

University of California, Riverside
Mechanical Engineering Department
ME133: Introduction to Mechatronics
Dr. Fabio Pasqualetti TA: Gianluca Bianchin
Lab 6: February 28, 2018
Lab Report Due on March 6, 2018

OBJECTIVES:

1. Get familiar with the Arduino microcontroller, its pin-out, input, output, principles of operation, and its software.
2. Use the Arduino S/W to write simple programs, debug and produce executable, and run programs on the microcontroller in stand-alone mode.
3. Design and test a simple game with Arduino.

BILL OF MATERIALS:

- 1x Arduino UNO
- 10x 220 Ohm Resistor
- 4x Red LED's
- 1x Green LED
- 4x Pushbuttons
- 1x Breadboard

USEFUL LINKS:

1. <http://www.melabs.com/products/usbprog.htm>
2. <http://arduino.cc/en/Tutorial/HomePage>.

SUBMISSION GUIDELINES: (READ CAREFULLY)

- (a) Before moving on to the next task SHOW THE TA YOUR CIRCUIT IS PROPERLY WORKING. Make sure your circuit is "clean" and well organized.
- (b) In your report your code will not be evaluated during the grading process. Attach a copy of your code at the appendix, after reducing the number of lines to a minimum and including proper comments.
- (c) Part of the grading will depend on the group's autonomy and results during the lab session.

ARDUINO CODE STRUCTURE:

- The **setup()** function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.
- After creating a setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond as it runs. Code in the loop() section of your sketch is used to actively control the Arduino board.
- The code below won't actually do anything, but its structure is useful for copying and pasting to get you started on any sketch of your own. It also shows you how to make comments in your code.
- Any line that starts with two slashes (//) will not be read by the compiler, so you can write anything you want after it. Commenting your code like this can be particularly helpful in explaining, both to yourself and others, how your program functions step by step.



```
sketch_oct29a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

9 Arduino/Genuino Uno on /dev/cu.usbmodem1411

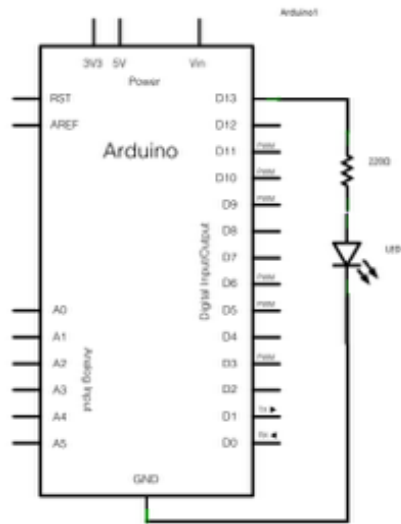
TASK 1: Wiring & coding Arduino for blinking LED

Figure: Arduino with LED connected for blinking Program

In the program below, the first thing you do is to initialize pin 13 as an output pin with the line **pinMode(13, OUTPUT);**

In the main loop, you turn the LED on with the line:

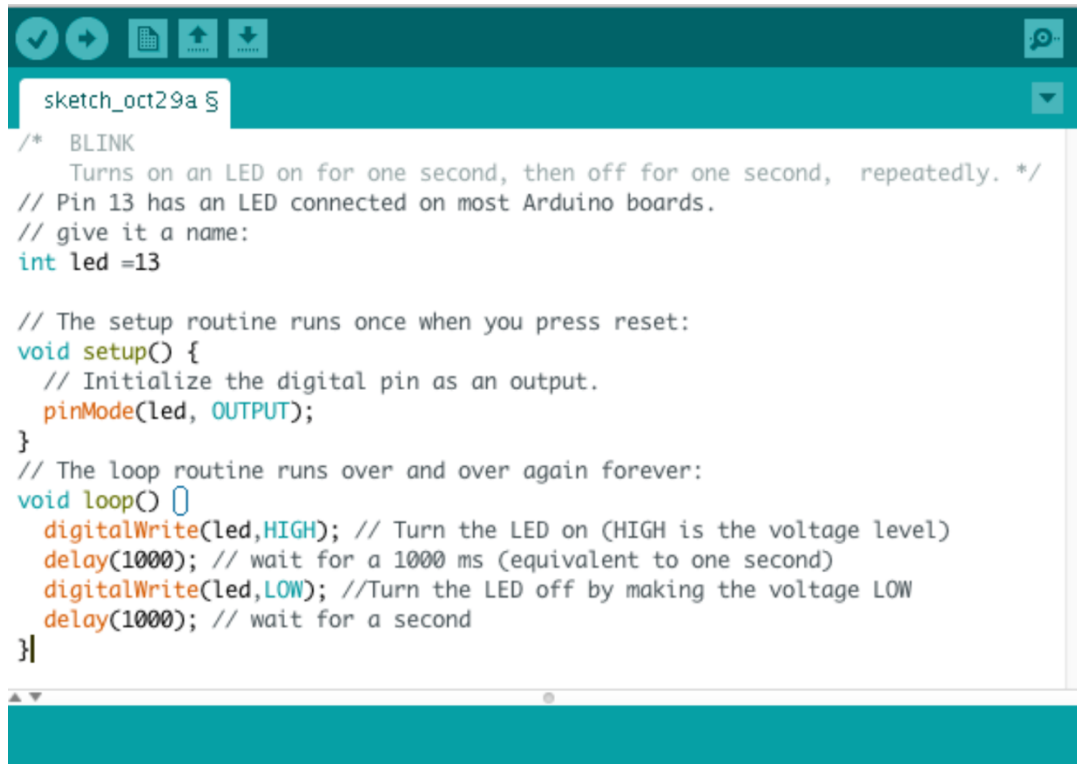
digitalWrite(13, HIGH);

This supplies 5 volts to pin 13. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:

digitalWrite(13, LOW);

That takes pin 13 back to 0 volts and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the delay() command tell the Arduino to do nothing for 1000 milliseconds. When you use the delay() command, nothing else happens for that amount of time.

- a) Demonstrate to the TA that code is properly working
- b) Accurately describe the procedure on your report

A screenshot of the Arduino IDE interface. The top toolbar shows icons for checking, running, saving, and uploading. The file name bar at the top says "sketch_oct29a §". The main text area contains the following C++ code:

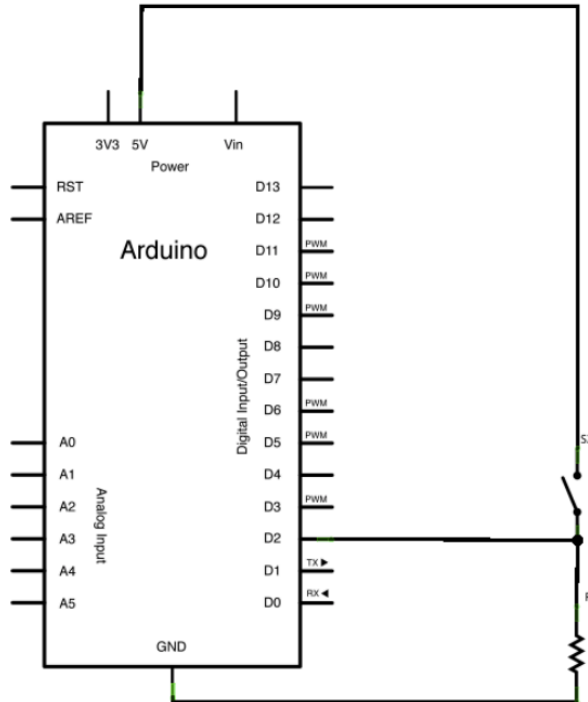
```
/* BLINK
   Turns on an LED on for one second, then off for one second,  repeatedly. */
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13

// The setup routine runs once when you press reset:
void setup() {
  // Initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
// The loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // Turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a 1000 ms (equivalent to one second)
  digitalWrite(led, LOW); //Turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

- Once you've understood the basic examples, check out the [BlinkWithoutDelay](#) example to learn how to create a delay while doing other things.
- Once you've understood this example, check out the [DigitalReadSerial](#) example to learn how read a switch connected to the Arduino.

TASK 2: Digital Read Serial

Wire the following circuit, where $R1 = 10K\ \Omega$:



Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or 1.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it doesn't have a solid connection to voltage or ground, and it will randomly return either HIGH or LOW. That's why you need a pull-down resistor in the circuit.

In the program below, the very first thing that you do in the setup function is to begin serial communication, at 9600 bits of data per second, between your Arduino and your computer. We use the line:

`Serial.begin(9600);`

Next, initialize digital pin 2, the pin that will read the output from your button, as an input:

`pinMode(2,INPUT);`

Now that your setup has been completed, move into the main loop of your code. When your button is pressed, 5 volts will freely flow through your circuit, and when it is not pressed, the input pin will be connected to ground through the 10-kilohm resistor. This is a digital input, meaning that the switch can only be in either an on state (seen by your Arduino as a "1", or HIGH) or an off state (seen by your Arduino as a "0", or LOW), with nothing in between.

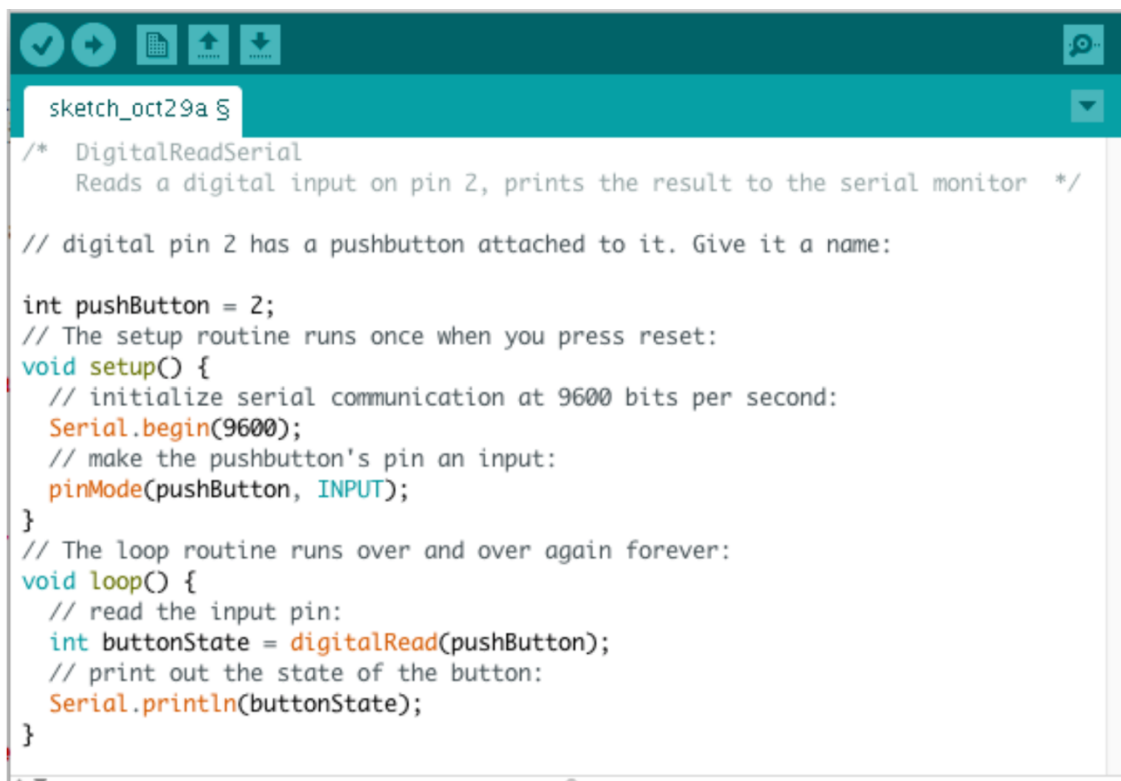
The first thing you need to do in the main loop of your program is to establish a variable to hold the information coming in from your switch. Since the information coming in from the switch will be either a "1" or a "0", you can use an int datatype. Call this variable `sensorValue` and set it to equal whatever is being read on digital pin 2. You can accomplish all this with just one line of code:

`int sensorValue = digitalRead(2);`

Once the Arduino has read the input, make it print this information back to the computer as a decimal (DEC) value. You can do this with the command `Serial.println()` in our last line of code:

`Serial.println(sensorValue, DEC);`

Now, when you open your Serial Monitor in the Arduino environment, you will see a stream of "0"s if your switch is open, or "1"s if your switch is closed.



```
sketch_oct29a §
/* DigitalReadSerial
   Reads a digital input on pin 2, prints the result to the serial monitor */

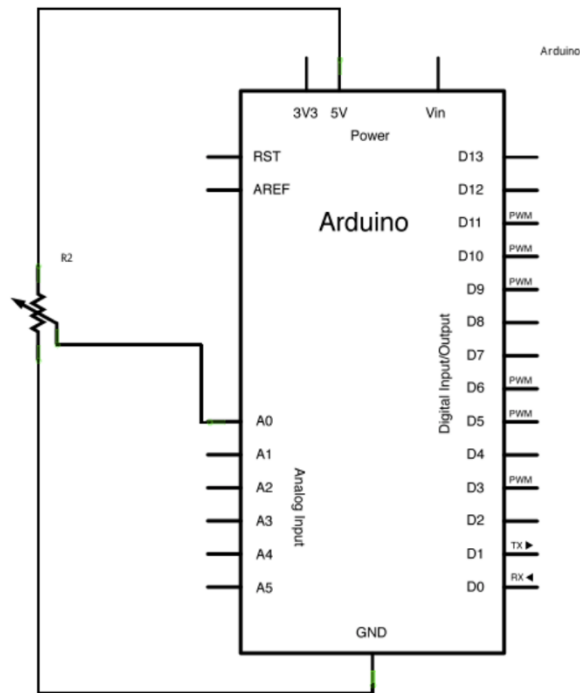
// digital pin 2 has a pushbutton attached to it. Give it a name:

int pushButton = 2;
// The setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}
// The loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
}
```

- a) Demonstrate to the TA that code is properly working
- b) Accurately describe the procedure on your report

TASK 3: Analog Read Serial

This example shows you how to read analog input from the physical world using a potentiometer. A **potentiometer** is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your Arduino, it is possible to measure the amount of resistance produced by a potentiometer (or *pot* for short) as an analog value. In this example you will monitor the state of your potentiometer after establishing serial communication between your Arduino and your computer.



Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 K Ohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the **analog voltage** that you're reading as an input.

The Arduino has a circuit inside called an **analog-to-digital converter** that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between,

`analogRead()` returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

In the program below, the only thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the command:

`Serial.begin(9600);`

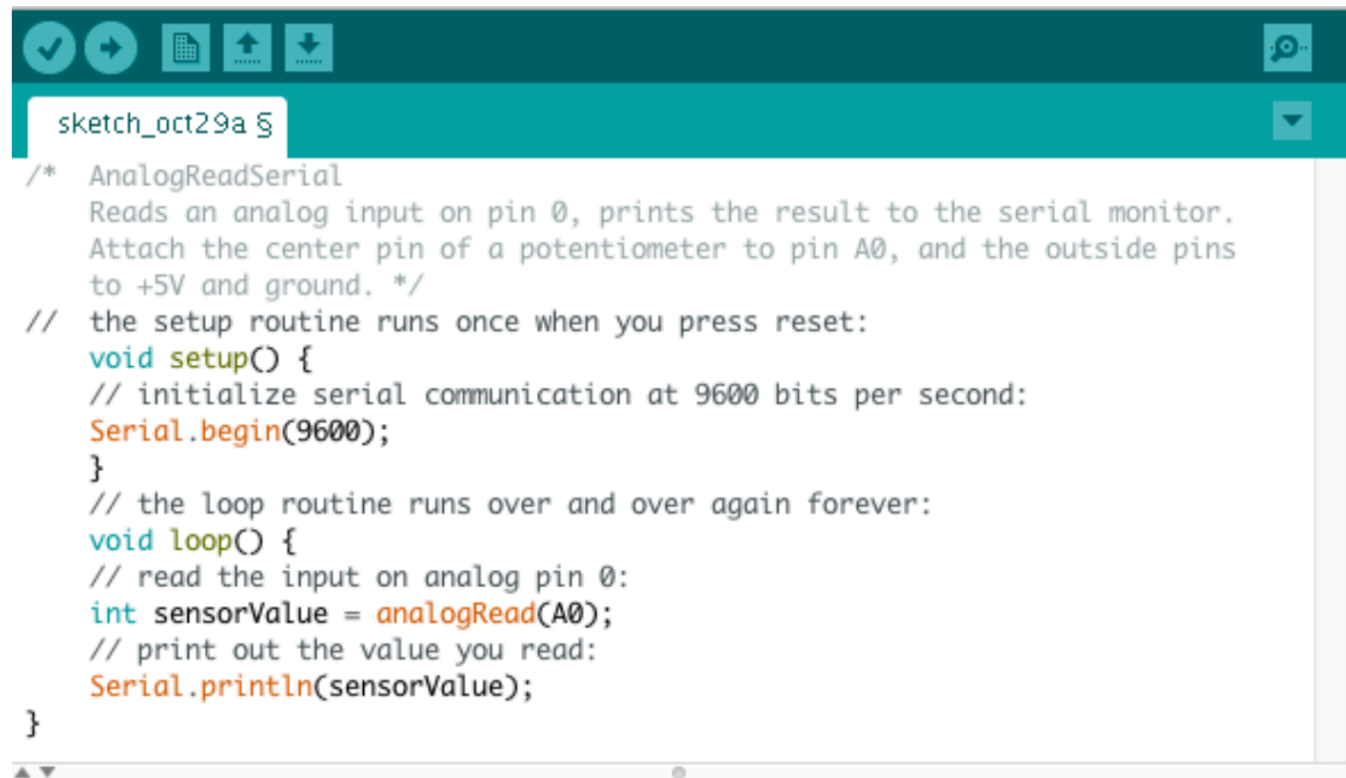
Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an `int datatype`) coming in from your potentiometer:

`int sensorValue = analogRead(A0);`

Finally, you need to print this information to your serial window as a decimal (DEC) value. You can do this with the command `Serial.println()` in your last line of code:

`Serial.println(sensorValue, DEC)`

Now, when you open your Serial Monitor in the Arduino development environment (by clicking the button directly to the right of the "Upload" button in the header of the program), you should see a steady stream of numbers ranging from 0-1023, correlating to the position of the pot. As you turn your potentiometer, these numbers will respond almost instantly.

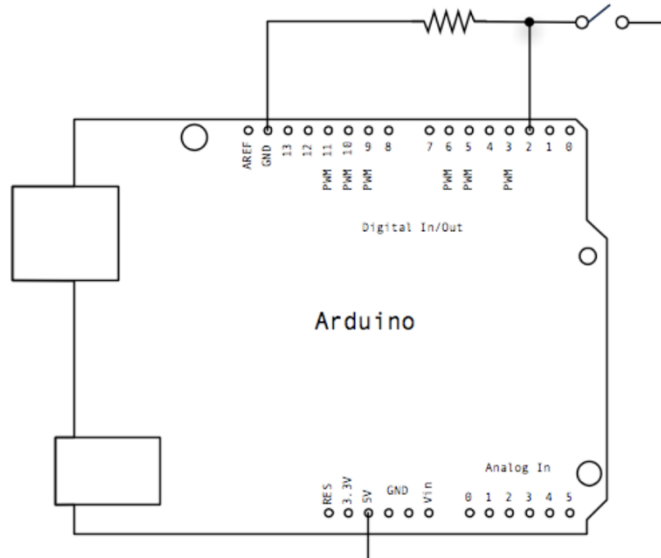


```
sketch_oct29a §
/* AnalogReadSerial
   Reads an analog input on pin 0, prints the result to the serial monitor.
   Attach the center pin of a potentiometer to pin A0, and the outside pins
   to +5V and ground. */
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
}
```

- Demonstrate to the TA that code is properly working
- Accurately describe the procedure on your report

TASK 4: Push Button Control of LED

Pushbuttons or switches connect two points in a circuit when you press them. This example turns on the built-in LED on pin 13 when you press the button. Use a 10K resistor.



Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 K Ohms) to ground. The other leg of the button connects to the 5 volt supply.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

You can also wire this circuit the opposite way, with a pull-up resistor keeping the input HIGH, and going LOW when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

If you disconnect the digital i/o pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resistor in the circuit.



```
sketch_oct29a §

The circuit:
* LED attached from pin 13 to ground
* pushbutton attached to pin 2 from +5V
* 10K resistor attached to pin 2 from ground
* Note: on most Arduinos there is already an LED on the board attached to
  pin 13. */
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;    // the number of the pushbutton pin
const int ledPin = 13;     // the number of the LED pin
// variables will change:
int buttonState = 0;        // variable for reading the pushbutton status

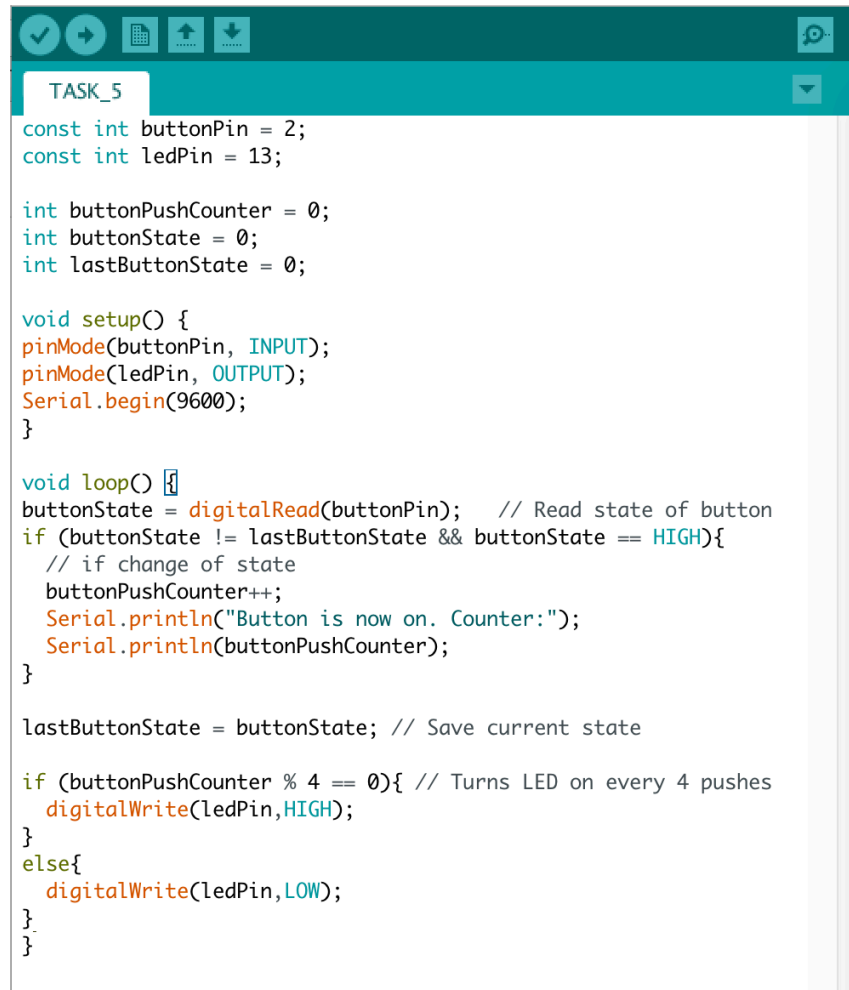
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

- a) Demonstrate to the TA that code is properly working
- b) Accurately describe the procedure on your report

TASK 5: Button State Change Detection

Once you've got a [pushbutton](#) (TASK 4) working, often it is desirable to execute a certain action based on the number of times the button is pushed. To do this, you need to know when the button changes state from off to on, and count how many times this change of state happens. This is called **edge detection**.



```
TASK_5
const int buttonPin = 2;
const int ledPin = 13;

int buttonPushCounter = 0;
int buttonState = 0;
int lastButtonState = 0;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  buttonState = digitalRead(buttonPin); // Read state of button
  if (buttonState != lastButtonState && buttonState == HIGH){
    // if change of state
    buttonPushCounter++;
    Serial.println("Button is now on. Counter:");
    Serial.println(buttonPushCounter);
  }

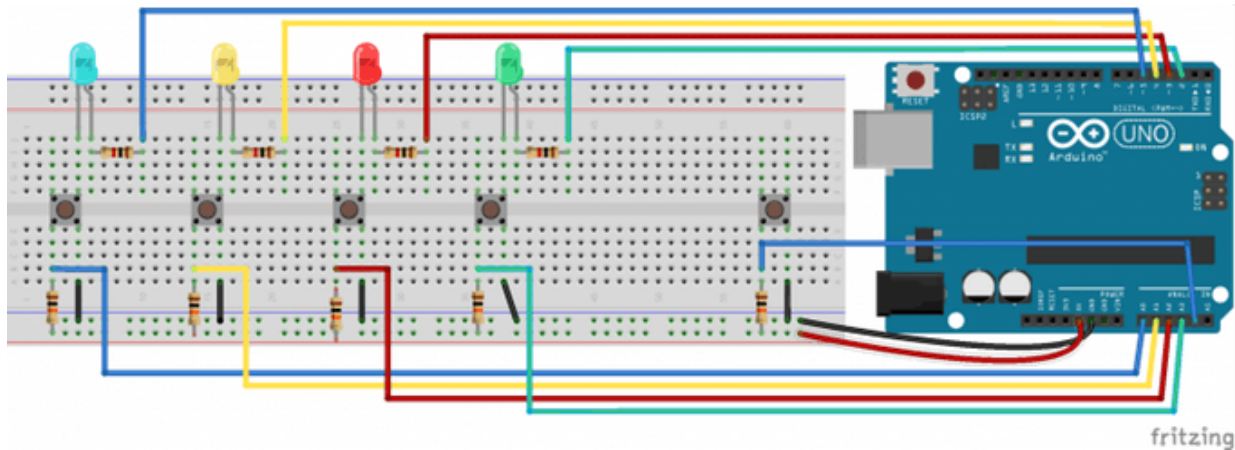
  lastButtonState = buttonState; // Save current state

  if (buttonPushCounter % 4 == 0){ // Turns LED on every 4 pushes
    digitalWrite(ledPin,HIGH);
  }
  else{
    digitalWrite(ledPin,LOW);
  }
}
```

Note: Button switches may not happen fast enough, and your code may count more than one switch at a time. Provide a solution to overcome this issue. Hint: Capacitor is a low pass filter.

SECTION 6: GAME CHALLENGE

Write an Arduino program to play a game of *Simon Says*. The program starts each round of the game by flashing all LEDs several times. The program flashes the four LEDs in a random sequence of length n . The player repeats the sequence by pressing the buttons next to the LEDs. The program verifies that the sequence of pressed buttons equals the sequence of flashed LEDs. If the sequence is correct, the program flashes all the LEDs once, generates a sequence of $n+1$ LEDs, and waits for the user to enter the sequence by pressing the buttons. If the sequence entered by the player is not correct, then the program flashes four times all LEDs, and starts over. The game terminates with $n = 8$, and it shows a winning flashing pattern.



Bill of Materials:

- 5x pushbuttons
- 1x Blue led
- 1x Yellow led
- 1x Red led
- 1x Green Led
- 4x 1k resistors
- 4x 10k resistors