POLITECNICO
MILANO 1863

Arduino

Introduction

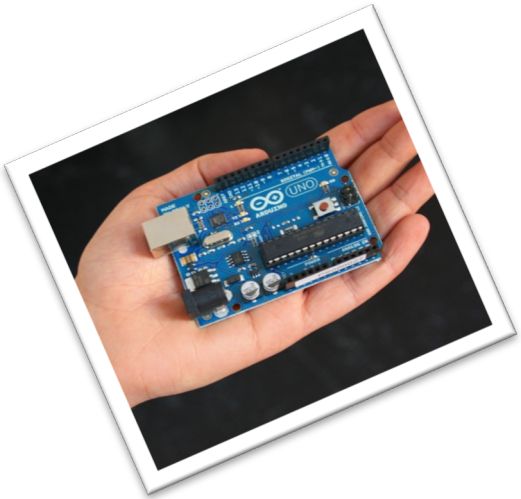
Ver 2.0 - 2020
Ver 1.0 - 2012 Oct

Fabio Salice

Arduino

- Open-source electronics prototyping platform based on microcontrollers
  - Hardware: simple open hardware design for the Arduino board with an Atmel AVR processor and on-board input/output support (source: wikipedia)
  - Software: standard programming language compiler and the boot loader that runs on the board (source: wikipedia)

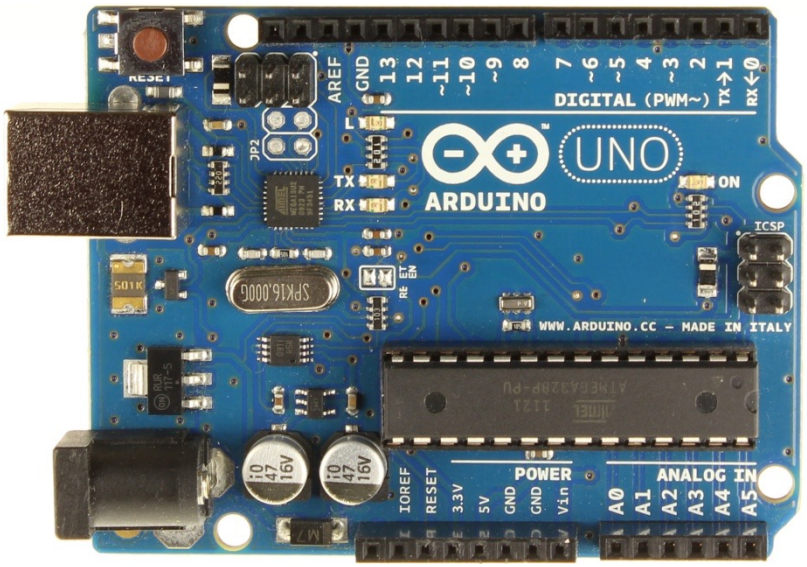
"**Arduino**" is an Italian masculine first name, meaning "**brave friend**"



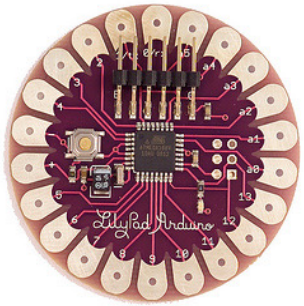
Official website:www.Arduino.cc

Name from King
Arduin of Ivrea (955–1015)

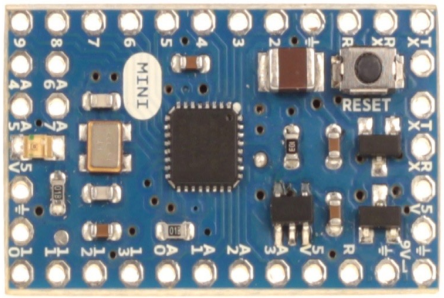
Arduino Boards



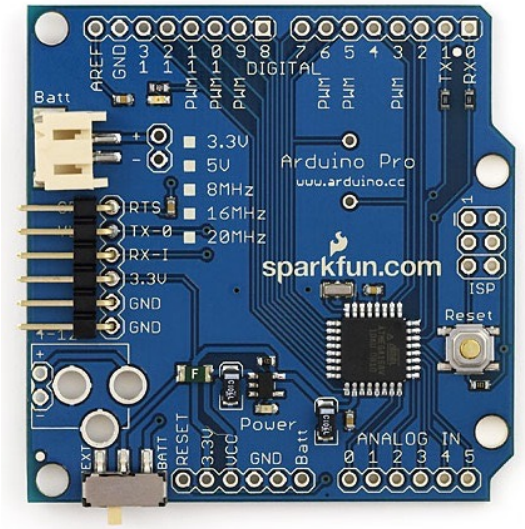
Arduino Uno_R3



Arduino LilyPad

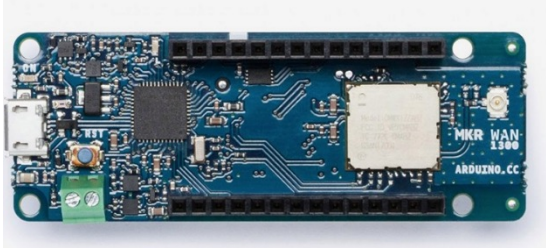


Arduino Mini

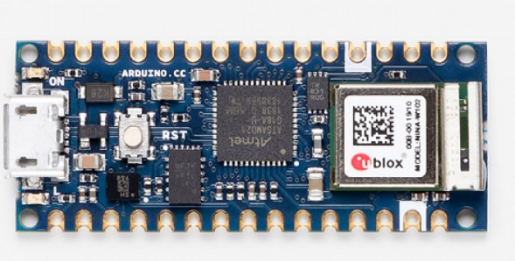


Arduino Pro

Arduino Boards

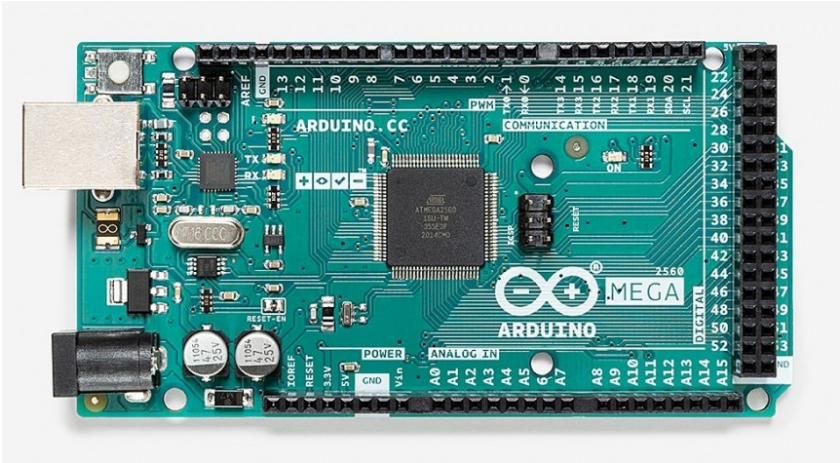


Arduino MKR WAN (Lora conn.)



Arduino NANO IOT

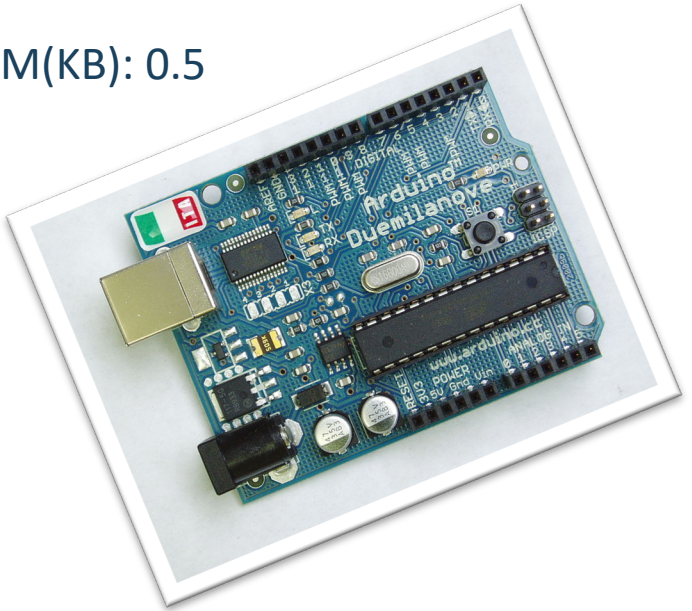
- WiFi
- BLE
- 6 axis IMU



Arduino Mega

Arduino Boards

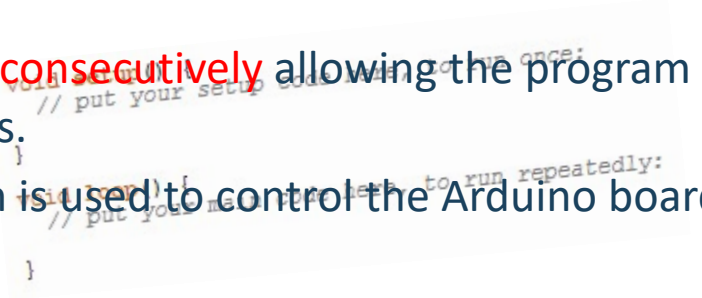
- Arduino in the lab : Arduino 2009
  - Microcontroller: ATmega168/328P
    - Frequency: 16 MHz
    - Voltage: 5 V
    - Flash (KB): 16/32 - EEPROM (KB): 0.5/1 - SRAM(KB): 0.5
    - Digital I/O: 14 where 6 with PWM
    - Analog Input: 6
    - USB interface: FTDI



http://arduino.cc/en/Main/ArduinoBoardDuemilanove

Arduino-Language

- Arduino programs can be divided into three main parts: structures, values (variables and costants), functions
- Structure
  - `[constant & variable]`
  - `setup()`
    - The `setup()` function is called when a sketch starts. It has to be used to initialize variables, pin modes, start using libraries, etc.
    - The `setup()` function will run only once, after each power-up or reset of the Arduino board.
  - `loop()`
    - the `loop()` function loops consecutively allowing the program to change and respond as it runs.
    - Code in the `loop()` section is used to control the Arduino board.



Arduino-Language

- Structure
  - `setup()`
  - `loop()`

```
int buttonPin = 3;

// setup initializes serial and the button pin
void setup()
{
  beginSerial(9600);
  pinMode(buttonPin, INPUT);
}

// loop checks the button pin each time,
// and will send serial if it is pressed
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');

  delay(1000);
}
```

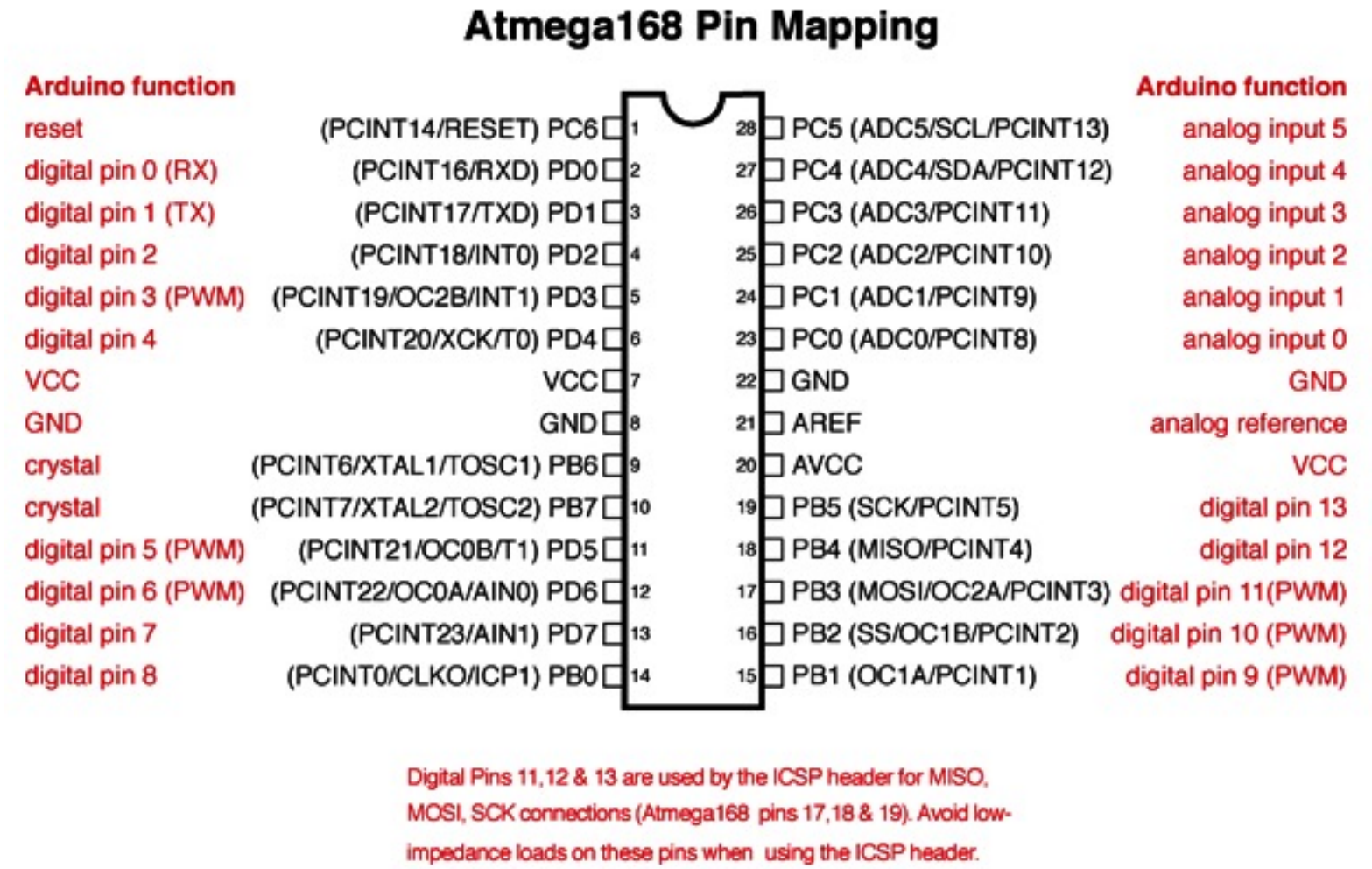
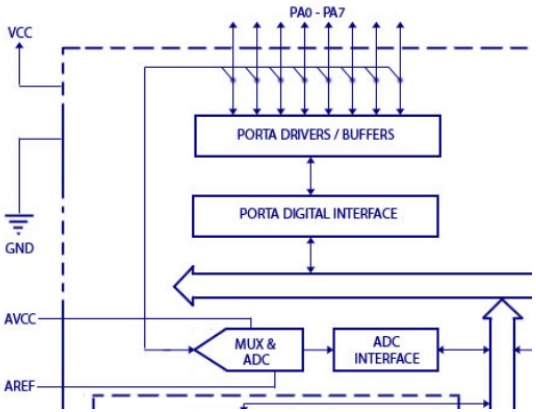
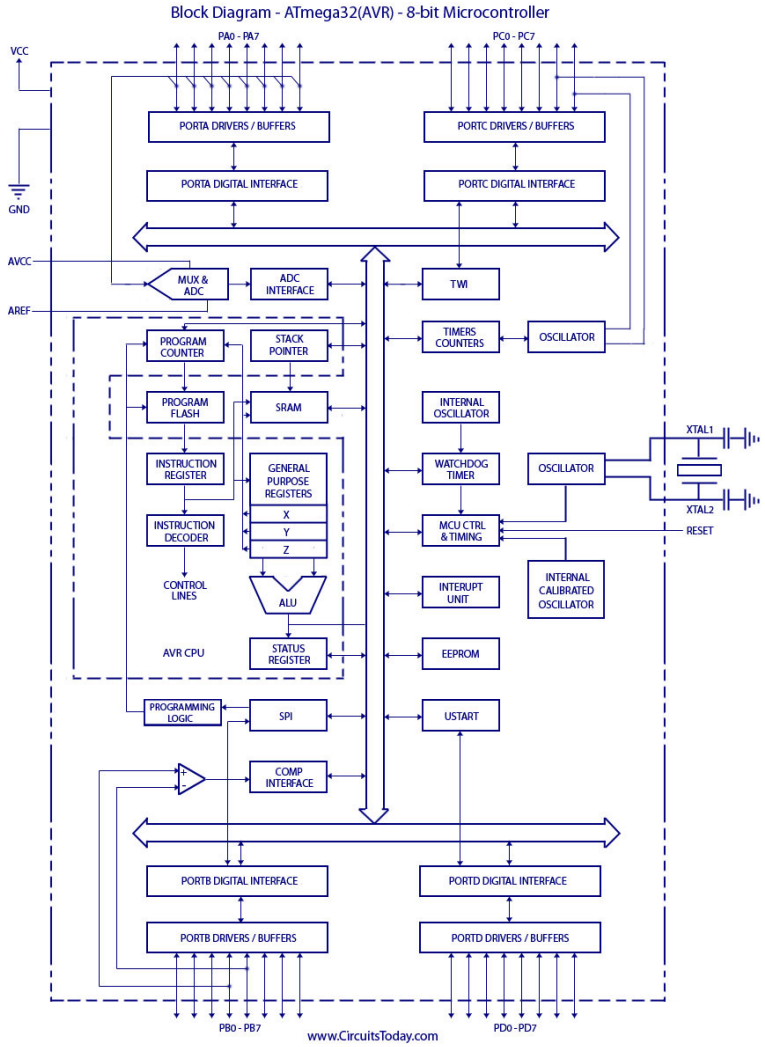
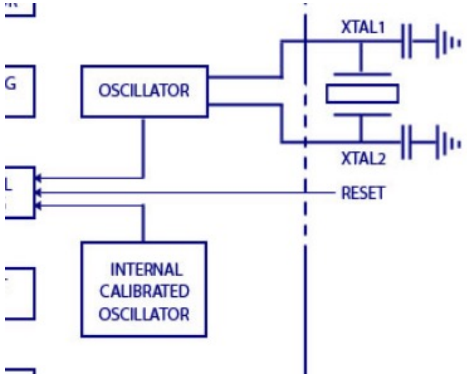
Arduino-Language

- The microntroller pins has to be configured. Two different types: digital and analog pins
  - Note: the vast majority of Atmega analog pins can be configured and used exactly as the digital pins.
- Digital pins
  - Configuration function: `pinMode`(*pin*, *mode*) where mode is `INPUT` or `OUTPUT`
    - Example:

```
int buttonPin = 3;
…
pinMode(buttonPin, INPUT);
```

  - Default: inputs (high impedence)
    - **Note**: a floating input pin will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacity coupling the state of a nearby pin. Pull-up/pull-down resistor is required (or a special configuration) .

Arduino-Language

- Digital pins as INPUT
  - Impedance: high
  - Pull-up or pull-down resistor:
    - 10k Ω (typical value)
    - 20K Ω pull-up resistors built into the Atmega can be accessed from software.
      - ```
        pinMode(pin, INPUT);      //set pin to input
        digitalWrite(pin, HIGH); //turn on pull-up resistors
        ```
- Digital pins as OUTPUT
  - Impedance: low (max current 40mA)
    - Note: not enough current to run most relays, solenoids, or motors.
  - ATTENTION: overcoming max current can damage or destroy the output transistors in the pin ("dead" pin).
    For this reason it is a good idea to connect OUTPUT pins to other devices with 470Ω or 1kΩ resistors, unless maximum current draw from the pins is required for a particular application.
        ```

- Digital pins
  - NOTE: the pull-up resistors are controlled by the same registers that control whether a pin is `HIGH` or `LOW`. Consequently a pin configured to have pull-up resistors turned on when the pin is an `INPUT`, will have the pin configured as `HIGH` if the pin is then switched to an `OUTPUT` with `pinMode()`. This works in the other direction as well, and an output pin that is left in a `HIGH` state will have the pull-up resistors set if switched to an input with `pinMode`().

Arduino-Language



ATMega 328


PA0 - PA7
VCC
PORTA DRIVERS / BUFFERS
PORTA DIGITAL INTERFACE
GND
AVCC
MUX & ADC
ADC INTERFACE
AREF


Block Diagram - ATmega32(AVR) - 8-bit Microcontroller
PA0 - PA7
PC0 - PC7
VCC
GND
AVCC
AREF
PORTA DRIVERS / BUFFERS
PORTC DRIVERS / BUFFERS
PORTA DIGITAL INTERFACE
PORTC DIGITAL INTERFACE
MUX & ADC
ADC INTERFACE
TWI
PROGRAM COUNTER
STACK POINTER
TIMERS COUNTERS
OSCILLATOR
PROGRAM FLASH
SRAM
INTERNAL OSCILLATOR
INSTRUCTION REGISTER
GENERAL PURPOSE REGISTERS
X
Y
Z
WATCHDOG TIMER
OSCILLATOR
INSTRUCTION DECODER
MCU CTRL & TIMING
CONTROL LINES
ALU
INTERUPT UNIT
INTERNAL CALIBRATED OSCILLATOR
AVR CPU
STATUS REGISTER
EEPROM
PROGRAMMING LOGIC
SPI
USTART
COMP INTERFACE
PORTB DIGITAL INTERFACE
PORTD DIGITAL INTERFACE
PORTB DRIVERS / BUFFERS
PORTD DRIVERS / BUFFERS
PB0 - PB7
PD0 - PD7
XTAL1
XTAL2
RESET
www.CircuitsToday.com


XTAL1
OSCILLATOR
XTAL2
RESET
INTERNAL CALIBRATED OSCILLATOR

Arduino-Language

- ATmega168/328-Arduino Pin Mapping
  - 14 digital pins: each of the 14 digital pins on the Duemilanove can be used as an input or output, using `pinMode()`, `digitalWrite()`, and `digitalRead()` functions (They operate at 5 volts)
    - Pins D0, D1: Serial transmission (0 (RX) and 1 (TX))
      - These pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.
    - Pin D2, D3: External Interrupts
      - These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
        - See the attachInterrupt() function for details.
    - Pins D3, D5, D6, D9, D10, and D11: PWM
      - Provide 8-bit PWM output with the analogWrite() function.

Arduino-Language

- ATmega168/328-Arduino Pin Mapping
  - Pins D10, D11, D12 and D13: SPI
    - 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
  - Pin D13: LED
    - There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
  - 6 analog inputs: each input provide 10 bits of resolution. By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the `AREF` pin and the `analogReference()` function. Additionally, some pins have specialized functionality:
    - Pins A4, A5: I2C
      - 4 (SDA) and 5 (SCL). Support I2C (TWI) communication using the Wire library.
    - AREF: Reference voltage for the analog inputs.
      - Used with analogReference().
    - Reset: Bring this line LOW to reset the microcontroller

Arduino-Language

- Variables
  - Constants
    - Digital pin direction (mode): **`INPUT`** and **`OUTPUT`**
      - E.g. `pinMode(pin, INPUT); //set pin to input`
    - Digital pin levels: **`HIGH`** and **`LOW`**
      - Pin configured as an `INPUT`:
        a) by reading (`digitalRead`) the pin, the microcontroller will report `HIGH` if a voltage of 3 volts or more is present at the pin.
        b) by writing `HIGH` (`digitalWrite`), this will set the internal 20K pullup resistors, which will steer the input pin to a `HIGH` reading unless it is pulled `LOW` by external circuitry.
           a) same as `INPUT_PULLUP`
      - Pin configured as on `OUTPUT`, and set to `HIGH` with `digitalWrite`, the pin is at 5 volts. In this state it can source current.
        - e.g. to another pin configured as an output, and set to `LOW`.
        - E.g. `digitalWrite(ledPin, HIGH); //sets the LED on`

Arduino-Language

- Variables (cont)
  - Constants (other)
    - Boolean (true, false), integer, floating
      - U & L formatters for integer constants.
        - By default integer constant are `int`. For different type:
        - `u` or `U` to force the constant into an unsigned data format.
          - Example: `33u`
        - `l` or `L` to force the constant into a long data format.
          - Example: `100000L`
          - Example: `32767ul //unsigned long`
  - Data types (like C/C++)
    - `void`, `boolean` (byte: 0 is false), `char` (byte), `unsigned char`, `byte`, `int` (2 bytes), `unsignet int`, `word` (like unsigned int), `long` (4 bytes), `unsigned long`, `float` (4 bytes), `double` (as a float), `string` - char array, `String` – object, `array`

Arduino-Language

- Variables (cont)
  - Data types (like C/C++)
    - string - char array
      - Examples:
        - `char Str1[15];`
        - `char Str2[8] = {'a','r','d','u','i','n','o'};`
        - `char Str3[8] = {'a','r','d','u','i','n','o','\0'};`
        - `char Str4[ ] = "arduino"; //implicit size`
        - `char Str5[8] = "arduino";`
        - `char Str6[15] = "arduino"; //over sized`

Arduino-Language

- Variables (cont)
  - Example (use of serial interface)
    - `char* myStrings[]={"This is string 1", "This is string 2", "This is string 3", "This is string 4", "This is string 5","This is string 6"};`

```
void setup(){
Serial.begin(9600);
}

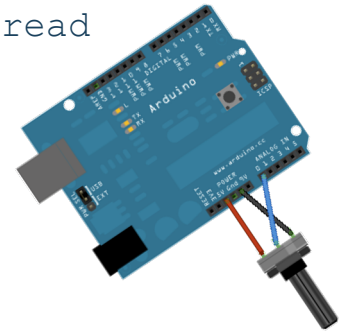
void loop(){
   for (int i = 0; i < 6; i++){
       Serial.println(myStrings[i]);
       delay(500);
   }
}
```

Arduino-Language

- Variables (cont)
  - Data types (like C/C++)
    - String – object
      - Functions:
        - `String()`: it constructs an instance of the String class.
          Examples:
          String thisString = String(13) → Returns you the String "13".
          String thisString = String(13, HEX) → Returns the String "D"
          String thisString = String(13, BIN) → Returns the String "1011"
        - `charAt()`, `compareTo()`, `concat()`, `endsWith()`, `equals()`, `equalsIgnoreCase()`, `getBytes()`, `indexOf()`, `lastIndexOf()`, `length()`, `replace()`, `setCharAt()`, `startsWith()`, `substring()`, `toCharArray()`, `toLowerCase()`, `toUpperCase()`, `trim()`

Arduino-Language

- Variables (cont)
  - Data types (like C/C++)
    - String – object
      - operators:
        - `[]`: element access (Same as `charAt()`)

```
char thisChar = string1[n]
thisChar = string1.charAt(n);
```

        - `+`: concatenation (Same as `string.concat()`)

```
string1 += string 2;
string1.concat(string2);
```

        - `==` : comparison (same as `string.equals()`)

```
if (stringOne.equals(stringTwo)) {…
if (stringOne ==stringTwo) {...
```

  - Variable scope: like C/C++
  - Static: static variables persist beyond the function call preserving their data between function calls

Arduino-Language

- Control structures
  - Like C/C++: `if`, `if … else`, `for`, `while`, `do … while`, `break`, `continue`, `return`, `goto`
- Operators
  - Arithmetic
    - `=` (assignment operator), `+` (addition), `–` (subtraction), `*` (multiplication), `/` (division), `%` (modulo), `++` (increment)
  - Comparison
    - `==` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than), `<=` (less than or equal to), `>=` (greater than or equal to)
  - Boolean
    - `&&` (and), `||` (or), `!` (not)
  - Bitwise (bit to bit)
    - `&` (and), `|` (or), `^` (xor), `~` (not), `<<` (bitshift left), `>>` (bitshift right)

Arduino-Language

- Functions
  - Digital I/O
    - ```
      int ledPin = 13; // LED  - pin D13
      int inPin = 7; // pushbutton - digital pin D7
      int val = 0; // variable to store the read value
      void setup()
      { pinMode(ledPin, OUTPUT); //sets pin D13 as output
        pinMode(inPin, INPUT);// sets pin D7 as input
      }
      void loop()
      { val = digitalRead(inPin); // read the input pin
        digitalWrite(ledPin, val); //sets the LED to the
                                   button's value
      }
      ```
      ```

- Functions
  - Analog I/O
    - `analogReference(type)`
      - Configures the reference voltage used for analog input (i.e. the value used as the top of the input range).
      - Type:
        - **`DEFAULT`**: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
        - **`INTERNAL`**: an built-in reference, equal to 1.1 volts on the ATmega168/ATmega328
        - **`EXTERNAL`**: the voltage applied to the AREF pin (0 to 5V only) is used as the reference.
      - Warning (see also http://arduino.cc/en/Reference/AnalogReference):
        1. Don't use anything less than 0V or more than 5V for external reference voltage on the AREF pin!
        2. If you're using an external reference on the AREF pin, you must set the analog reference to EXTERNAL before calling analogRead().

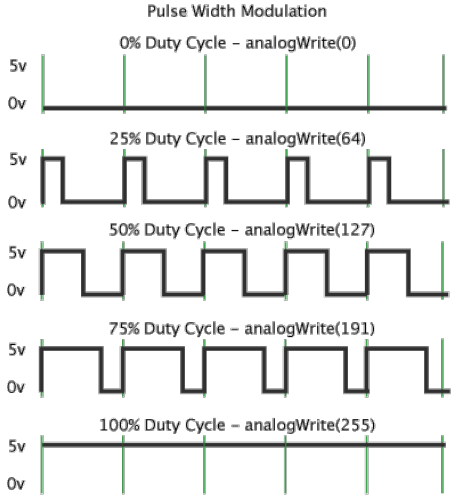
Arduino-Language

- Functions
  - Analog I/O (cont.)
    - `analogRead(pin)` where `pin` is from `A0` to `A6`
      - Returns a value from 0 to 1023; if reference is 5V the resolution is 4.9 mV per unit.
    - `// potentiometer wiper (middle terminal) is connected to analog pin A3 outside leads to ground and +5V`

```
int val = 0; // variable to store the value read
int InPin = A3;
void setup() {
  Serial.begin(9600);//  setup serial
}
void loop() {
  val = analogRead(InPin); // read the input pin
  Serial.println(val); // debug value
}
```



- Functions
  - Analog I/O (cont.)
    - `analogWrite(pin, value)` where `pin` is 3, 5, 6, 9, 10 and 11 and `value` is the duty cycle between 0 (always off) and 255 (always on).
      - pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` (or a call to digitalRead() or digitalWrite() on the same pin).
      - The frequency of the PWM signal is approximately 490 Hz.



Arduino-Language

- Functions
  - Analog I/O (cont.)
    -
```
int ledPin = 9; // LED connected to digital pin 9
int analogPin = A3; // potentiometer connected to analog 3
int val = 0; // variable to store the read value

void setup(){
   pinMode(ledPin, OUTPUT); // sets the D pin as output
}
void loop() {
  val = analogRead(analogPin); // read the input pin
  analogWrite(ledPin, val/4);   // analogRead values go
                                from 0 to 1023,
                                analogWrite values
                                from 0 to 255

}
```

Arduino-Language

- Functions
  - Advance I/O
    - `tone(pin, frequency, [duration])` where `pin` is a digital pin, `frequency` is the frequency in Hz (`unsigned int`) and `duration` is the duration time in milli seconds (`unsigned long`)
      - Generates a square wave of the specified frequency (and 50% duty cycle) on a pin.
      - A duration can be specified, otherwise the wave continues until a call to `noTone()`.
      - Only one tone can be generated at a time
      - `tone()` function will interfere with PWM output on pins 3 and 11.
    - `map(value, fromLow, fromHigh, toLow, toHigh)`
      - ```
        long map(long x, long in_min, long in_max, long out_min, long out_max)
        {
           return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
        out_min;
        }
        ```
        ```

- Functions
  - Advance I/O
    - Plays a pitch that changes based on a changing analog input

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  int sensRead = analogRead(A0); // read the sensor
  Serial.println(sensRead); //print the read value
// map the analog input range (in this case, 400 - 1000 from the
photoresistor) to the output pitch range (120 - 1500Hz) change
the minimum and maximum input numbers below depending on the
range your sensor's giving:
  int thisPitch = map(sensRead, 400, 1000, 120,
1500);
  tone(9, thisPitch, 10); // play the pitch
  delay(1);  // delay in between reads for stability
}
```

- Functions
  - Advance I/O
    - `shiftOut(dataPin, clockPin, bitOrder, value)` where `pin` is the pin on which to output each bit, `ClockPin` is the pin to toggle once the dataPin has been set to the correct value (`int`) , `bitOrder` which order to shift out the bits (either `MSBFIRST` or `LSBFIRST`) and `value` the data to shift out (`byte`)
      - Shifts out a byte of data one bit at a time.
      - The `dataPin` and `clockPin` must already be configured as outputs by a call to `pinMode()`.
      - `shiftOut` is currently written to output 1 byte (8 bits) so it requires a more step operation to output values larger than 255.
        - ```
          long int data;
          for (i=0; i<4; i++) {
             shiftOut(dataPin, clockPin, MSBFIRST, (data >> 8));
          }
          ```
      - Each bit is written in turn to a data pin, after which a clock pin is pulsed (taken high, then low) to indicate that the bit is available.
          ```

- Functions
  - Advance I/O
    - `pulseIn(pin, value, [timeout])` where `pin` is the input pin, `value` is type of pulse to read (`HIGH` or `LOW`) (`int`) and `timeout` is the number of microseconds to wait for the pulse to start (default is one second) (`unsigned long`)
      - the length of the pulse (in microseconds) or 0 if no pulse started before the timeout (`unsigned long`).
    -
```
int pin = 7;
unsigned long duration;
void setup() {
  pinMode(pin, INPUT);
}
void loop() {
  duration = pulseIn(pin, HIGH);
}
```

Arduino-Language

- Functions
  - Time
    - `millis()`
      - Returns the number of milliseconds (`unsigned long`) since the Arduino board began running the current program.
        - Back to 0 in 50 days
    - `micros()`
      - Returns the number of microseconds (`unsigned long`) since the Arduino board began running the current program (resolution in 4 microsec)
        - Back to 0 in 70min
    - `delay(value)` where `value` is the delay time in milliseconds `(unsigned long)`
    - `delayMicroseconds(value)` where `value` is the delay time in microseconds `(unsigned long)`

Arduino-Language

- Functions
  - Math
    - `min(x,y)`
      - Calculates the minimum of two numbers.
    - `max(x,y)`
      - Calculates the maximum of two numbers.
    - `abs(x)`
      - Computes the absolute value of a number.
    - `map(value, fromLow, fromHigh, toLow, toHigh)`
      - Re-maps a number from one range to another.
    - `pow(base, exponent)` where `base` and `exponent` are `float` (return `double`)
      - Calculates the value of a number raised to a power.

Arduino-Language

- Functions
  - Math
    - `sqrt(x)` where `x` in a number of any data type (return `double`)
    - `constrain(x, a, b)` where `x` is the number to constrain (all data types), `a` is the lower end of the range (all data types) and `b` is the upper end of the range (all data types)
      - Constrains a number to be within a range.
  - Note: avoid using other functions inside the brackets, it may lead to incorrect results
    - `min(a++, 100); // avoid this, yields incorrect results`

Arduino-Language

- Functions
  - Trigonometry
    - `sin(x), cos(x), tan(x)` where `x` is the angle in radians `(float)` (return `double`)
  - Random Numbers
    - `randomSeed()`
    - `random([min],max)`
      -
```
long randNumber;
void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
}
void loop(){
  randNumber = random(300);
  Serial.println(randNumber);
  delay(50);
}
```

if analog input pin 0 is unconnected, random analog noise will cause the call to randomSeed() to generate different seed numbers each time the sketch runs. randomSeed() will then shuffle the random function.

- Functions
  - Bits and Bytes
    - `lowByte(x)` where `x` is a value of any type
      - Extracts the low-order byte of a variable
    - `highByte(x)` where `x` is a value of any type
      - Extracts the high-order byte of a word (or the second lowest byte of a larger data type)
    - `bitRead(x,n)` where `x` is the number from which to read and `n is` which bit to read, starting at 0 for the least-significant bit
      - Reads a bit of a number.
    - `bitWrite(x,n,b)` where `x` is the numeric variable to which to write , `n` is bit of the number to write starting at 0 for the least-significant bit and, `b is` the value to write to the bit (`0` or `1`)
    - `bitSet(x,n)` where `x` is the number whose to set and `n` is which bit set (`1`), starting at 0 for the least-significant bit
    - `bitClear(x,n)` where `x` is the number whose bit to clear and `n` is which bit clear (`0`), starting at 0 for the least-significant bit

- Functions
  - External Interrupts
    - Arduino 2009 board have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3).
    - `attachInterrupt(interrupt, function, mode)` where `interrupt` is the number of the interrupt (int), `function` (interrupt service routine) is the function to call when the interrupt occurs and `mode` defines when the interrupt should be triggered.
      - Specifies a function to call when an external interrupt occurs. Replaces any previous function that was attached to the interrupt.
      - NOTE: the function has no parameters and return nothing
      - Mode has 4 constants predefined as valid values:
        - LOW to trigger the interrupt whenever the pin is low,
        - CHANGE to trigger the interrupt whenever the pin changes value
        - RISING to trigger when the pin goes from low to high,
        - FALLING for when the pin goes from high to low.

Arduino-Language

- Functions
  - External Interrupts (cont.)
    - example

```
int pin = 13;
volatile int state = LOW;
void setup(){
  pinMode(pin, OUTPUT); //led
  attachInterrupt(0, blink, CHANGE);
}
void loop(){
   digitalWrite(pin, state);
}

void blink() {
  state = !state;
}
```

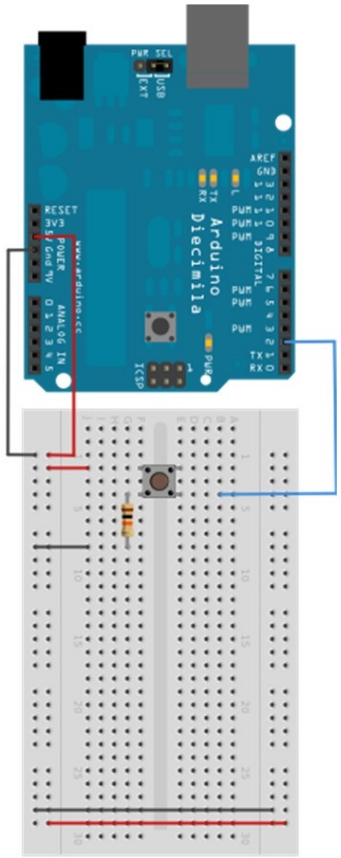
Note: variables modified within the attached function should be declared *volatile*

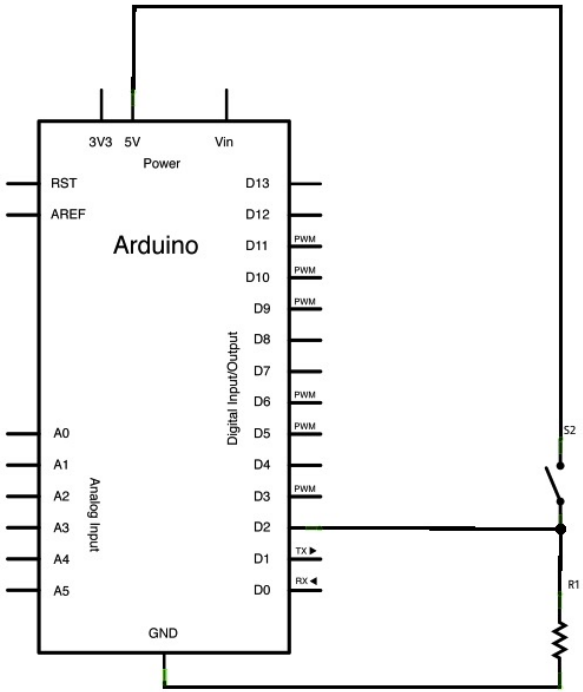
A Volatile variable is loaded from RAM and not from a storage register. A variable should be declared volatile whenever its value can be changed by something beyond the control of the code section in which it appears (e,g, concurrently executing thread. In the Arduino, the only place that this is likely to occur is in an interrupt service routine.

- Functions
  - External Interrupts (cont.)
    - `detachInterrupt(interrupt)` where `interrupt` is the number of interrupt to disable (0 or 1).
      - Turns off the given interrupt.
      -
```
switch (state) {
    case RISING: //we have just seen a rising edge
      detachInterrupt(i);
      attachInterrupt(i, fall, FALLING);
                              //attach the falling edge
      break;
    case FALLING: //we just saw a falling edge
      detachInterrupt(i);
      i++;  //move to the next pin
      i = i % 2;
      attachInterrupt(i, rise,RISING);
      break;
   }
```

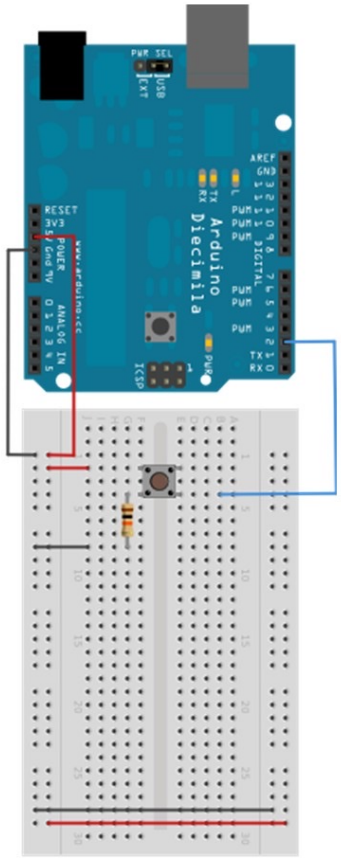
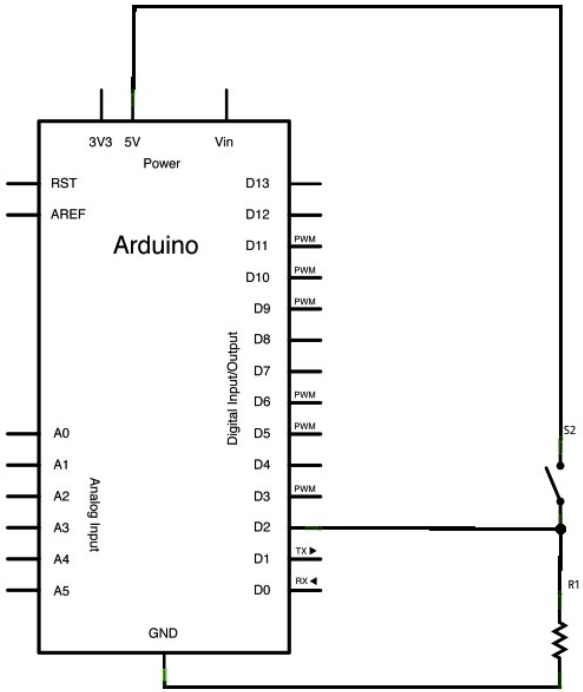
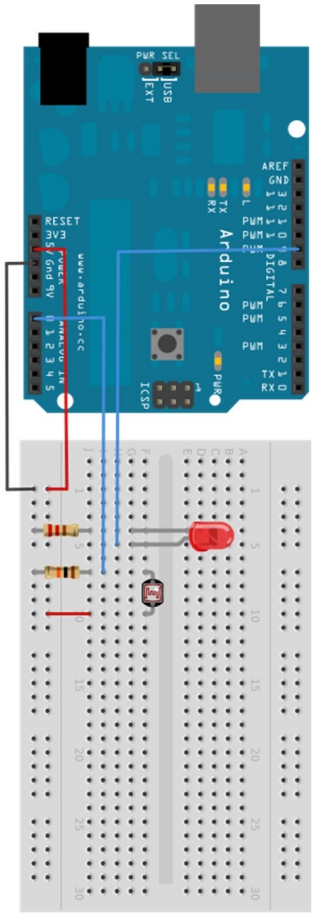
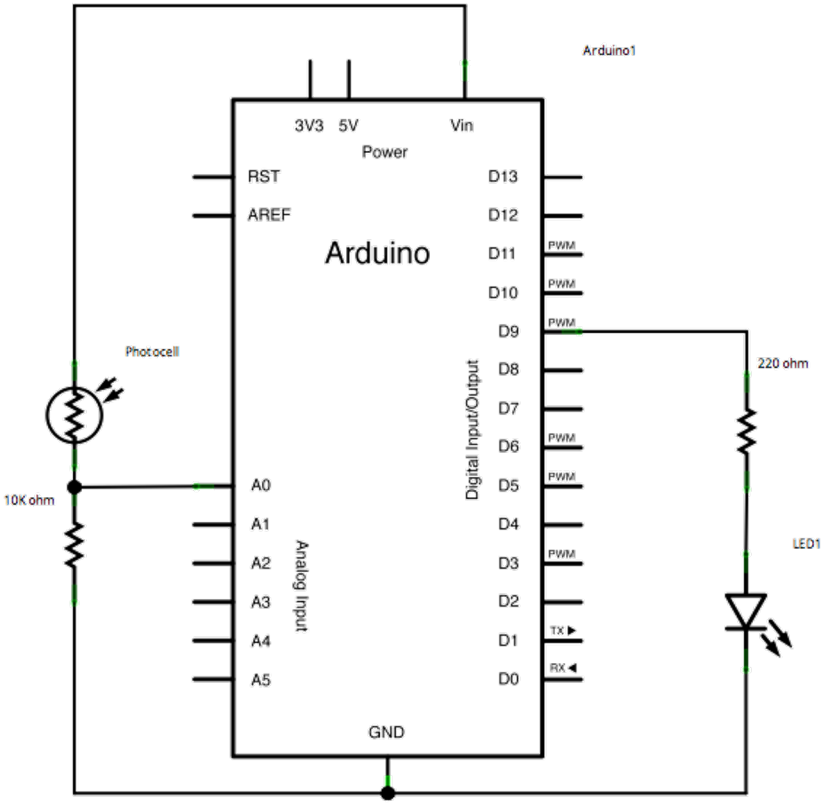
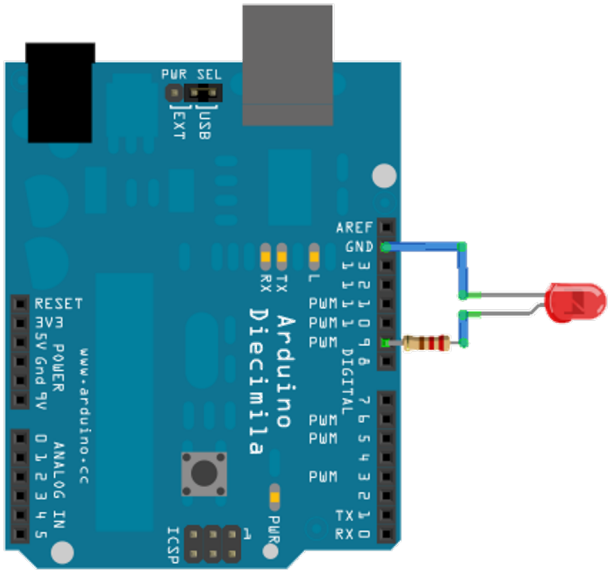
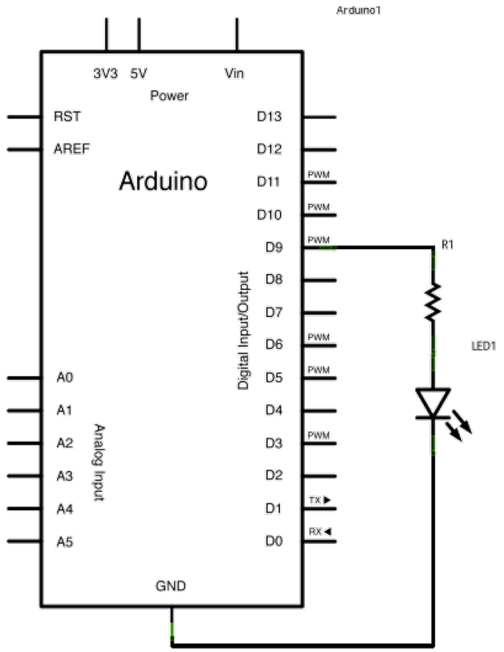
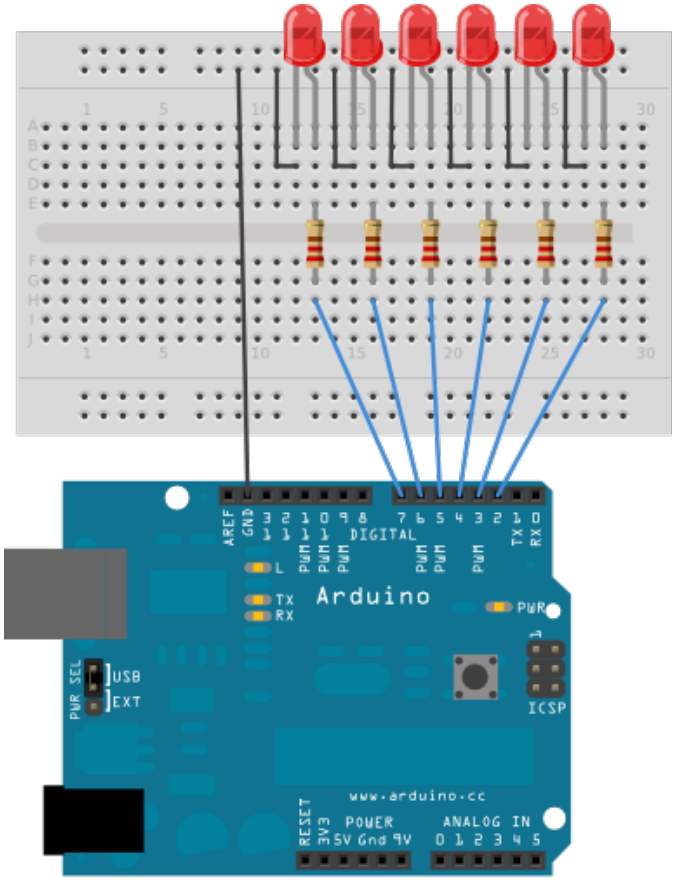
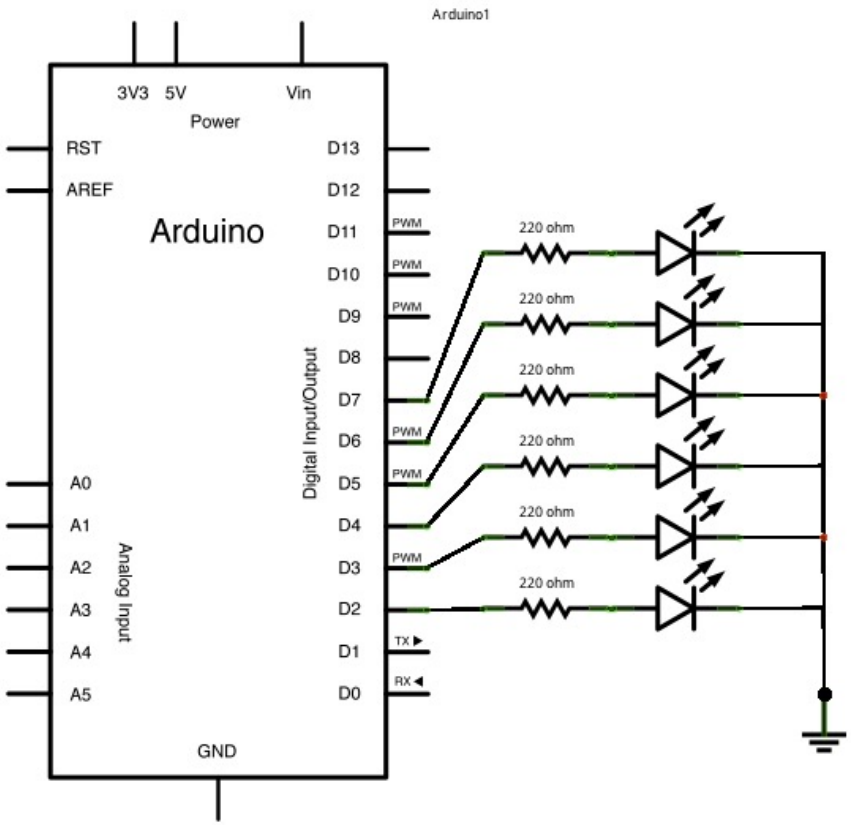
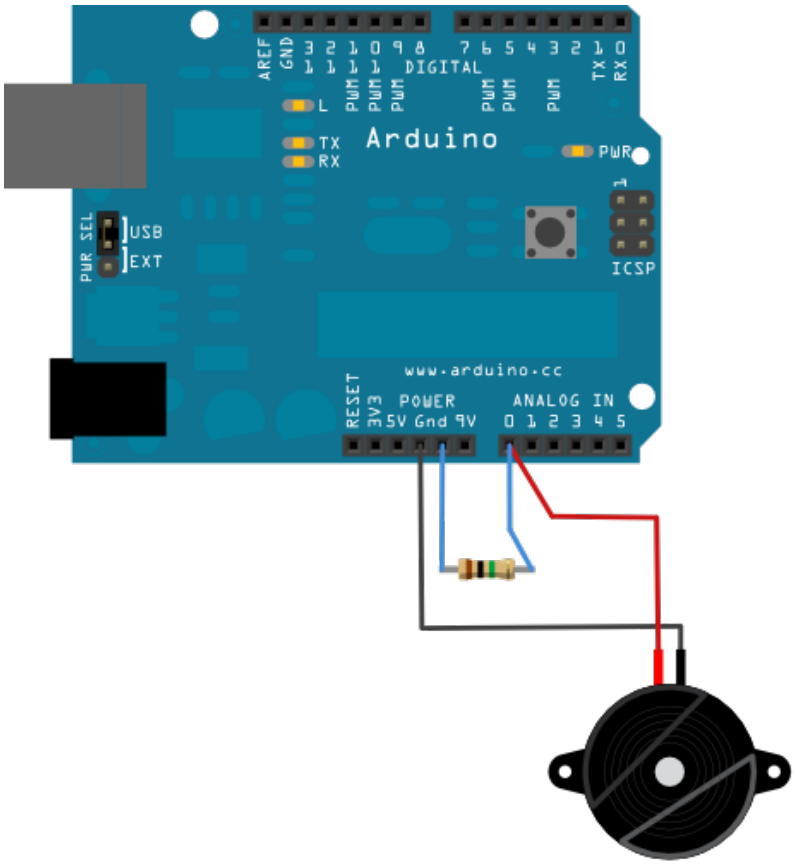
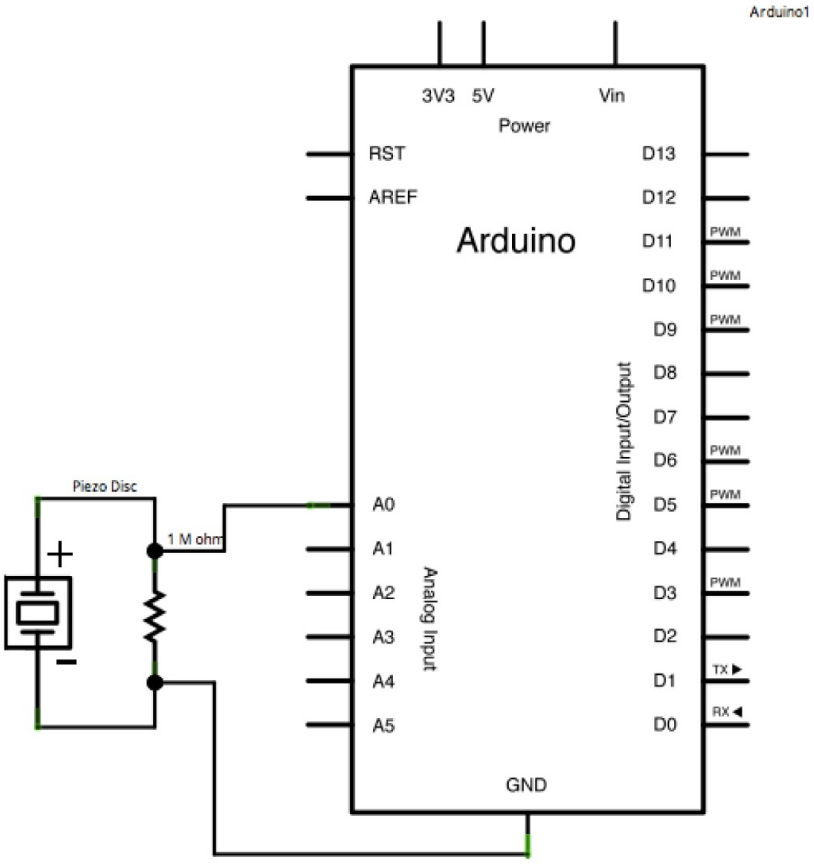
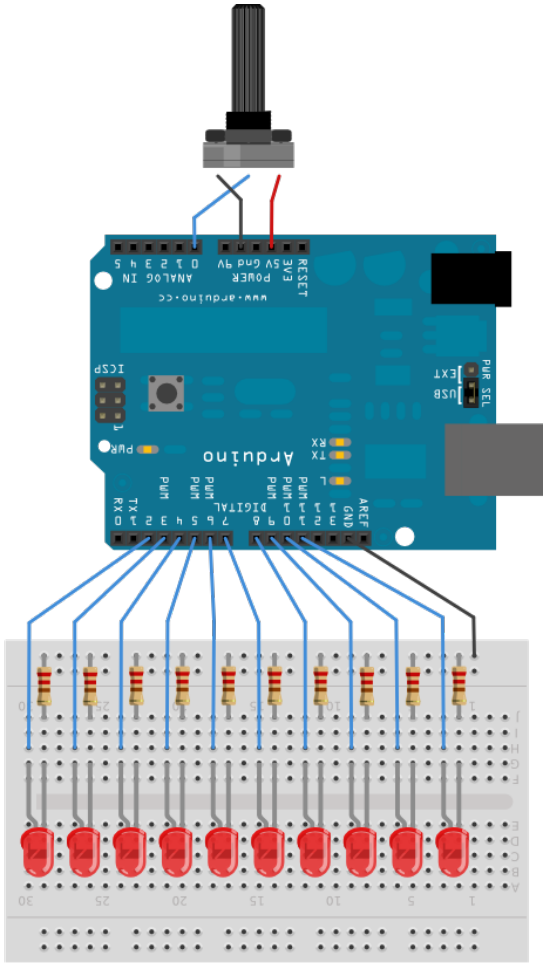
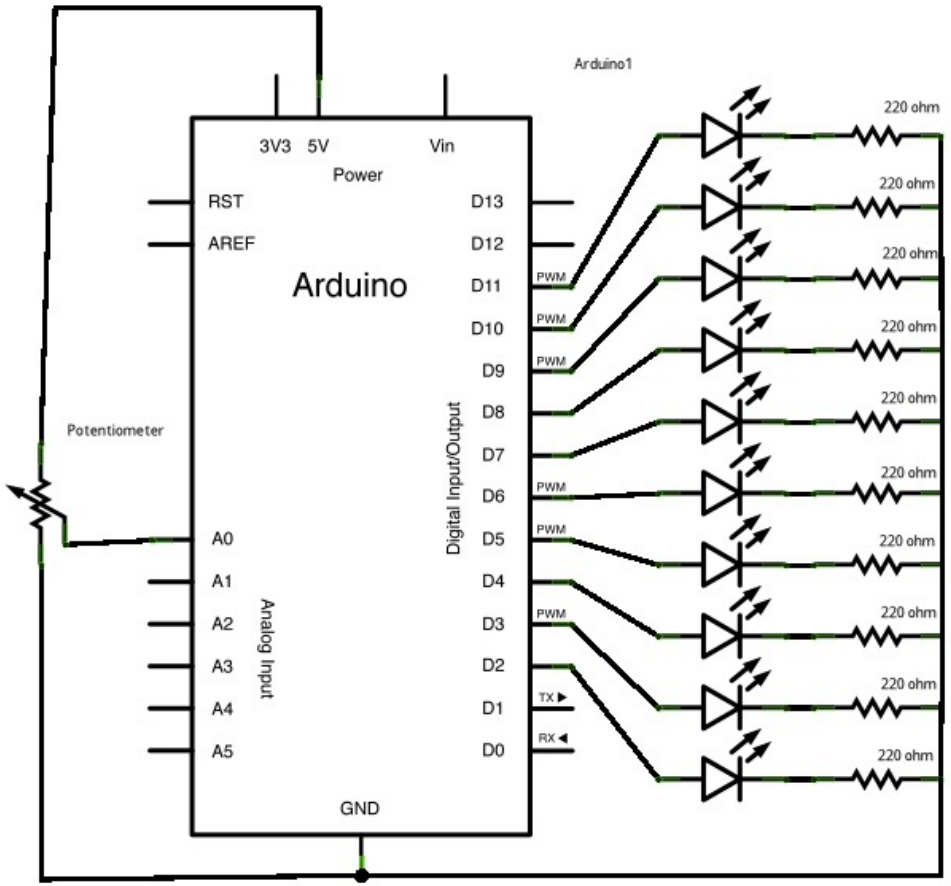
Examples

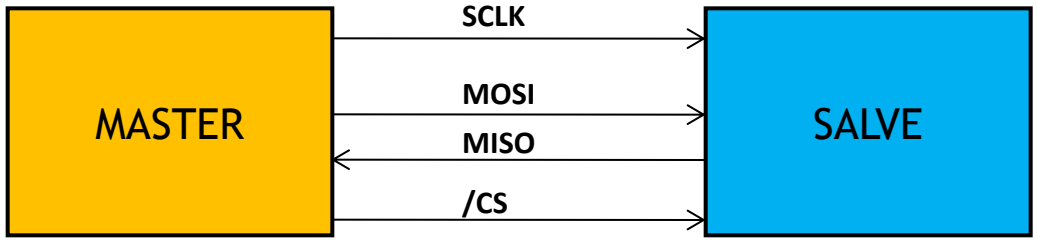
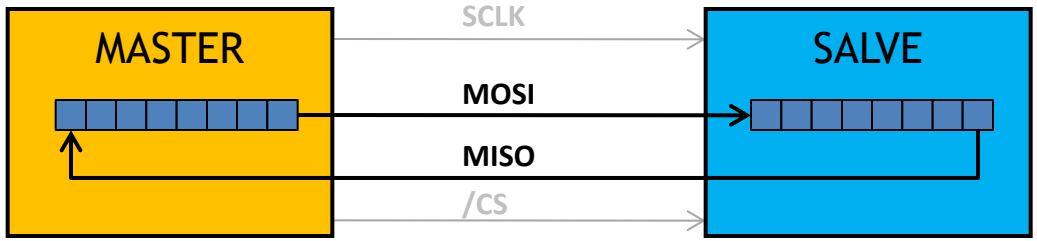
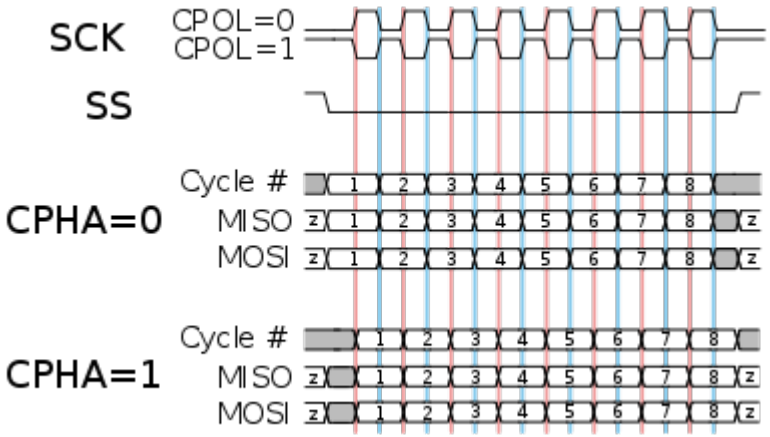
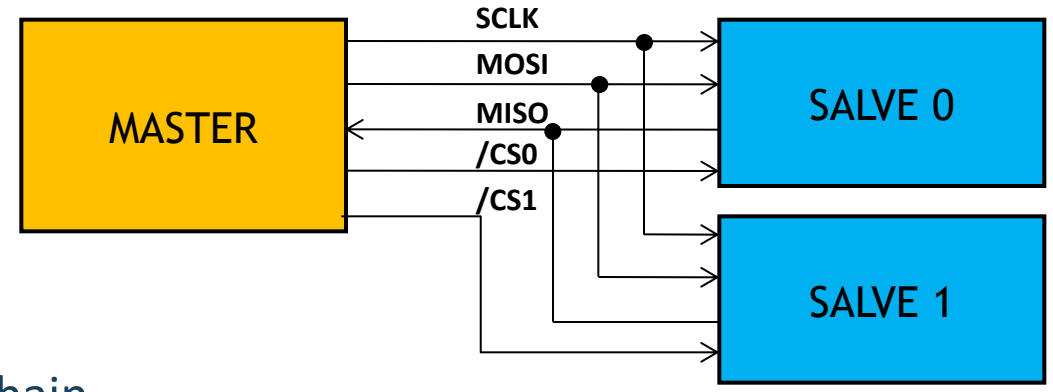
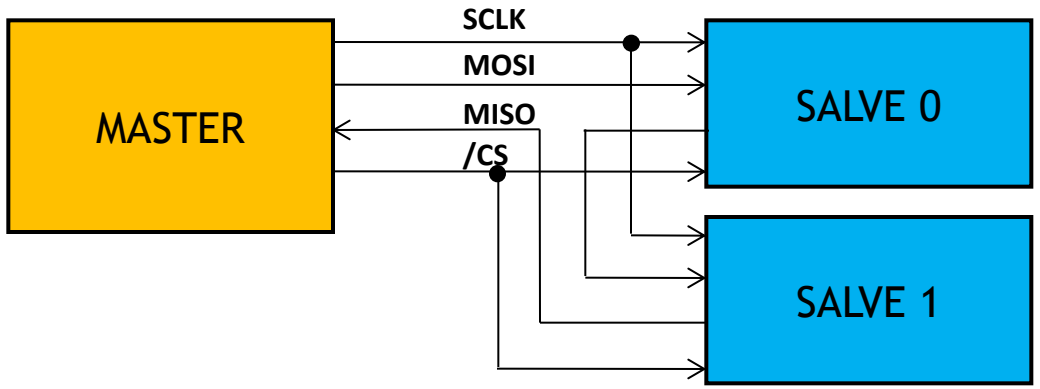
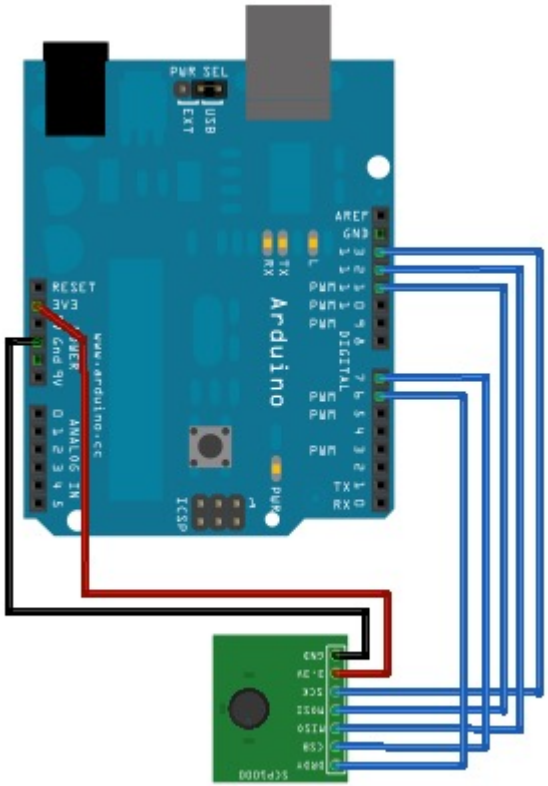
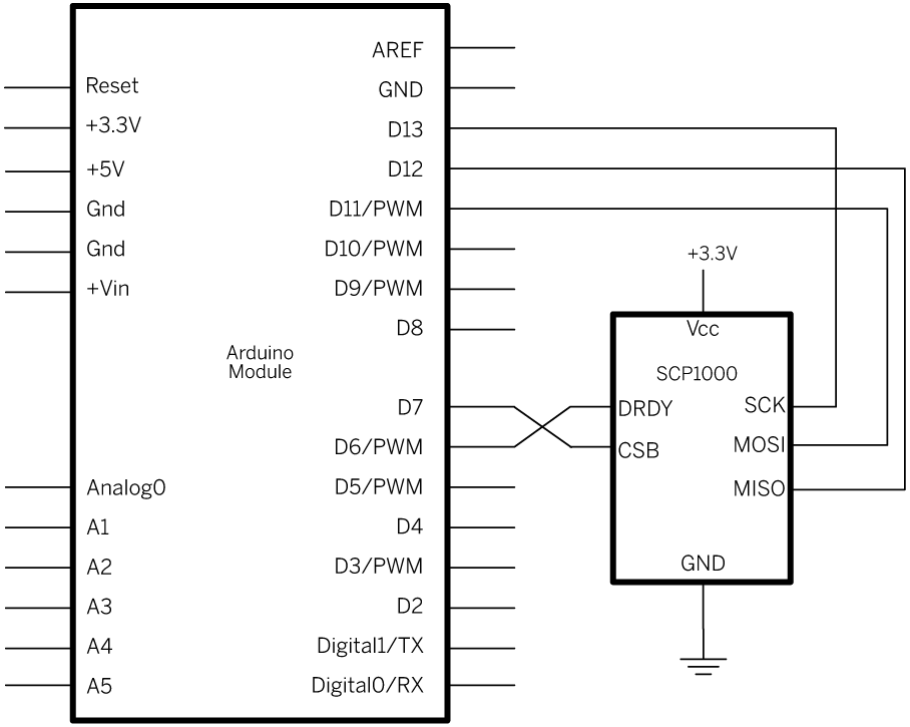
- Read a switch, print the state out to the Arduino Serial Monitor.





Examples

- cont.
  - ```
    // digital pin 2 has a pushbutton attached to it.
    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);
      delay(1);        // delay in between reads for stability
    }
    ```
    ```

Examples

- Counting the number of button pushes turns on the LED on the board every 4 button pushes





Examples

- cont.
  - Variable and constant section

```
// Constant (will not change):
const int buttonPin = 2;//the pin that the pushbutton is attached to
const int ledPin = 13;  // the pin that the LED is attached to
// Variables (will change):
int buttonPushCounter = 0;//counter for the number of button presses
int buttonState = 0; // current state of the button
int lastButtonState = 0; // previous state of the button
```

  - Set Up (system initialization)

```
void setup() {
  // initialize the button pin as a input:
  pinMode(buttonPin, INPUT);
  // initialize the LED as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communication:
  Serial.begin(9600);
}
```

Examples

- cont.
  - Loop

```
void loop() {
 // read the pushbutton input pin:
 buttonState = digitalRead(buttonPin);
 // compare the buttonState to its previous state
 if (buttonState != lastButtonState) {
 // if the state has changed, increment the counter
   if (buttonState == HIGH) {
     // if the current state is HIGH then the button went from off to on:
     buttonPushCounter++;
     Serial.println("on");
     Serial.print("number of button pushes:  ");
     Serial.println(buttonPushCounter);
   }
   else {
       // if the current state is LOW then the button wend from on to off:
     Serial.println("off");
   }
 }
 // save the current state as the last state,  for next time through the loop
 lastButtonState = buttonState;
```

Examples

- cont.
  - Loop (cont.)

```
void loop() {
  . . .
   // turns on the LED every four button pushes by checking the modulo of the button
     push counter. the modulo function gives you the remainder of the division of two
     numbers:
  if (buttonPushCounter % 4 = = 0) {
        digitalWrite(ledPin, HIGH);
   } else {
        digitalWrite(ledPin, LOW);
   }
}
```

Examples

- Define a maximum and minimum for expected analog sensor values (sensor calibration)





Examples

- cont.
  - Variable and constant section

```
const int sensorPin = A0;  // pin that the sensor is attached to
const int ledPin = 9;      // pin that the LED is attached to
// variables:
int sensorValue = 0;       // the sensor value
int sensorMin = 1023;      // minimum sensor value
int sensorMax = 0;         // maximum sensor value
```

  - Set Up (system initialization)

```
void setup() {
// turn on LED to signal the start of the calibration period:
pinMode(13, OUTPUT); digitalWrite(13, HIGH);
// calibrate during the first 5 seconds
while (millis() < 5000) {
   sensorValue = analogRead(sensorPin);
   // record the maximum sensor value
   if (sensorValue > sensorMax) { sensorMax = sensorValue; }
   // record the minimum sensor value
   if (sensorValue < sensorMin) { sensorMin = sensorValue; }
 }
// signal the end of the calibration period
  digitalWrite(13, LOW);
}
```

Examples

- cont.
  - Loop

```
void loop() {
  // read the sensor:
  sensorValue = analogRead(sensorPin);
  // apply the calibration to the sensor reading
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 255);
  // in case the sensor value is outside the range seen during calibration
  sensorValue = constrain(sensorValue, 0, 255);
  // fade the LED using the calibrated value:
  analogWrite(ledPin, sensorValue);
}
```

Reminder

`analogWrite(pin, value)` (PWM control)

where `pin` is 3, 5, 6, **9**, 10 and 11 and `value` is the duty cycle between 0 (always off) and 255 (always on).

Examples

- Move the mouse to change the brightness of a LED.
  - Bytes can be sent to the Arduino from any software that can access the computer serial port.
    - Shows how to send data from the computer to the Arduino board. The data is sent in individual bytes, each of which ranges from 0 to 255.





Examples

- cont.
  - Variable and constant section

```
const int ledPin = 9;      // the pin that the LED is attached to
```

  - Set Up (system initialization)

```
void setup()
{
  // initialize the serial communication:
  Serial.begin(9600);
  // initialize the ledPin as an output:
  pinMode(ledPin, OUTPUT);
}
```

  - Loop

```
void loop() {
 byte brightness;
 // check if data has been sent from the computer:
 if (Serial.available()) {
   // read the most recent byte (which will be from 0 to 255):
   brightness = Serial.read();
   analogWrite(ledPin, brightness); // set the brightness of the LED:
 }
}
```

Examples

- cont. (example of processing port)

```
import processing.serial.*;
Serial port;
void setup() {
      size(256, 150);
      println("Available serial ports:");
      println(Serial.list());
// Uses the first port in this list (number 0).  Change this to select the port corresponding to the Arduino board.  The last parameter (e.g. 9600) is the speed of the communication.  It has to correspond to the value passed to Serial.begin() in Arduino sketch.
 port = new Serial(this, Serial.list()[0], 9600);
 // If  know the name of the port used by the Arduino board, we can specify it directly using the code: port = new Serial(this, "COM1", 9600);
 }
 void draw() {
    // draw a gradient from black to white
    for (int i = 0; i < 256; i++) {
        stroke(i);
        line(i, 0, i, 150);
     }
    // write the current X-position of the mouse to the serial port as a single byte
 port.write(mouseX);
 }
```

Examples

- Controlling multiple LEDs with a `for()` loop.





Examples

- cont.
  - Variable and constant section

```
// The higher the number, the slower the timing.
int timer = 100;
```

  - Set Up (system initialization)

```
void setup() {
  // use a for loop to initialize each pin as an output:
  for (int thisPin = 2; thisPin < 8; thisPin++)  {
      pinMode(thisPin, OUTPUT);
  }
  digitalWrite(13, LOW);
 }
```

Examples

- cont.
  - Loop

```
void loop() {
  // loop from the lowest pin to the highest:
  for (int thisPin = 2; thisPin < 8; thisPin++) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
    // turn the pin off:
    digitalWrite(thisPin, LOW);
  }
  // loop from the highest pin to the lowest:
  for (int thisPin = 7; thisPin >= 2; thisPin--) {
    // turn the pin on:
    digitalWrite(thisPin, HIGH);
    delay(timer);
    // turn the pin off:
    digitalWrite(thisPin, LOW);
  }
}
```

Examples

- Detect knocks with a piezo element.





Examples

- cont.
  - Variable and constant section

```
const int ledPin = 13;       // led connected to digital pin 13
const int knockSensor = A0; // the piezo is connected to analog pin 0
const int threshold = 100;   // threshold value to decide when the detected
                                sound is a knock or not
// variables
int sensorReading = 0;//store the value read from the sensor pin
int ledState = LOW; // store the last LED status, to toggle the light
```

  - Set Up (system initialization)

```
void setup() {
   pinMode(ledPin, OUTPUT); // declare the ledPin as as OUTPUT
   Serial.begin(9600);      // use the serial port
}
```

Examples

- cont.
  - Loop

```
void loop() {
  // read the sensor and store it in the variable sensorReading:
  sensorReading = analogRead(knockSensor);
  // if the sensor reading is greater than the threshold:
  if (sensorReading >= threshold) {
    // toggle the status of the ledPin:
    ledState = !ledState;
    // update the LED pin itself:
    digitalWrite(ledPin, ledState);
    // send the string "Knock!" back to the computer, followed by newline
    Serial.println("Knock!");
  }
  delay(100);  // delay to avoid overloading the serial port buffer
}
```

Examples

- Read the EEPROM and send its values to the computer.
  - The microcontroller has 1K bytes of EEPROM. The example illustrates how to store values read from analog input 0 into the EEPROM using theEEPROM.write() function.
  -
```
int addr = 0;
void setup() { }
void loop()
{
  // inputs range is from 0 to 1023 while a byte single is stored.
  int val = analogRead(0)>>2;
  // write the value to the appropriate byte of the EEPROM.
  EEPROM.write(addr, val);
  // advance to the next address.
  // there are 1K bytes in the EEPROM, so go back to 0 when we hit 1024.
  addr = addr + 1;
  if (addr == 1024)
    addr = 0;
  delay(100);
}
```

Examples

- Control a series of LEDs in a row to make an LED graph.





Examples

- cont.
  - Variable and constant section

```
const int analogPin = A0;    // the pin that the potentiometer is attached to
const int ledCount = 10;     // the number of LEDs in the bar graph
// an array of pin numbers to which LEDs are attached
int ledPins[] = {2,3,4,5,6,7,8,9,10,11 };
```

  - Set Up (system initialization)

```
void setup() {
  // loop over the pin array and set them all to output:
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    pinMode(ledPins[thisLed], OUTPUT);
  }
}
```

Examples

- cont.
  - Loop

```
void loop() {
 // read the potentiometer:
 int sensorReading = analogRead(analogPin);
 // map the result to a range from 0 to the number of LEDs:
 int ledLevel = map(sensorReading, 0, 1023, 0, ledCount);
 // loop over the LED array:
 for (int thisLed = 0; thisLed < ledCount; thisLed++) {
   // if the array element's index is less than ledLevel,
  // turn the pin for this element on:
   if (thisLed < ledLevel) {
     digitalWrite(ledPins[thisLed], HIGH);
   }
   // turn off all pins higher than the ledLevel:
   else {
     digitalWrite(ledPins[thisLed], LOW);
   }
 }
}
```

- Power and Energy consumption reduction
  - Library: Enerlib.h
  - Methods
    - `void PowerDown(void)`
      - Puts Arduino in PowerDown mode.
    - `void Idle(void)`
      - Puts Arduino in Idle mode.
    - `void SleepADC(void)`
      - Puts Arduino in SleepADC mode.
    - `void PowerSave(void)`
      - Puts Arduino in PowerSave mode.
    - `void Standby(void)`
      - Puts Arduino in Standby mode.
    - `bool WasSleeping(void)`
      - Check if Arduino was sleeping. Use it in any ISR which IRQ wakes up the microcontroller to see if the ISR was called by a wake up event.

Advanced programming

- Low Power
  - `#include <Enerlib.h>`
  - `Energy energy;`
  -
```
void INT0_ISR(void) //interrupt routine
{  /* The WasSleeping function will return true if Arduino was
      sleeping before the IRQ. Subsequent calls to WasSleeping will
      return false until Arduino reenters in a low power state. The
      WasSleeping function should only be called in the ISR. */
   if (energy.WasSleeping()) {
    /* Arduino was waked up by IRQ.If you shut down external
       peripherals before sleeping, you can reinitialize them here.
       Look on ATMega's datasheet for hardware limitations in the
       ISR when microcontroller just leave any low power state. */
   }
   else
   { /* The IRQ happened in awake state.
     This code is for the "normal" ISR. */
   }
 }
```

- Low Power
  - . . . (cont.)

```
void setup()
{
   attachInterrupt(0, INT0_ISR, LOW);
   /* Pin 2 will be the "wake button". Due to uC limitations, it needs to be a
      level interrupt.
      For experienced programmers: ATMega's datasheet contains information
      about the rest of wake up sources. The Extended Standby is not
      implemented. */
   energy.Idle();
 }

void loop()
{
 . . .
}
```

- SPI
  - SPI or Serial Peripheral Interface is a synchronous, full duplex, Master/Slave serial data bus
  - Multiple slave devices are allowed with individual slave select (chip select) lines



  - SCLK (CLK): clock
  - MOSI (SDI, DI): Master Output/Slave Input
  - MISO (SDO, DO): Master Input/Slave Output
  - CS (SS, nCS): Chip Select (Active Low)

SPI

- SPI (cont.)
  - The SPI bus is a single master-more slave devices; CS low logic level or high→low transition selects the slave device.
    - Example: Maxim MAX1242 ADC starts the conversion on the falling edge of the signal.
  - Most slave devices have tri-state outputs: MISO signal becomes high impedance ("disconnected") when the device is not selected.
    - Devices without tri-state outputs can't share SPI bus segments with other devices.

- SPI: Data Transmission
  - The master configures the clock.
    - Typical frequency are from 1MHz to 70 MHz.
  - The master selects the slave.
    - If a waiting period is required (such as for analog-to-digital conversion) then the master must wait for at least that period of time before starting to issue clock cycles.
  - The master starts the full-duplex communication.
    - In each SPI clock cycle:
      1. the master sends a bit on the MOSI line
      2. the slave sends a bit on the MISO line

- SPI: Data Transmission (cont.)
  - Transmissions normally involve two shift registers (one in the master and one in the slave):
    - they are connected in a ring.
    - data is usually shifted out with the most significant bit first
    - transmissions may involve any number of clock cycles.
    - transmissions often consist of 8-bit words.
      - 16-bit words for touchscreen controllers or audio codecs
      - 12-bit words for many digital-to-analog or analog-to-digital converters.
  - **NOTE**: The master must select only one slave at a time.



- SPI: Clock Polarity Phase
  - Master must to configure the clock polarity (CPOL) and phase (CPHA) with respect to the data:
    - CPOL=0 (1) the base value of the clock is zero (one)
    - CPHA=0 and CPOL=0 or CPHA=1 and CPOL=1, data is captured on the clock's rising edge and data is propagated on a falling edge.
    - CPHA=1 and CPOL=0 or CPHA=0 and CPOL=1, data is captured on the clock's falling edge and data is propagated on a rising edge.
  - NOTE: data are always sample in the half of the cycle.



- SPI: Configuration
  - Parallel



  - Daisy-chain



- SPI: Pros and Cons
  - Pros
    - Full duplex communications
    - Higher throughput then I^2C
    - Simple Hardware interface
      - No arbitration or associated failure modes
      - Slaves use the master's clock and don't need precision oscillators
  - Cons
    - Requires more pins than I^2C
    - No in-band addressing and no hardware flow control
      - master can delay the next clock edge to slow the transfer rate
    - No hardware slave acknowledgment
    - Supports only one master device
    - Generally prone to noise spikes causing faulty communication
    - Only handles short distances compared

- SPI Arduino methods
  - `SPI.begin()`
  - `SPI.end()`
  - `SPI.setBitOrder(order)` where order is `LSBFIRST` or `MSBFIRST`
    - Sets the order of the bits shifted out of and into the SPI bus
  - `SPI.setClockDivider(divider)` where `divider` is $\texttt{SPI\_CLOCK\_DIV}_n$ and n is 2, 4, 8, 16, 32, 64, or 128
    - Sets the SPI clock divider relative to the system clock
  - `SPI.setDataMode(mode)` where `mode` is `SPI_MODE0`, `SPI_MODE1`, `SPI_MODE2`, or `SPI_MODE3`
    - Sets the SPI data mode: that is, clock polarity and phase.
  - `SPI.transfer(val)` where `val` is the byte to send out over the bus

Example SPI

- read data from a SCP1000 Barometric Pressure sensor using SPI





Example SPI

- Cont.

```
// the sensor communicates using SPI, so include the library:
#include <SPI.h>
//Sensor's memory register addresses:
const int PRESSURE = 0x1F;       //3 most significant bits of pressure
const int PRESSURE_LSB = 0x20;   //16 least significant bits of pressure
const int TEMPERATURE = 0x21;    //16 bit temperature reading
const byte READ = 0b11111100;      // SCP1000's read command
const byte WRITE = 0b00000010;    // SCP1000's write command
// pins used for the connection with the sensor
// the other you need are controlled by the SPI library):
const int dataReadyPin = 6;
const int chipSelectPin = 7;

 void setup() {
 . . .
```

Example SPI

- Cont.
  - ```
    void setup() {
      Serial.begin(9600);
      // start the SPI library:
      SPI.begin();
      // initalize the  data ready and chip select pins:
      pinMode(dataReadyPin, INPUT);
      pinMode(chipSelectPin, OUTPUT);
      //Configure SCP1000 for low noise configuration:
      writeRegister(0x02, 0x2D);
      writeRegister(0x01, 0x03);
      writeRegister(0x03, 0x02);
      // give the sensor time to set up:
      delay(100);
    }

    void loop() {
    . . .
    ```
    ```

Example SPI

- Cont.
  - ```
    void loop() {
      //Select High Resolution Mode
      writeRegister(0x03, 0x0A);
      // don't do anything until the data ready pin is high:
      if (digitalRead(dataReadyPin) == HIGH) {
        //Read the temperature data
        int tempData = readRegister(0x21, 2);
        // convert the temperature to celsius and display it:
        float realTemp = (float)tempData / 20.0;
        Serial.print("Temp[C]=");
        Serial.print(realTemp);
        //Read the pressure data highest 3 bits:
        byte  pressure_data_high = readRegister(0x1F, 1);
        pressure_data_high &= 0b00000111; //you only needs bits 2 to 0
        //Read the pressure data lower 16 bits:
        unsigned int pressure_data_low = readRegister(0x20, 2);
        //combine the two parts into one 19-bit number:
        long pressure = ((pressure_data_high << 16) | pressure_data_low)/4;
        // display the temperature:
        Serial.println("\tPressure [Pa]=" + String(pressure));
     }
    }
    . . .
    ```
    ```

Example SPI

- Cont.
  - ```
    //Read from or write to register from the SCP1000:
    unsigned int readRegister(byte thisRegister, int bytesToRead ) {
       byte inByte = 0;           // incoming byte from the SPI
       unsigned int result = 0;   // result to return
       Serial.print(thisRegister, BIN);
       Serial.print("\t");
       // SCP1000 expects the register name in the upper 6 bits of the byte.
      // So shift the bits left by two bits:
      thisRegister = thisRegister << 2;
      // now combine the address and the command into one byte
      byte dataToSend = thisRegister & READ;
      Serial.println(thisRegister, BIN);
      // take the chip select low to select the device:
      digitalWrite(chipSelectPin, LOW);
      // send the device the register you want to read:
      SPI.transfer(dataToSend);
      // send a value of 0 to read the first byte returned:
      result = SPI.transfer(0x00);
      // decrement the number of bytes left to read:
      bytesToRead--;
      . . .
    ```
    ```

Example SPI

- Cont.
  - . . .

```
// if you still have another byte to read:
if (bytesToRead > 0) {
  // shift the first byte left, then get the second byte:
  result = result << 8;
  inByte = SPI.transfer(0x00);
  // combine the byte you just got with the previous one:
  result = result | inByte;
  // decrement the number of bytes left to read:
  bytesToRead--;
 }
 // take the chip select high to de-select:
 digitalWrite(chipSelectPin, HIGH);
 // return the result:
 return(result);
}
```

Example SPI

- Cont.
  - void writeRegister(byte thisRegister, byte thisValue) {

```
// SCP1000 expects the register address in the upper 6 bits
// of the byte. So shift the bits left by two bits:
thisRegister = thisRegister << 2;
// now combine the register address and the command into one byte:
byte dataToSend = thisRegister | WRITE;
// take the chip select low to select the device:
digitalWrite(chipSelectPin, LOW);
SPI.transfer(dataToSend); //Send register location
SPI.transfer(thisValue);  //Send value to record into register
// take the chip select high to de-select:
digitalWrite(chipSelectPin, HIGH);
}
```

Arduino IDE

recommended

ONLINE IDE

- Save your sketches in the cloud
- Includes all libreries
- Supports new Arduino Boards

«The Arduino Web Editor can run on a variety of Platforms. If you are using Windows, Mac or Linux follow a simple flow to install the Arduino Web Editor plugin, which permits you to upload sketches from the browser onto your boards.»

https://create.arduino.cc/editor

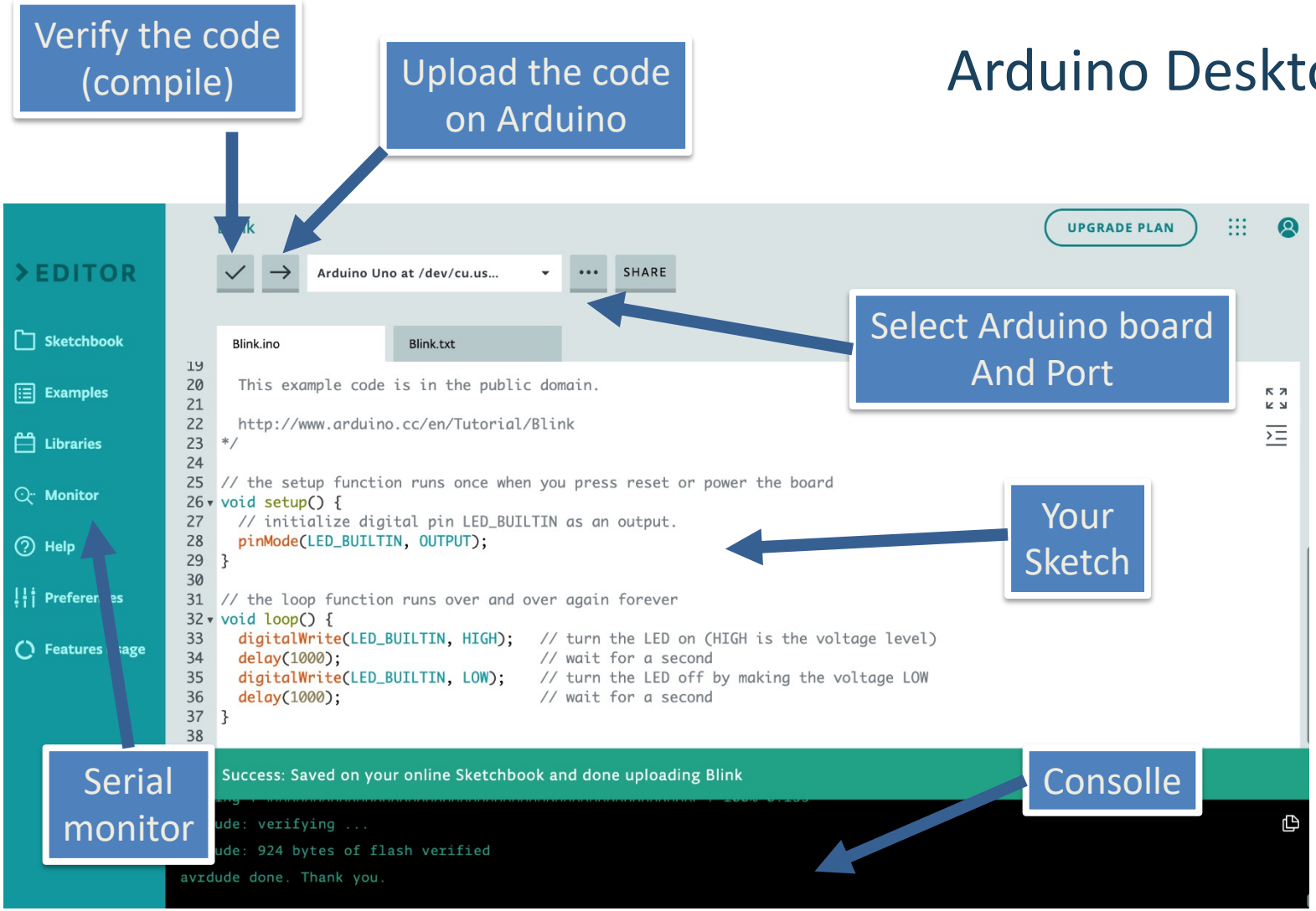
DESKTOP IDE

- Works offline

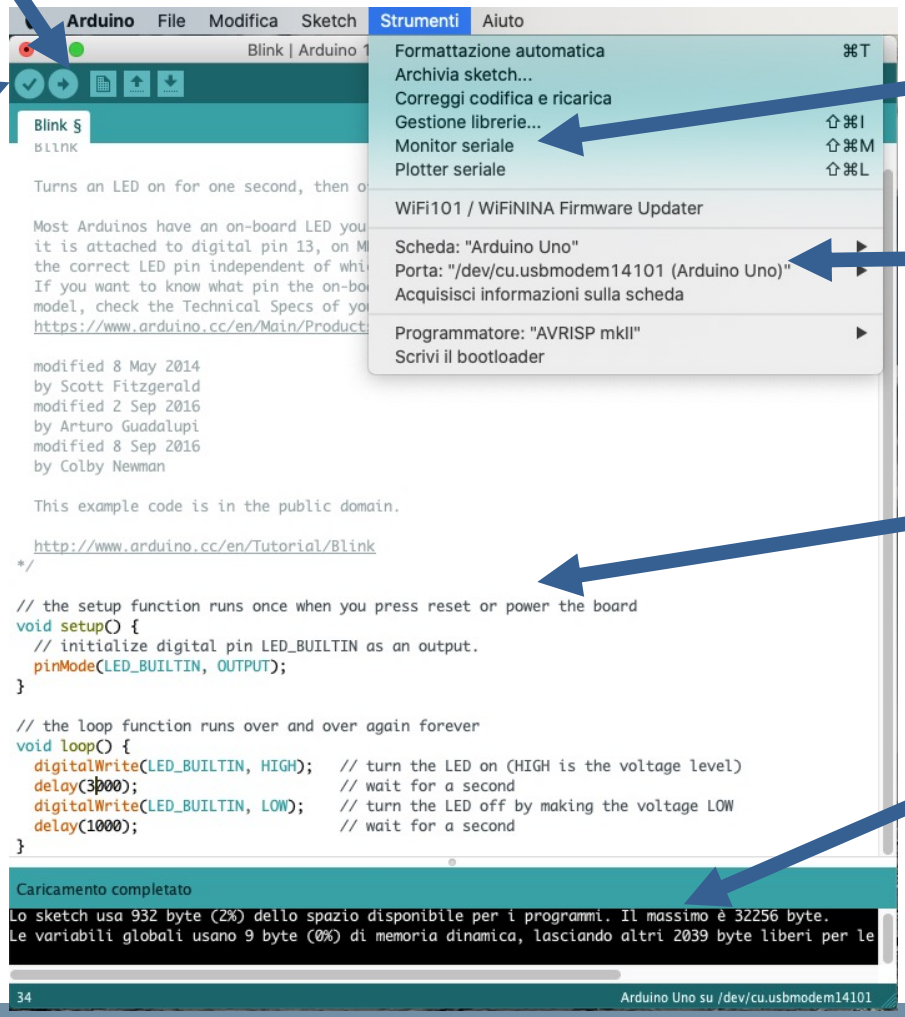
Refer to the official instructions to install the software on Windows, Mac, and Linux:

https://www.arduino.cc/en/software

Arduino Desktop IDE



Arduino Desktop IDE



Upload the code on Arduino

Verify the code (compile)

Serial monitor

Select Arduino board And Port

Your Sketch

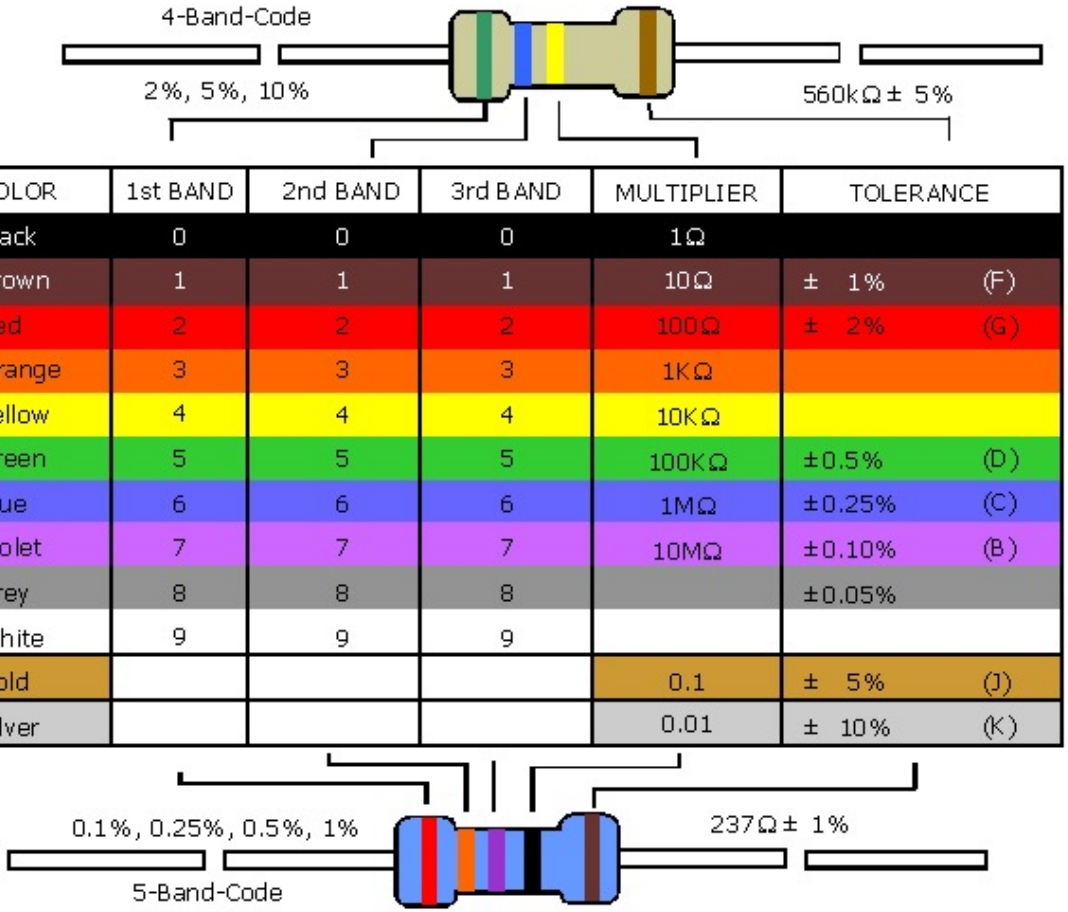
Consolle

Appendix

- For more Arduino cases, refer to: http://arduino.cc/playground/Projects/Ideas
- Arduino development environment http://arduino.cc/en/Guide/HomePage
- Fritznig http://fritzing.org/download/
- Examples (processing, web, …) https://studioarduino.wordpress.com/2013/02/13/arduino-server-arduino-client/

Appendix

- Resistors color code



COLOR	1st BAND	2nd BAND	3rd BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	±0.5% (D)
Blue	6	6	6	1MΩ	±0.25% (C)
Violet	7	7	7	10MΩ	±0.10% (B)
Grey	8	8	8		±0.05%
White	9	9	9		
Gold				0.1	± 5% (J)
Silver				0.01	± 10% (K)