Programming Basics Part 1

LEDs and Buzzers

Learn About

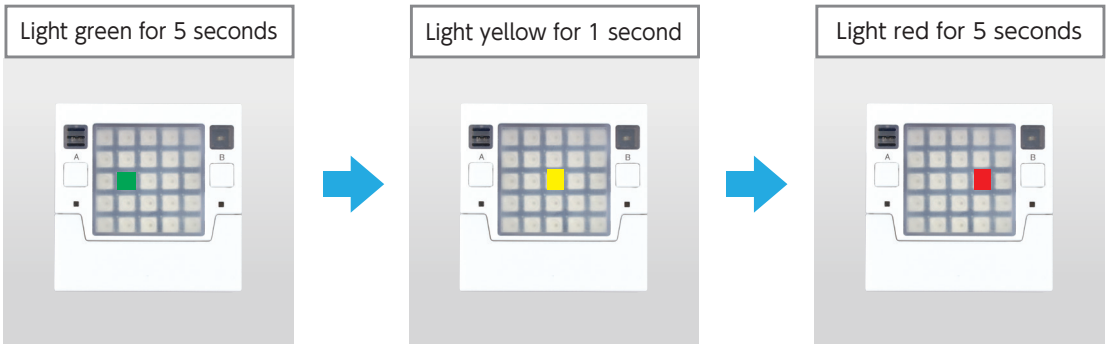
- Programming LEDs and Buzzers
- Programming Sequences
- Loops
- Conditions

Teacher's Edition

Index

1. Programming Sequences

A sequential program is a program that gives a series of commands to the computer in the same order they're written in. Let's make a traffic light using LEDs and program it to work using a sequential program!



1.1 Building a Traffic Light

1.2 Lighting Up an LED

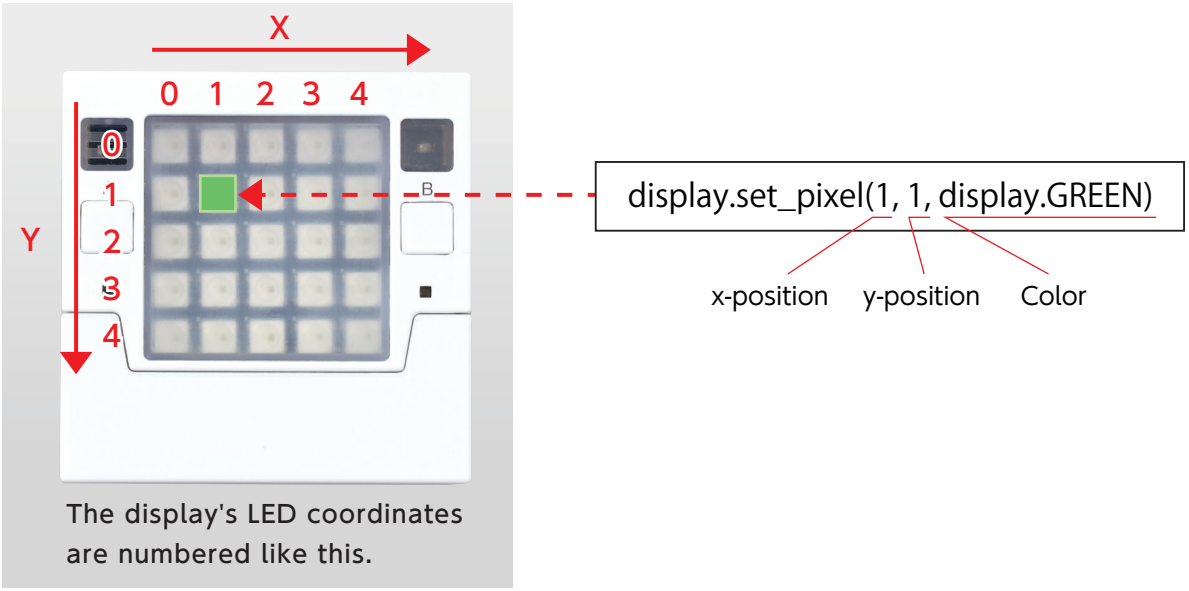
Here's a program you can use to make an LED light up green.

```
1:  from pystubit.board import display
2:
3:  display.set_pixel(1, 1, display.GREEN)
```

We'll be using the Studuino:bit library to control the hardware parts in the Core Unit.
Line 1 of this program, **from pystubit.board import display**, imports a **display** object from the Studuino:bit library's **pystubit.board** module. **display** objects have methods you can use to turn the LED display on and off, data that describes different light colors, and more. See **Appendix A** to learn more about using **display** objects.
The first method we'll be using from the **display** object is **set_pixel()**. **set_pixel()** is a method that specifies a single LED from the display and turns it on or off. It's defined like this:

```
set_pixel(x, y, color)
```

The parameters in **set_pixel** first specify the position of the LED, and then what color it should light up in. Line 3 of the program above, **display.set_pixel(1, 1, display.GREEN),** makes the LED at the x-y coordinates (**1, 1**) in the display light up green (**display.GREEN**).



The display's LED coordinates are numbered like this.

The code **display.GREEN** uses a property from the **display** object called **GREEN** to describe the color green to the computer.
Now **Transfer** your program to the Core Unit and see if it works!

1.3 Lighting Up an LED for 1 Second

The program below will make an LED light up in green for exactly 1 second.

```
1:  from pystubit.board import display
2:  import time
3:
4:  display.set_pixel(1, 1, display.GREEN)
5:  time.sleep(1)
6:  display.clear()
```

Line 2, **import time**, tells the program you'll be using the **time** module from the standard Python library. The **time** module has objects and functions you can use to keep track of how long your Core Unit has been turned on, make the unit pause/go to sleep, etc. Its **sleep()** function pauses the program for the number of seconds you specify in its parameter. So line 5, **time.sleep(1)**, make the program pause for 1 second! Line 6, **display.clear()**, turns off all the LEDs in the LED display. The **clear** method has no parameters for you to set, so it doesn't have anything written inside its parentheses.

Changing the value in **time.sleep(1)**'s parameter lets you set how many seconds the LED will stay lit for. You can set this parameter using floating-point numbers too, not just integers, so pick any number you like!

Your program will always run from top to bottom.

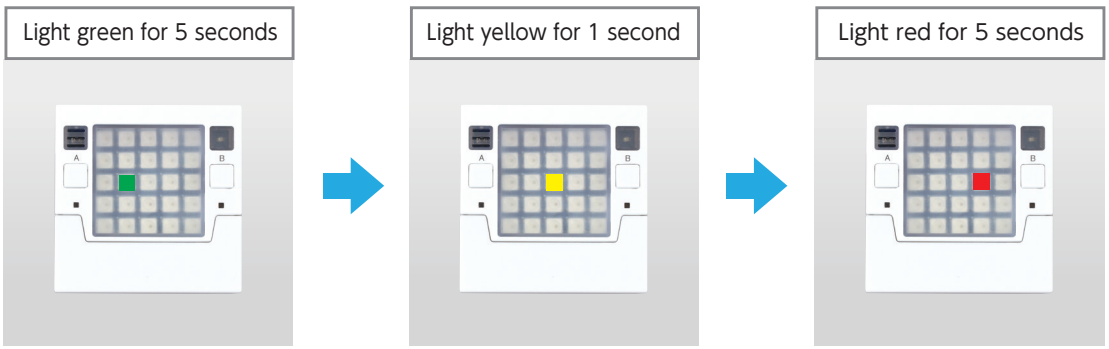
```
1:  from pystubit.board import display
2:  import time
3:
4:  display.set_pixel(1, 1, display.GREEN)
5:  time.sleep(1)
6:  display.clear()
```

Do you know why we need the **time.sleep(1)** line? If you run a program without it (like the one on the next page), the LED basically won't light up! This happens because the Core Unit runs through programs very fast, so as soon as it sends the command to turn the LED on it immediately sends the following command to turn the LED off. The result is that the LED turns off as soon as it turns on!

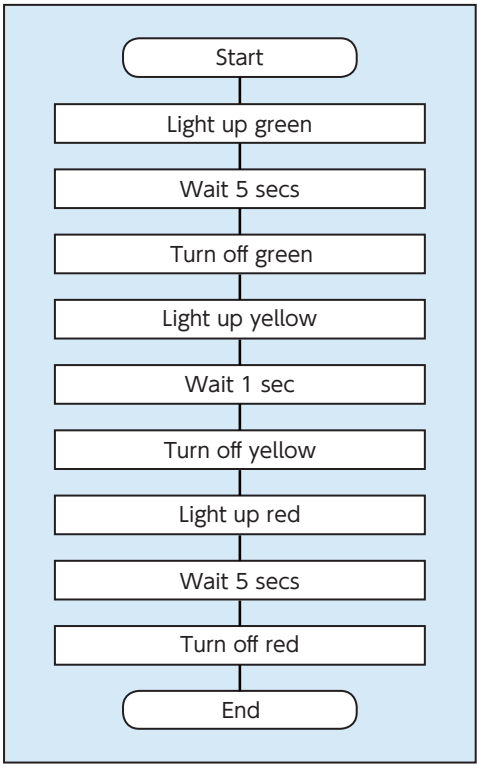
```
1:  from pystubit.board import display
2:  import time
3:
4:  display.set_pixel(1, 1, display.GREEN)
5:  display.clear()
```

Programming Practice: Programming a Traffic Light

Draw up a flowchart that breaks down the lighting sequence shown below, then see if you can write a program for it!



Example Program




```
1:  from pystubit.board import display
2:  import time
3:
4:  display.set_pixel(1, 2, display.GREEN)
5:  time.sleep(5)
6:  display.clear()
7:  display.set_pixel(2, 2, display.YELLOW)
8:  time.sleep(1)
9:  display.clear()
10: display.set_pixel(3, 2, display.RED)
11: time.sleep(5)
12: display.clear()
```

2. Programming Loops

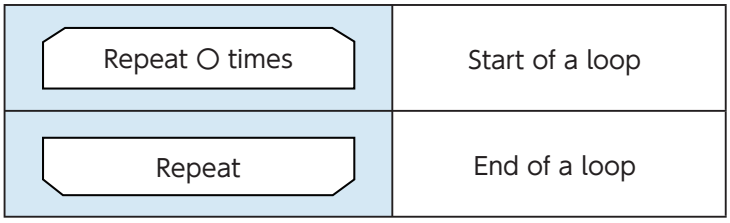
A program that repeats the sequence of commands inside of it over and over again is called a **loop**. The traffic light program we made on the last page turns the light off after it finishes lighting up red, and the program completely stops after that. To make a traffic light that can go through the same sequence of lights over and over like a real one does, we'll need to use a loop.

There are two main kinds of loop programs in Python; **for** loops that repeat a set number of times or until they finish processing a set amount of information, and **while** loops that keep going until a particular condition is fulfilled.

2.1 Flowcharts and Python Syntax

Let's see how to write out a loop that repeats a set number of times, both as a flowchart and in Python. When you use a **for** statement in Python, make sure to end it with a colon (:). It's also important to use blank spaces like tabs and spaces correctly since Python uses them to organize its programs. You'll need to press Space four times at the start of each line of the program you want to loop to indent them.

Flowchart



Python

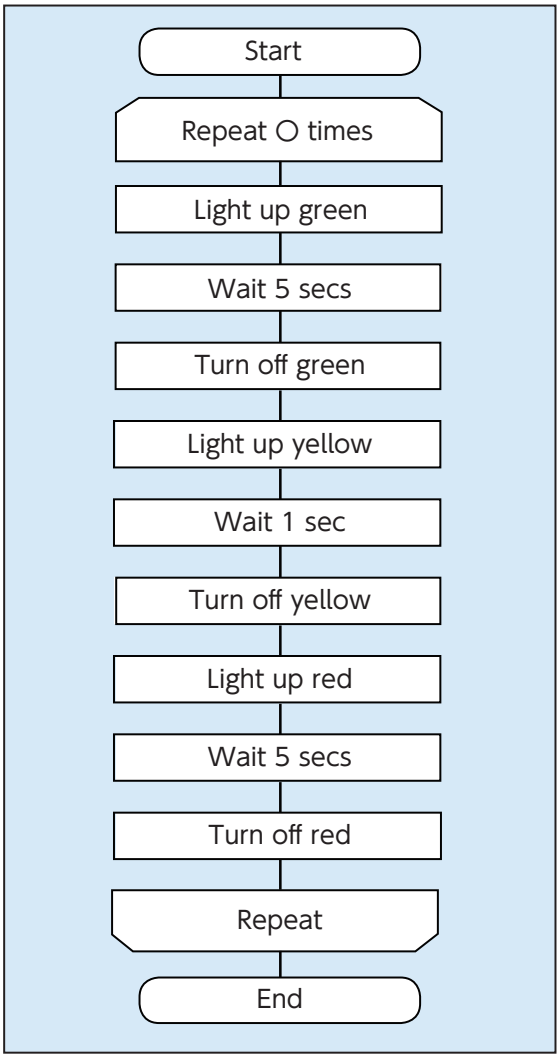
```
for Variable in range(Number of Loops):
    Program you want to loop
```

Indent every line of the program you want to loop!

Now let's take the traffic signal program we made in the last section and write is as a loop using a flowchart and Python **pseudocode**(*), too. In the program below we want to make the whole traffic light program loop, so each line of it is indented.

(*): **Pseudocode** is code you write out using words to explain how a program will work before you write it in actual code. There aren't any set rules for writing pseudocode, so write it however makes sense to you!

Flowchart



Python

```
for Variable in range(# of Loops):
    Light up green
    Wait 5 seconds
    Turn off green
    Light up yellow
    Wait 1 second
    Turn off yellow
    Light up red
    Wait 5 seconds
    Turn off red
```

Indent

2.2 Improving Your Program (Adding Loops)

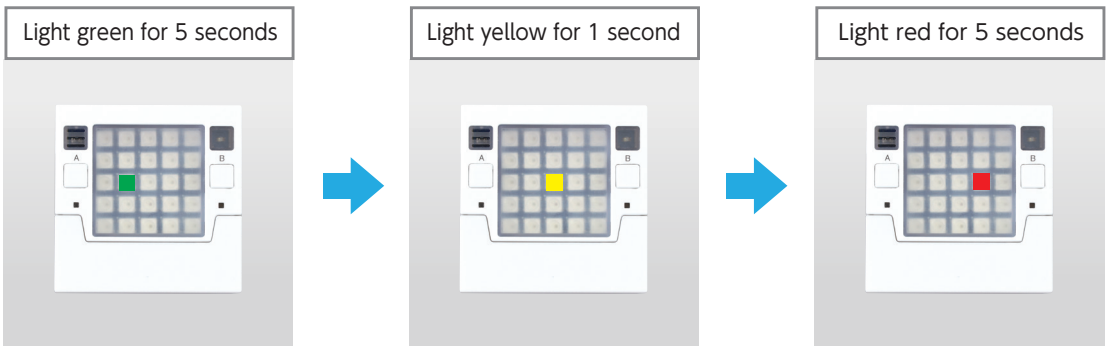
Here's a program you can use to make your traffic signal program loop ten times!

```
 1:  from pystubit.board import display
 2:  import time
 3:
 4:  for i in range(10):
 5:      display.set_pixel(1, 2, display.GREEN)
 6:      time.sleep(5)
 7:      display.clear()
 8:      display.set_pixel(2, 2, display.YELLOW)
 9:      time.sleep(1)
10:      display.clear()
11:      display.set_pixel(3, 2, display.RED)
12:      time.sleep(5)
13:      display.clear()
```

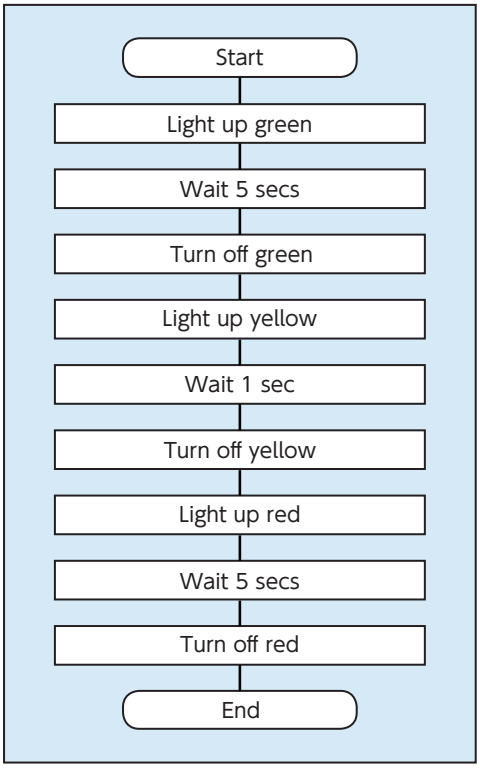
```
1:  from pystubit.board import display
2:  import time
3:
4:  display.set_pixel(1, 1, display.GREEN)
5:  display.clear()
```

Programming Practice:　Programming a Traffic Light

Draw up a flowchart that breaks down the lighting sequence shown below, then see if you can write a program for it!



Example Program




```
1:  from pystubit.board import display
2:  import time
3:
4:  display.set_pixel(1, 2, display.GREEN)
5:  time.sleep(5)
6:  display.clear()
7:  display.set_pixel(2, 2, display.YELLOW)
8:  time.sleep(1)
9:  display.clear()
10: display.set_pixel(3, 2, display.RED)
11: time.sleep(5)
12: display.clear()
```

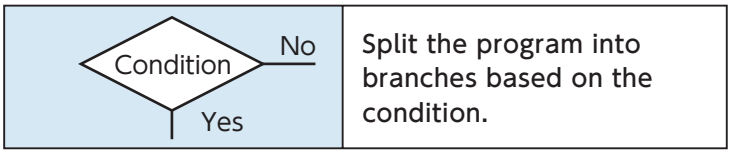
3. Programming Conditions

A **condition** is a statement that makes a program carry out different commands depending on whether it's true or false. In this section, we'll learn about conditions and how they make programs branch in different directions by programming a push-signal pedestrian traffic light! In Python, conditions are created using **if-else** statements.

3.1 Flowcharts and Python Syntax

Let's see how to write conditions in flowcharts and in Python. In the Python version, both possible branches of the program should be indented!

Flowchart



Python

```
if Conditional Statement:
↔Program that runs when the condition is fulfilled.
else:
↔Program that runs when the condition is NOT fulfilled.
```

Both of the programs you want to run should be indented!

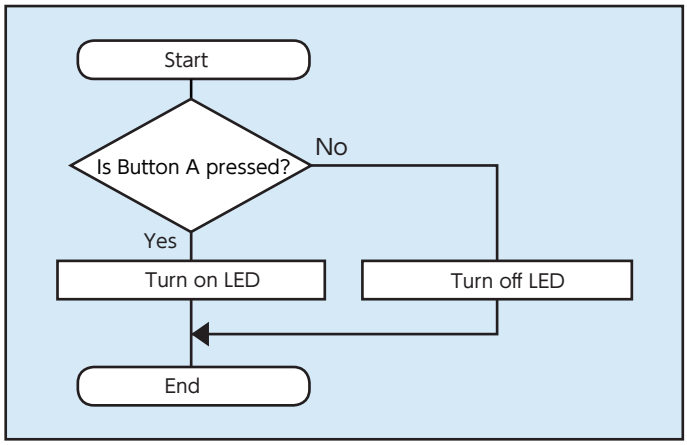
You can write conditional statements like this in Python by using relational operators like these:

Relational Operator	Conditional Statement	Meaning
==	A == B	A is equal to B
!=	A != B	A is not equal to B
>	A > B	A is greater than B
>=	A >= B	A is greater than or equal to B
<	A < B	A is less than B
<=	A <= B	A is less than or equal to B

3.2 Pressing Button A to Light Up an LED

You can make this happen with the following flowchart or pseudocode.

Flowchart



Python

```
if Button A is pressed:
↔Turn on LED
else :
↔Turn off LED
```

3.3 Values from Buttons

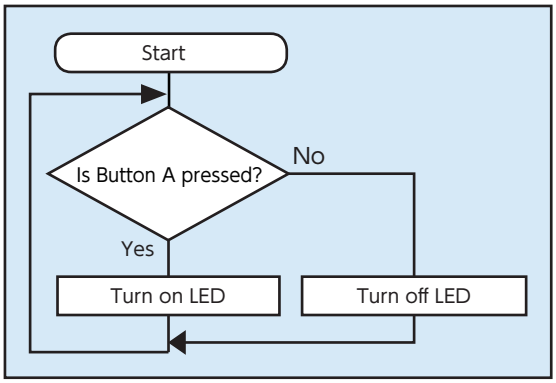
button_a objects from the Studuino:bit library let you write programs that use Button A on the Core Unit. Use the line of code below to make **button_a** objects available in your program, like we did with **display** objects before.

```
from pystubit.board import button_a
```

Running the method **get_value()** from a **button_a** object will retrieve a value of either **1** or **0**. This tells you whether Button A is currently being pressed.
See **Appendix B** to learn more about using button objects.

You can use this flowchart/program to make an LED light up when Button A is pressed:

Flowchart



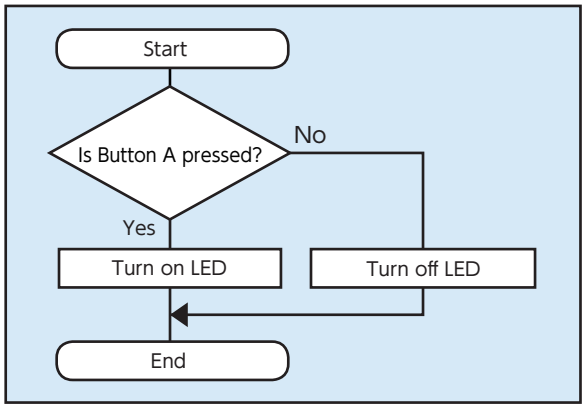
Python

```
1:  from pystubit.board import display, button_a
2:
3:  while True:
4:      if button_a.get_value() == 0:
5:          display.set_pixel(0, 0, display.RED)
6:      else:
7:          display.clear()
```

Line 1 of this program, **from pystubit.board import display, button_a** imports both **display** and **button_a** objects, letting you use them to control both the LED display and Button A in this program.
Using the **get_value()** method from the **button_a** object on line 4 lets you find the current status of Button A. If Button A is pressed, the value found here will be 0, fulfilling the condition. When the condition is fulfilled, the LED at the x-y coordinates (0,0) on the display will light up red (based on line 5 the program). If the value found for Button A is not 0 (meaing it isn't being pressed), the LED display will turn off (see line 7 of the program). Lines 4-7 are all indented, making them an endless loop under the **while True**: statement on line 3.

What happens if we don't use a **while** statement here? Let's look at the flowchart and program with the **while** statement removed.

Flowchart



Python

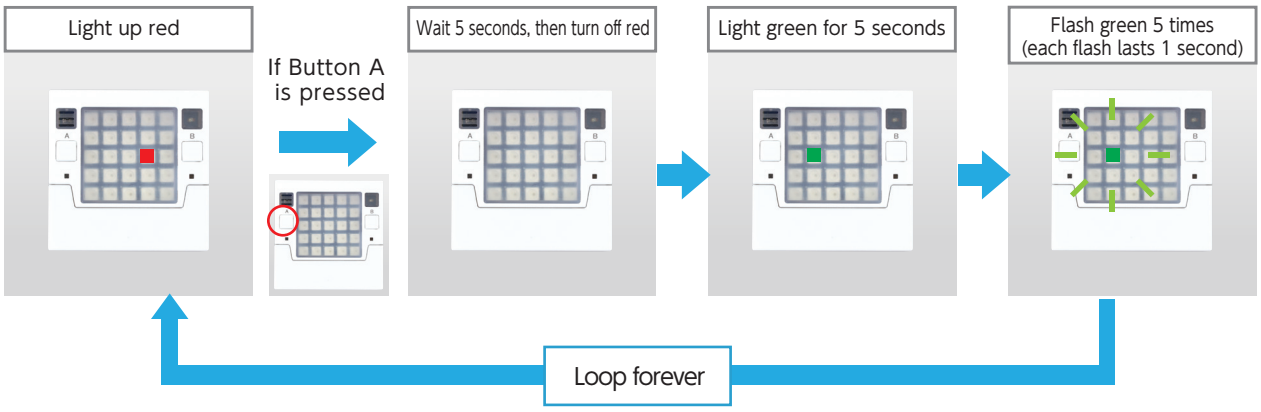
```
1:  from pystubit.board import display, button_a
2:
3:  if button_a.get_value() == 0:
4:      display.set_pixel(0, 0, display.RED)
5:  else:
6:      display.clear()
```

If you run the program like this, the device won't respond at all when you press Button A! This is also caused by how fast the Core Unit is running your program - the program will be finished running as soon as you start it! That's why you need the **while** statement to make the program never stop checking whether Button A is being pressed. That makes it work how we want it to!

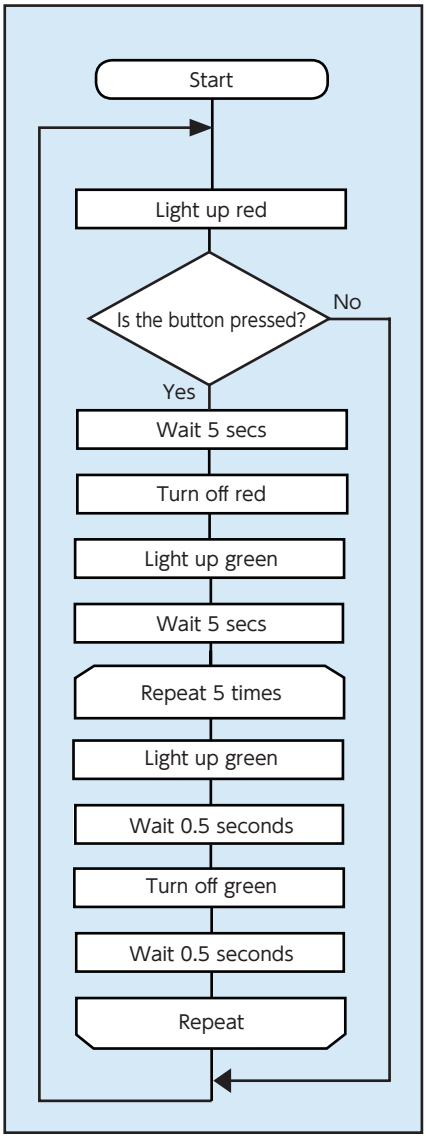
Programming Practice

Programming a Push-Button Traffic Signal

Draw up a flowchart that breaks down the lighting sequence shown below, then see if you can write a program for it!



Example Program



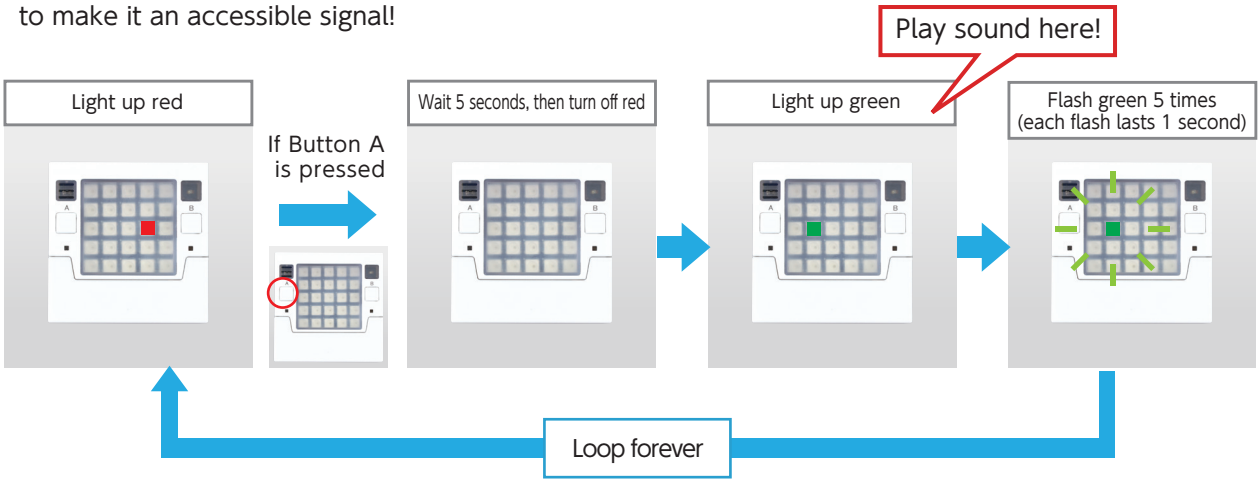

```
1:  from pystubit.board import display, button_a
2:  import time
3:
4:  while True:
5:      display.set_pixel(3, 2, display.RED)
6:      if button_a.get_value() == 0:
7:          time.sleep(5)
8:          display.clear()
9:          display.set_pixel(1, 2, display.GREEN)
10:         time.sleep(5)
11:         for i in range(5):
12:             display.set_pixel(1, 2, display.GREEN)
13:             time.sleep(0.5)
14:             display.clear()
15:             time.sleep(0.5)
```

4. Advanced Studies 1: Programming with Buzzers

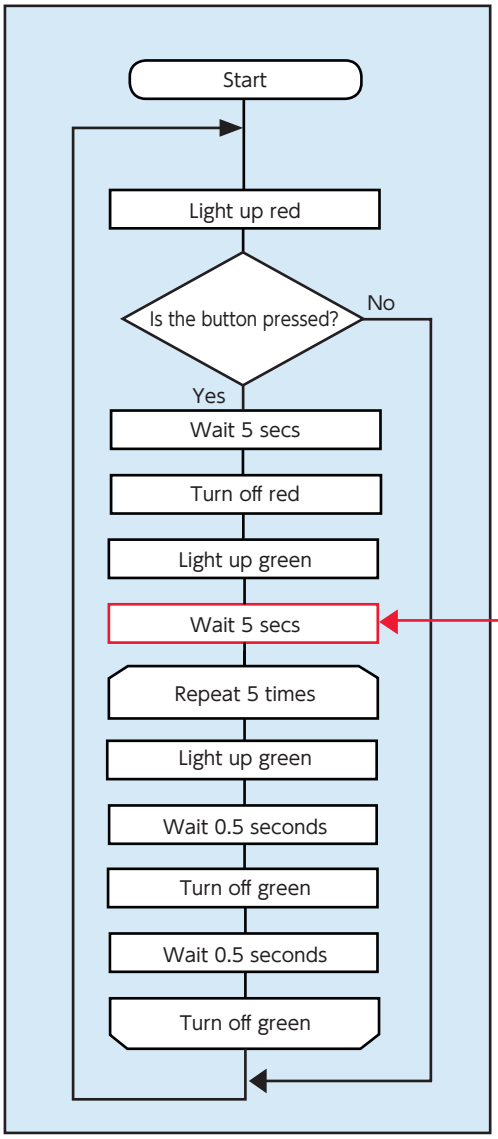
Programming an Accessible Traffic Signal

Some traffic signals play sounds when they change color to let visually impaired people know when it's safe to cross the street. We call these accessible signals because they can be used by more people than visual-only signals can!

Let's expand our push-button signal's program so it plays sound from the Buzzer while the light is green to make it an accessible signal!



Example Program



We'll add the program that plays the Buzzer here.

4.1 Playing the Buzzer

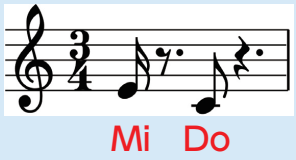
We'll need to use a **buzzer** object from the Studuino:bit library to let us program the Buzzer in the Core Unit. The program shown below will make the Buzzer play sound. See **Appendix C** to learn more about using **buzzer** objects.

```
1:  from pystubit.board import buzzer
2:  import time
3:
4:  buzzer.on('60')
5:  time.sleep(1)
6:  buzzer.off()
```

Line 1 of this program, **from pystubit.board import buzzer**, imports a **buzzer** object from the Studuino:bit library, letting you use it to program the Buzzer.
buzzer.on('60') in line 4 makes the Buzzer play the note Do (60). On line 5, **time.sleep(1)** pauses the program for 1 second before **buzzer.off()** makes the Buzzer stop playing on line 6. The number we used to set the value of the parameter in the **on** method is a MIDI note number. These numbers match up to the solfa scale, so 60 is a Do, 61 is a Do#, and so on. See **Appendix E** for a list of MIDI notes you can set to play with **buzzer** objects' **on** method. Then change how long they play with a **time** object's **sleep** method.

4.2 Making a Cuckoo Sound

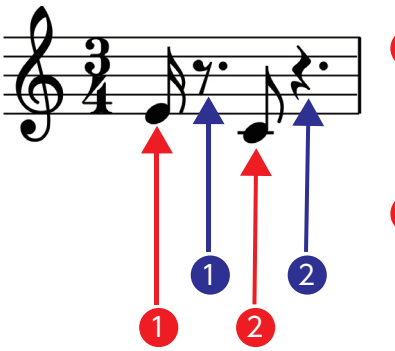
You need two notes to make a cuckoo sound: **Mi (64)** and **Do (60)**.



How Long is Each Note?

Musicians read sheet music to tell how long to play notes and when to rest when they play a particular song. You'll need to make a musical score like this for your cuckoo sound, too. You can do this by programming **two notes** and **two rests** with different lengths of time.

A Cuckoo Sound Score



A Cuckoo Sound Program

```
buzzer.on('64')
time.sleep(0.1)
buzzer.off()
time.sleep(0.3)
buzzer.on('60')
time.sleep(0.2)
buzzer.off()
time.sleep(0.6)
```

To make the cuckoo sound play five times, you can set up a loop using a **for** statement, as shown on the right.

```
1:  from pystubit.board import buzzer
2:  import time
3:
4:  for i in range(5):
5:      buzzer.on('64')
6:      time.sleep(0.1)
7:      buzzer.off()
8:      time.sleep(0.3)
9:      buzzer.on('60')
10:     time.sleep(0.2)
11:     buzzer.off()
12:     time.sleep(0.6)
```

4.3 Playing Sound on a Green Light

Making a sequence of programming commands into a function can make your program easier to read. To create (or define) a function in Python, write a line like this:

```
def Function(Parameter):
    Program
```

Remember to start with an indent!

Make sure to indent the program you're going to make into a function! Function names have to follow the same rules as variable names, so they need to be written with Roman letters, numbers, or underscores, but can't start with a number. If your function doesn't use any parameters, you can leave the parentheses empty. If it uses multiple parameters, remember to separate them with commas!

```
 1: from pystubit.board import buzzer
 2: import time
 3:
 4: def cuckoo():
 5:     for i in range(5):
 6:         buzzer.on('64')
 7:         time.sleep(0.1)
 8:         buzzer.off()
 9:         time.sleep(0.3)
10:         buzzer.on('60')
11:         time.sleep(0.2)
12:         buzzer.off()
13:         time.sleep(0.6)
```

The **cuckoo** function doesn't have any parameters we need to set, so you can make it by writing just **def cuckoo():** on line 4 of this program. Remember to indent lines 5-13 to show that they're the program for the **cuckoo()** function!

Replace line 10 of your push-button signal program from **Programming Practice** on page 35 (**time.sleep(5)**, after lighting up green) with a line that calls the **cuckoo** function.

```
 1:  from pystubit.board import display, button_a, buzzer
 2:  import time
 3:
 4:  def cuckoo():
 5:      for i in range(5):
 6:          buzzer.on('64')
 7:          time.sleep(0.1)
 8:          buzzer.off()
 9:          time.sleep(0.3)
10:          buzzer.on('60')
11:          time.sleep(0.2)
12:          buzzer.off()
13:          time.sleep(0.6)
14:
15:  while True:
16:      display.set_pixel(3, 2, display.RED)
17:      if button_a.get_value() == 0:
18:          time.sleep(5)
19:          display.clear()
20:          display.set_pixel(1, 2, display.GREEN)
21:          cuckoo()
22:          for i in range(5):
23:              display.set_pixel(1, 2, display.GREEN)
24:              time.sleep(0.5)
25:              display.clear()
26:              time.sleep(0.5)
```

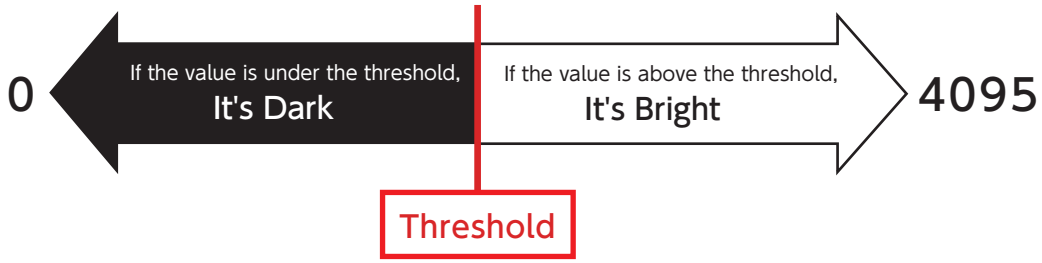
The **cuckoo()** on line 21 runs the **cuckoo** function created on line 4.

5. Advanced Studies 2: Programming with Light Sensors

Programming a Sensor Light

Let's try programming a sensor light that turns on when its surroundings get dark.

We'll use the Core Unit's built-in Light Sensor to measure the brightness of the unit's surroundings. The Light Sensor measures brightness on a scale from **0** (dark) to **4095** (bright). Our sensor light will use the Light Sensor to judge when the surroundings are sufficiently dark, and then turn on an LED. We call the number used to judge when a condition like this is fulfilled a **threshold.**



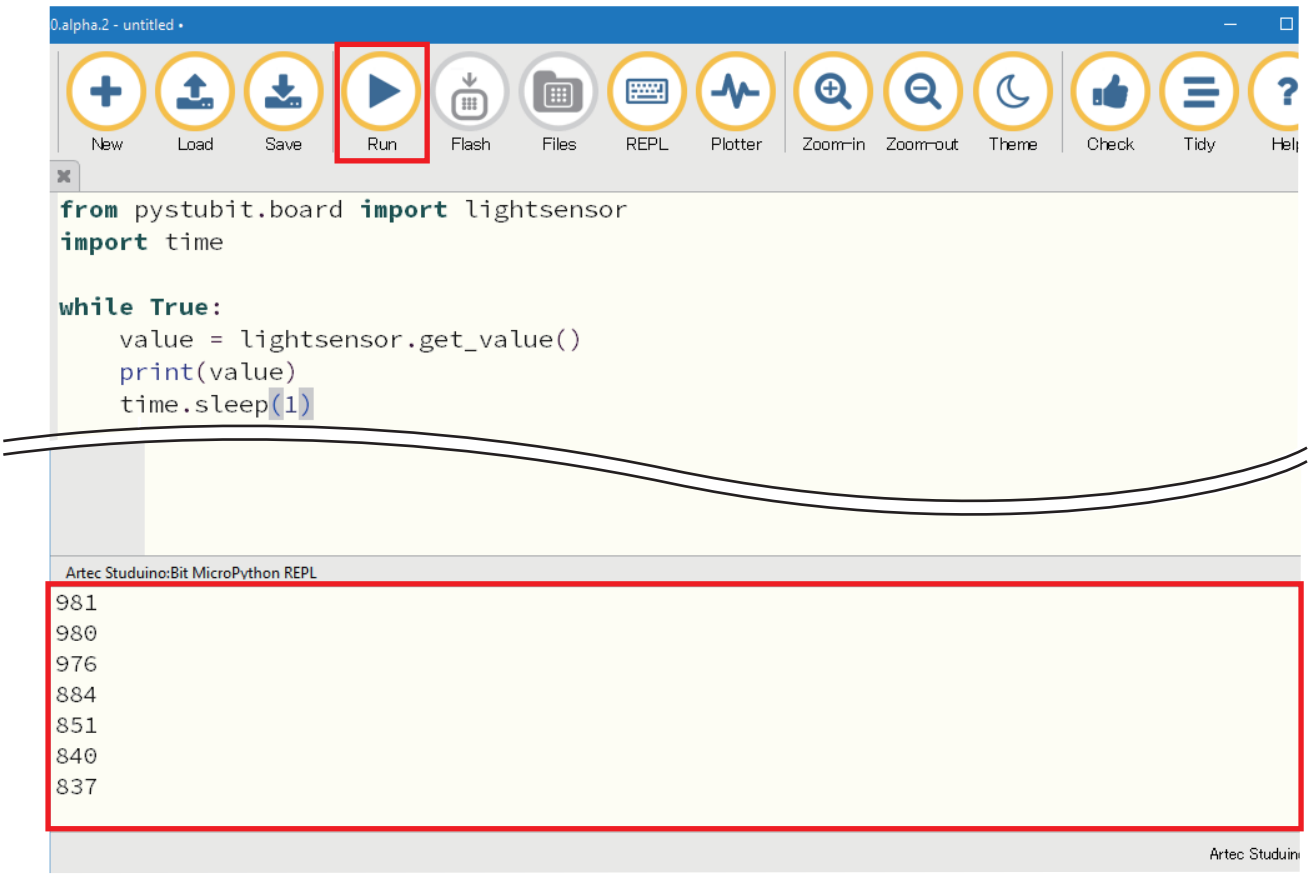
To pick a threshold for this program you'll need to check the Light Sensor's values.

5.1 Checking the Light Sensor's Values in Mu

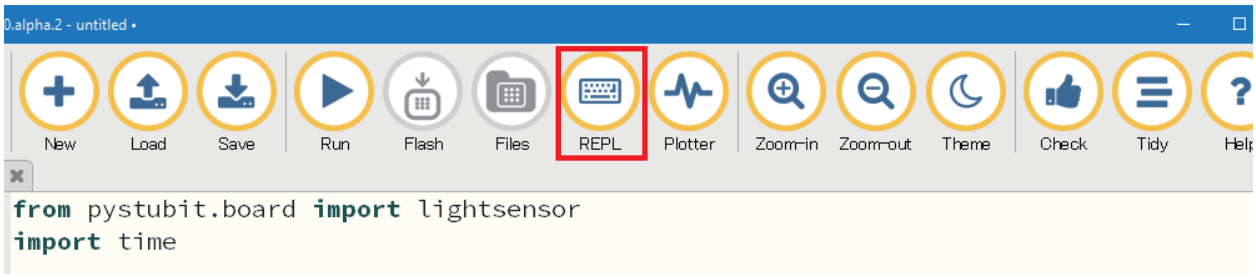
Write the following code into the text editor.

```
1:  from pystubit.board import lightsensor
2:  import time
3:
4:  while True:
5:      value = lightsensor.get_value()
6:      print(value)
7:      time.sleep(1)
```

We'll need to use a **lightsensor** object from the Studuino:bit library to let us program the Light Sensor in the Core Unit. You can retrieve values from the Light Sensor by using the method **get_value()**. See **Appendix D** to learn more about using **lightsensor** objects.
All the lines after **while True**: on line 4 are an endless loop. Line 5, **value = lightsensor.get_value()**, saves the Light Sensor's value in a variable called **value**. Line 6, **print(value)**, will make REPL display the current value of the **value** variable. Line 7, **time.sleep(1)**, makes the program pause for 1 second.
Click the **Run** button in the Mu editor to run this program, and REPL will start displaying the Light Sensor's value every second. Try covering the Core Unit's Light Sensor with your hand to see how its value changes.



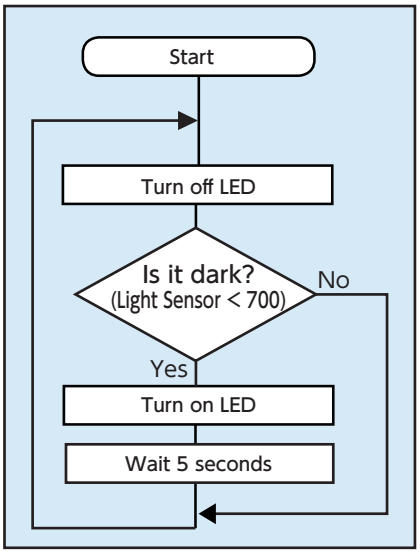
When you want to stop your program, click Mu's **REPL** button and close REPL.



5.2 Completing Your Sensor Light

Below is an example sensor light program.

Flowchart




```
1:  from pystubit.board import display, lightsensor
2:  import time
3:
4:  while True:
5:      display.clear()
6:      if lightsensor.get_value() < 700:
7:          display.set_pixel(0, 0, display.RED)
8:          time.sleep(5)
```

Appendix A. display Object Methods

Method	What It Does
get_pixel(x, y)	Find the color of the LED at row x/column y. The return value will be in (R,G,B) format.
set_pixel(x, y, color)	Set the color of the LED at row x/column y using the **color** parameter. **color** can be set using (R,G,B), [R,G,B], or #RGB format.
clear()	Set the brightness of all LEDs to 0 (off).
show(iterable, delay=400, *, wait=True, loop=False, clear=False, color=None)	Show the **iterable** parameter on the LED display (images, text, and numbers can all be set in this parameter).
scroll(string, delay=150, *, wait=True, loop=False, color=None)	Scroll the **value** parameter across the LED display horizontally (strings of letters and numbers can be set in this parameter).
on()	Turn on power to the display.
off()	Turn off power to the display. (This allows you to use the GPIO ports connected to the display for other purposes.)
is_on()	Return **True** if power to the display is ON and **False** if it's OFF.

display objects also contain the following properties that can be used to set LED colors:
BLACK (this turns off the LED), WHITE, RED, LIME, BLUE,YELLOW, CYAN, MAGENTA, SILVER, GRAY, MAROON, OLIVE,GREEN, PURPLE, TEAL, NAVY

Appendix B. button_a Object Methods

Method	What It Does
get_value()	Return **0** if the button is being pressed and **1** if it isn't.
is_pressed()	Return **True** if the button is being pressed.
was_pressed()	Button objects store information about when their buttons are pressed. This method will return **True** if the button has been pressed in the past. This information will be reset after you call this method in a program.
get_presses()	Button objects store information about when their buttons are pressed. This method will return the number of times the button has been pressed in the past. This number will be reset after you call this method in a program.

Appendix C. buzzer Object Methods

Method	What It Does

on(sound, *, duration=-1)	The **sound** parameter can be set to a specific note using a note name (as a string, 'C3'-'G9'), a MIDI note number (as a string, '48'-'127') or a frequency (as an integer). The **duration** parameter is used to set how long the sound should play for and can be set in ms (milliseconds). If you don't set the **duration** parameter the Buzzer will keep playing until you run the **off** method.
	bzr.on('50', duration=1000)
	# This plays MIDI note 50 for 1 second.
	bzr.on('C4', duration=1000)
	# This plays the note C4 for 1 second.
	bzr.on(440)

off()	Return **True** if the button is being pressed.

Appendix D. lightsensor Object Methods

Method	What It Does
get_value()	Find the value of the Core Unit's built in Light Sensor (0-4095).

Appendix E. MIDI Notes for buzzer Objects

		Note											
		Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Ti
Octave	3	48	49	50	51	52	53	54	55	56	57	58	59
	4	60	61	62	63	64	65	66	67	68	69	70	71
	5	72	73	74	75	76	77	78	79	80	81	82	83
	6	84	85	86	87	88	89	90	91	92	93	94	95
	7	96	97	98	99	100	101	102	103	104	105	106	107
	8	108	109	110	111	112	113	114	115	116	117	118	119
	9	120	121	122	123	124	125	126	127				

088375 K0620