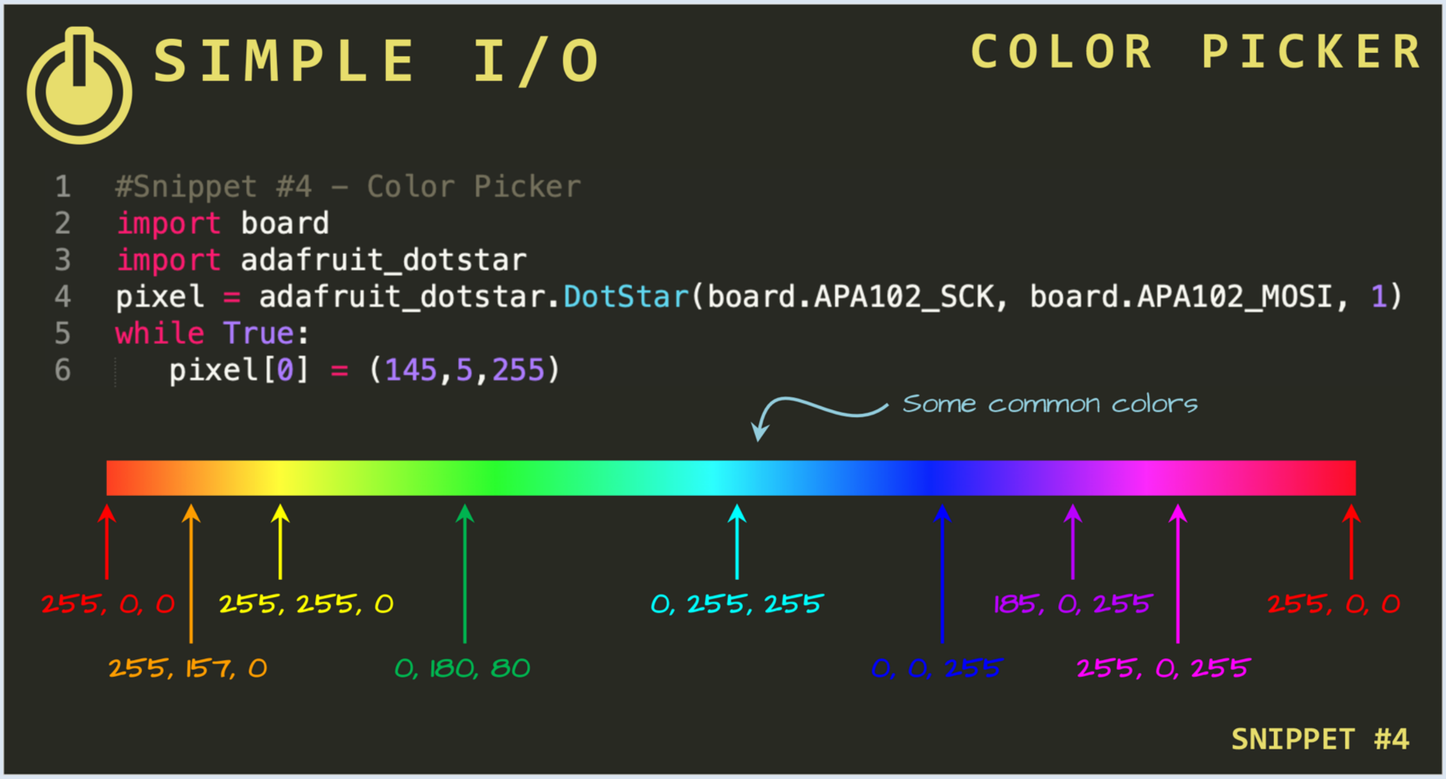
This code lets you control the color of light emitted by the internal neopixel.



SIMPLE I/O

COLOR PICKER

SNIPPET #4



SIMPLE I/O

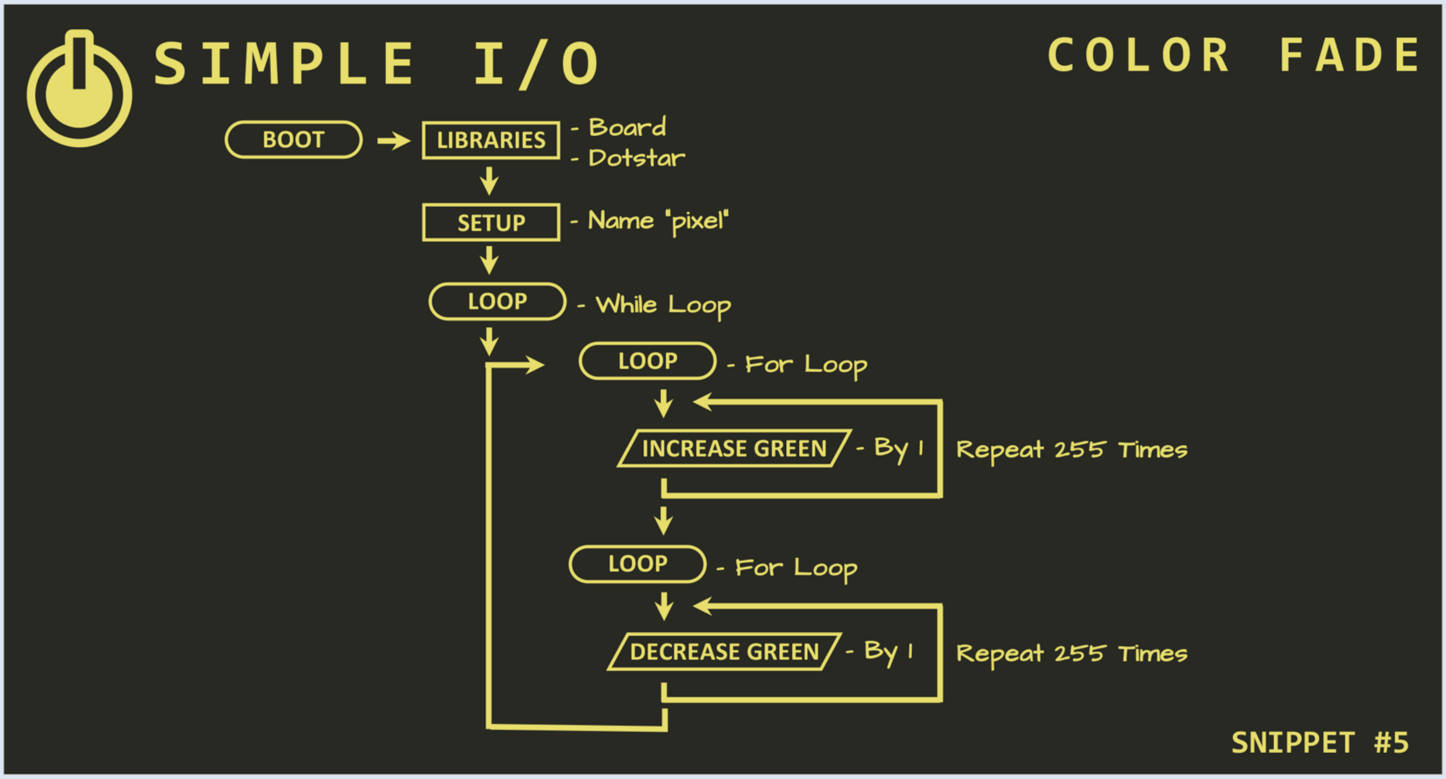
COLOR PICKER

```
#Snippet #4 - Color Picker
import board
import adafruit_dotstar
pixel = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
while True:
    pixel[0] = (145,5,255)
```

SNIPPET #4

See if you can produce your favorite color.

This code uses FOR LOOPS to make colors fade in or out.



SIMPLE I/O

COLOR FADE

BOOT → LIBRARIES - Board - Dotstar

SETUP - Name 'pixel'

LOOP - While Loop

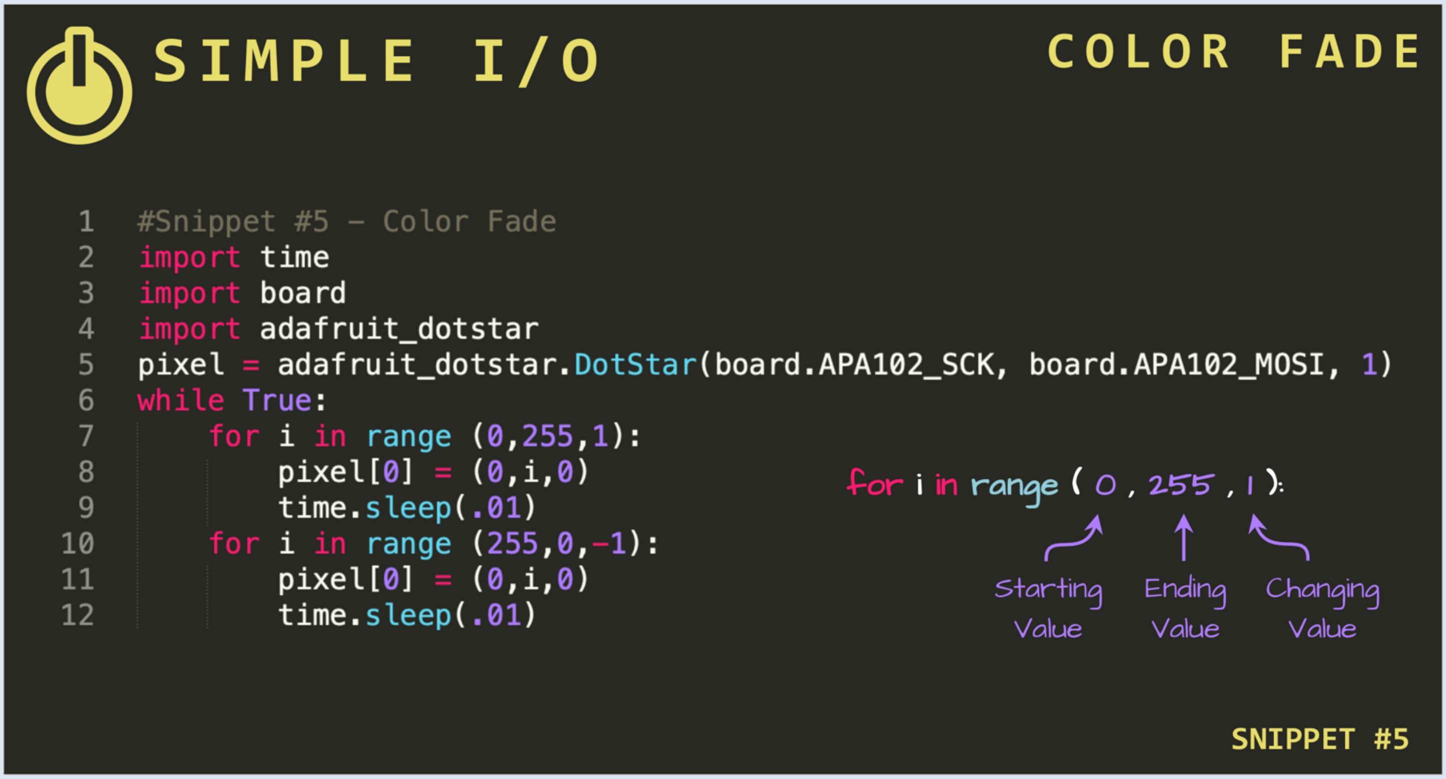
LOOP - For Loop

INCREASE GREEN - By 1 Repeat 255 Times

LOOP - For Loop

DECREASE GREEN - By 1 Repeat 255 Times

SNIPPET #5



SIMPLE I/O

COLOR FADE

```python
#Snippet #5 – Color Fade
import time
import board
import adafruit_dotstar
pixel = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
while True:
    for i in range (0,255,1):
        pixel[0] = (0,i,0)
        time.sleep(.01)
    for i in range (255,0,-1):
        pixel[0] = (0,i,0)
        time.sleep(.01)
```

for i in range (0 , 255 , 1):

Starting Value

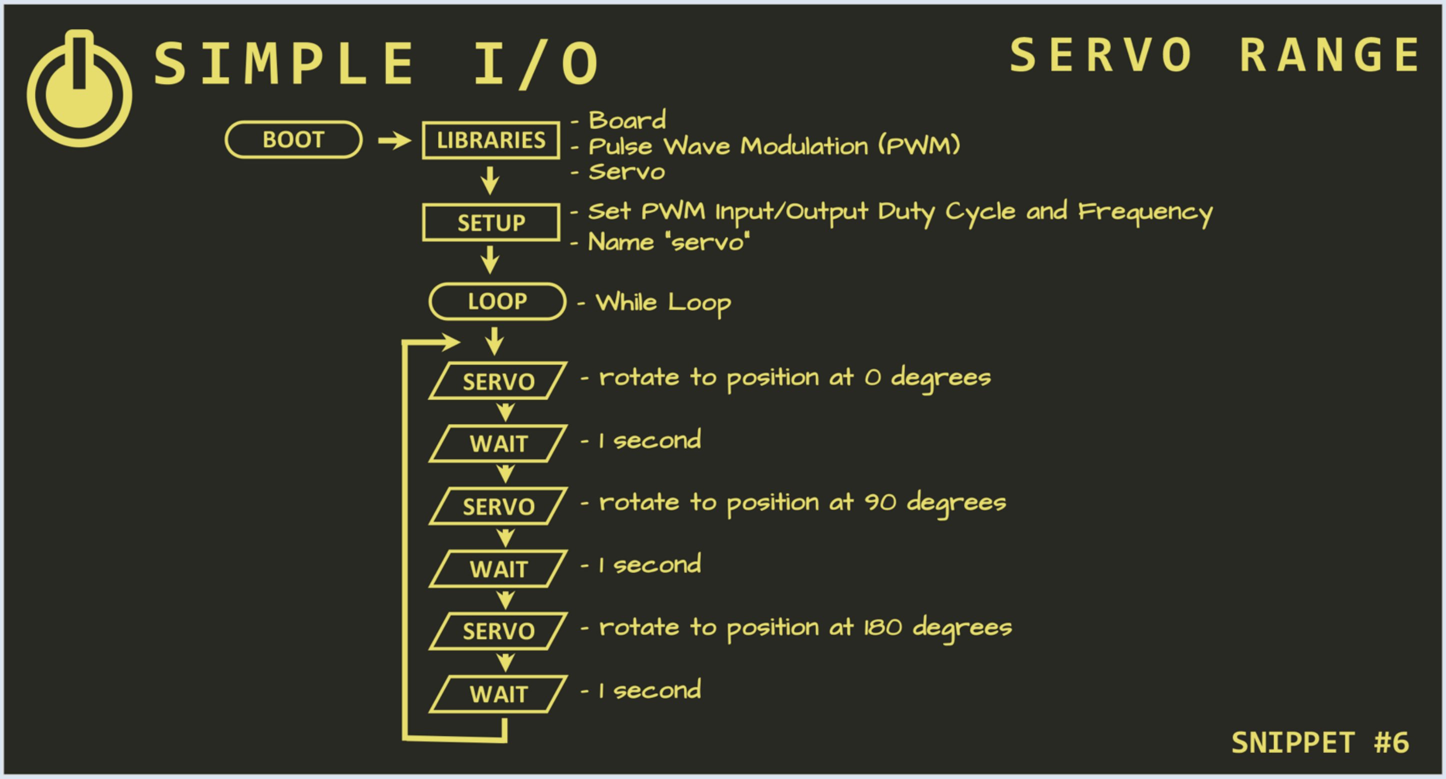
Ending Value

Changing Value

SNIPPET #5

See if you can get one color to fade into another.

This code controls the position of the servo motor.



SIMPLE I/O

SERVO RANGE

BOOT → LIBRARIES
- Board
- Pulse Wave Modulation (PWM)
- Servo

SETUP
- Set PWM Input/Output Duty Cycle and Frequency
- Name 'servo'

LOOP - While Loop

SERVO - rotate to position at 0 degrees

WAIT - 1 second

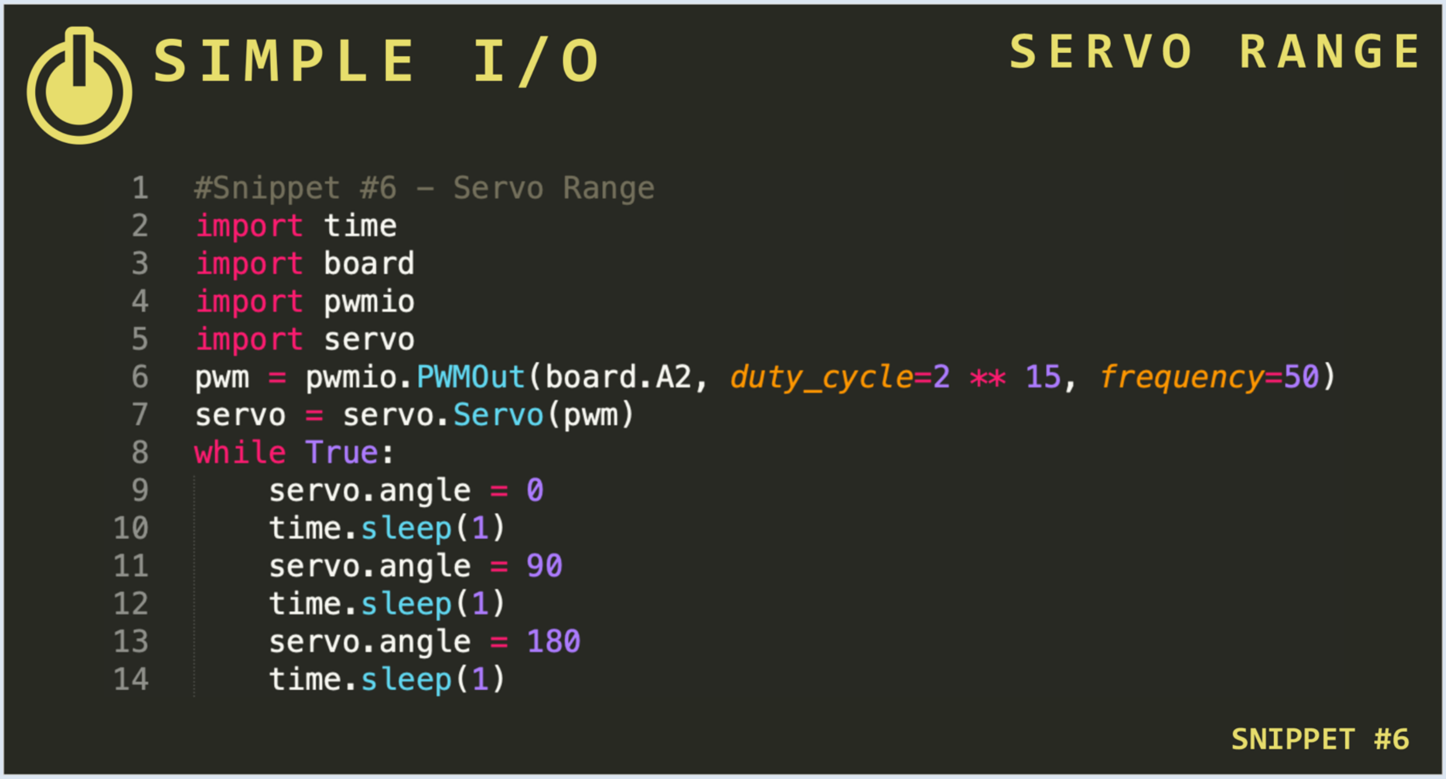
SERVO - rotate to position at 90 degrees

WAIT - 1 second

SERVO - rotate to position at 180 degrees

WAIT - 1 second

SNIPPET #6



SIMPLE I/O

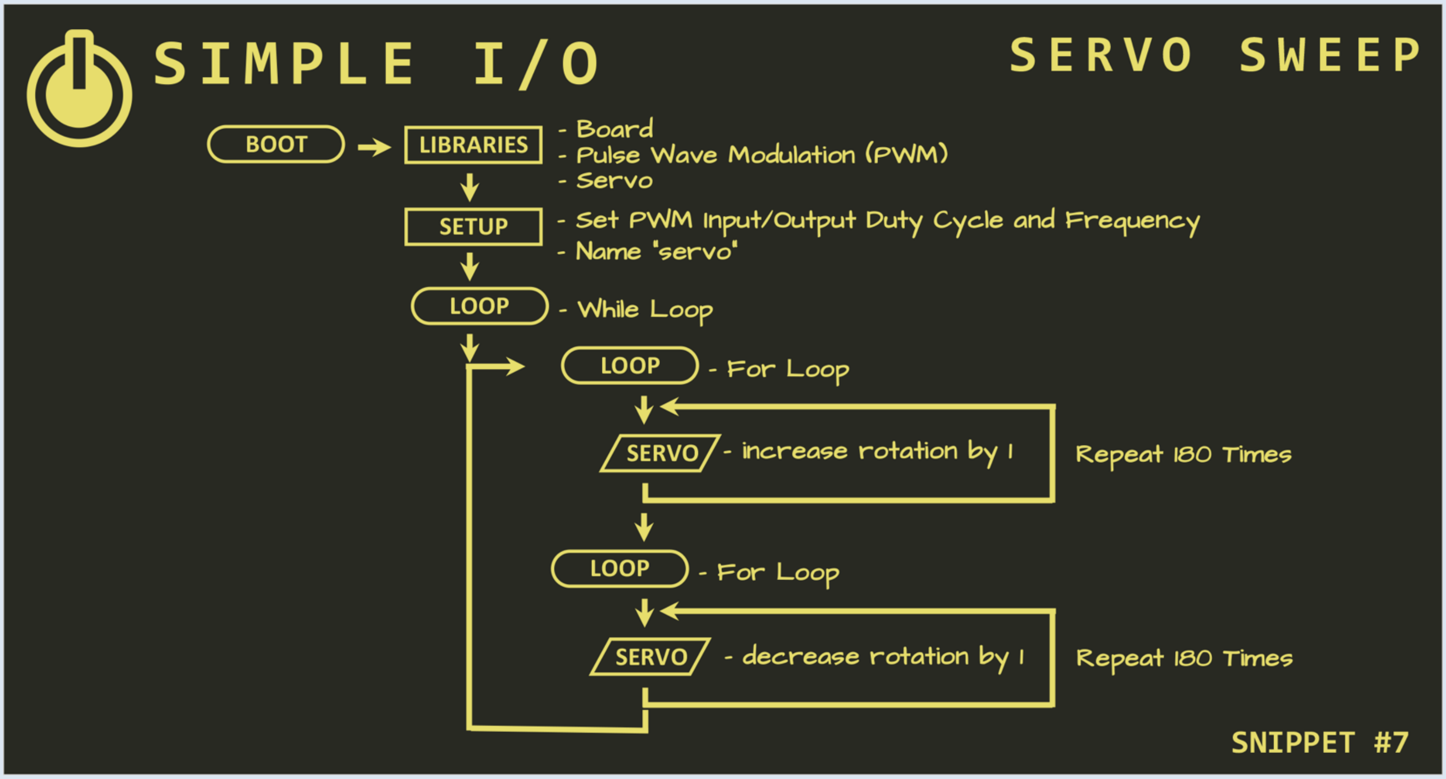
SERVO RANGE

```
#Snippet #6 - Servo Range
import time
import board
import pwmio
import servo
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm)
while True:
    servo.angle = 0
    time.sleep(1)
    servo.angle = 90
    time.sleep(1)
    servo.angle = 180
    time.sleep(1)
```

SNIPPET #6

See if you can get the servo to move in an interesting or complex pattern.

This code uses a FOR LOOP to get the servo to change positions in steps.



SIMPLE I/O SERVO SWEEP

BOOT → LIBRARIES - Board - Pulse Wave Modulation (PWM) - Servo

SETUP - Set PWM Input/Output Duty Cycle and Frequency - Name 'servo'

LOOP - While Loop

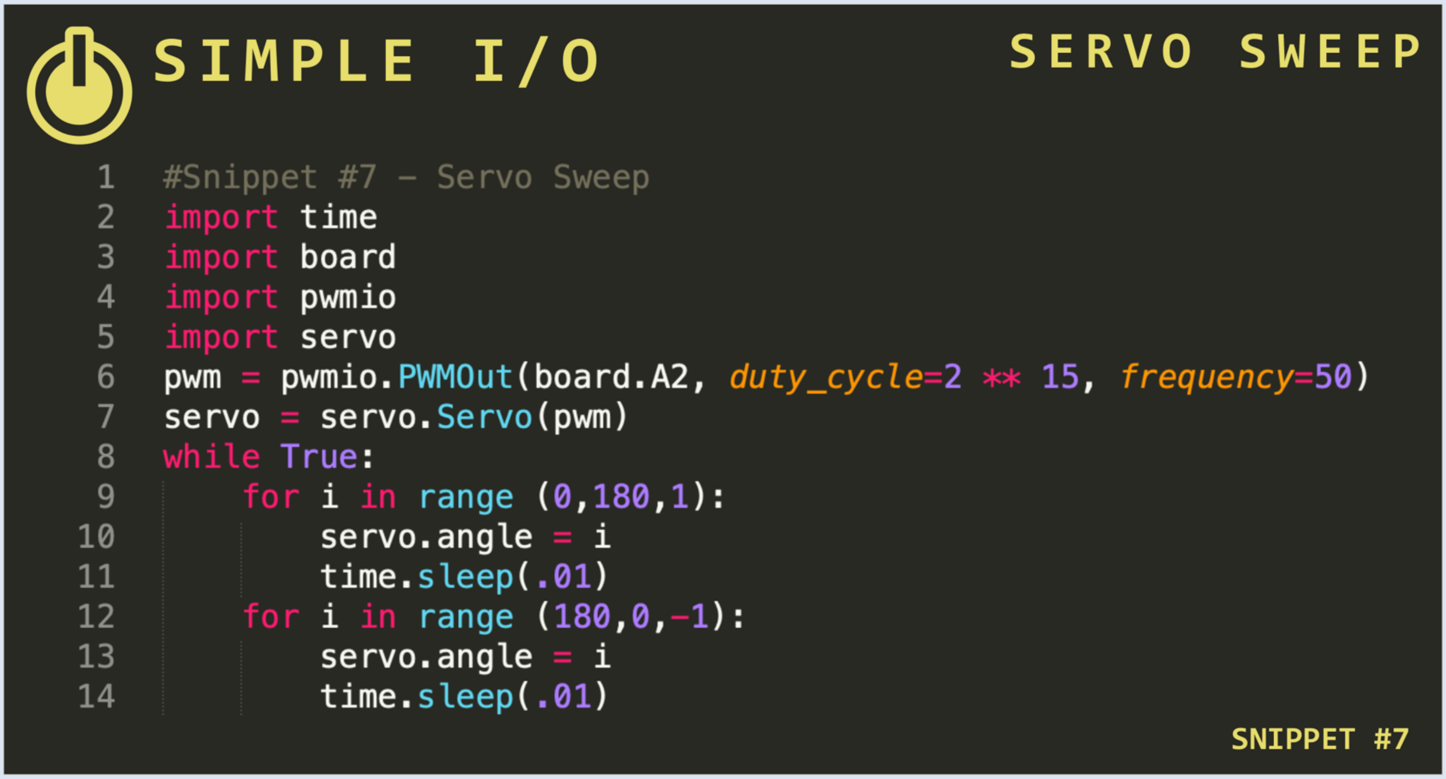
LOOP - For Loop

SERVO - increase rotation by 1 Repeat 180 Times

LOOP - For Loop

SERVO - decrease rotation by 1 Repeat 180 Times

SNIPPET #7



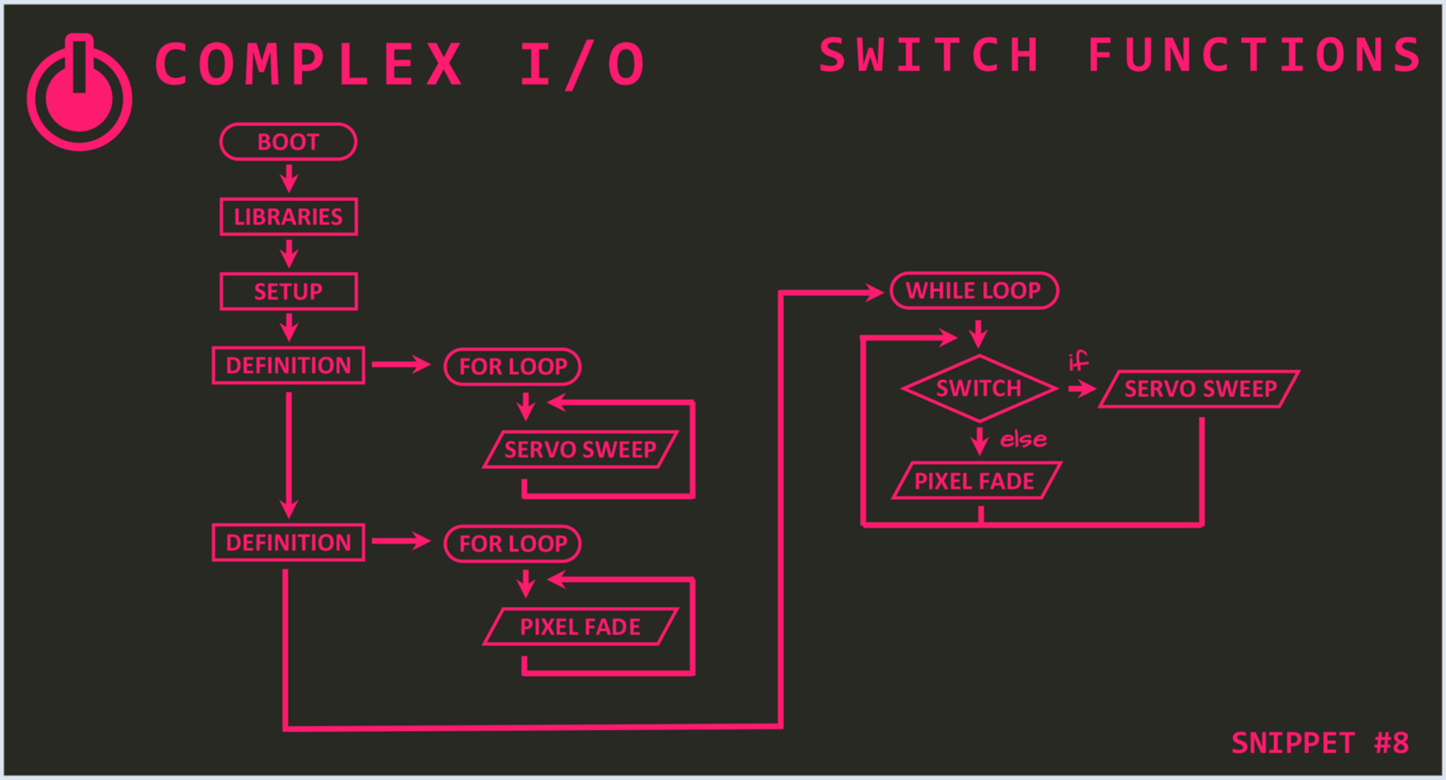
SIMPLE I/O SERVO SWEEP

```
#Snippet #7 - Servo Sweep
import time
import board
import pwmio
import servo
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm)
while True:
    for i in range (0,180,1):
        servo.angle = i
        time.sleep(.01)
    for i in range (180,0,-1):
        servo.angle = i
        time.sleep(.01)
```

SNIPPET #7

See if you can change the speed and amount of rotation.

This code defines functions that are initiated with the toggle switch.

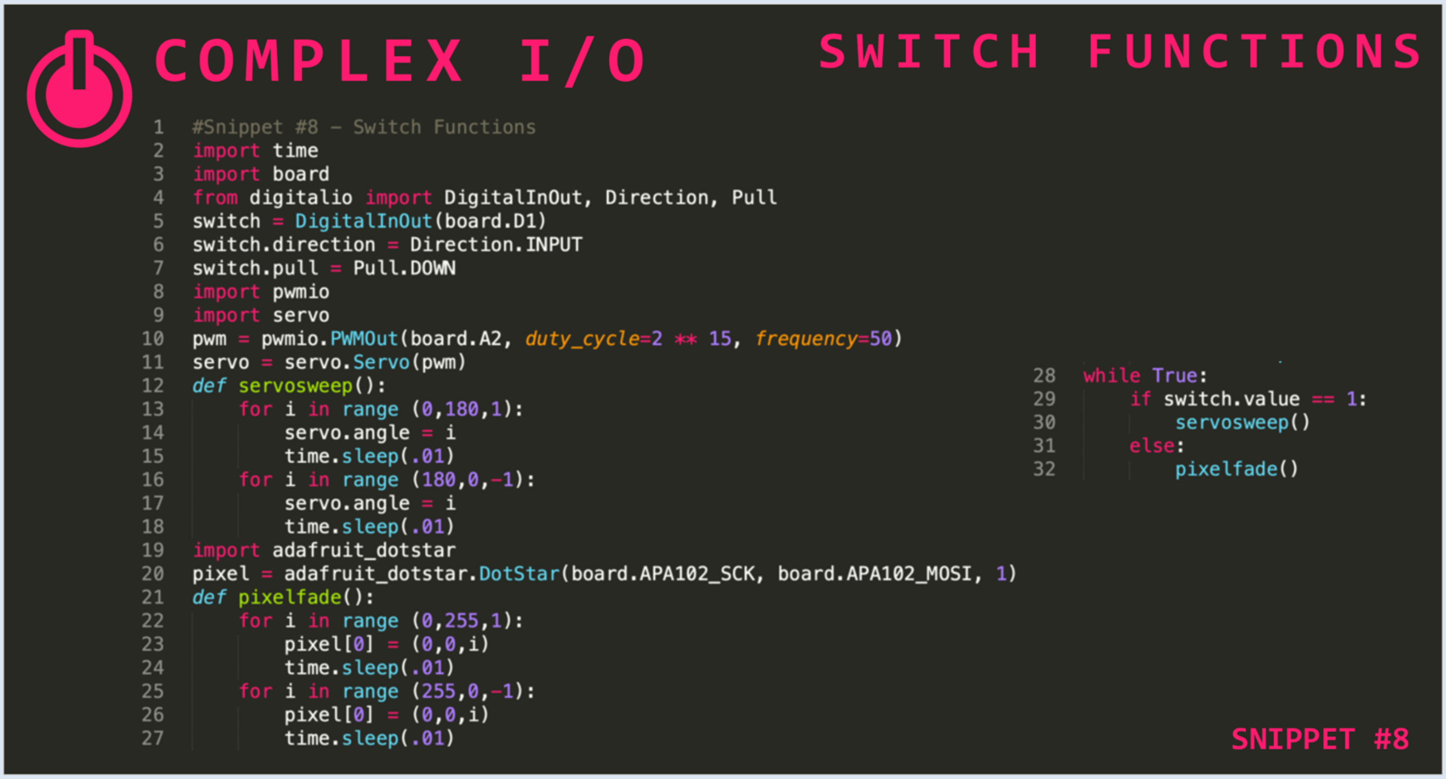


COMPLEX I/O

SWITCH FUNCTIONS

BOOT
LIBRARIES
SETUP
DEFINITION
FOR LOOP
SERVO SWEEP
DEFINITION
FOR LOOP
PIXEL FADE
WHILE LOOP
SWITCH
if
SERVO SWEEP
else
PIXEL FADE

SNIPPET #8



COMPLEX I/O

SWITCH FUNCTIONS

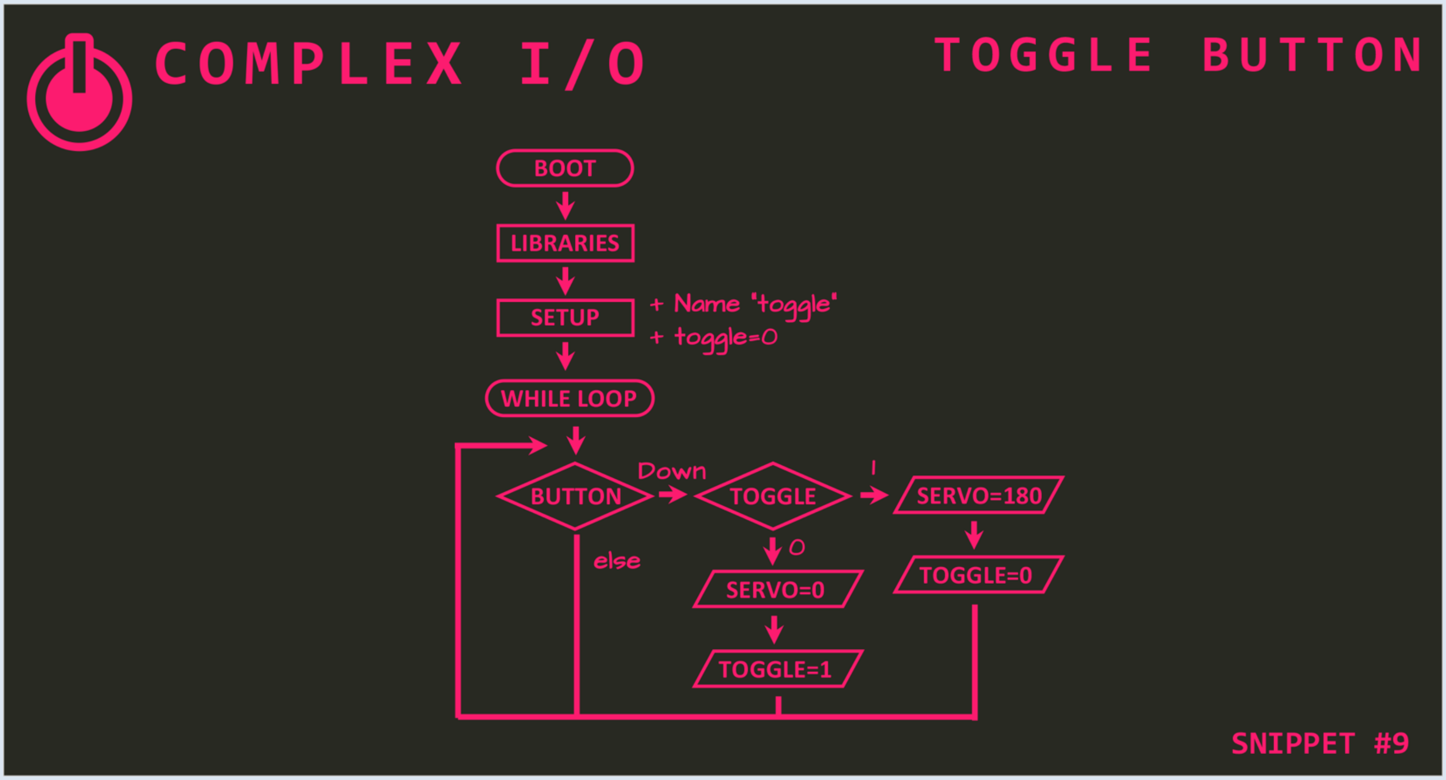
```python
#Snippet #8 - Switch Functions
import time
import board
from digitalio import DigitalInOut, Direction, Pull
switch = DigitalInOut(board.D1)
switch.direction = Direction.INPUT
switch.pull = Pull.DOWN
import pwmio
import servo
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm)
def servosweep():
    for i in range (0,180,1):
        servo.angle = i
        time.sleep(.01)
    for i in range (180,0,-1):
        servo.angle = i
        time.sleep(.01)
import adafruit_dotstar
pixel = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
def pixelfade():
    for i in range (0,255,1):
        pixel[0] = (0,0,i)
        time.sleep(.01)
    for i in range (255,0,-1):
        pixel[0] = (0,0,i)
        time.sleep(.01)
```

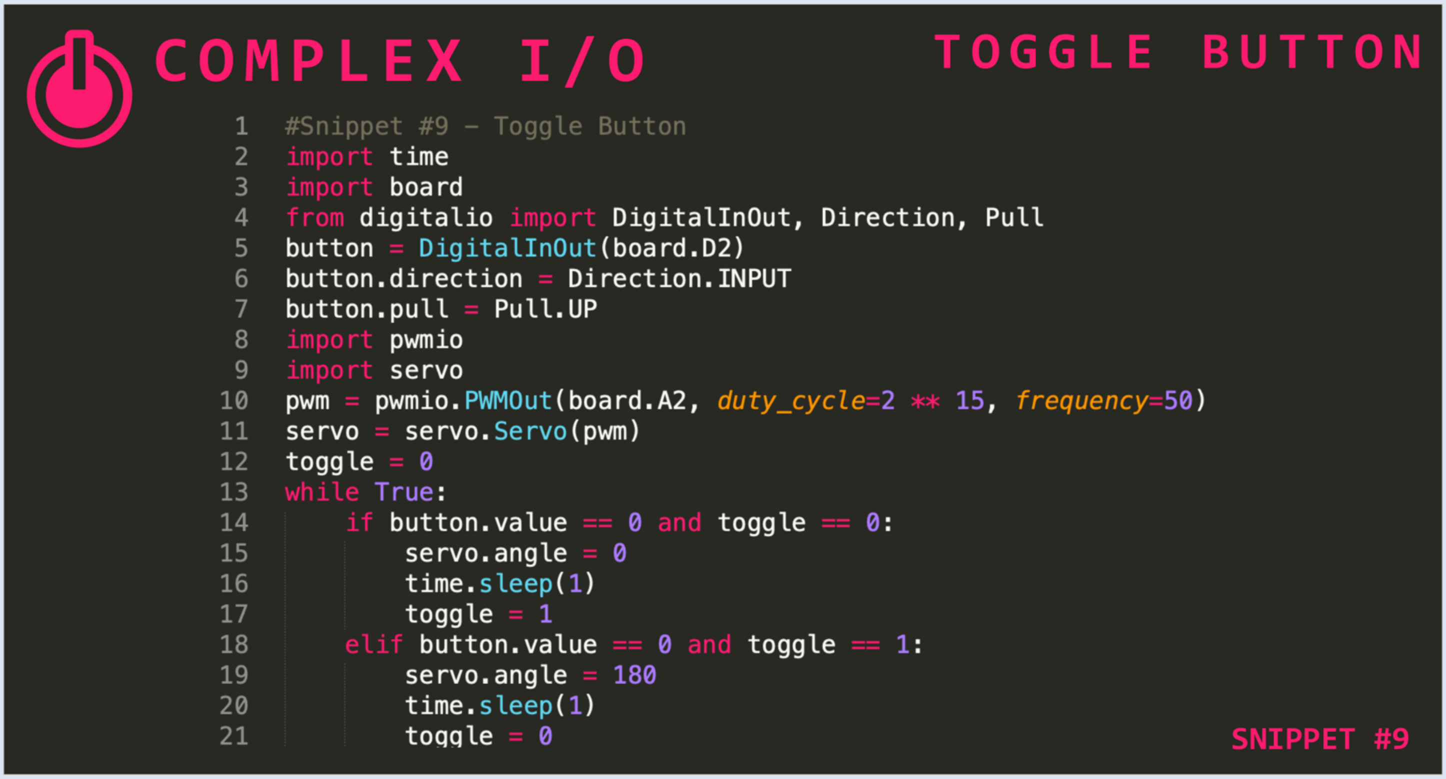
```python
while True:
    if switch.value == 1:
        servosweep()
    else:
        pixelfade()
```

SNIPPET #8

See if you can modify these functions.

This code changes the button input from push for “on” and let go for “off” to push once for “on” and push again for “off”.




COMPLEX I/O

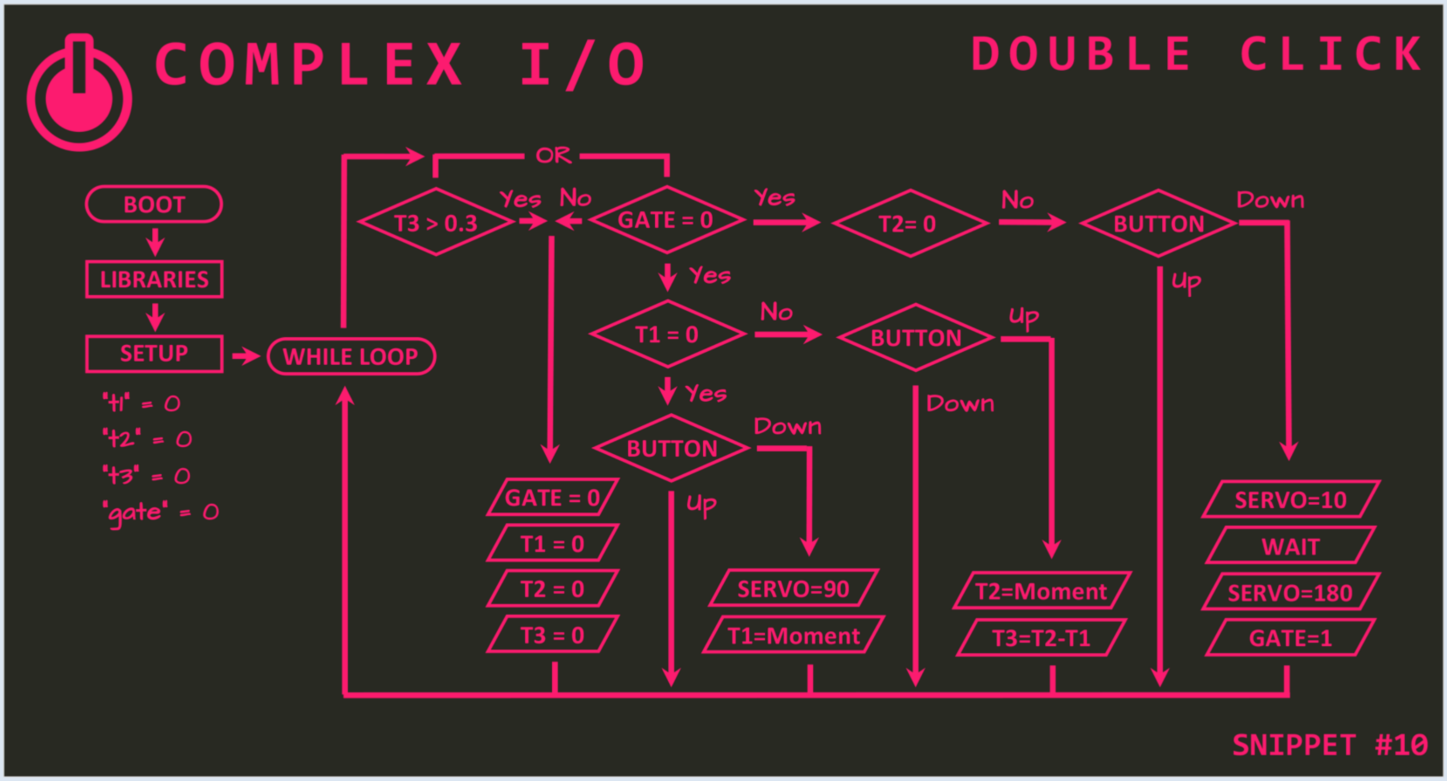
TOGGLE BUTTON

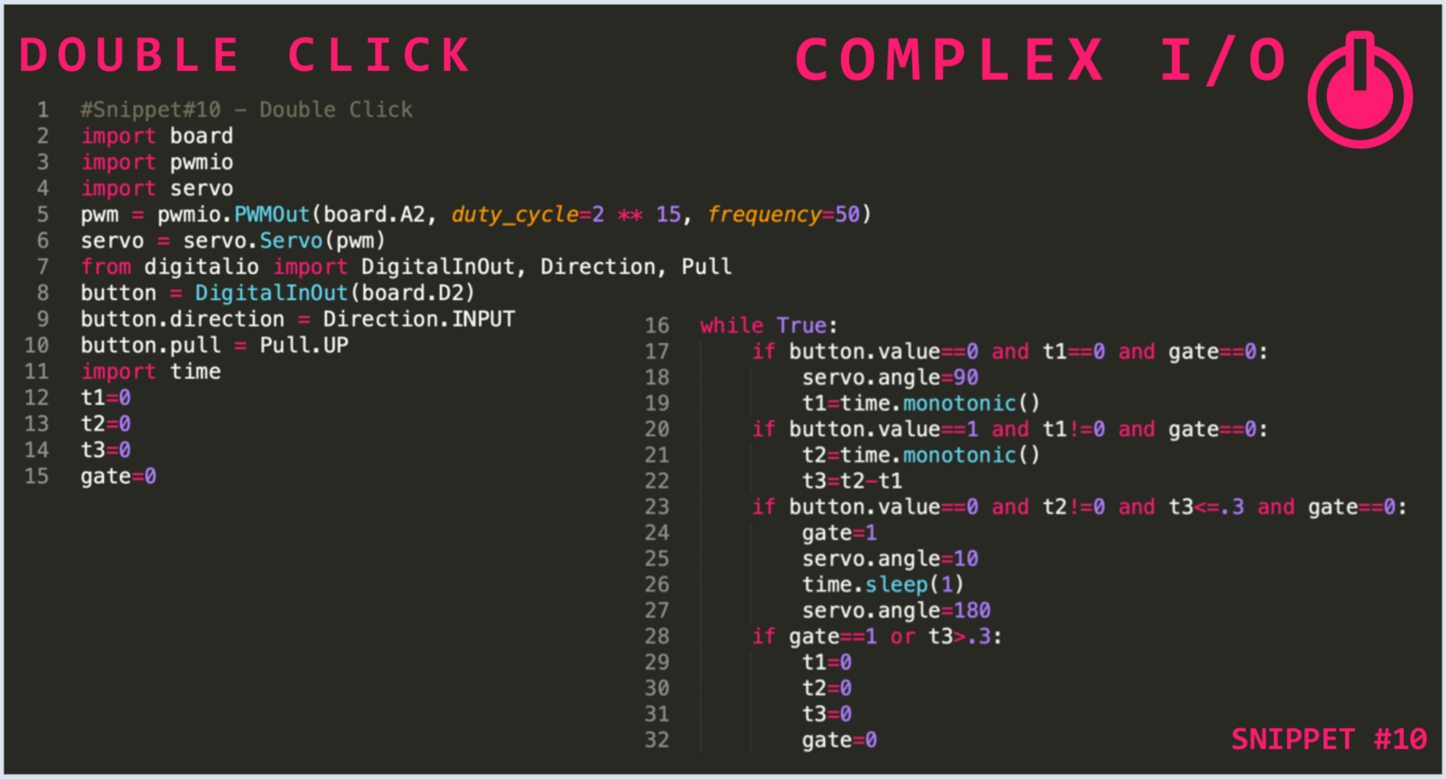
```
#Snippet #9 - Toggle Button
import time
import board
from digitalio import DigitalInOut, Direction, Pull
button = DigitalInOut(board.D2)
button.direction = Direction.INPUT
button.pull = Pull.UP
import pwmio
import servo
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm)
toggle = 0
while True:
    if button.value == 0 and toggle == 0:
        servo.angle = 0
        time.sleep(1)
        toggle = 1
    elif button.value == 0 and toggle == 1:
        servo.angle = 180
        time.sleep(1)
        toggle = 0
```

SNIPPET #9

See if you can make the servo do interesting things with this toggle mode.

This code expands the button input options by introducing a double-click.





```python
#Snippet#10 - Double Click
import board
import pwmio
import servo
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm)
from digitalio import DigitalInOut, Direction, Pull
button = DigitalInOut(board.D2)
button.direction = Direction.INPUT
button.pull = Pull.UP
import time
t1=0
t2=0
t3=0
gate=0
while True:
    if button.value==0 and t1==0 and gate==0:
        servo.angle=90
        t1=time.monotonic()
    if button.value==1 and t1!=0 and gate==0:
        t2=time.monotonic()
        t3=t2-t1
    if button.value==0 and t2!=0 and t3<=.3 and gate==0:
        gate=1
        servo.angle=10
        time.sleep(1)
        servo.angle=180
    if gate==1 or t3>.3:
        t1=0
        t2=0
        t3=0
        gate=0
```

See if you can tweak how fast you need to double click to unlock an interesting action.

CONGRATULATIONS!

You've now gone through the steps to plug in and use your Smart Servo and you've learned to change and alter the code built into the Smart Servo. You're now ready to start adding simple mechanisms that will allow your Smart Servo to do so much more.

REMINDER ABOUT CODING SNIPS

If you want to return your code to the original "factory setting", just copy and paste from here: tinyurl.com/SmartServoSnips

THE BIGGER PICTURE

UNDERSTANDING PHYSICAL COMPUTING WITH YOUR SMART SERVO

What is Physical Computing?

When you modified those code snippets to control your Smart Servo, you were doing something fundamentally different from regular programming. Traditional computer programs work entirely with information - processing data, displaying images, or sending messages. But your Smart Servo connects code to the physical world. When you write a command in CircuitPython, something actually moves in real space. This bridge between digital code and physical action is called physical computing.

Your Smart Servo is a complete physical computing system in one package. The microcontroller runs your code, reads input from the assistive switch port, checks the kill switch status, and sends precise electrical signals to control the servo motor - all coordinating together to transform your programming logic into real movement. Every robot, smart home device, or interactive installation uses these same principles: sensors detect the physical world, code makes decisions, and actuators (like your servo) create physical responses.

Where Physical Computing Changes the World

Physical computing is everywhere once you start looking for it. The automatic doors at stores use motion sensors and motors. Fitness trackers read your heartbeat and movement to display information. 3D printers translate digital designs into physical objects layer by layer. Self-driving cars use dozens of sensors and actuators working together, making thousands of decisions per second to navigate safely.

What makes physical computing powerful is the feedback loop between sensing and action. You experienced this when you used the assistive switch input - the physical world (someone pressing a button) changed what your code did, which changed how the servo moved. This sense-think-act cycle is the foundation of robotics, automation, and interactive technology. The loops and conditionals you practiced aren't just programming concepts - they're the logic that lets machines respond intelligently to an unpredictable physical world.