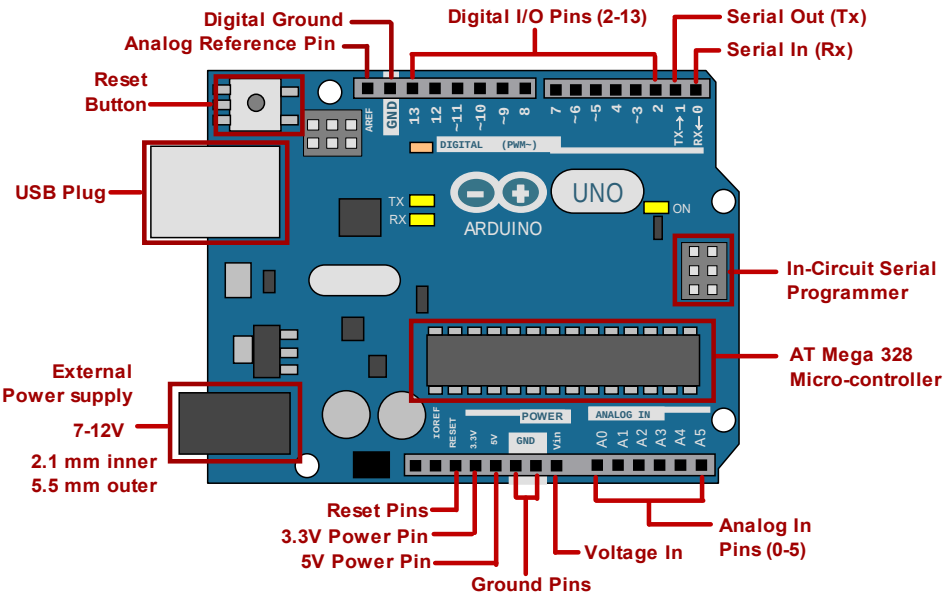
LAB 1 - ARDUINO BLINK

PROJECT, LAB, or HW

1 - ARDUINO BLINK



ITEMS NEEDED

- Arduino UNO R3 or Mega 2560 R3
- USB Cable
- LED
- 220-ohm resistor
- Pull-down resistor (at least 1k-ohm)
- Breadboard or breadboard shield
- Computer with Arduino IDE software installed
- Tact switch
- Wire

INTRODUCTION & BASICS

In this project (or exercise) we will have the ARDUINO micro-controller control the lighting of an LED. There are 5 exercises to do. Save all 5 programs (do not delete or over-write any of them). Name each file as indicated on the given code. For example, the exercise 1 program should be named "1_blink".

Micro-controllers are computers that are designed to sense and control things in the physical world. They can be used to turn lights on/off, run motors, sense sound, light, acceleration, etc. The Arduino platform has become popular in the hobbyist and academic communities.

We will program Arduino to receive signals and also send out signals through its input/output (I/O) pins. We will program Arduino to control both input and output of its digital pins.

We will also need to build some simple circuitry around the Arduino micro-controller. This requires us to use some standard electrical components breadboards, resistors, and LEDs.

Please read the sections in the handouts specified below.

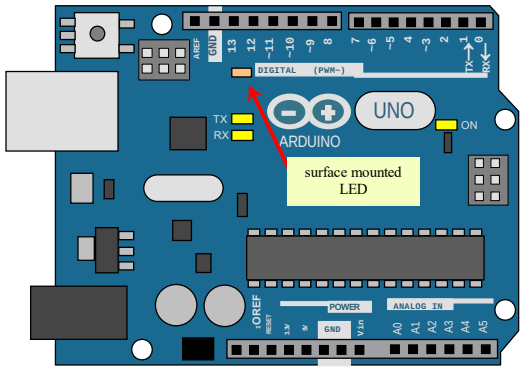
REFER to the ARDUINO GUIDE for more information on Arduino.
- Chapter 1 (Intro)
- Chapter 2 (Arduino IDE or software)
- Chapter 4 (I/O, digital vs. analog, PWM)

REFER to the CT GUIDE
- Chapter 6 (6.1 Wire; 6.2 Breadboards; 6.3 Resistors; 6.7 Diodes; 6.8 LEDs)

REFER to the C PROGRAMMING GUIDE
- (Section 4.2, "if" statement)

EXERCISES

1.1 BLINK SURFACE-MOUNTED LED



Here we get the "surface mounted" (built onto the board) LED (connected to pin 13) to blink on and off. In this exercise you only need the Arduino board and cable. We aren't using the breadboard or circuit components yet.

LED blinking is often the first and simplest exercise with a micro-controller. This is done by changing digital pin 13 (HIGH or LOW). There is a nice YouTube video series by Jeremy Blum (in association with element14.com). His first video tutorial (#1) shows you how to set up the software and do the LED blink project (https://www.youtube.com/watch?v=fCxzA9_kg6s). Although this is a simple exercise, the basic process of controlling the Arduino input/output is fundamental and forms the basis for more advanced tasks.

Here are some important points:
- Arduino boards have a built in "surface-mounted" LED connected to pin 13.
- All Arduino programs must have 2 functions: setup( ) and loop( ).
- setup ( ) is done first and once. loop ( ) is done after setup() and is done repeatedly forever.
- In digital electronics, LOW means 0V and HIGH means 5V.

"pinMode()" configures an I/O pin as either an INPUT or an OUTPUT.
"digitalWrite()" commands either a HIGH or LOW to a certain pin
"delay()" pauses the Arduino computer for a certain number of milliseconds (1 ms = 1/1000 seconds).

The delay( ) function pauses the Arduino processor for a period of time. The state of the I/O pins is held unchanged during this period. Thus, the delay command is used to control the duration of the LED on and off times.

CODE

Type the code given below into the Arduino software EXACTLY. C++ is case sensitive, so duplicate the case exactly as well. Save the program as indicated on the first line of code below ("1_blink").

```
// 1_Blink.ino ----------------
// Blink LED on Arduino (pin 13) repeatedly

int ledPin = 13;     // specify pin


void setup()   // done once
{
   pinMode(ledPin, OUTPUT);  // set pin as output
}

void loop()    // done repeatedly
{
  digitalWrite(ledPin, HIGH);          // LED on
  delay (300);                         // hold it for this many milliseconds
  digitalWrite(ledPin, LOW);           // LED off
  delay(300);                          // hold it
}

// end ---------------------------------
```

FOLLOW-UP ACTIVITIES

Alter the delay times (change the number in the delay ( ) function) and observe the changes in the blinking LED. Make the LED blink faster and then record the result. This demonstrates how software can easily alter the behavior of hardware.

1.2 MOMENTARY LED

Exercises 2 through 5 require the use of the Arduino, breadboard, and the circuit components (LED, wire, resistors, tact switch, etc.).

In exercise 2, the LED turns ON while the tact switch button is pushed down, and it turns OFF when the button is released. Jeremy has this on YouTube as well on his tutorial #2. (https://www.youtube.com/watch?v=_LCCGFSMOr4).

THINGS TO KNOW

- Always build and modify circuits with power OFF.
- IMPORTANT! NEVER create any "short circuits". This means never connect different voltage source values (e.g.,5 VDC & GND, which is 0 V). This can damage the Arduino or your computer.
- "Ground" (GND) in electric circuits usually means the (-) side of the power supply (or battery). GND is often considered 0 V.

The momentary "tact" (or tactile, or button) switch. It is a spring-loaded switch that, while pushed, electrically connects one set of legs to another set of legs. An internal spring pushes the button back out when the button is released, causing the legs to disconnect. The switch has 2 pairs of legs, with each pair connected internally all the time.

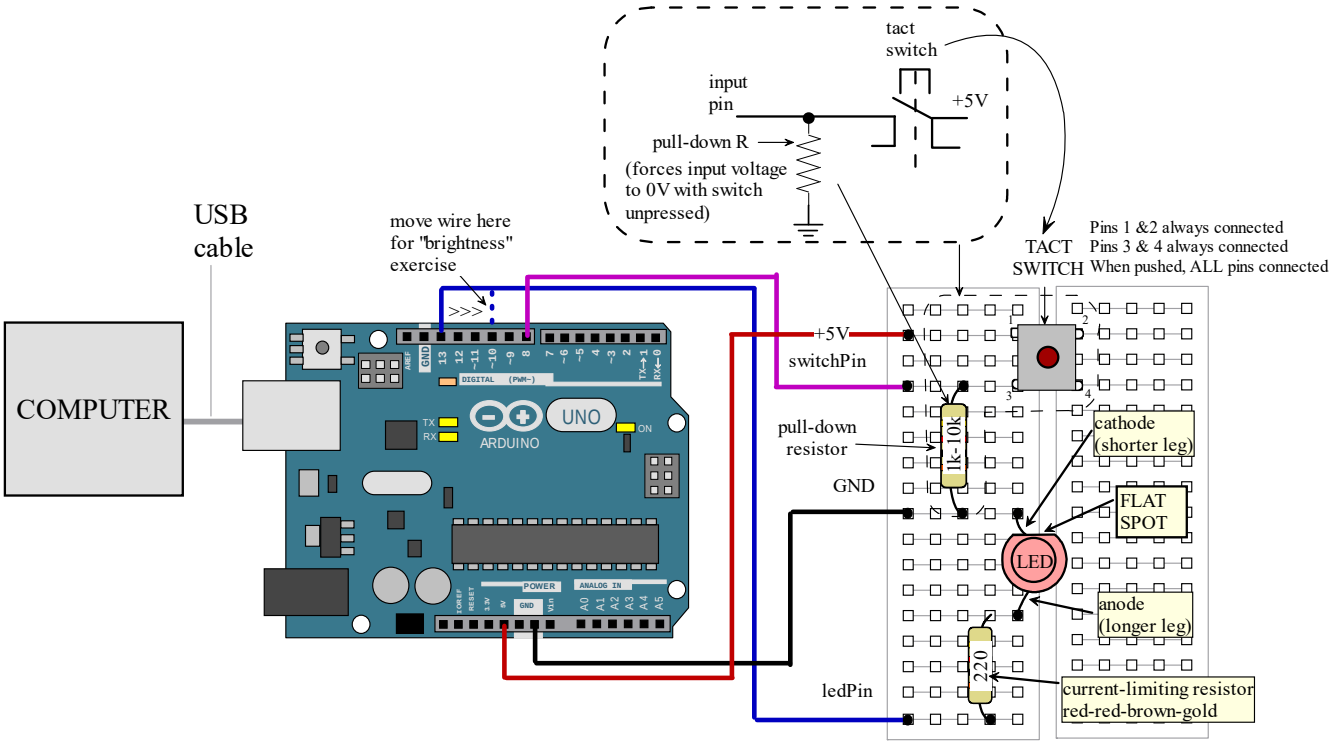
PROGRAMMING

This code uses the "if" statement. We use the "if" statement when we only want to execute certain statements when certain CONDITIONS are met. The "if" statement tests whether a condition is TRUE or FALSE. If it is true, then a "block" of statements (inside the following curly braces) is executed. If false, the block of statements is not executed.

CIRCUIT DIAGRAM

The programming code MUST match the physical circuit (e.g., the specified pins).
The LED must be inserted in the correct direction (cathode closer to GND).
There are 2 resistors used. The 220-ohm resistor (red-red-brown) must be in series with the LED (it limits the current through the LED). The pull-down resistor must be greater than 1k-ohm. The purpose of the pull-down resistor is to force the voltage at the "switch pin" to be 0V unless the button is pushed (in which case, the voltage at the switch pin goes to 5V).

Note will be the same circuit for exercises 2 through 5, except there is one very small change to be made for exercise 5. If your kit has a breadboard shield, you may assemble the shield onto the Arduino board. The breadboard itself has 2-sided sticky tape so it sticks to the shield.



CODE

You may copy and paste the given code into the Arduino software, but afterwards look through it and insert spacing and hard returns so it looks EXACTLY like the code given. Save the file as "2_momentary".

```
// 2_Momentary.ino ---------------------------------------
// LED turns on while button is pushed, off when button released

int ledPin = 13;
int switchPin = 8;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(switchPin, INPUT);
}

void loop()
{
  if (digitalRead(switchPin) == HIGH)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  digitalWrite(ledPin, LOW);
}
// end ---------------------------------------------------------------------
```

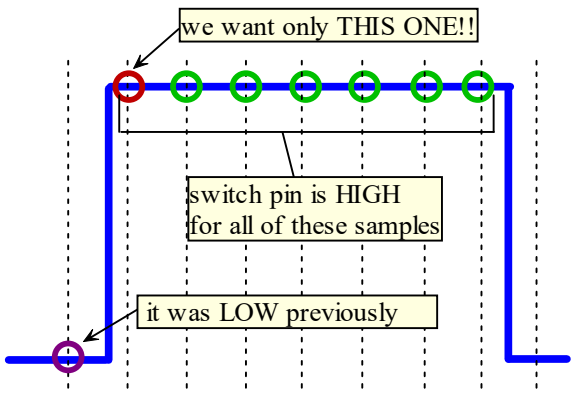
1.3 TOGGLE LED

Now we will change the behavior of the circuit without changing the circuit hardware at all! This is really cool and is the great advantage of computer-controlled systems. Now the LED should TOGGLE on and off with each push and release of the button. (Jeremy covers this in the same video as before)

THINGS TO KNOW

In the prior program, you simply had to program based on the current state of the button and the LED. Now the program must keep track of both the current and prior states, and so you'll need to add a variable that keeps track of the PREVIOUS state the button.

Remember that you are programming inside of the loop ( ) function which repeats over and over. Programming in this loop environment can get tricky. While physical events happen continuously, the computer can only sense these events in discrete glimpses (like peaking by briefly opening your eyes)! Each time through the LOOP function, the "digitalRead" command gives Arduino a brief glimpse at a physical event. For example, we want the LED to toggle (HIGH to LOW, or LOW to HIGH) with a button push. So we must sense the button push and use the result in a condition. We may be tempted to write the condition as: "if (currentButton == HIGH)". But this will not work right.



During the brief period that we press the button, Arduino will glimpse the button as HIGH for many time cycles (e.g. 30 cycles), and it will toggle the LED each time (not what we wanted!). We really just want the FIRST time the button is HIGH. In code, this means we want the button is HIGH during current time loop and it was LOW in the previous time loop, so "if (currentButton == HIGH && lastButton = = LOW)".

In our code we will a variable called "ledState" which is of the data type BOOLEAN. Boolean data types are either TRUE or FALSE. This makes it easy to toggle using the NOT logical operator ("!"). NOT TRUE = FALSE and NOT FALSE = TRUE. Equivalently, TRUE = HIGH and FALSE = LOW.

CODE

```
//3_Toggle.ino -------------------------------
// Toggle LED on and off with each push and release of the button

int ledPin = 13;
int switchPin = 8;
boolean ledOn = false;                  // booleans are true or false, or high or low
boolean lastButton = LOW;
boolean currentButton = LOW;

void setup()
{
     pinMode(ledPin, OUTPUT);           //pin that powers the LED
     pinMode(switchPin, INPUT);         //pin that READS the state of switch
}

void loop()
{
     currentButton = digitalRead(switchPin); //get current switch state

     //test if button was just pushed (vs. last time cycle)
     if (lastButton == LOW && currentButton == HIGH)
     {
         ledOn = !ledOn;                        //toggle LED state
     }
     lastButton = currentButton;                // reset last button state
     digitalWrite(ledPin, ledOn);               // light LED
}
```

RESULT

Try switching the LED on and off by pushing the button quickly. You may find that sometimes the LED does not respond.

1.4 TOGGLE WITH DE-BOUNCE

Let's fix that problem where the button push did not seem to work all the time.

THINGS TO KNOW

Why does is the LED periodically not respond?

This is because mechanical switches are not perfect. When pushed the contacts tend to bounce for a short period of time which causes the input signal to not switch "cleanly" as shown by the figure. The electronics are fast enough to register this unclean signal and, as a result, our program sometimes works incorrectly. This is a GREAT example of how an engineer must solve a problem in the real world. We could try to solve it with a mechanical or electrical design change in the physical switch to make it perform better, but it is much easier to solve it with software programming.

Our approach will be to sense the button push, then wait briefly for the button to stop bouncing. Then after the brief wait, read the button's pin. In addition, we will perform these steps inside of our own custom new function called "debounce( )".

CODE

```
// 4_debounce.ino ------------------------------
// toggle LED with debounce function

int ledPin = 13;                //pin to power LED
int switchPin = 8;              //pin that tracks the switch's state
boolean ledState = false;
boolean lastButton = LOW;
boolean currentButton = LOW;

void setup()
{
  pinMode(ledPin, OUTPUT);      //LED pin is an output
  pinMode(switchPin, INPUT);    //switch pin is an input
}


void loop()
{
   currentButton = debounce(lastButton);     // get button state, use debounce()

  if (lastButton == LOW && currentButton == HIGH)  // button pushed?
  {
    ledState = !ledState;                   // toggle LED
  }
  lastButton = currentButton;               // reset last button
  digitalWrite(ledPin, ledState);           // light LED
}

boolean debounce(boolean last)  // define debounce() fcn -------------
{
   boolean current = digitalRead(switchPin); // read switch pin
  if (last != current)                       // did switch pin change?
  {
    delay(5);                                // wait for bouncing to stop
    current = digitalRead(switchPin);        // now read pin
  }
  return current;                            //return pin reading
}

// end ----------------------------------------------------------
```

1.5 LED STEP BRIGHTNESS

Now let's change the behavior again, mostly with a programming change and one small hardware/wiring change. In this case we will change the LED brightness with each button push and release. You can change LED brightness by changing the voltage applied to it. Lower voltages produce less brightness. You'd have to connect the LED to a so-called "analog output" pin, which can vary its voltage output over a range of values. Arduino does not have true analog output pins. Instead, Arduino has so-called PULSE-WIDTH MODULATION (or PWM). In reality the LED will blink, but it will do so too fast for us to see the blinking. Instead we would see a dimmed LED.

CIRCUIT

There is a SMALL HARDWARE CHANGE from the last exercise!
Switch the LED pin from 13 to a PWM pin (say pin ~10) (or any other pin marked "~" or PWM). But if you use a pin other than pin 10, make sure you alter the given code below to reflect the pin you used!

CODE

```
// 5_Brightness.ino ----------------------------------------
// Make LED step up in brightness with button push

int ledPin = 10;                      //switch led to pwm pin ~10
int switchPin = 8;
int ledLevel = 0;
boolean lastButton = LOW;             // booleans are high or low
boolean currentButton = LOW;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(switchPin, INPUT);
}

void loop()
{
  currentButton = debounce(lastButton);           // get button state (with debounce)

  if (lastButton == LOW && currentButton == HIGH) // is button pushed?
  {
    ledLevel = ledLevel + 51;        //increment LED state (255/51 = 5 equal levels)

  }
  lastButton = currentButton;                     // reset last button
  if (ledLevel > 255) ledLevel = 0;               // reset brightness when reach end
  analogWrite(ledPin, ledLevel);                  // light LED
}

boolean debounce(boolean last) //-------------------------------------------
{
   boolean current = digitalRead(switchPin); // read switch pin
  if (last != current)                       // did switch pin change?
  {
    delay(5);                                // wait for bouncing to stop
    current = digitalRead(switchPin);        // now read pin
  }
  return current;                            //return pin reading
}
```

```
// end ---------------------------------------
```

2 - SUBMISSION

Complete ALL exercises. Save ALL programs.
Then demonstrate the exercises to the instructor.

The instructor MAY allow the option of video submission (but he will let you know if this is the case)

If video upload is allowed, then follow the steps below:

Submit short videos onto Canvas. The videos should show the hardware working.
VIDEOS MUST BE NAMED THE SAME AS PROGRAM FILES (*.ino) given above.
VIDEO SIZES ARE LIMITED!! (max 30 Mb each).
Submit videos all at once. You may produce the videos at different times, but upload all at once.
Videos should just show the behavior.
Don't video the code. Don't narrate. These are not needed and make the videos longer.
Each video should not be more than about 5 SECONDS LONG.

Submit short videos for 5 exercises (see below):

1. Arduino Blink - show the surface-mounted LED blinking
2. Momentary - show yourself pushing the button and the LED responding.
3. Toggle (do NOT submit this video - but still do the exercise!)
4. Debounce - push the button ~ 8 times in about 4 seconds (& LED responding).
5. LED brightness - push button & show LED changing brightness.