Stop Light

2

40-Pin

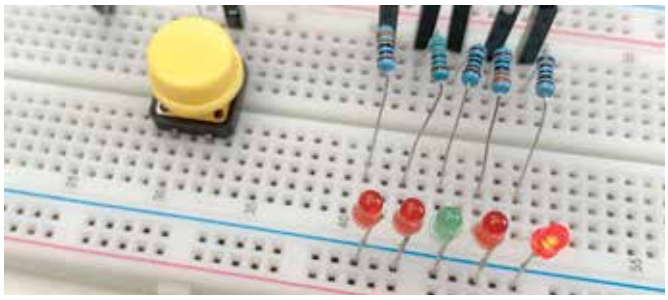
A simple game to stop the light on the centre LED
Covers lists, conditional statements, and introduces push buttons

60 mins

1. Purpose of the game

Five LEDs light-up one-at-a-time in a backward-and-forward sequence. The middle LED is green. The player attempts to stop the cycle when the middle LED is lit.

It's harder than it looks, and there are lots of upgrades you can make!

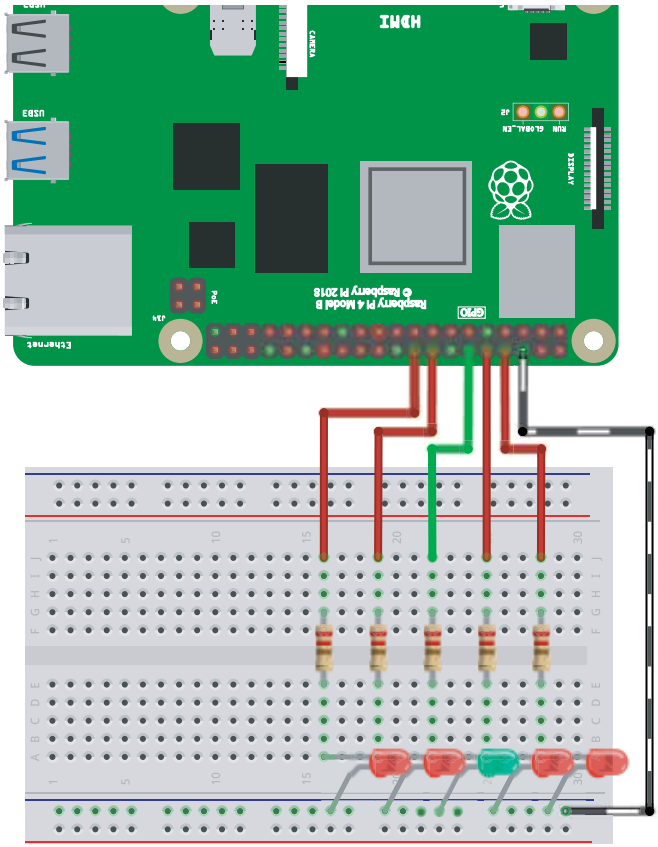


2. Component list

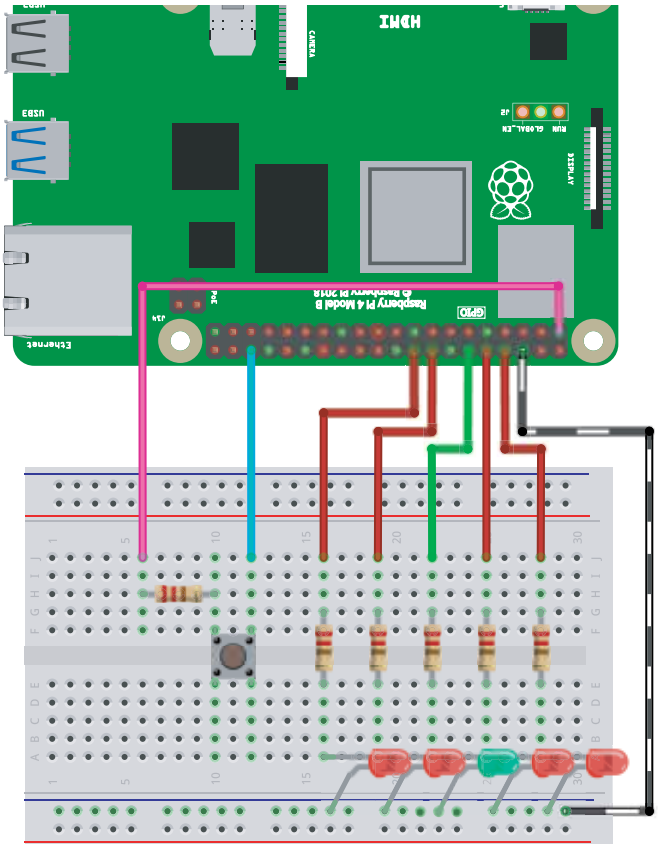
In addition to your Raspberry Pi and breadboard, this project requires the following:

- 1 x momentary push switch
- 4 x red LEDs
- 1 x green LED
- 6 x 220 ohm resistors (red, red, brown)
- 8 x jumpers

3. Add the LEDs



4. Add the button



5. The code

```
1.  # Import libraries
2.  import RPi.GPIO as GPIO
3.  from time import sleep
4.
5.  # Hide warnings.
6.  GPIO.setwarnings(False)
7.
8.  # Use BOARD pin numbering
9.  GPIO.setmode(GPIO.BOARD)
10.
11. # Set up pin 36 for button
12. buttonNumber = 36
13. GPIO.setup(buttonNumber, GPIO.IN,
    pull_up_down = GPIO.PUD_DOWN)
14.
15. # List of LED pins
16. LEDpins = [18, 16, 12, 10, 8]
17. LEDlength = len(LEDpins) # Length of list
18.
19. # Determine the pin number of the middle LED
20. LEDmiddle = LEDpins[int(LEDlength / 2)]
21.
22. # Reverse list and append to itself
23. LEDpins = LEDpins + LEDpins[::-1]
24.
25. # Initialise LED pins
26. for i in LEDpins:
27.     GPIO.setup(i, GPIO.OUT)
28.
29. # Define what happens when the button
    is pressed.
30. def onPress(LEDlit, LEDmiddle):
31.     # Check if the LED currently lit is
        the same as the middle one
32.     if LEDlit == LEDmiddle:
33.         print ("Well done")
34.     else:
35.         print("Try again")
36.     sleep(3)
37.
38. # Main program
39. while True:
40.     # Step through LEDs
41.     for i in LEDpins:
42.         GPIO.output(i, GPIO.HIGH)
43.         sleep(0.05)
44.         if (GPIO.input(buttonNumber)):
45.             onPress(i, LEDmiddle)
46.         GPIO.output(i, GPIO.LOW)
```

6. How does it work?

The LED pins are not consecutively numbered, so they need to be stored in a list which can be iterated-through *(line 16)*.

To find the middle of the list, we use the `len` command to first determine the list's length *(line 17)*, then divide it by two and cast to an integer to round down any decimal values *(line 20)*. So why not just type-in the middle number? This method allows us to have any number of LEDs, and always calculate the middle one.

If we were to use the list as it currently stands, the LEDs would only light in one direction. *Line 23* appends a reversed version of the list to the end of itself. This allows us to use a single `for` loop *(line 41)* to achieve the lighting pattern we want.

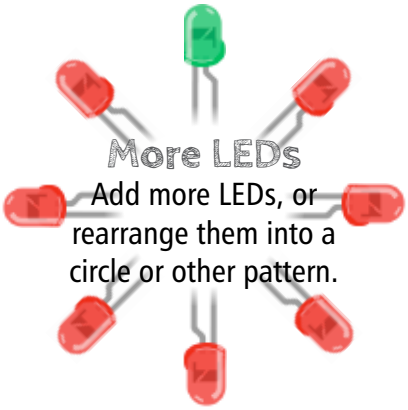
When the user presses the button, the current iteration of the loop (`i`) and the middle LED pin are passed to the `onPress()` function *(lines 44 and 45)*.

`onPress()` *(line 30)* compares the currently lit LED pin number with the middle LED pin number. If they are the same, it prints "Well Done", otherwise it prints "Try again"; pauses for 3 seconds, and continues the game.

7. Ideas for extending the game

Keep score

Add a digital display (see Tutorial X01) to keep a score. +10 points each time the user presses correctly, -3 for each incorrect press!



More LEDs

Add more LEDs, or rearrange them into a circle or other pattern.

Speed it up

Each time the user presses the button at the correct time, decrease the value of the sleep timer on line 43