Assignment: 6
Title: Create a program that illuminates the green LED if the counter is less than 100, illuminates the yellow LED if the counter is between 101 and 200 and illuminates the red LED if the counter is greater than 200.
Objective: Connectivity, configuration and control of LED using Arduino circuit under different conditions.
Hardware Requirement: Arduino, LED, 220 ohm resistor etc.
Software Requirement: Arduino IDE
Theory: The problem statement is like Arduino traffic light, a fun little project that you can build in under an hour. Here's how to build your own using an Arduino, and how to change the circuit for an advanced variation. What You Need to Build an Arduino Traffic Light Controller Apart from the basic Arduino, you'll need: ● 1 x 10k-ohm resistor ● 1 x pushbutton switch ● 6 x 220-ohm resistors ● A breadboard ● Connecting wires ● Red, yellow and green LEDs Arduino Traffic Light: The Basics Let's start small. A basic, single traffic light is a good place to start. Here's the circuit: Connect the anode (long leg) of each LED to digital pins eight, nine, and ten (via a 220-ohm resistor). Connect the cathodes (short leg) to the Arduino's ground.
Diagram: Pin Diagrams,Semantic

Code for the Arduino Traffic Light
Start by defining variables so that you can address the lights by name rather than a number. Start a new Arduino project, and begin with these lines:
 	
``
int red = 10; int
yellow = 9; int
green = 8;
``
 	
Next, let's add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:
 	
``
void setup(){ pinMode(red,
    OUTPUT);
pinMode(yellow, OUTPUT);
    pinMode(green, OUTPUT);
}
``
 	
The pinMode function configures the Arduino to use a given pin as an output. You have to do this for your LEDs to work at all. Now for the actual logic of the traffic light. Here's the code you need. Add this below your variable definitions and setup function:

```
void loop(){
  changeLights
  ();
  delay(15000);
}
void changeLights(){
  // green off, yellow on for 3
  seconds digitalWrite(green, LOW);
  digitalWrite(yellow, HIGH);
```
// turn off yellow, then turn red on for 5
seconds digitalWrite(yellow, LOW);
digitalWrite(red,
HIGH); delay(5000);
// red and yellow on for 2 seconds (red is already on though)
digitalWrite(yellow, HIGH);
delay(2000);
// turn off red and yellow, then turn on
green digitalWrite(yellow, LOW);
digitalWrite(red, LOW);
digitalWrite(green,
HIGH); delay(3000);
```
}
```

Upload this code to your Arduino, and run (make sure to select the correct board and port from the Tools > Board and Tools > Port menus). You should have a working traffic light that changes every 15 seconds, like this (sped up):

Let's break down this code. The changeLights function performs all the hard work. This rotates the traffic light through yellow and red, then back to green. As this gets called inside the loop function, the Arduino will run this code forever, with a 15-second pause every time.

The changeLights function consists of four distinct steps:
● Green on, yellow off
● Yellow off, red on
● Yellow on, red on
● Green on, red off, yellow off
These four steps replicate the process used in real traffic lights. For each step, the code is very similar. The appropriate LED gets turned on or off using **digitalWrite**. This is an Arduino function used to set output pins to HIGH (for on), or LOW (for off).

After enabling or disabling the required LEDs, the **delay** makes the Arduino wait for a given amount of time. Three seconds in this case.

Going Deeper: Arduino Pedestrian Crossing

Now that you know the basics, let's improve it. Add in a pushbutton for pedestrians to change the light whenever they like:

Notice how the traffic light is exactly the same as the previous example. Connect the button to digital pin 12. You'll notice that the switch has a high-impedance 10k-ohm resistor attached to it, and you may be wondering why. This is a pull-down resistor.

A switch either lets the current flow or doesn't. This seems simple enough, but in a logic circuit, the current should be always flowing in either a high or low state (remember, 1 or 0, HIGH or LOW). You might assume that a pushbutton switch that isn't actually pressed would be in a LOW

state, but in fact, it's said to be 'floating', because no current gets drawn at all.

In this floating state, it's possible that a false reading will occur as it fluctuates with electrical interference. In other words, a floating switch is giving neither a reliable HIGH nor LOW reading. A pull-down resistor keeps a small amount of current flowing when the switch gets closed, thereby ensuring an accurate low state reading.
In other logic circuits, you may find a pull-up resistor instead, and this works on the same principle, but in reverse, making sure that particular logic gate defaults to high.
Now, in the loop part of the code, instead of changing the lights every 15 seconds, you're going to read the state of the pushbutton switch instead, and only change the lights when it's activated.
Code for the Arduino Pedestrian Crossing
Start by adding a new variable to store your button pin:

```
int button = 12; // switch is on pin 12
```

Now, in the setup function, add a new line to declare the switch as an input. Add a line to set the traffic lights to the green stage. Without this initial setting, they would off until the first time changeLights runs.

```
pinMode(button, INPUT);
digitalWrite(green, HIGH);
```

Change the entire loop function to the following instead:

```
void loop() {
  if (digitalRead(button) == HIGH){
    delay(15); // software debounce if
    (digitalRead(button) == HIGH) {
// if the switch is HIGH, ie. pushed down - change the lights!
      changeLights();
delay(15000); // wait for 15 seconds
}
}
}
```

That should do it. You may be wondering why the button checking happens twice (digitalRead(button)), separated by a small delay. This is debouncing. Much like the pull-down resistor for the button, this simple check stops the code detecting minor interference as a button press.

By waiting inside the if statement for 15 seconds, the traffic lights can't change for at least that duration. Once 15 seconds is over the loop restarts. Each restart of the loop, it reads the state of the button again, but if it isn't pressed, the if statement never activates, the lights never change, and the program restarts again.

Here's how this looks (sped up):

Arduino Traffic Light with Junction
Let's try a more advanced model. Instead of a pedestrian crossing, change your circuit to have two traffic lights:
Connect the second traffic light to digital pins 11, 12, and 13.
Code for the Arduino Traffic Light with Junction
First, assign your new traffic light pins to variables, and configure them as outputs, like

in the first example:

```
// light one int
red1 = 10;
int yellow1 = 9;
int green1 = 8;
// light two int
red2 = 13;
int yellow2 = 12;
int green2 = 11;
void setup(){
// light one pinMode(red1,
    OUTPUT);
pinMode(yellow1, OUTPUT);
    pinMode(green1, OUTPUT);
// light two pinMode(red2,
    OUTPUT);
pinMode(yellow2, OUTPUT);
    pinMode(green2, OUTPUT);
}
```

Now, update your loop to use
the code from the first example (instead of the pedestrian
crossing):

```
void loop(){
    changeLights();
    delay(15000);
}
```

Once again, all the work is carried out in the changeLights function. Rather than going **red** > **red & yellow** > **green,** this code will alternate the traffic lights. When one is on green, the other is on red. Here's the code:

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

LED	Arduino Pin
LED 1	0
LED 2	1
LED 3	2
LED 4	4

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open

Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the

Arduino IDE go to tools⊡Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running and you can observe the LED's getting illuminated after the count is changed on the board.

Observation:

You can observe the LED's turning ON after the specific counter has expired. Additionally you can print the value of the counter on the serial terminal using the *serial.println* function.

Code:

```
const int green_led = 0;
const int yellow_led = 1;
const int red_led = 2;

unsigned int counter;

void setup() {
// put your setup code here, to run once:
 pinMode(green_led, OUTPUT);
 pinMode(red_led, OUTPUT);
 pinMode(yellow_led, OUTPUT);

 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,HIGH);
}

void loop() {
counter = counter+1;
delay(200);

if(counter <= 100)
{
 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,LOW);
 digitalWrite(red_led,HIGH);
 }

else if((counter > 100)&& (counter <= 200)) {
 digitalWrite(yellow_led,LOW);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,HIGH);
 }

else if(counter > 200)
{
 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,LOW);
```

```
 }

if (counter > 210) counter =0;
 }
```

Conclusion: Executed the program that illuminates the green LED if the counter is less than 100, illuminates the yellow LED if the counter is between 101 and 200 and illuminates the red LED if the counter is greater than 200.