

## LECTURE IV

# Microcontroller Architecture and Arduino

© 2023 Open Project Space, Institute of Electrical and Electronics Engineers at the University of California, Irvine. All Rights Reserved.

## **SECTION I**

# **Binary Numbers and Digital Signals**

# Number Systems and Bases

- The **Decimal** number system contains the set of ten digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, making it a **Base-10** number system
- **There exist different bases** beyond the Decimal number system
  - For example, **Base-8** is the **Octal** number system with the set of only eight digits... {0, 1, 2, 3, 4, 5, 6, 7}
  - **Base-16**, or the **Hexadecimal** number system, has a whopping sixteen digits... {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

# Number Systems and Bases (Cont'd)

- We can convert a number of a different base to Base-10 with the following method
- Ex) Convert Hex number  $B32_{16}$  to Base-10

B is represented  
by the decimal  
number 11

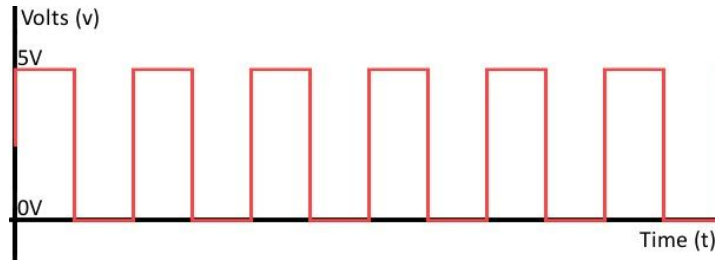
$$\begin{aligned} & (B \cdot 16^2) + (3 \cdot 16^1) + (2 \cdot 16^0) \\ &= (B \cdot 16^2) + (3 \cdot 16^1) + 2 \\ &= (B \cdot 16^2) + 48 + 2 \\ &= (11 \cdot 16^2) + 48 + 2 \\ &= 2816 + 48 + 2 = 2866 \end{aligned}$$

The base of a  
hexadecimal number  
is 16

Each hexadecimal  
digit represents a  
power of 16

# Binary Signals

- **Computers transfer data** across wires/lines as **digital** (discrete) **voltage signals**
  - These signals are either a **HIGH** or **LOW** voltage
  - Contrasts from **analog signals** which can be values in a continuous range

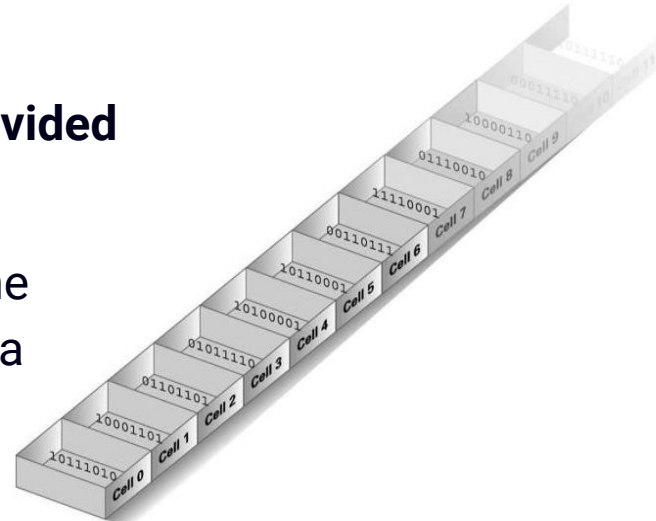


← This is a **digital signal** where the waveform is either **5V** or **0V** (two *discrete* values)

- **Digital signals are translated** to the **Binary** number system or **Base-2** (1s and 0s)

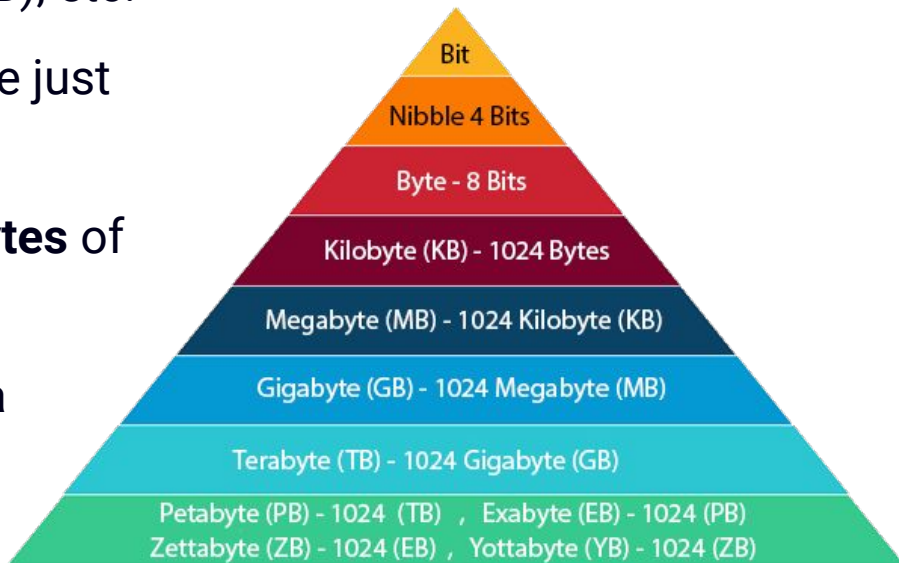
# Bits, Bytes, and Beyond

- **Single Binary digits** are called **bits**
  - Bits compose binary numbers like  $10010_2$  ( $18_{10}$ )
- **Eight bits** form a **byte**
  - For example...  $10100010_2$   $01011111_2$
- Data in a computer is stored in **memory** which is **divided** into cells each a **byte** in width
  - This means that on many systems, a **byte** is the **smallest unit of data** that can be accessed by a computer



# Bits, Bytes, and Beyond (Cont'd)

- Decimal units are used to express the size of binary data
  - i.e. Kilobytes (KB), Megabytes (MB), etc.
- **Small embedded computers** may store just **kilobytes** or **megabytes** of data
- **Laptops** can store **gigabytes** or **terabytes** of information
- **Google servers** store **exabytes** of data
  - That's billions of gigabytes!



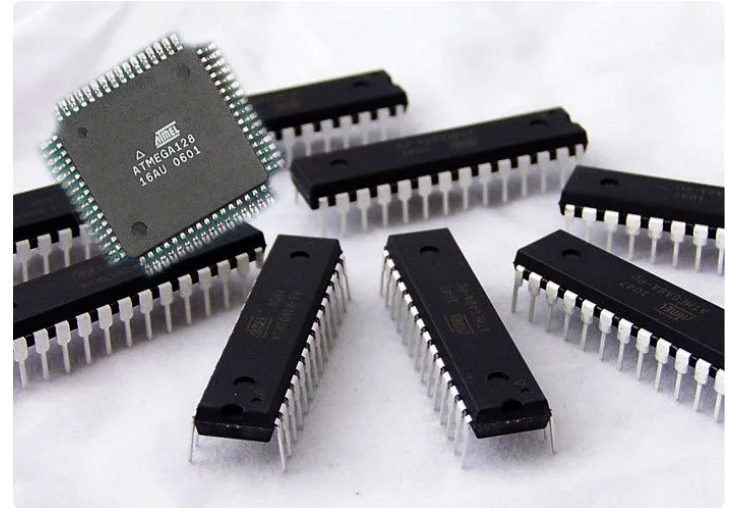
## SECTION II

# Microcontrollers



# Microcontrollers

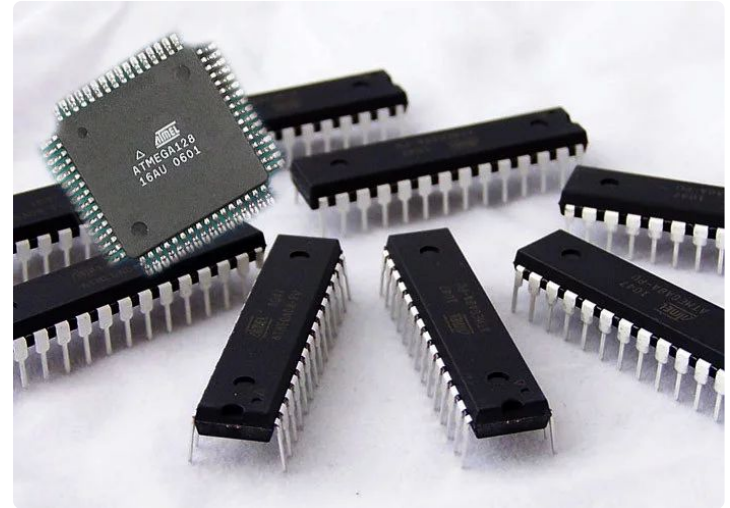
- A **microcontroller** (or **MCU**) is a **small computer** on a single integrated circuit
  - It is quite literally a computer, complete with everything needed to run and store programs
- An **architecture** defines the organization of hardware within the computer
  - **MCUs have different architectures**
    - Sometimes we say they “belong to different families”



**AVR Microcontroller Family**

# Microcontrollers (Cont'd)

- We will focus on a microcontroller which adheres to the **AVR architecture**
  - It runs programs made with the **AVR instruction set**
- To better understand how an AVR microcontroller works, we will examine its key components...



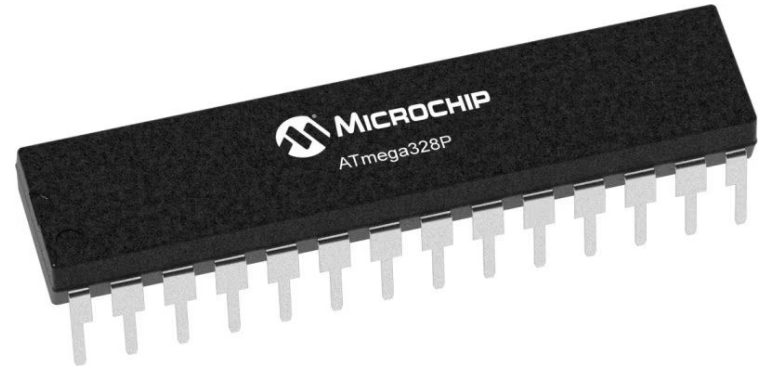
**AVR Microcontroller Family**

## **SECTION III**

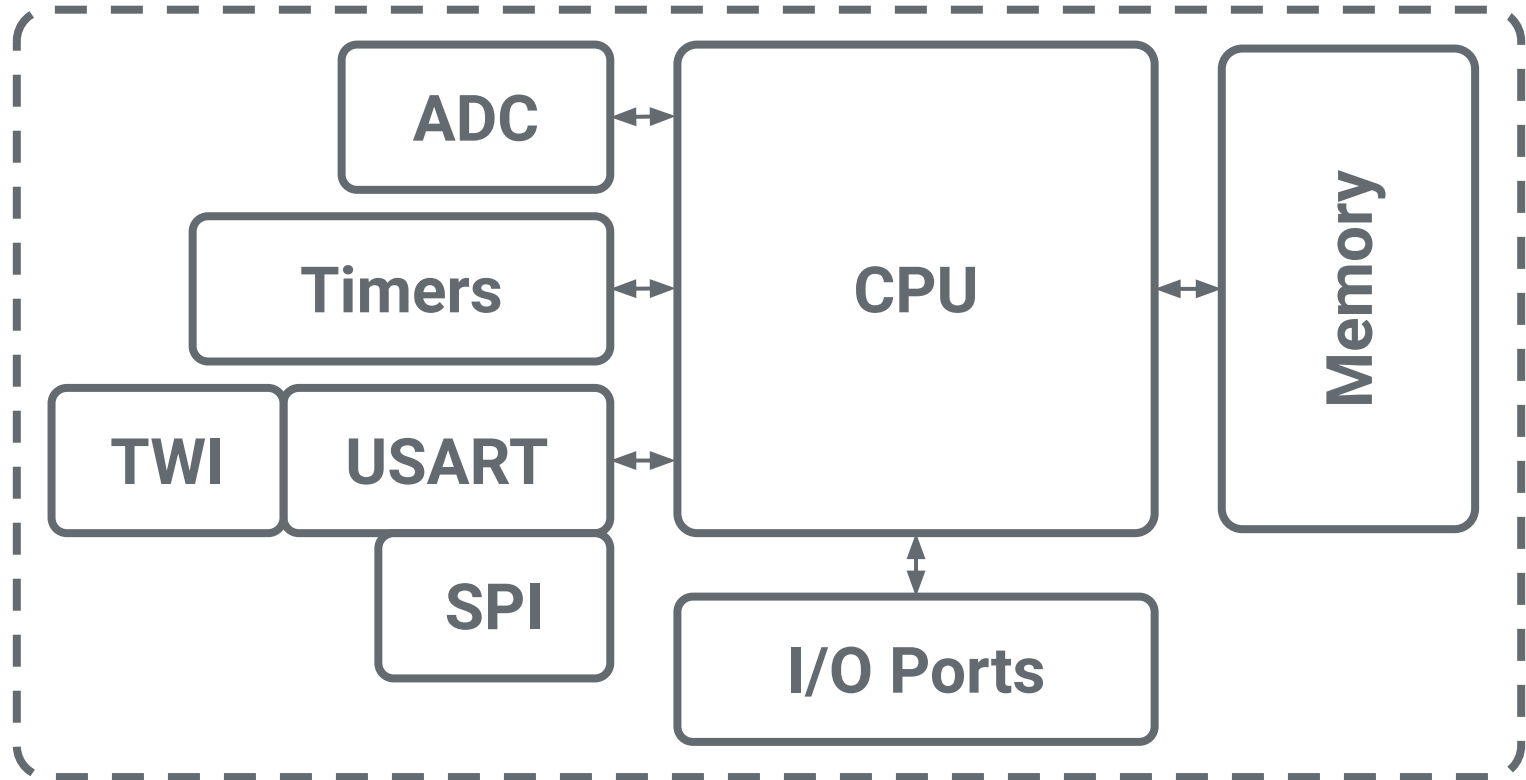
# **AVR Microcontroller Architecture**

# AVR Architecture

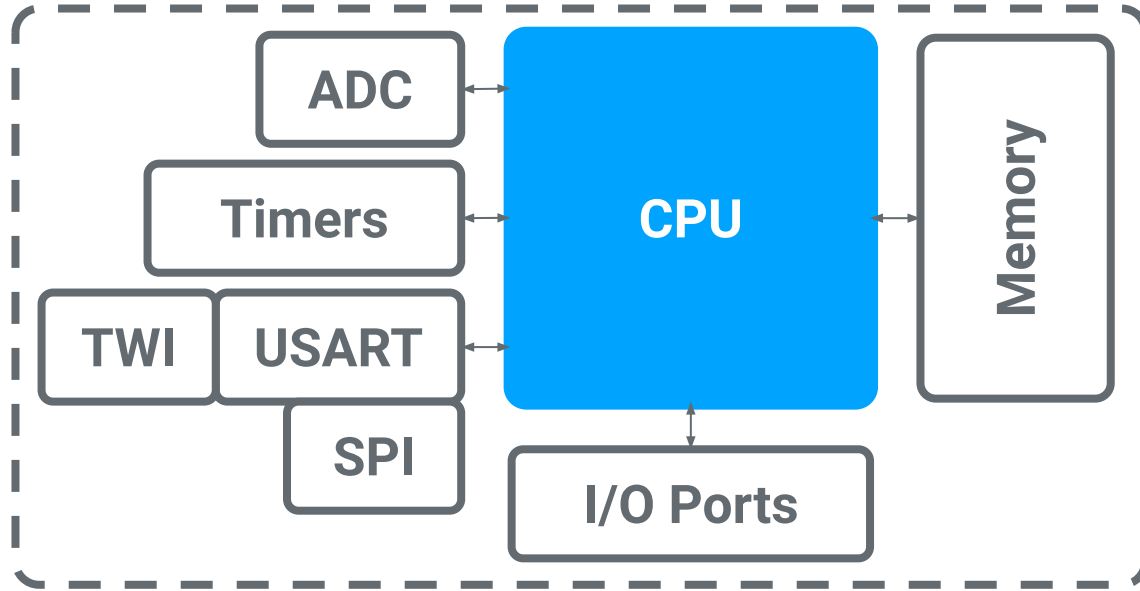
- Main units of an AVR microcontroller:
  - **Central Processing Unit (CPU)**
  - **Memory**
  - **Analog-to-Digital Converter**
  - **Timers**
  - **Input/Output Ports**
  - **Serial Communication Units**



# AVR Architecture (Cont'd)

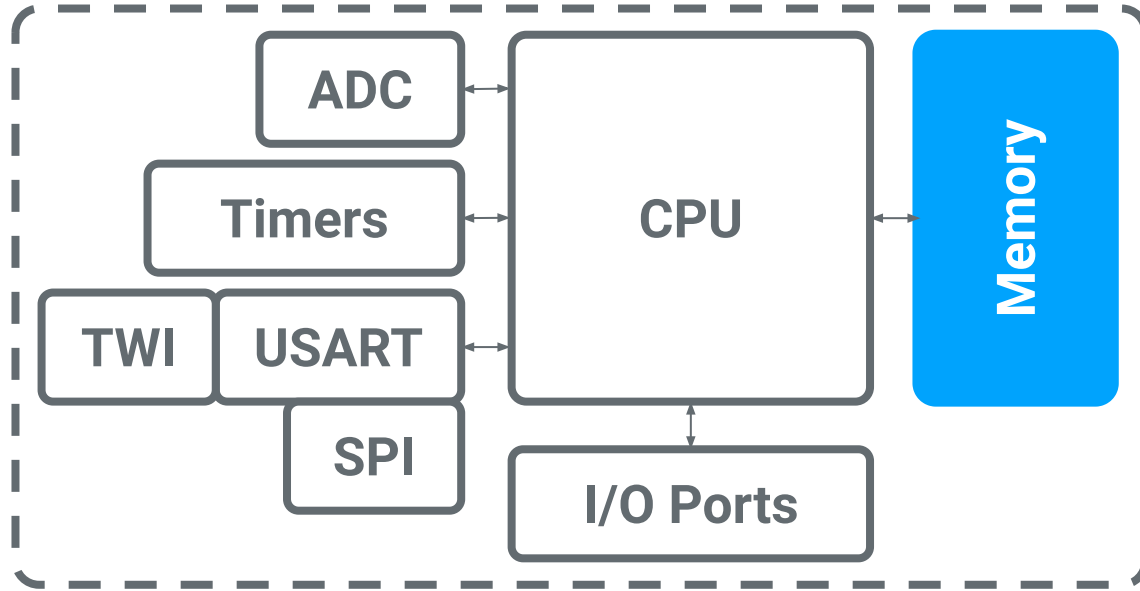


# Central Processing Unit (CPU)



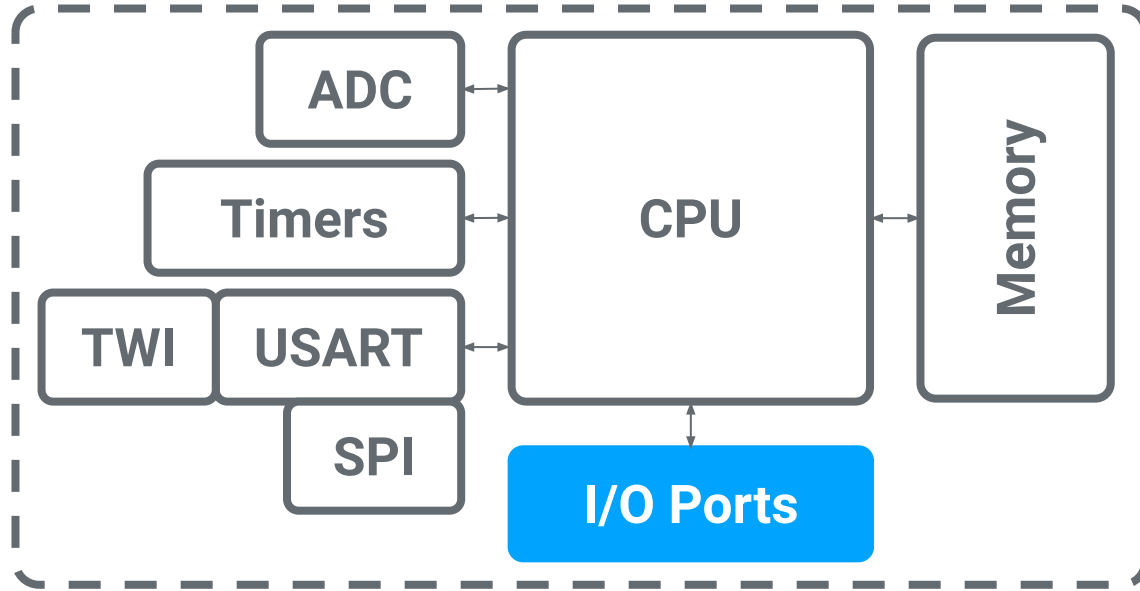
A **CPU** **executes instructions** provided by a memory unit. These instructions may be mathematical **computations**, memory **data transfer**, or signals to **input and output** ports.

# Memory



A **memory** unit stores data and instructions. There are subunits of memory, some of which are for short-term storage, erased when the microcontroller is disconnected from its power supply, while others are long-term.

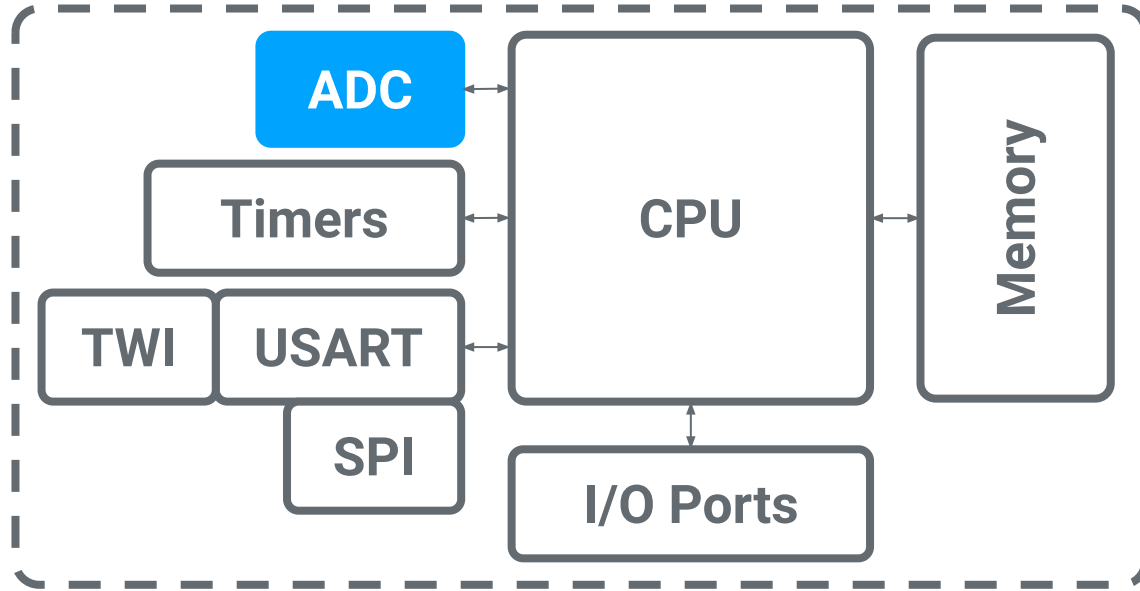
# Input/Output (I/O) Ports



**Input/Output Ports (I/O Ports)** connect the microcontroller to peripheral devices that transfer data to and from the microcontroller. This data may be stored in memory or interpreted by the CPU. The I/O Ports are wired to **physical leads on the IC** called **I/O pins**.

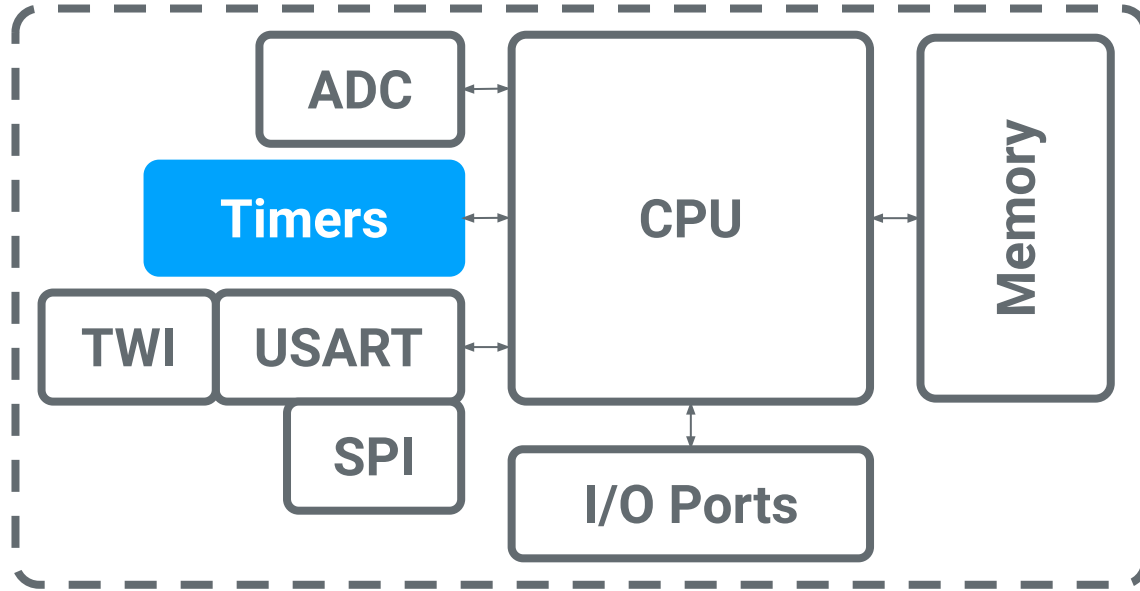


# Analog-To-Digital Converter (ADC)



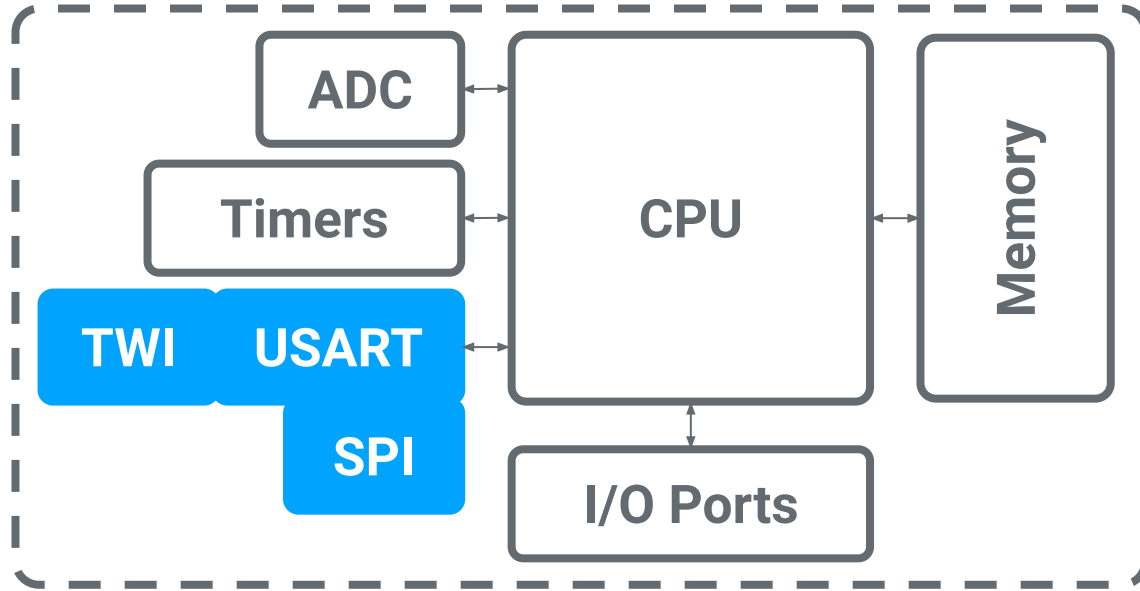
The **analog-to-digital converter (ADC)** measures real-world, analog signals from the I/O pins (i.e. temperature, pressure, acceleration) and **converts them into digital signals** that the microcontroller can interpret.

# Timers



**Timers** are used by the microcontroller to **control the timing** of program execution or output signals and **measure time**.

# Serial Communication Units



The **TWI**, **USART**, and **SPI** are all units that **generate and interpret serial communication signals** between the microcontroller and I/O peripherals. This facilitates data transfer across devices. We will expand on these terms in a future lecture on communication.

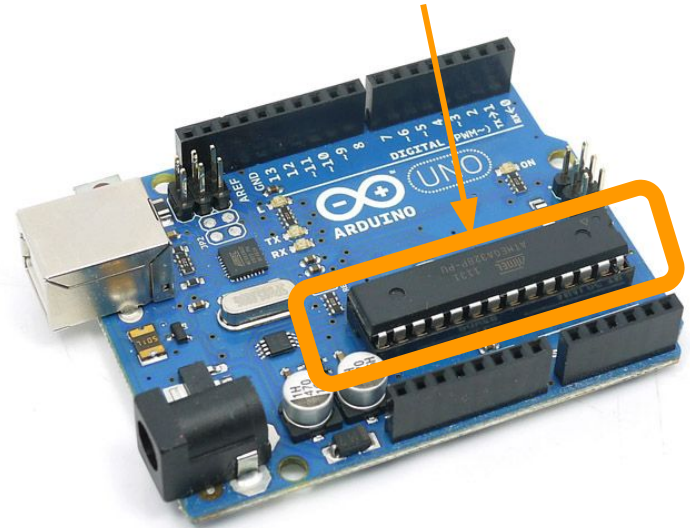
## **SECTION IV**

# **Arduino Boards**

# Arduino Boards

- The Arduino company designed **microcontroller boards** to make several AVR microcontrollers **hobbyist-friendly**
  - Added a USB port, input voltage regulator, onboard LEDs, among other things
  - This family of boards includes the **Arduino Uno, Arduino Nano**, and many, many more!

The MCU is **an IC that sits in a DIP socket** on the board

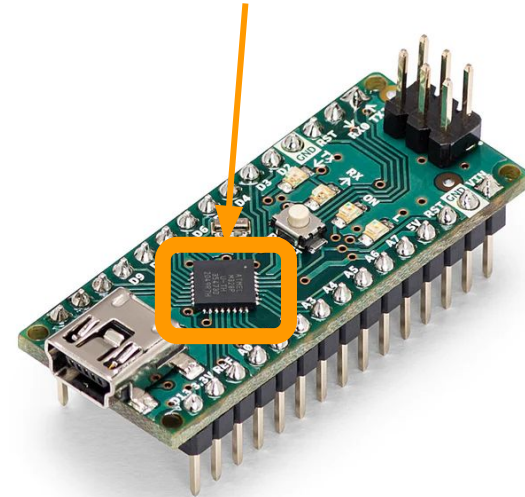


**Arduino Uno**

# Arduino Nano

- The current and future lectures use the **Arduino Nano** (based on the **ATmega328P** microcontroller)
  - The ATmega328P belongs to the AVR family
  - The **Arduino Uno** uses the same **microcontroller**
  - Almost all code written for the Arduino Nano is directly transferable to the Arduino Uno

The MCU is an **SOIC** soldered to the board



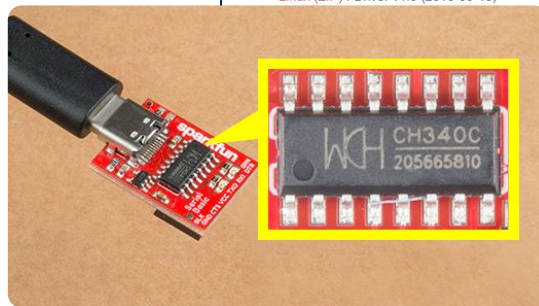
**Arduino Nano**

## **SECTION V**

# **Arduino IDE**

# CH340 Driver

- We will use the CH340 driver so that our personal computers understand how to communicate with the Arduino board
- Download the CH340 installer [here](#)



## Drivers (If You Need Them)

The CH340 has been tested on:

- Windows 7/10
- Mac OSX
  - v10.10.5 (Yosemite)
  - v10.11.6 (El Capitan)
  - v10.13.0 (High Sierra)
  - v10.14.5 (Mojave)
- Linux
  - Raspbian Stretch (11-13-2018 release) for the Raspberry Pi
  - Raspbian Buster (2019-07-10 release) for the Raspberry Pi
  - Ubuntu v18.04.2, 64-bit

These operating systems have the CDC drivers pre-installed, which means you shouldn't need to install any extra software. However, there are a wide range of operating systems out there, so if you run into driver problems, you can get the archived drivers linked below:

- **Windows (EXE)** : Driver executable
- **Windows (ZIP)** : Driver v3.4 (2016-09-27)
- **Mac (ZIP)** : Driver v1.5 (2018-07-04)
- **Linux (ZIP)** : Driver v1.5 (2018-03-18)

version of their drivers in their [English translated website](#).

[DRIVERS \(ENGLISH PAGE\)](#)

Find the latest version of their drivers from their [website](#) in had the option to have the web page translated. However, you in either language. For those interested in heading to the


[DRIVERS \(MANDIRIN PAGE\)](#)



# Arduino IDE

- We will write code, compile, and upload it to the Arduino board from our personal computers using the **Arduino IDE** software kit
- Download the IDE installer [here](#)





## Arduino IDE 2.0.1

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

**SOURCE CODE**

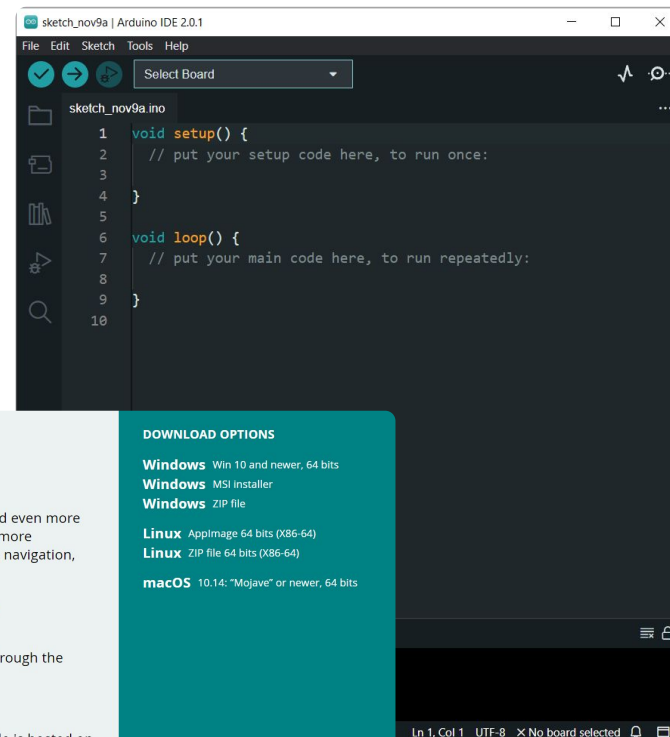
The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

### DOWNLOAD OPTIONS

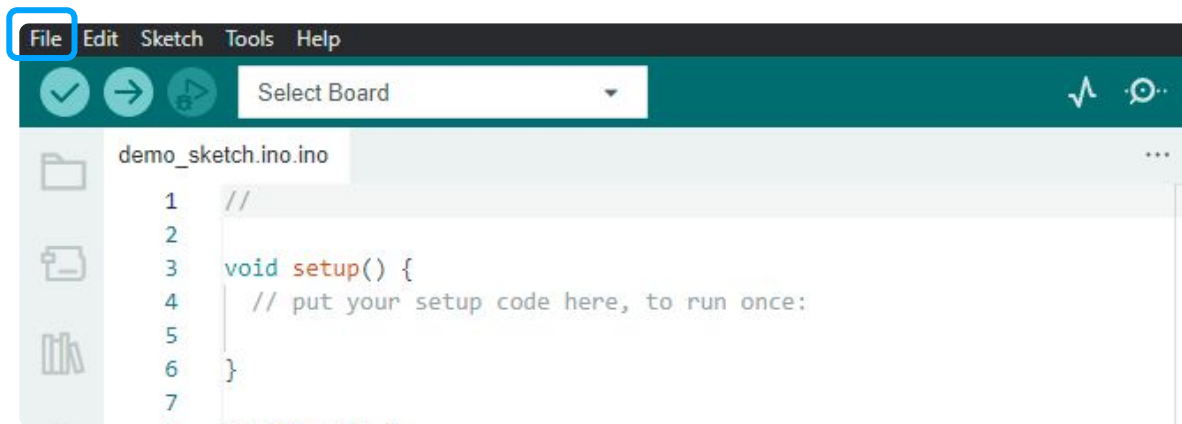
**Windows** Win 10 and newer, 64 bits  
**Windows** MSI installer  
**Windows** ZIP file

**Linux** AppImage 64 bits (X86-64)  
**Linux** ZIP file 64 bits (X86-64)

**macOS** 10.14: "Mojave" or newer, 64 bits

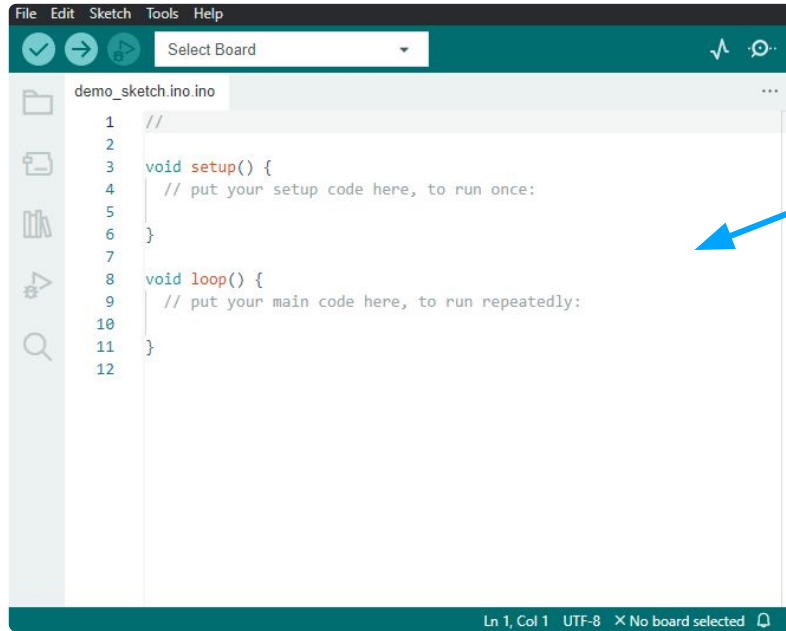


# Creating a Sketch



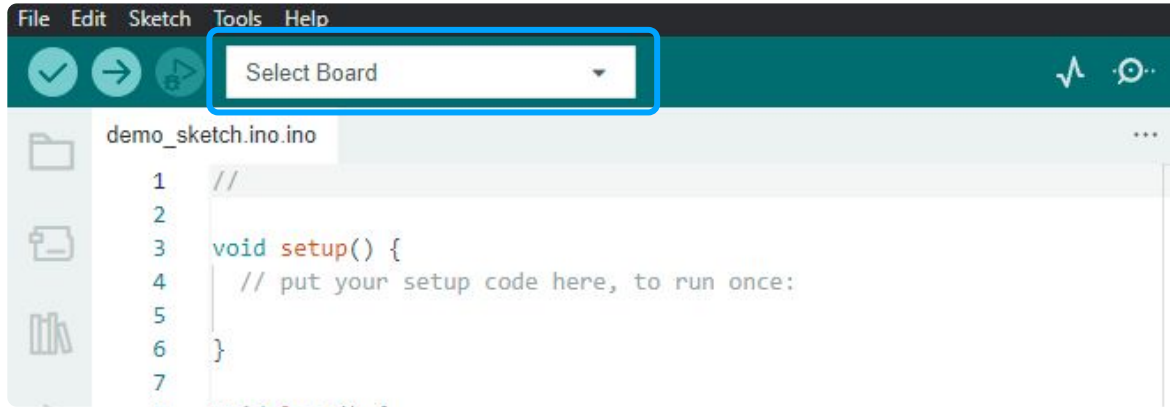
- An Arduino source code file is called a **sketch** and has the **.ino** extension
- When the IDE opens, a new sketch will be created automatically or a previous sketch will open
  - To create a new sketch, select **File** → **New Sketch** from the menu

# Editing a Sketch



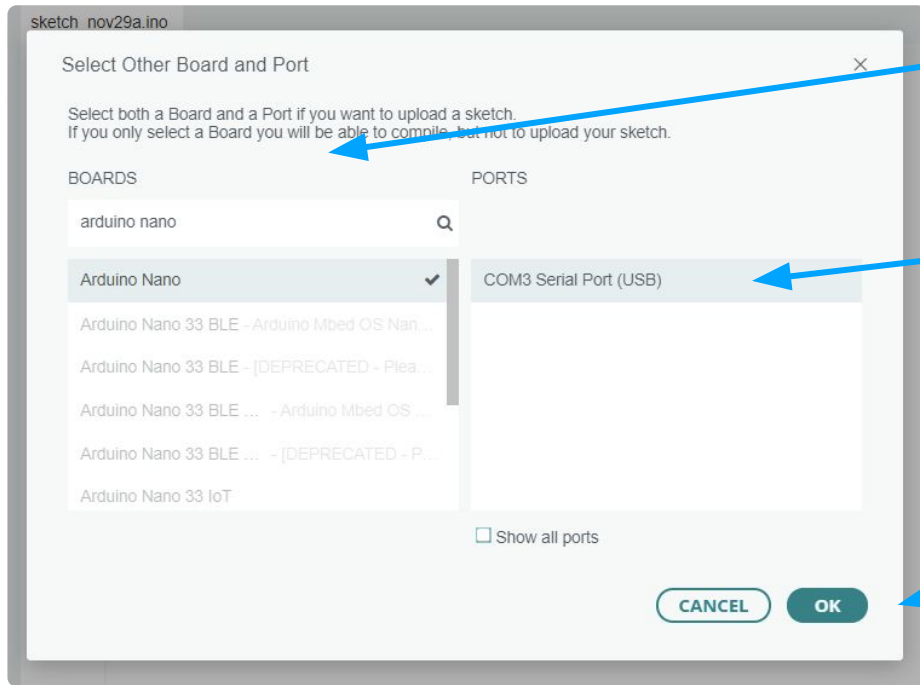
- Sketches can be modified from the **code editor**, which appears just below the menu
- New sketches opened in the editor come with **template code**
- The editor is also enhanced with text **autocompletion**

# Setting the Target Board



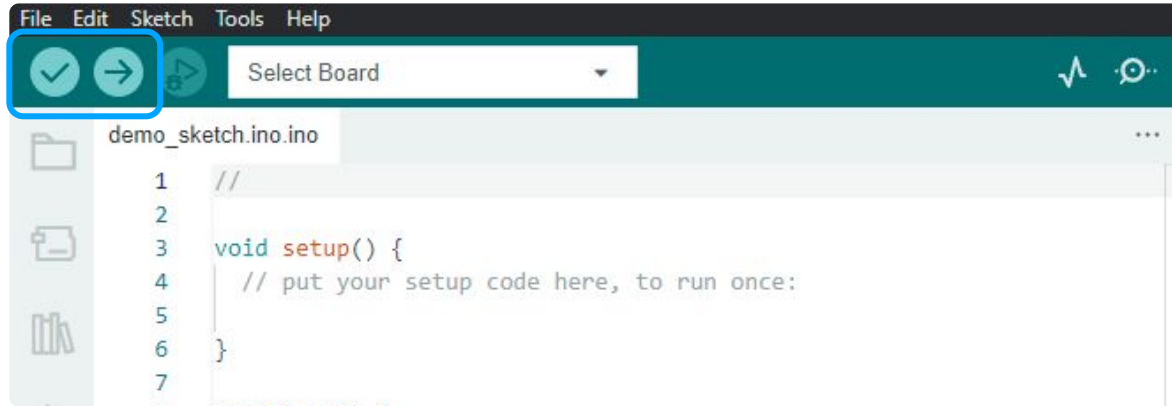
- Before we can upload a sketch, we must **connect the Arduino board via USB to the computer** and configure the IDE to the correct board and USB port
- Open the **Select Board** dropdown and select an available Arduino Nano board
  - If no option appears, click **Select other board and port...**

# Setting the Target Board (Cont'd)



- In the popup window, search **Boards** and select **Arduino Nano**
- From the available items under **Port**, select the USB port to which the board is currently connected
- Confirm the selections by clicking **OK**

# Verifying and Uploading a Sketch



- Before uploading the sketch, select **Verify** (the **checkmark** icon) to compile the sketch
- Once the sketch compiles successfully, select **Upload** (the **right arrow** icon) to upload the compiled sketch to the Arduino board

# Troubleshooting Ports

Did the upload time out? Are active ports missing from the dropdown?

**Here's what to do:**

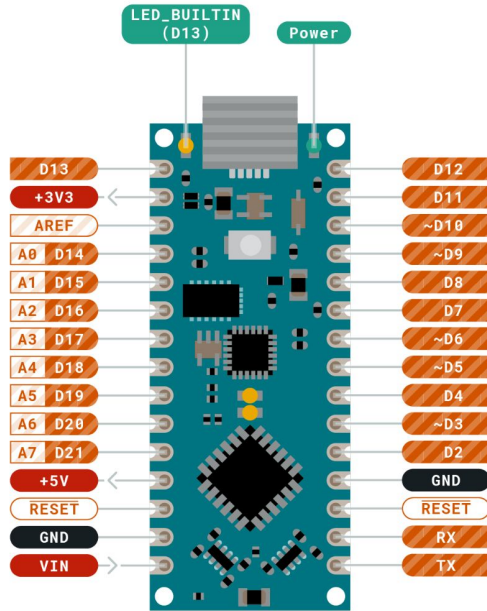
- Check your USB connection
  - Check your system's device manager to verify the Arduino is connected
  - Replace the USB cable or switch to a different USB port
- Verify your CH340 driver installation
- In the IDE, select **Tools** → **Processor** → **ATmega328P (Old Bootloader)**
- Restart the IDE and/or the personal computer

## **SECTION VI**

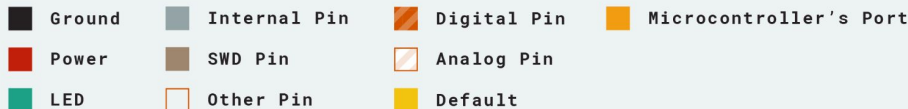
# **Basic Arduino Pins and Functions**



# Arduino Pinout Diagram



- A pinout diagram is included in the Arduino Nano datasheet
- **General Purpose Input/Output (GPIO)** pins are pins with no predefined purpose
  - These are the **analog** and **digital pins** marked by the pinout diagram
  - We will design an Arduino sketch to control these pins



ARDUINO.CC



This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

# Sketch Structure

- Sketches are written in the **Arduino language**, which is much like C++
- **Assign pins as global variables** at the beginning of the sketch:  
`const int pinName = pin#;`
  - Each GPIO pin is associated with an integer
- Just like the C++ **main** function, Arduino has built-in functions **setup** and **loop**, which must be defined by the programmer

```
const int RECEIVER = 14;
const int LED = 7;

void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(9600);
}

void loop()
{
    Serial.println(analogRead(RECEIVER));
    digitalWrite(LED, HIGH);
}
```

# Sketch Structure (Cont'd)

- **setup()** runs once at the beginning of program execution
  - Used to initialize pin modes and global variables
- **loop()** runs repeatedly in an infinite loop
  - This is where you implement the “main” code
- **Do not define the main function.** Arduino does this for you

```
const int RECEIVER = 14;  
const int LED = 7;
```

```
void setup()  
{  
    pinMode(LED, OUTPUT);  
    pinMode(RECEIVER, INPUT);  
    Serial.begin(9600);  
}
```

```
void loop()  
{  
    Serial.println(analogRead(RECEIVER));  
    digitalWrite(LED, HIGH);  
}
```

# Things to do in setup

- `void pinMode(int pin, int mode)`
  - Call this function for each digital pin you want to use
  - Pass the global variable assigned to each pin to the `pin` param
  - Specify if your pin is **INPUT** or **OUTPUT** mode
    - We **read data from input pins** and **send data to output pins**

```
void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(9600);
}
```

# Things to do in setup (Cont'd)

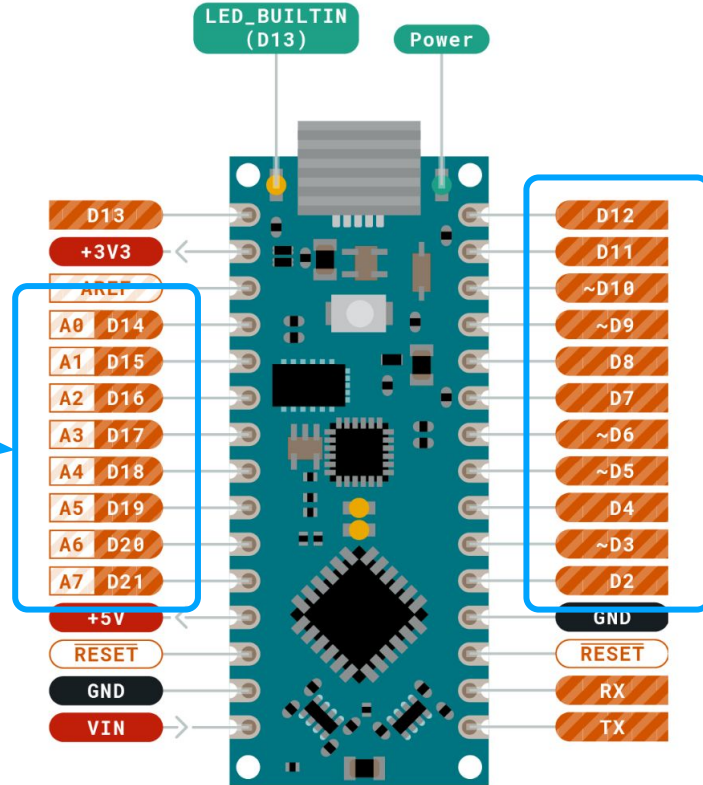
- `Serial.begin(int baud_rate)`
  - `Serial` is an object that facilitates communication between the Arduino board and the computer connected to it via USB
  - Pass 9600 bits per second as the argument for this member function
    - To be elaborated in future lectures

```
void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(9600);
}
```

# Digital Pins

## Digital Output Pins

These pins are numbered  
as 2, 3, 4... 20, 21



## Digital Input/Output Pins

# Digital Pin Functions

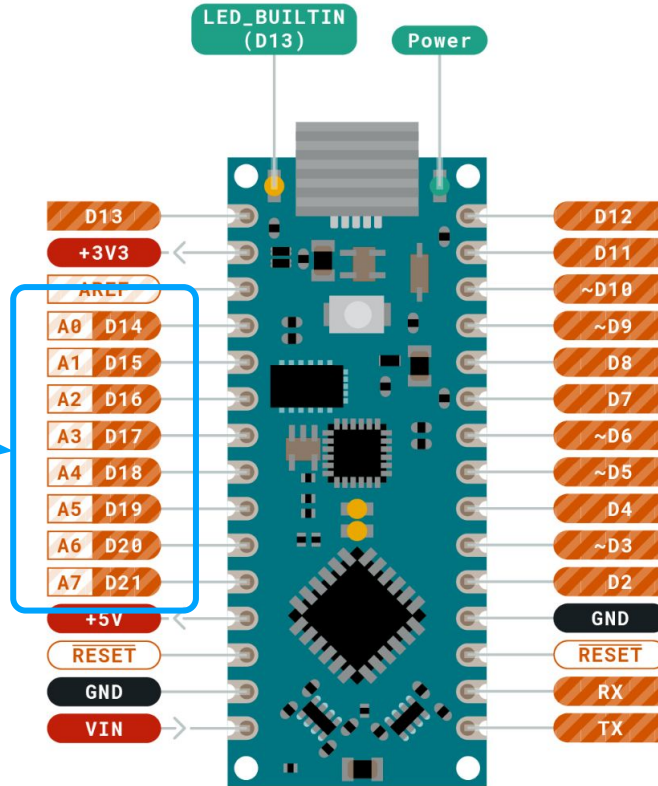


- `int digitalRead(int pin)`
  - **Reads the voltage** at the input pin, returning **HIGH** (5V) or **LOW** (0V) as an integer (1 or 0)
- `void digitalWrite(int pin, int value)`
  - **Sets the voltage** at the output pin to either a **HIGH** (5V) or **LOW** (0V) value
- Analogy - light switch and light bulb:
  - You use the switch to set the bulb to either MAX brightness or MIN brightness

# Analog Pins

## Analog Input Pins

These pins have the aliases **A0**, **A1**, **A2**, and so on... instead of **0**, **1**, **2**



## PWM-Capable Digital Output Pins

These pins are marked with ~ on the pinout diagram



# Analog Pin Functions



- `int analogRead(int pin)`
  - **Reads the voltage** at the input pin, maps it to a value in the **discrete range 0–1023** (0V to 5V) and returns that value
  - Use the aliases `A0`, `A1`, `A2...` for the pin number
- `void analogWrite(int pin, int value)`
  - **Sets the average voltage** on digital output pin to a value in the **discrete range 0–255** (0V to 5V)
  - This is a function for ~ **PWM pins only**
- Analogy - light dimmer:
  - You use the slide to set the bulb to anywhere *between* MAX brightness or MIN brightness

# ADC and analogRead

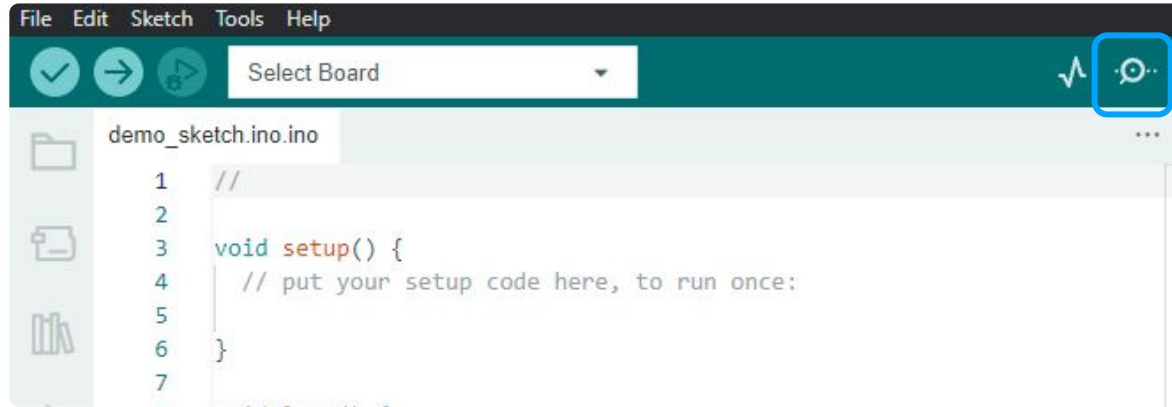
- **analogRead** utilizes the **analog-to-digital converter** inside the Arduino's AVR microcontroller to **measure the real-world, analog signal** and **convert it to a digital signal**
  - The measurement resolution is 10 bits, which is why the function returns values from 0–1023



# More Basic Functions

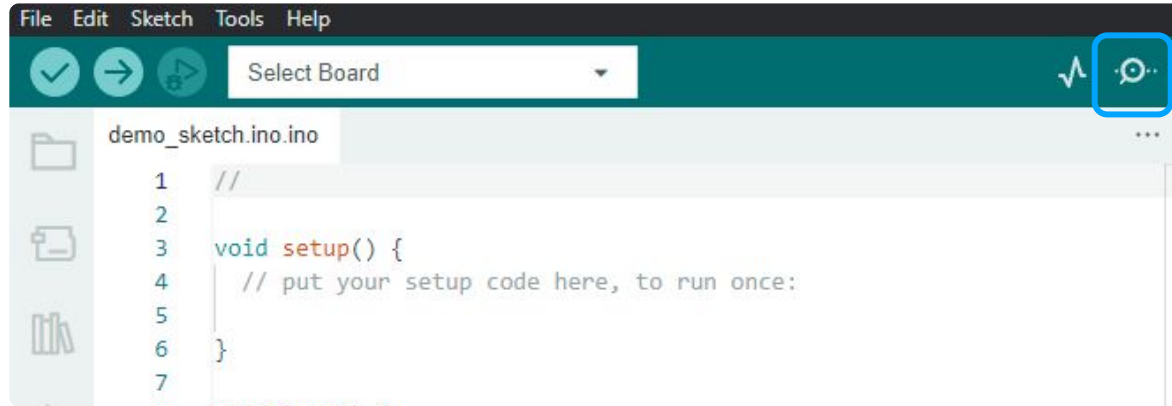
- `void delay(int ms)`
  - **Pauses the program execution** by `ms` milliseconds
- `Serial.print(val)`
  - Sends `val` to the computer connected via USB and **displays `val` on the Serial Monitor** in the IDE
- `Serial.println(val)`
  - Sends `val` to the computer connected via USB and **displays `val` on the Serial Monitor** in the IDE, **followed by a newline**

# Using the Serial Monitor



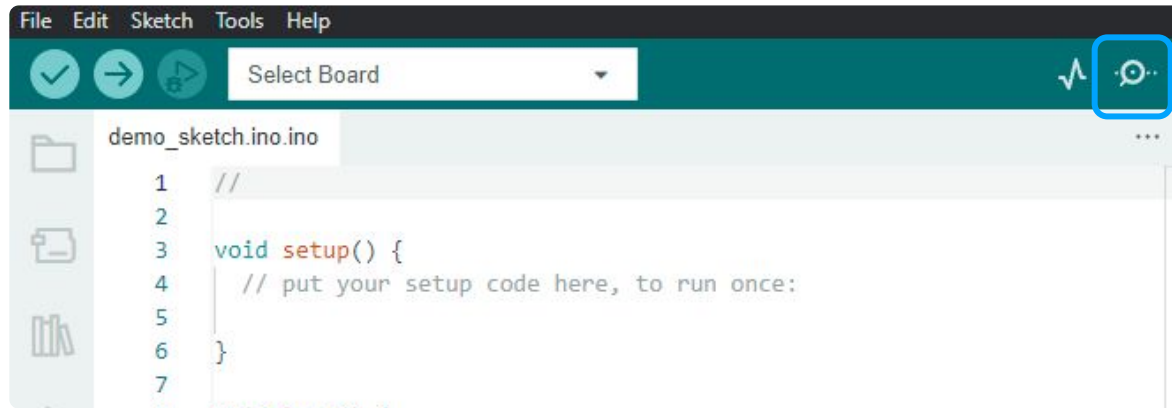
- While the Arduino board is connected to the personal computer via USB, select **Serial Monitor** (the **magnifying glass** icon) in the IDE
  - A pane will appear at the bottom of the IDE window which displays all data sent by the Arduino board using **Serial.print**

# Using the Serial Monitor (Cont'd)



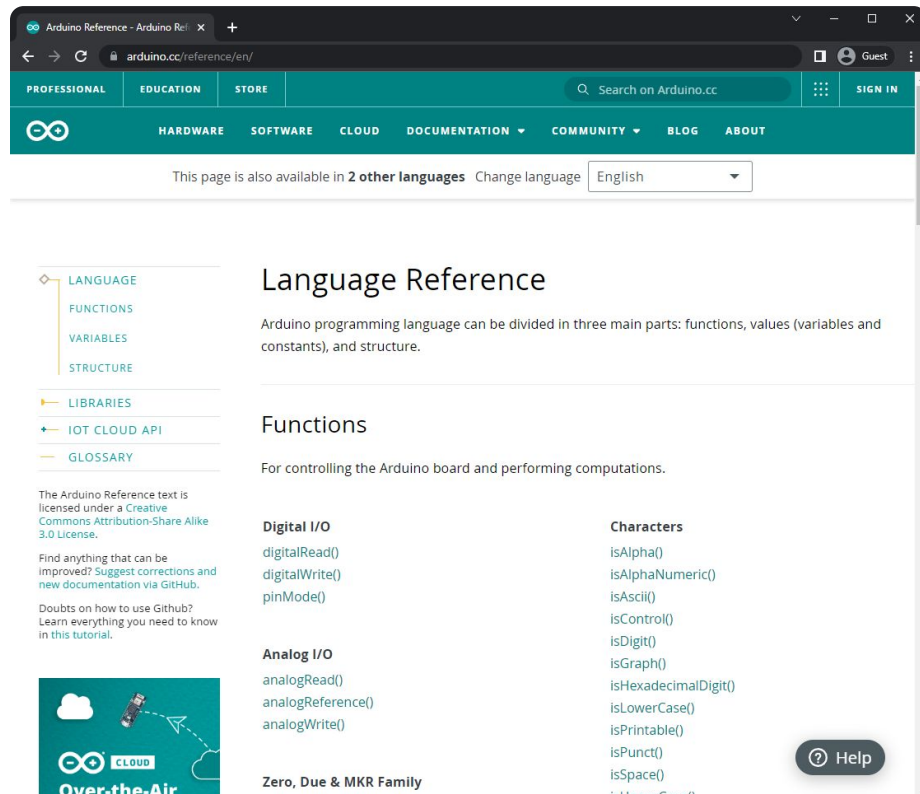
- In the absence of a debugger (the Arduino Nano is not capable of using one), **Serial.print** is an excellent tool to **help debug programs**
  - Print values to track across parts of your program
    - Unexpected values displayed to the Serial Monitor indicates an error

# Using the Serial Monitor (Cont'd)



- The C++ **iostream** library is not compatible with Arduino
  - Do not use `cin` or `cout` to print text
- **Serial.print** is the primary way to print text to the Serial Monitor

# Arduino Reference Library



The screenshot shows the Arduino Reference Library website. The browser address bar displays 'arduino.cc/reference/en/'. The navigation bar includes links for PROFESSIONAL, EDUCATION, STORE, and a search bar. Below the navigation bar, there are links for HARDWARE, SOFTWARE, CLOUD, DOCUMENTATION, COMMUNITY, BLOG, and ABOUT. A language selector indicates the page is available in 2 other languages, with 'English' selected. The main content area is titled 'Language Reference' and explains that Arduino programming language can be divided into three main parts: functions, values (variables and constants), and structure. A sidebar on the left lists categories: LANGUAGE (selected), FUNCTIONS, VARIABLES, STRUCTURE, LIBRARIES, IOT CLOUD API, and GLOSSARY. Below the sidebar, there is a section for 'Digital I/O' with a list of functions: digitalRead(), digitalWrite(), pinMode(), and analogRead(). Another section for 'Analog I/O' lists analogReference() and analogWrite(). A 'Characters' section lists various is\* functions like isAlpha(), isAlphaNumeric(), isAscii(), isControl(), isDigit(), isGraph(), isHexadecimalDigit(), isLowerCase(), isPrintable(), isPunct(), isSpace(), and isUpperCase(). At the bottom left, there is a 'Zero, Due & MKR Family' section. A 'Help' button is visible in the bottom right corner.

Arduino Reference - Arduino Re... x +

← → ↻ 🔒 arduino.cc/reference/en/ 👤 Guest ⋮

PROFESSIONAL EDUCATION STORE 🔍 Search on Arduino.cc ⋮ SIGN IN

🔗 HARDWARE SOFTWARE CLOUD DOCUMENTATION ▼ COMMUNITY ▼ BLOG ABOUT

This page is also available in 2 other languages Change language English ▼

## Language Reference

Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

### Functions

For controlling the Arduino board and performing computations.

#### Digital I/O

- `digitalRead()`
- `digitalWrite()`
- `pinMode()`

#### Analog I/O

- `analogRead()`
- `analogReference()`
- `analogWrite()`

#### Characters

- `isAlpha()`
- `isAlphaNumeric()`
- `isAscii()`
- `isControl()`
- `isDigit()`
- `isGraph()`
- `isHexadecimalDigit()`
- `isLowerCase()`
- `isPrintable()`
- `isPunct()`
- `isSpace()`
- `isUpperCase()`

The Arduino Reference text is licensed under a Creative Commons Attribution-Share Alike 3.0 License.

Find anything that can be improved? Suggest corrections and new documentation via GitHub.

Doubts on how to use GitHub? Learn everything you need to know in this tutorial.

Zero, Due & MKR Family

🔗 CLOUD Over-the-Air

🔗 Help

- Learn more about Arduino functions and libraries [here](#)
- The **Arduino Reference Library** includes support for devices like LCDs, Sensors, and WiFi modules

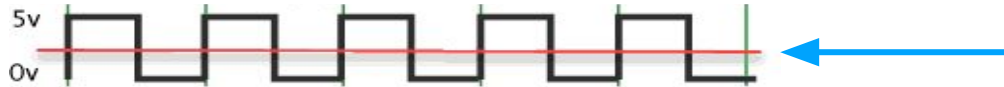
## **SECTION VII**

# **Pulse Width Modulation**



# PWM

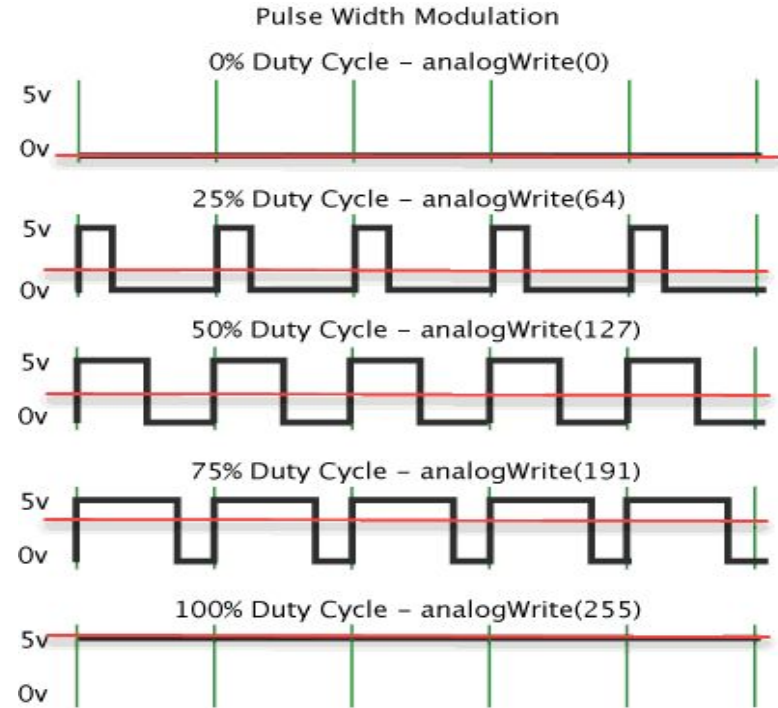
- The Arduino board (the underlying AVR microcontroller) is a **digital source**, meaning it can only output a **HIGH** (5V) or **LOW** (0V) voltage
  - Then how does **analogWrite** output analog signals?
- **Pulse Width Modulation (PWM)** is an oscillating digital waveform that emulates an analog output
  - By oscillating a signal from **HIGH** to **LOW** quickly, the average voltage over time will be *between HIGH and LOW* - an analog value



The average value, the *analog value*, of this waveform is **2.5V**

# PWM Duty Cycle

- The **duty cycle** of the PWM wave is the percentage of time where the signal is **HIGH**
  - For example, a 50% duty cycle translates to an average value of 2.5V (50% of 5V)
  - Allows us to output a continuous range of voltages between **HIGH** and **LOW**
  - Duty cycle is **controlled by a timer** inside the AVR microcontroller

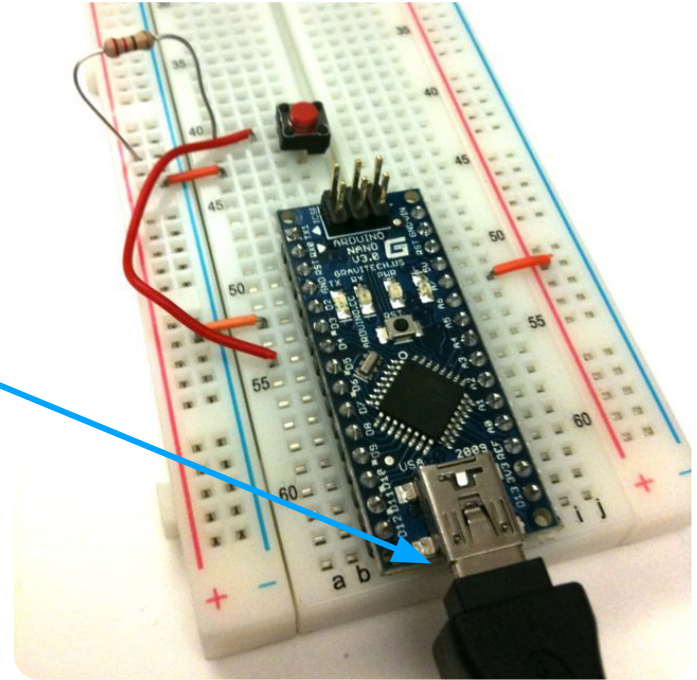


## **SECTION VIII**

# **Digital LED Circuit Exercises**

# Prototyping with the Arduino Nano

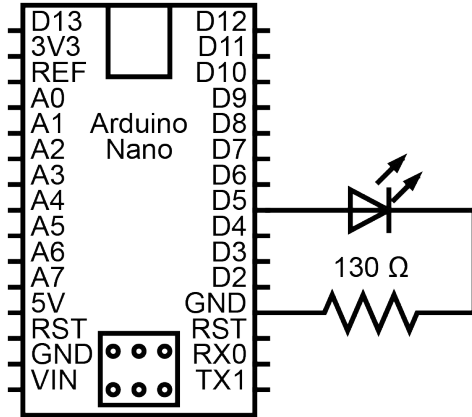
- The Arduino Nano can be **seated along the DIP channel** of a standard breadboard just like a DIP IC
- The **Arduino Nano's USB port should be oriented away from the board**, so the connected cable doesn't obstruct the breadboard
- Circuits that interface with the Arduino board must **share a common ground with the GND pin of the board**



# Digital LED Circuit

I/A

Build the circuit below from the schematic. Then, complete the template code, flash it to the Arduino board, and verify the circuit.



```
// Assign variable to pin number for LED

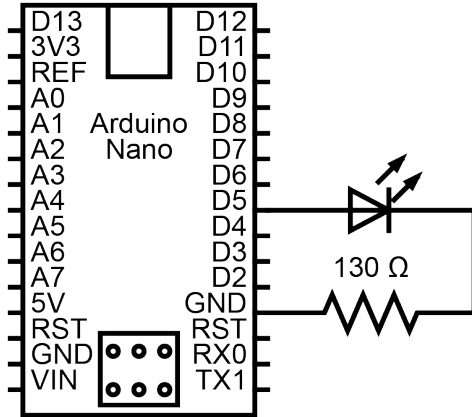
void setup() {
    // Configure LED pin's behavior to OUTPUT
    // Configure the Serial baud rate
}

void loop() {
    // Set LED pin to HIGH
}
```

# Digital Blinking LED Circuit

I/A

Build the circuit below from the schematic. Then, complete the template code, flash it to the Arduino board, and verify the circuit.



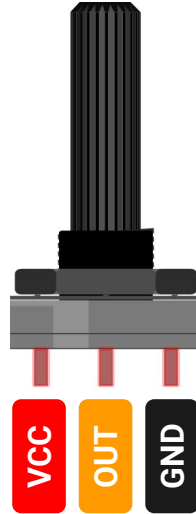
```
// Assign variable to pin number for LED

void setup() {
    // Configure LED pin's behavior to OUTPUT
    // Configure the Serial baud rate
}

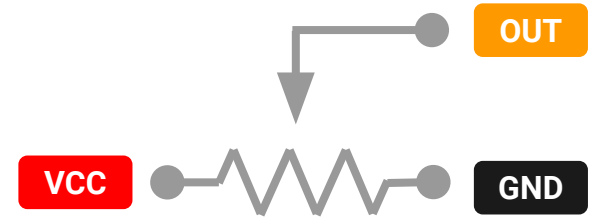
void loop() {
    // Set LED pin to HIGH
    // Delay
    // Set LED pin to LOW
    // Delay
}
```

# Potentiometers

- A **potentiometer** is a **three-terminal variable resistor**
- We will use the potentiometer as a **voltage divider** - a circuit which accepts a supply voltage and outputs a voltage which is a fraction of the supply voltage
- The voltage of the potentiometer's output pin ranges between the VCC and GND pin voltages

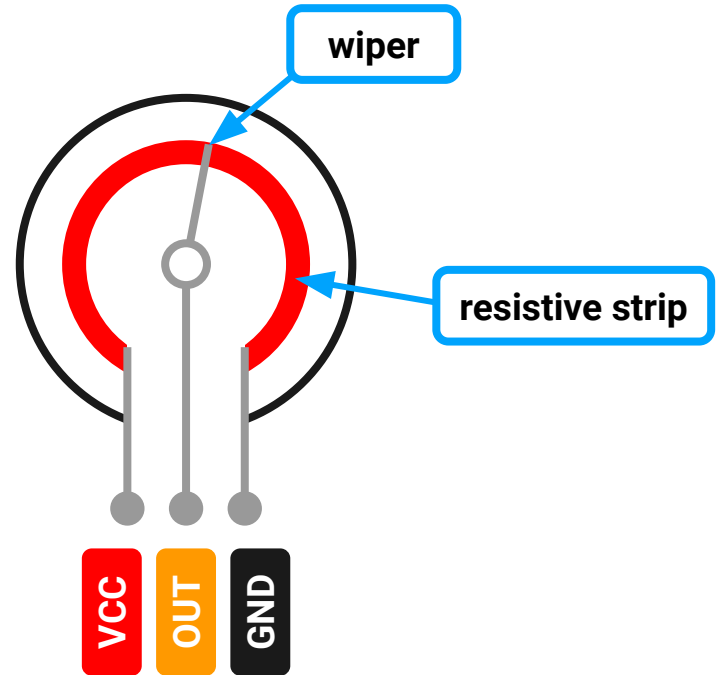


The positions of **VCC** and **GND** can be swapped



# Potentiometers (Cont'd)

- Internally, a **resistive strip** connects its VCC and GND pins
  - A rotating **wiper** connects the output pin to the strip
- **The greater the distance** along the strip between the wiper and the VCC pin, **the greater the resistance** between the wiper and VCC
- The wiper **reduces the voltage** at the output pin the further it is **turned clockwise** (toward GND)

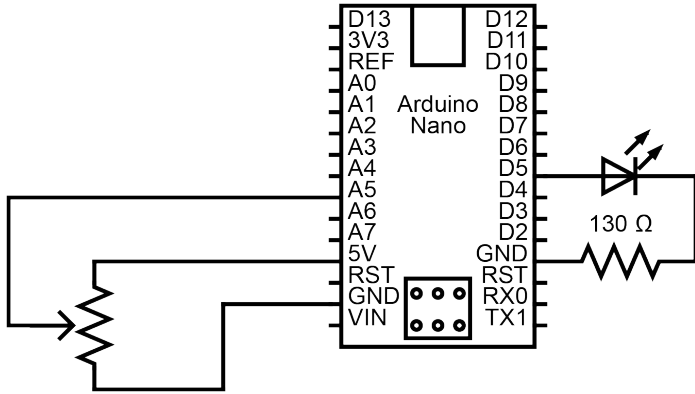




# Digital Dimmable LED Circuit

I/A

Build the circuit below from the schematic. Then, complete the template code, flash it to the Arduino board, and verify the circuit.



```
// Assign variable to pin number for LED
// Assign variable to pin number for Pot

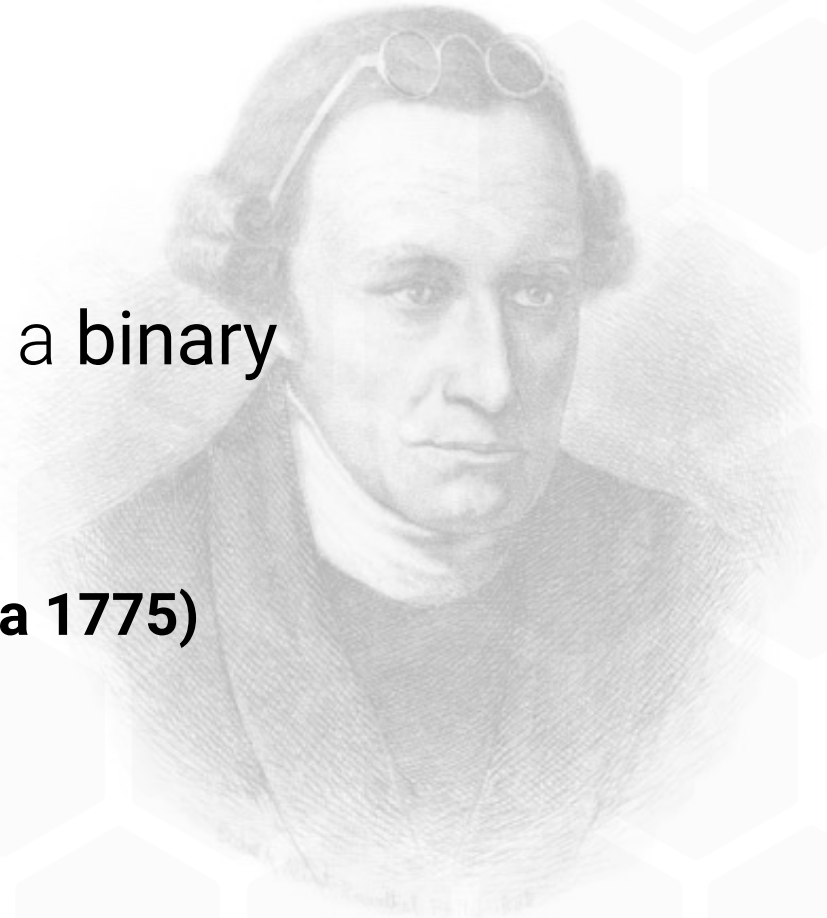
void setup() {
    // Configure the LED pin's behavior to OUTPUT
    // Configure the Pot pin behavior to INPUT
    // Configure the Serial baud rate
}

void loop() {
    // Read pot pin value
    // Set LED pin to the pot pin value
}
```

“Give me **PWM** or give me a **binary state** of existence!”

**Patrick Henry (circa 1775)**

Famous Misquotes



# FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.