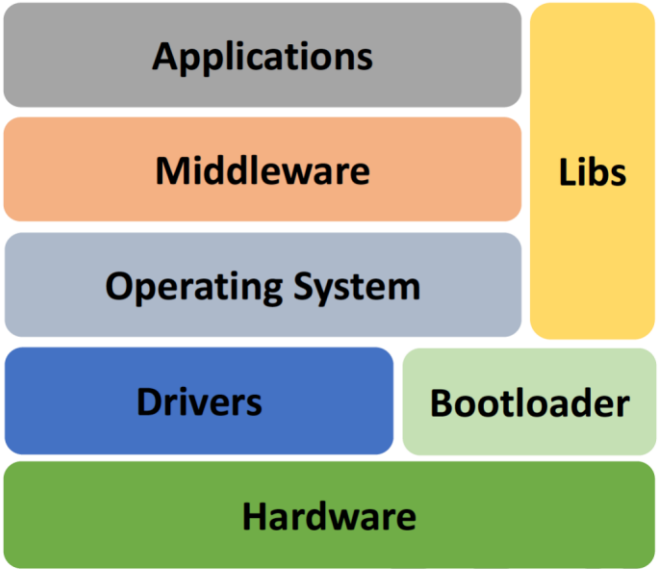
Introduction to Embedded Systems

Unit 1.6: Introduction to Embedded Programming in Arduino/C++ and the Arduino Uno, ESP32, and ESP8266 MCUs

Alexander Yemane
BCIT
INCS 3610
Fall 2025

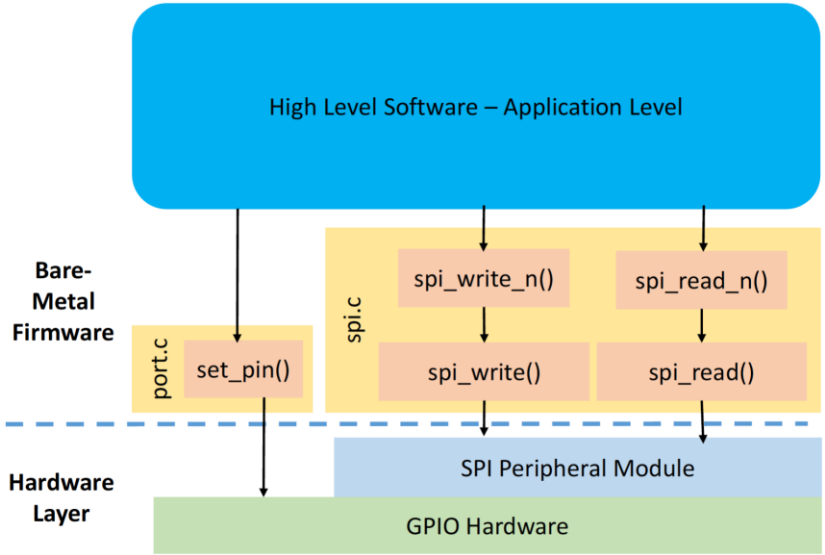
Embedded system software in layers

- I/O access exposed via
  - Device drivers
  - Hardware abstraction layer (HAL)
- Code booting via bootloader
- If OS is used, likely to be a “minimalist” real-time operating system (RTOS)
  - Typically no OS, but instead simple scheduler
- Libraries and middleware for unified services / utilities, e.g. networking, storage management
  - aka “software glue”

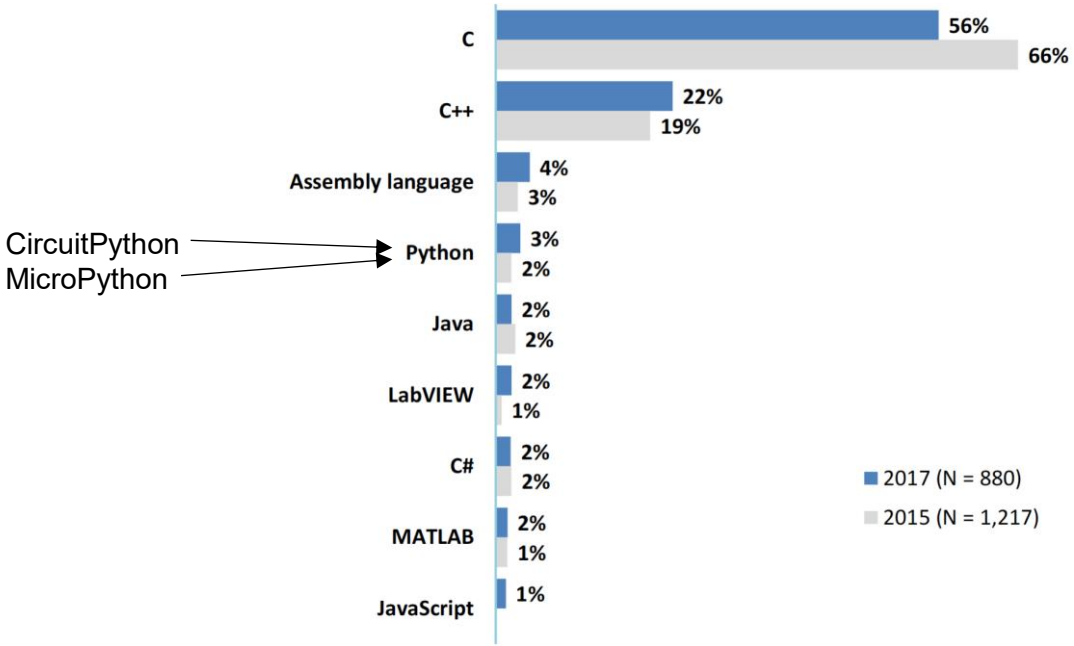


Application layer & how your code talks to hardware

- You will write your application code in an embedded programming language
- You will call libraries that use the drivers and HAL underneath
- Eventually interact with hardware, e.g.
  - GPIO
  - I2C/SPI/UART



Embedded programming language landscape



- ASPENCORE. (2017). 2017 Embedded Markets Study Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments. April, 1–102.

Embedded programming language trends

- The rise of AI developed systems
- Open-source code
- Security will be king
- More modern languages (C++, Rust)
- Simulation-first development
- DevOps (IT involved) and observability
- Edge (of network) AI

“7 Embedded Software Trends to Watch in 2025” – DesignNews, December 17th, 2024
https://www.designnews.com/embedded-systems/7-embedded-software-trends-to-watch-in-2025

Why C/C++ dominate for embedded programming

- Availability of compilers for almost any MCU
- Small executable
- Modest requirements for ROM, RAM
- Predictable resource use when you avoid unbounded dynamic allocation
- Timing-centric operations
- Direct access to hardware

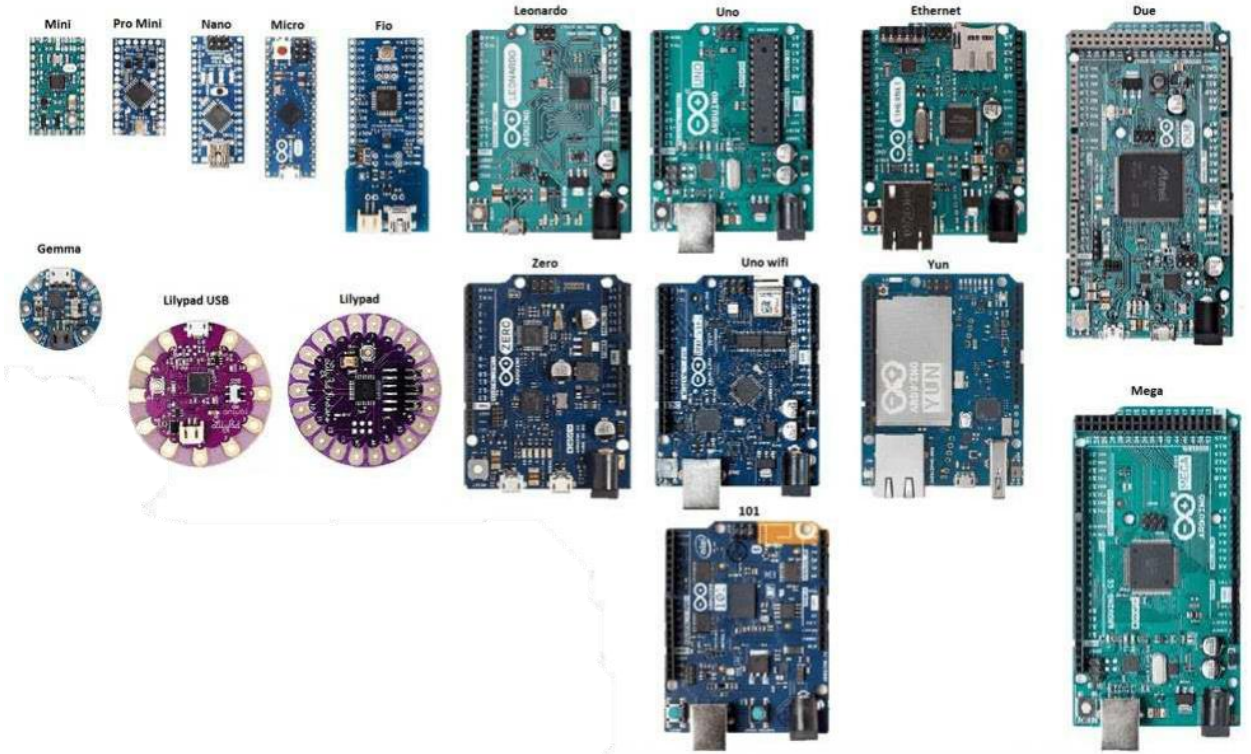
- Note: Modern C++ is as efficient as C and may slowly replace C in the future

Introduction to Arduino Systems

- A hardware and software company based in Italy
- Initial development – software and later hardware boards
- Produces and markets “official” boards*: Uno, Due, Leonardo, Diecimila, Mega, Nano
- Software and hardware is open source
- Acquired recently by Qualcomm (2025)

*See https://www.arduino.cc/en/Main/Products for more information

Big family of boards



Arduino Systems

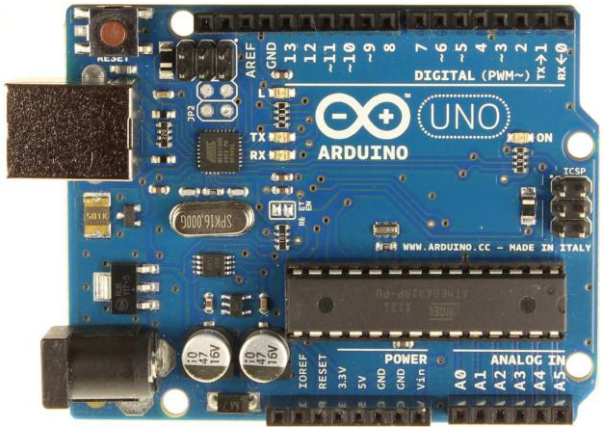
- An Arduino embedded system comprises of
  - Software & Software Tools
    - Integrated Development system (IDE)
    - Arduino programming language (based on C++)
    - Development & Debugging Tools
    - Software libraries
  - Hardware
    - ATmel processor boards (and others)
    - Shields (add-on modules)
    - Sensors, actuators, peripherals
  - Open Source Platform

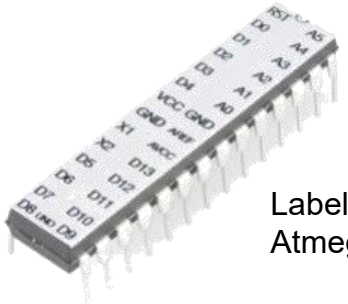
Why use Arduino Systems?

- Simple, clear programming environment using GUI
- Large selection of open-source libraries
- Inexpensive, lots of clones, 3rd party manufacturers
- Cross-platform (Windows, macOS, Linux)
- Global community of students, professionals, hobbyists

Arduino Uno

- Most common microcontroller board to begin Arduino projects.
- Uses an Atmel ATmega328P processor with a separate programmable interface using another Atmel processor and USB.
- Has sockets for interfacing and power.
- So popular that it is called Arduino (Please don’t make this mistake!)

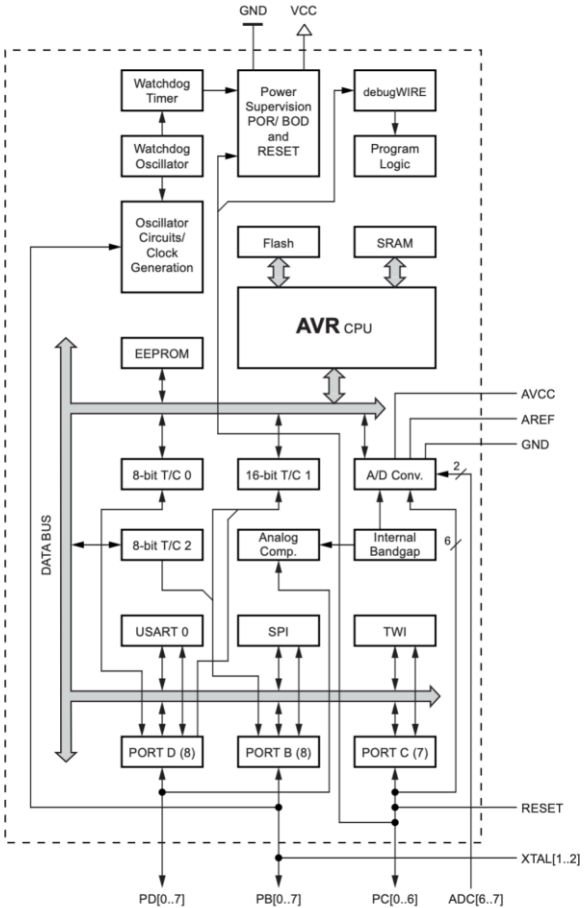




Labeled
Atmega328P processor

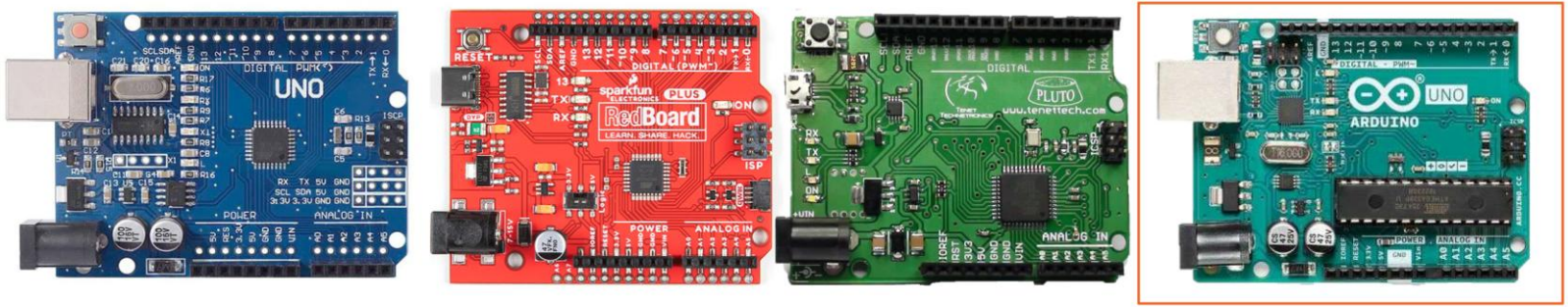
Atmel ATmega328P MCU

- 8-bit AVR processor running at 16MHz
- Harvard architecture
- 32 KB Flash, 1 KB EEPROM, 2 KB SRAM
- 14 digital input/output pins (6 can be used as PWM)
- 6 analog inputs (10-bit ADC)
- 1x USART, 1x SPI, 1x I2C (aka TWI) interfaces
- 2x 8-bit timers, 1x 16-bit timer



Variations of the Uno

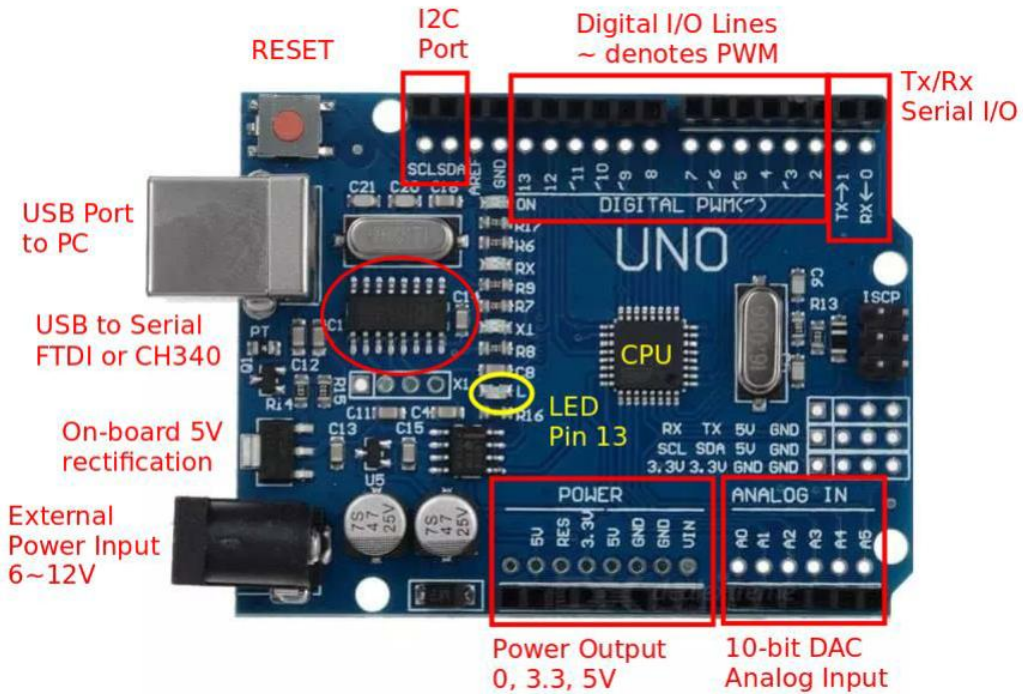
- Being open-source, there are many variations.
- Programming and usage are basically the same with some minor variations.
- All boards use the ATMega328P processor (may be in different formats)
- All boards have the same I/O pins
- Difference is in $$$



Main Uno board difference

- Original boards use the FTDI chip for USB
- Chinese clones use the CH340 chip
  - Need to install the CH340 drivers
  - Lots of tutorials from Google

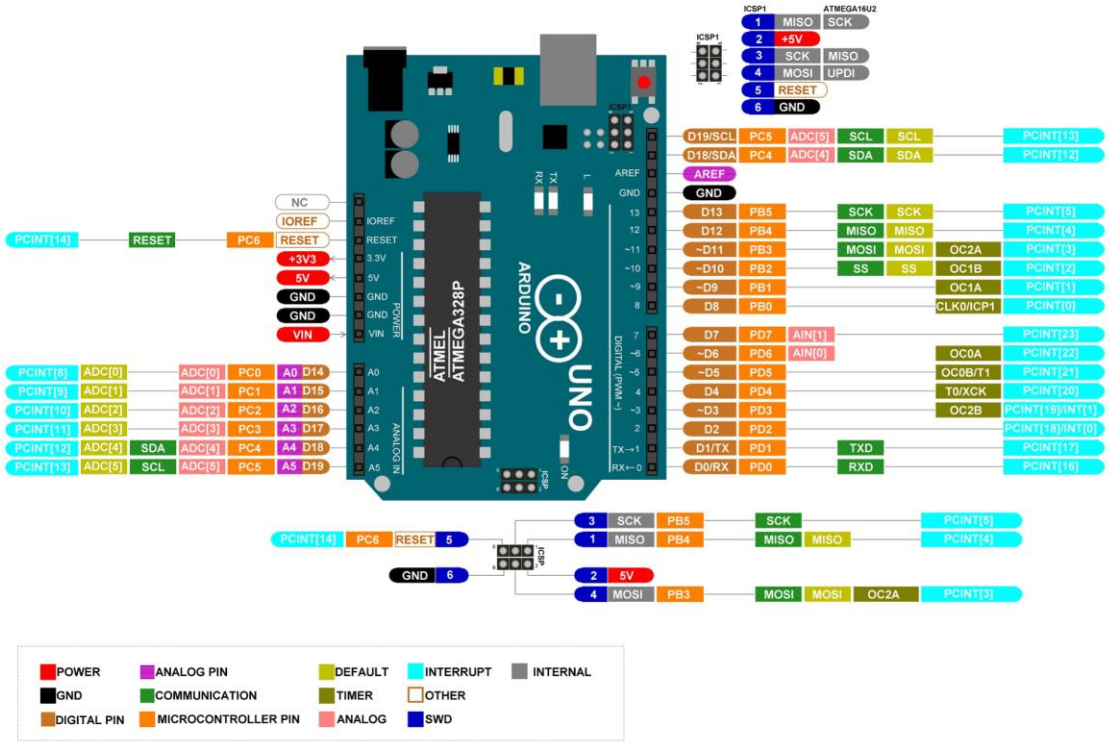
Uno board features



Uno board features

- Power
  - Can be powered from USB port (<0.5A)
  - External power from DC jack (6V-12V)
  - Power outputs: 0 (GND), 3.3V (voltage-regulated), 5.0V (up to 0.3A)
- Digital Input/Output pins
  - Can be configured as input, output or pull-up Inputs
  - Has pulse width modulation (PWM) on ~ indicated pins
  - Built-in LED on Pin 13
- Analog Inputs
  - 10-bit analog inputs
- Others
  - Serial I/O interface
  - I2C interface
  - SPI interface

Uno pin mapping



Programming the Uno

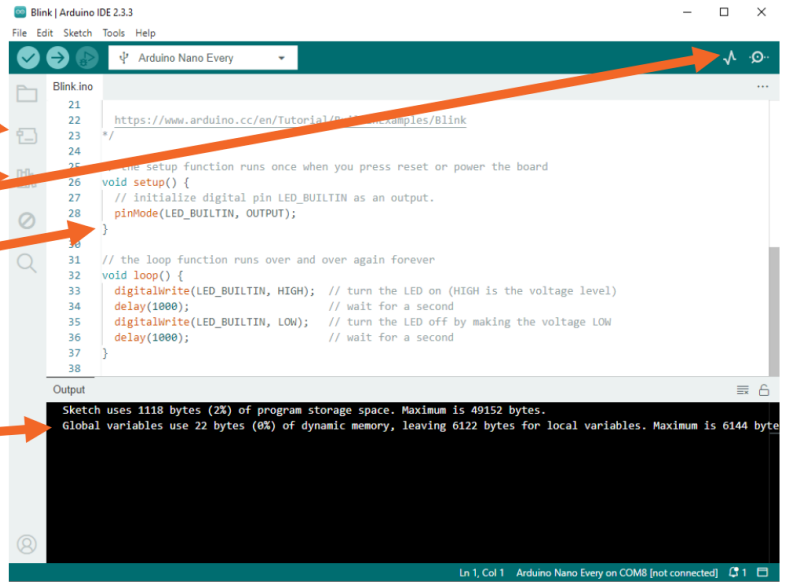
- The Uno uses a bootloader (2K code) which allows programming through the USB or FTDI interface.
  - When processor starts up or is reset, it loads and runs bootloader
  - If there is a programming command from the serial interface (USB/FTDI), it loads the program that you are sending via USB/FTDI
  - Else runs the last loaded program
- Bootloader and USB interface makes your work so much easier.

Programming the Uno

- Requires an Integrated Development Environment (IDE)
  - Arduino IDE is built upon an ATMEL C/C++ compiler (AVRdude)
  - Suggest using IDE v2 for features like:
    - Syntax highlighting
    - Auto Assist in typing keywords (Ctrl+Spacebar)
  - Differences with traditional C/C++ structure:
    - No direct access to main() function
    - Encourages use of functions instead of direct register programming

Arduino IDE

- Managing driver and compilers for all Arduino (and not only models)
- Library manager
- Serial Port Monitor
- Editor
- Compiler Output



Arduino vs C/C++ program example

C/C++

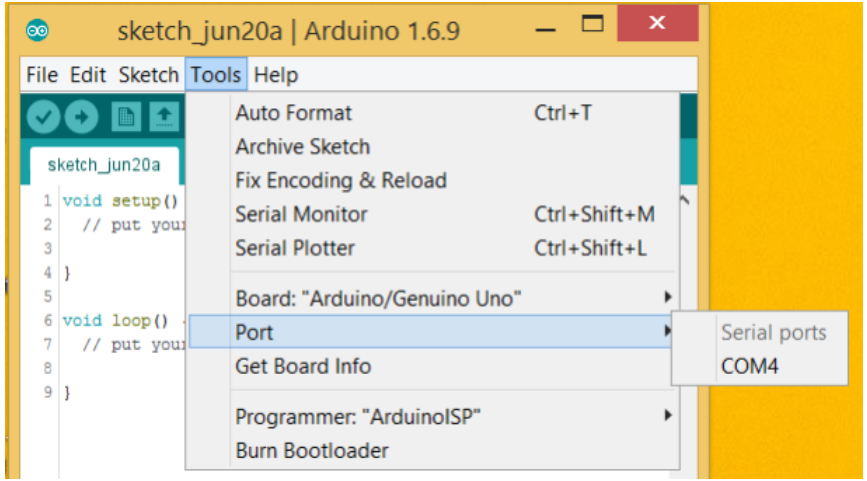
```
// programa de prueba blink

#define F_CPU 16000000UL    // frecuencia XTAL

#include <avr/io.h>
#include <util/delay.h>

int main()
{
   // inicializacion
   DDRC= 0B00100000;          //PC5 salida, resto entrada

   // ciclo infinito
   while (1)
   {
       PORTC= 0B00100000;    //hacer PC5 1
       _delay_ms(1000);
       PORTC= 0B00000000;    //hacer PC5 0
       _delay_ms(1000);
    }

    return 0;
}
```

Arduino Code

```
/*
  Blink
*/

void setup()
{
   // initialize digital pin LED_BUILTIN as an output.
   pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop()
{
   digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
   delay(1000);                       // wait for a second
   digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
   delay(1000);                       // wait for a second
}
```

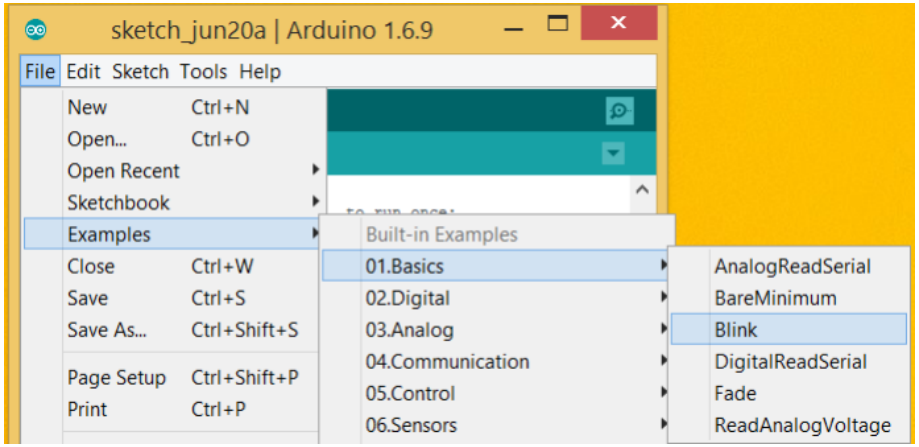
Using the Arduino IDE

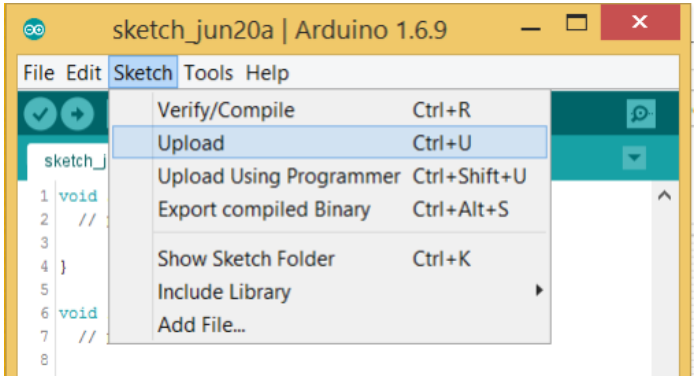
- Connect your Uno board to the host computer.
- Launch the Arduino IDE
- Select the correct board that you are using via Tools > Board
- Identify and check the port the board is connected to via Tools > Serial Port > (your COM port)



Test an example program

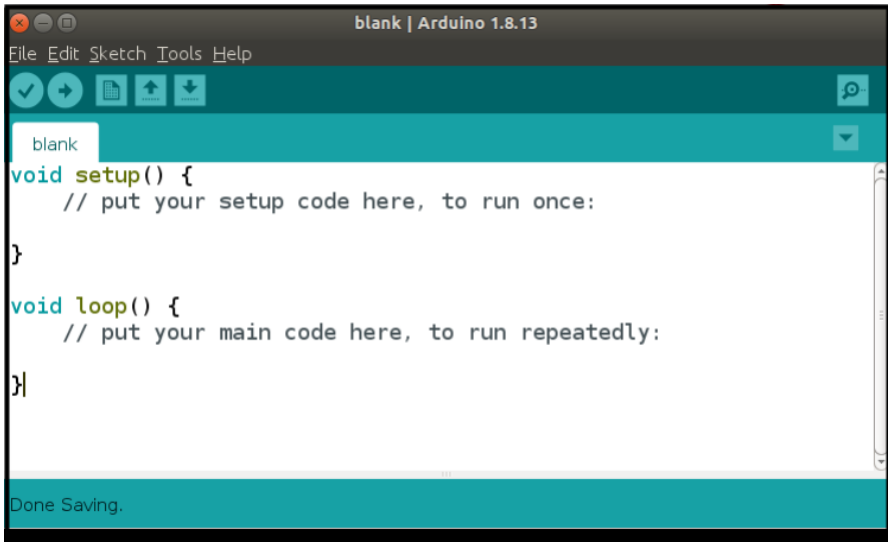
- Load an example program from the built-in sketches, e.g. “Blink”.
- Compile the program
- Upload to the Uno board
- Program executes after loading





Sketches

- Arduino programs are called sketches
  - In text
  - Have extension .ino
- All Arduino sketches have 2 functions:
  - setup()
  - loop()



setup()

- Executed only ONCE after each powerup or reset of the Uno.
- Uno is automatically reset after each successful sketch upload
- Contains
  - initialization code
  - initialization of variables
  - setup and configuration of I/O ports
  - setup of other interfaces
- Tip: Use identifiers to name your I/O pins, it makes programming and code recognition much easier

loop()

- The loop() function is executed after the setup() code completes.
- Loops infinitely executing code within the loop() function.
- Place your code/program within this function, there is NO stopping this code

Coding tips

- Follow good C++ programming habits:
  - Use a consistent coding style (indenting, function/variable name capitalization, brackets, etc.)
  - Write comments (//), and explain why rather than how
  - Name your variables and functions intelligently
  - Declare variables and function inputs and outputs with the correct data types
  - Don’t forget semicolons!

C++-like coding

- If
- Switch
- For loop
- While loop
- Functions
- Pointers
- Arrays
- Libraries

Variables

- Variables are memory set aside to hold changing data.
- Variables use different amounts of memory depending on the data type used.
- Common data types are
  - char 8-bit
  - int 16-bit
  - float 32-bit
  - String stores a sequence of characters
- We use identifiers (names) to name the variable and locate it.

Variable scope

- Variable declared inside a function is local
- Variable declared outside a function is visible anywhere inside that sketch
- Declaring a variable as ‘extern’ within a local scope tells the compiler you are referring to a global variable defined elsewhere

```
fcn1 ()
{
  // Local variable
  int x = 1;
}

// Global variable
int x = 2;
```

More data types

- array
- bool
- boolean
- byte
- char
- double
- float
- int
- long
- short
- size_t
- string
- String()
- unsigned char
- unsigned int
- unsigned long
- void
- word

Operators

Arithmetic operators

- % (remainder)
- * (multiplication)
- + (addition)
- - (subtraction)
- / (division)
- = (assignment operator)

Operators work only with **similar** data types

Use conversion functions to help.

Boolean operators

- ! (logical not)
- && (logical and)
- || (logical or)

Bitwise operators

- & (bitwise and)
- << (bitshift left)
- >> (bitshift right)
- ^ (bitwise xor)
- | (bitwise or)
- ~ (bitwise not)

Compound operators

- %= (compound remainder)
- &= (compound bitwise and)
- *= (compound multiplication)
- ++ (increment)
- += (compound addition)
- -- (decrement)
- -= (compound subtraction)
- /= (compound division)
- ^= (compound bitwise xor)
- |= (compound bitwise or)

Typecasting

- You can convert variables from one type to another by “casting”

```
// "cast" my_float to instead BE an int
int x = (int) my_float;
print(x);
```

- Creates an int with the float in it and prints it like any other integer.

Boolean

- Boolean variables have only 2 values: True/False
- Boolean expressions have only 2 results: True/False
- Comparison operators give a Boolean result
- Boolean operators work only with Booleans

- Used in conditionals and loops

Conditional: if … else

- Control structure which checks the condition expression and if true executes the following code block.
- Condition expression is a Boolean expression.
- When used with the else, control transfers to the else block.
- Can have nested or have multiples else-if conditionals for more granular control.

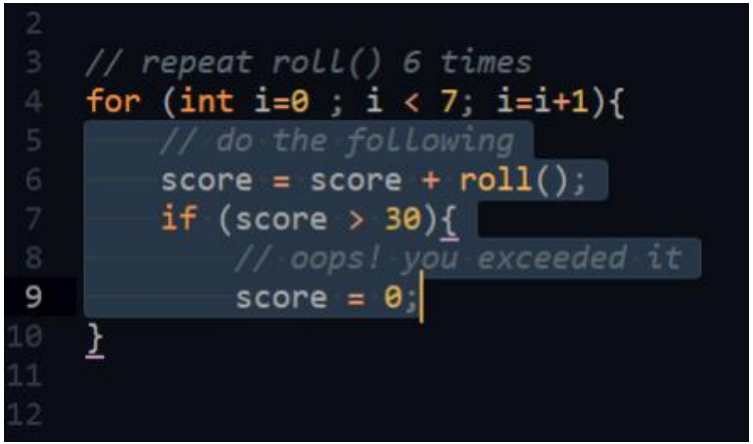
Conditional: switch … case

- Control structure which checks the value in switch (preferably ordinal) and transfers control to the matching case code block.
- Each case code block must be terminated with a break.
- Each case must match exactly.
- Control is transferred to the default code block (if any) if there is no match.

```
1
2  die = roll();        // rolls a die
3
4  switch(die){
5      case 6:
6          // scores and rolls again
7          score = score + 6 + roll();
8          break;
9      case 1:
10         // forfeits turn (no score)
11         break;
12     case 4:
13         // deducts points
14         score = score - 4;
15         break;
16     default:
17         // all other values
18         score = score + die;
19         break;
20 }
21
```

Loop: for

- Structure of for
  - Initialization
  - Conditional expression
  - Increment
- Used when we know exactly the number of times we wish to loop



Loop: while

- Tests conditional expression, if true the code block is executed.
- Indefinite loop, code block is executed zero, once or many times.
- If the condition results always in True, we have an endless loop.

```
1
2  // read the sensor
3  while (readSensor() > 25){
4      // adjust the settings
5      value = value - 1;
6      result = setLight(value);
7
8      if (result == -1)
9      {
10         // error has occured
11         break;
12     }
13 }
14
15 // what is the state here?
```

Loop: do … while

- Executes the code block before testing the conditional expression.
- If conditional expression is true the code block is repeated.
- Indefinite loop, code block is executed once or many times.
- If the condition results always in True, we have an endless loop

```
2
3  // execute code with switch is pressed
4  do
5  {
6    // control light intensity
7    value = value + 1;
8    light(value);
9    // check brightness
10   intensity = measureBrightness();
11 }
12 while ( intensity < 25 );
13
14 // what happens if the intensity
15 // was originally 10 before the loop
16 // is entered
17
```

Control: break

- When used in a loop, break exits the loop, control transfers to next statement after loop.
- break is also used to transfer control out of a matching case in a switch statement.
- Control: **continue** ignores the remaining statements and transfers control to the loop condition.

  (not commonly used)

```
1
2  // read the sensor
3  while (readSensor() > 25){
4      // adjust the settings
5      value = value - 1;
6      result = setLight(value);
7
8      if (result == -1)
9      {
10         // error has occured
11         break;
12     }
13 }
14
15 // what is the state here?
```

Functions

- A function is identified using ()
- A function is a block of code that can accept parameters.
- Executes the code when called, returns a single value as its name.
- **return** is used to return the value in the indicated data type.

```
1
2  /* function
3      name: cube
4      parameters: int value
5      returns: int
6  */
7  int cube(int value){
8      result = value * value * value;
9      return result
10 }
11
12 ...
13
14 // execution
15 myAnswer = cube(4);
16
17 int data = 6;
18 yourAnswer = cube(data);
19
20
```

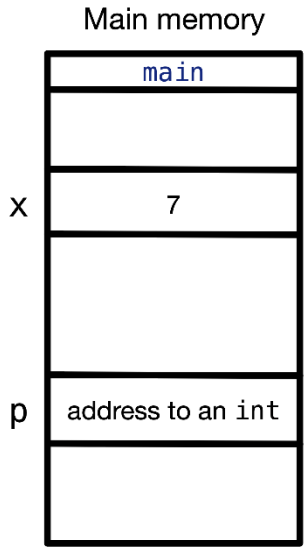
Arrays

- Arrays is a data type which can hold multiple values in a single variable.
- The values must be of the same data type, as defined by the array.
- Each element can be accessed using an index, which starts from 0.
- You can identify an array by “[ ]”
- Arrays are initialized with garbage values if not specified any

```
2
3   // create an array of size 5 elements
4   // populate the array
5   int dataset[5] = {45,55,65,75,86};
6
7   // display the values
8   for (int i=0; i < 6; ++i){
9     Serial.print("Index i = ");
10    Serial.println(dataset[i]);
11  }
12
```

Pointers

- A pointer is a variable whose value is the memory address of another variable



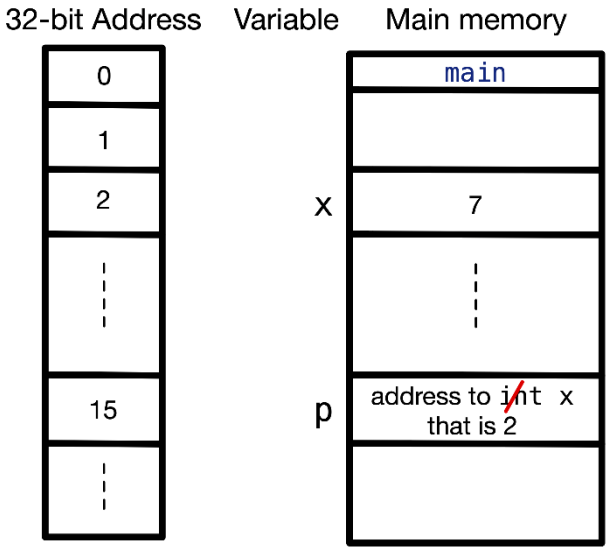

```c
#include <stdio.h>

int main(void) {
    int x = 7;
    int *p;

    return 0;
}
```

① data type of x is int

② data type of p is int*

Assigning value of a pointer

- Using the reference operator &, we can get the address of a variable




```
#include <stdio.h>

int main(void) {
    int x = 7;
    int *p;
    p = &x;
    return 0;
}
```

① data type of x is `int`

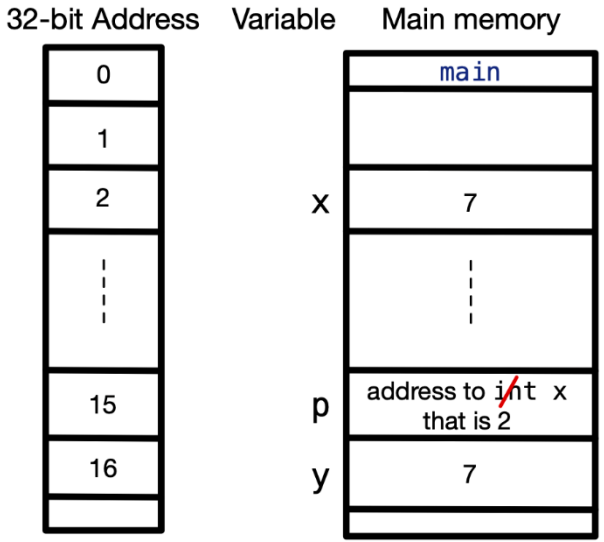
② data type of p is `int*`

③ p is assigned the address of x

Reference operator: this means "address of"
Recall its usage in `scanf`

Dereferencing a pointer

- Using the dereference operator * on a pointer variable, we can get the value stored at an address




```
#include <stdio.h>

int main(void) {
1  int x = 7;      data type of x is int
2  int *p;         data type of p is int*
3  p = &x;         p is assigned the address of x
4  int y;          declare variable y
5  y = *p;         y is assigned the value stored at address p — x
   return 0;
}
```

Dereference operator: this means "value at" address p

Class

- A user-defined data type that is used to create objects.
- An object has
  - attributes (constants, variables)
  - methods (functions)
- An object's attributes and methods are accessed using the dot (.) operator
- Classes are predominantly used in code libraries
  - E.g. Serial

HC-SR04 lab example

```
class HCSR04
{
    public:
        HCSR04(int trigPin, int echoPin);
        float calculate_distance();

    private:
        int _trigPin;
        int _echoPin;
};
```

```
#include "HCSR04.h"
#include <Arduino.h>


HCSR04::HCSR04(int trigPin, int echoPin)
{
  pinMode(trigPin, OUTPUT); // set the trigger
     pin as output
  pinMode(echoPin, INPUT);  // set the echo pin
     as input
  _trigPin = trigPin;
  _echoPin = echoPin;
}

float HCSR04::calculate_distance()
{
  // write a pulse: when trigger pin is set to
     HIGH, start transmitting
  digitalWrite(_trigPin, LOW); // clear the
     trigger pin
  delayMicroseconds(10);
  digitalWrite(_trigPin, HIGH); //set the
     trigger pin to HIGH for 10 ms
  delayMicroseconds(10);
  digitalWrite(_trigPin, LOW);

  long duration = pulseIn(_echoPin, HIGH); //
     time elapsed for sending and receiving
     pulse wave
  float distance = duration * 0.034 / 2;     //
     calculate distance in cm

  return distance; //return distance
}
```

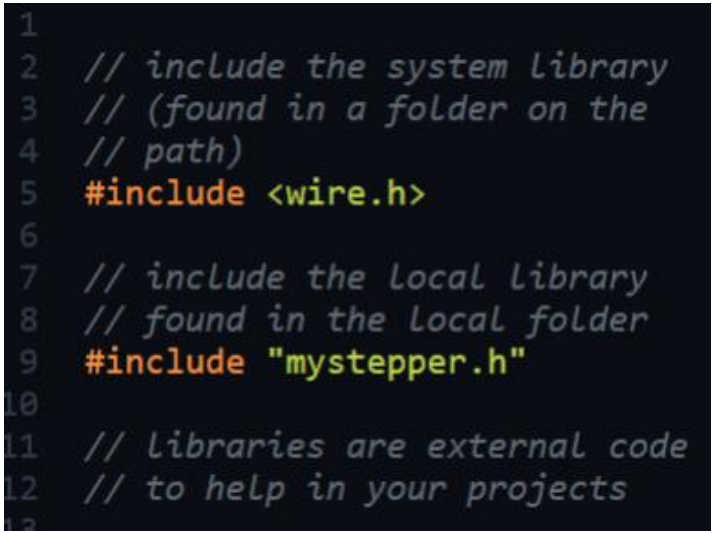
Directive: #define

- #define is a compiler directive and not a code statement.
- Does not end with a semicolon
- Used to name a constant and assign the value
- **const** is the preferred method of defining constants

```
1
2  // Replace all occurences of LED_RED
3  // with the value 7
4  #define  LED_RED  7
5
6  // alternative
7  // const is a keyword
8  const int LED_RED = 7;
9
```

Directive: #include

- #include is a compiler directive and not a code statement.
- Does not end with a semicolon
- Instructs the compiler to read and insert code from the target file
- < file > indicates system library, found along the library path
- “ file “ indicates local file in same folder



Digital I/O

- ATMega328 has 14 digital I/O pins
  - Labelled Pin 0 to Pin 13
  - Pin 0 (Tx), Pin 1 (Rx) are assigned as Serial I/O
  - Pin 3, Pin 5, Pin 6, Pin 9, Pin 10 and Pin 11 have PWM functionality
- A digital I/O pin can input or output digital (0V, 5V) signals.

- Some pins are multifunctional, i.e. have different functions depending on how they are initialized.
  - inputs (default)
  - inputs with pull-up resistors
  - outputs
  - PWM outputs

Digital I/O functions

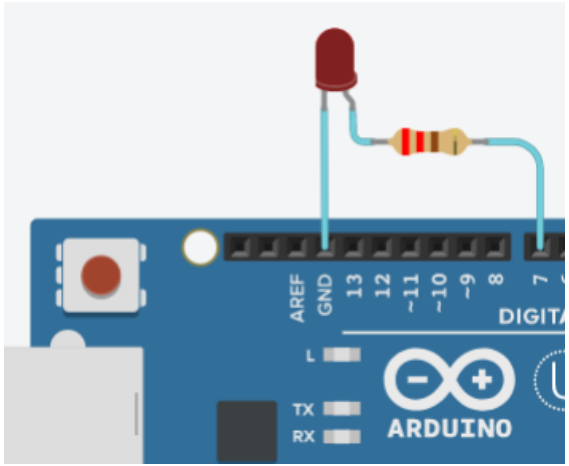
- The Arduino system provides 3 functions for the manipulation of digital I/O.
- You need to
    1. Configure the pin (pinMode()), before
    2. Using the pin
        - digitalWrite() output
        - digitalRead() input

pinMode(pin, MODE)

- Configures specified pin to behave either as input or output.
- Modes available
  - INPUT
    - digital input mode (high-impedance states)
  - INPUT_PULLUP
    - digital input mode with internal 20K-50K ohm pull-up resistor
  - OUTPUT
    - digital output mode
    - able to source up to 40mA per pin, total of 200mA per chip

digitalWrite(pin, {LOW|HIGH})

- Outputs a LOW (0V) or HIGH (5V) to a digital pin.
- The digital pin must be configured as OUTPUT.




```
2
3   // Red LED connected to pin 7
4   #define RED   7
5
6   ...
7
8   // set as digital OUTPUT
9   pinMode(RED,OUTPUT);
10
11  // flash the LED
12  digitalWrite(RED, HIGH);
13  delay(300);
14  digitalWrite(RED, LOW);
15  delay(300);
16
```

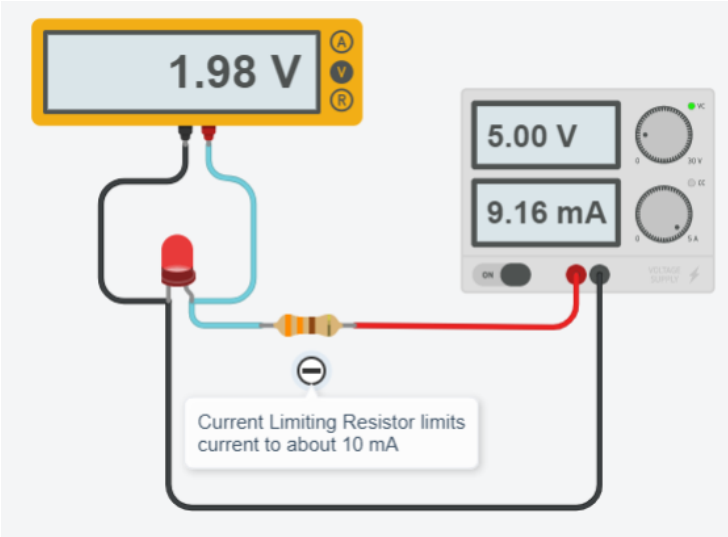
Driving an LED

- LED lights up (conducts) if a correct voltage is applied to the pins.
- When the LED conducts, current is allowed to pass through. The LED drops about 2V.
- We need to limit this current (10-20mA) otherwise, we will get a short-circuit.
- Current-limiting resistor value:

  R = V / I

  = (5 – 2V) / 10 mA

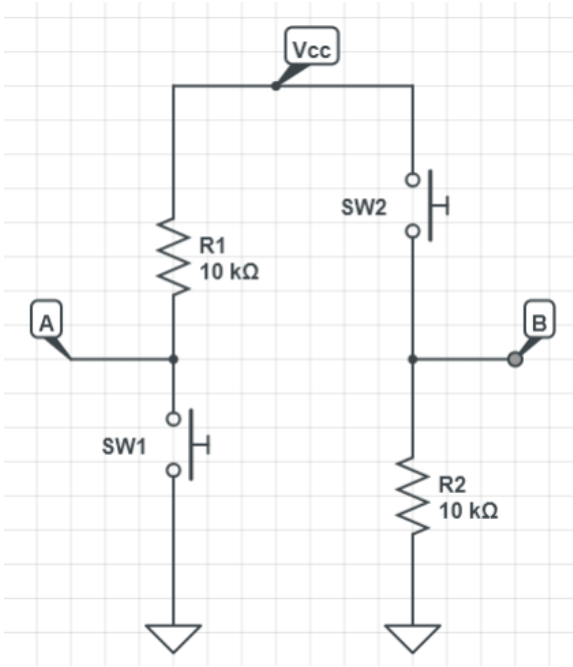
  = 300 ohms



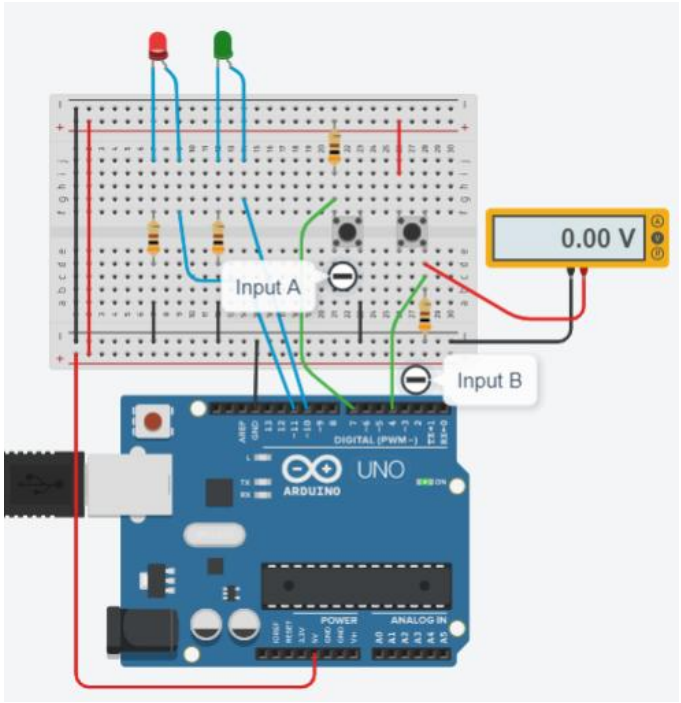
LED current depends on type, check data sheet for forward voltage and current limits.

Digital input using pull-up/down resistors

- We can use digital inputs to read the status of the switches in a circuit.
- Need to add a current-limiting resistor to prevent short circuits.
- Usual value is 10k Ohm
- States:
  - A – normal HIGH, when closed LOW
    - Pull-up
  - B – normal LOW, when closed HIGH
    - Pull-down



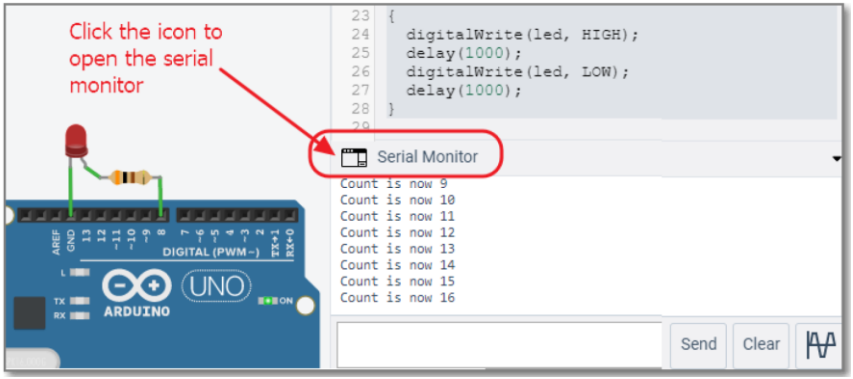
Digital input example




```
#define RED 11
#define GREEN 10
#define A   7
#define B   4

void setup()
{
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(A, INPUT);
  pinMode(B, INPUT);
}

void loop()
{
  if (digitalRead(A) == HIGH){
    digitalWrite(RED, HIGH);
  }
  else{
    digitalWrite(RED, LOW);
  }
  if (digitalRead(B) == LOW){
    digitalWrite(GREEN, LOW);
  }
  else{
    digitalWrite(GREEN, HIGH);
  }
}
```

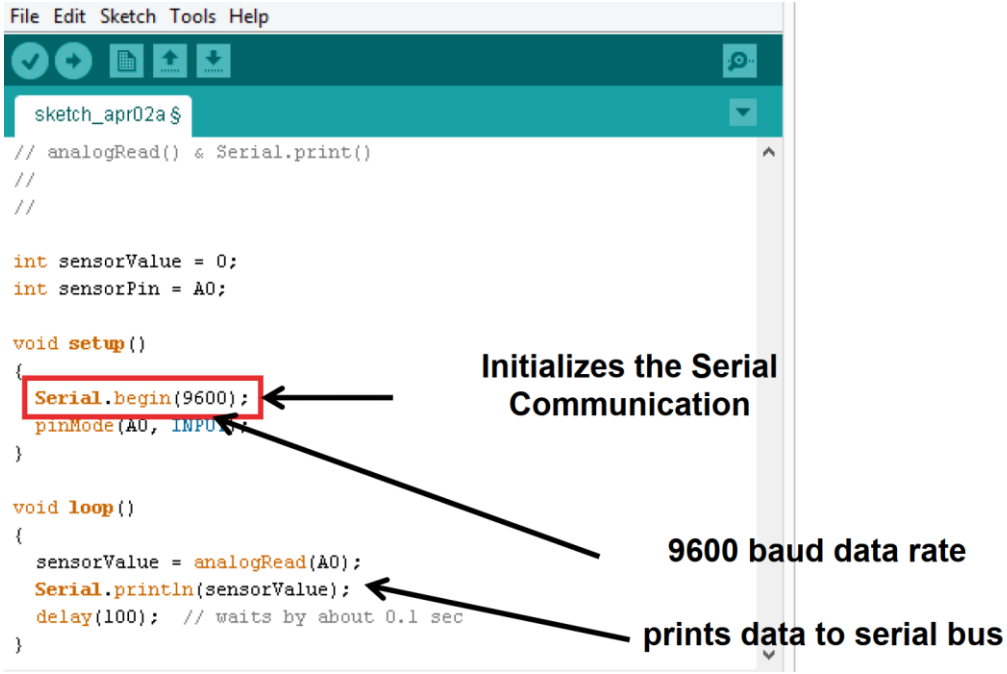
Arduino Serial communication

- Most microcontrollers do not have display
- When connected to a computer, output can be displayed by the computer, mostly for debugging purposes
- In most cases, the connection is realized using serial interface (UART)
- The Arduino IDE provides a Serial Monitor for displaying text messages
  - You can also input data to it
- Also provides a Serial Plotter to graph values
- When using Serial library functions, you must not use the Tx (0) or Rx (1) pins for any I/O



Serial library

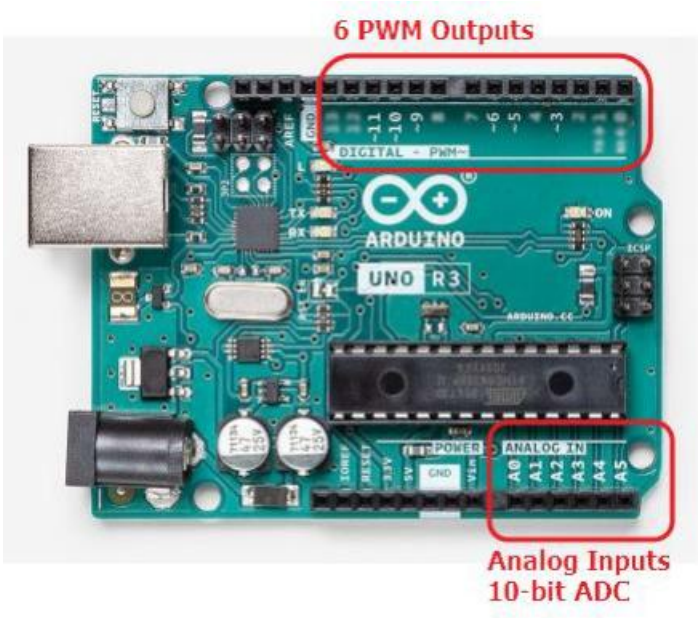
- Use Serial to display the value from an analog sensor. The sketch displays a new sensor value every 0.1 seconds.
- There are other Serial functions, but usually not used in embedded circuits.



Analog I/O

- ATMega328 has 6 analog input pins
  - Labelled A0 to A5
- Each analog input has a 10-bit analog-to-digital converter that can produce an equivalent binary value for an analog voltage between 0 and Vref.

- Analog output is done using PWM which can be used to control LEDs and motors.
- Pins that can perform PWM are denoted with a ~



Reading analog signals

- Analog signals need to be converted to digital values before they can be processed.
- Analog-to-Digital conversion required
  - Sampling
  - Vref
  - Timing

Simplified ADC with Arduino system

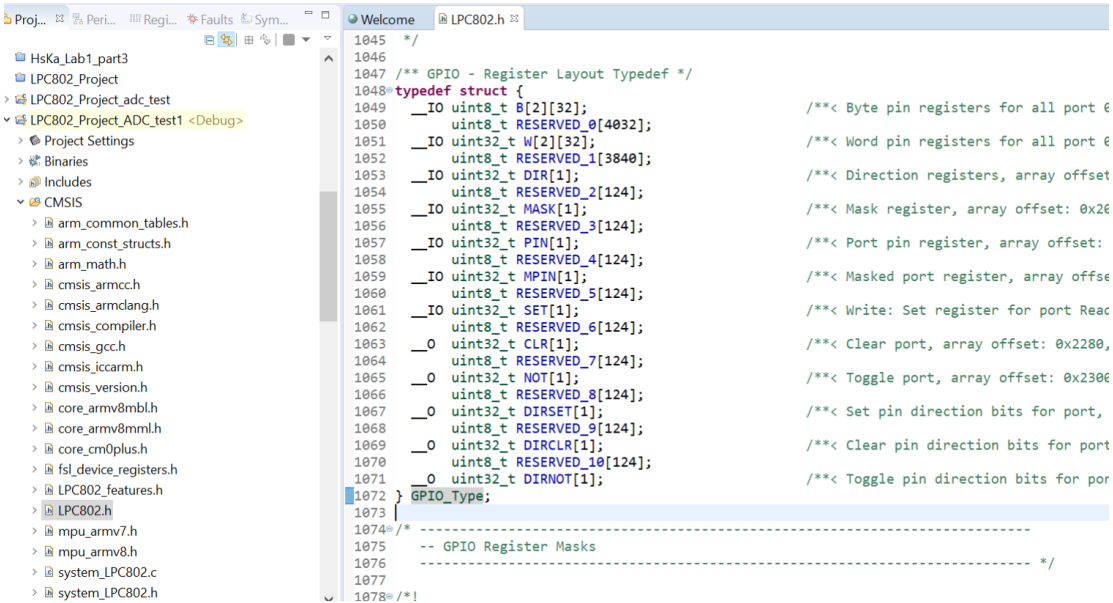
- Assumes that input signals are stable and do not change quickly.
- Uses a default Vref of 5V
- Up to 2^{10} or 1024 (10 bits) voltage levels / states
  - 0V – 0
  - 5V – 1023
- Smallest measurable voltage change is 5V/1024 or 4.88mV
- Maximum sample rate is 10,000 times a second (one sample every 0.0001s)

Libraries

- Abundance of code libraries for every known sensor, device, application
- No need to write your own code, use a library
- Lots of examples available
- Library consists of
  - C++ file (.cpp) which holds the code (usually a Class) which talks to the device/application
  - Header file (.h) which holds the definitions of the Class and functions available.
  - Since Arduino use an ATMEL compiler, you can move any C/C++ code to Arduino
- Once library is loaded and included in your source code, we just need to use the functions/methods to implement our application.

Accessing hardware registers

- All modern vendor IDEs come with chip-specific memory-mappings and data structures in header files.
  - LPC802.h part of SDK for LPC802 MCU



Loading the library

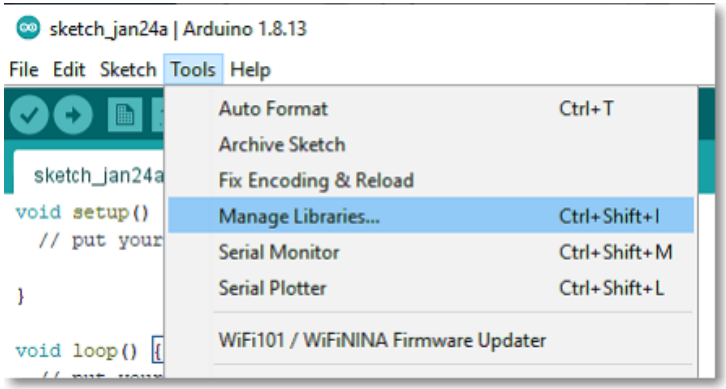
- 3 ways of loading the library
  1. Use the Arduino Library Management system
  2. Download the compressed library (.zip) and use the IDE to load the library
  3. Download the compressed library (.zip), extract the .h and .cpp files, manually copy them into the correct folders, restart the IDE.
- Library folder location

Library	Folder location
Arduino IDE System Libraries	C:\Program Files (x86)\Arduino\libraries
User libraries	C:\Users\<username>\Documents\Arduino\libraries

- Setting the User Library folder: **IDE > File > Preferences**

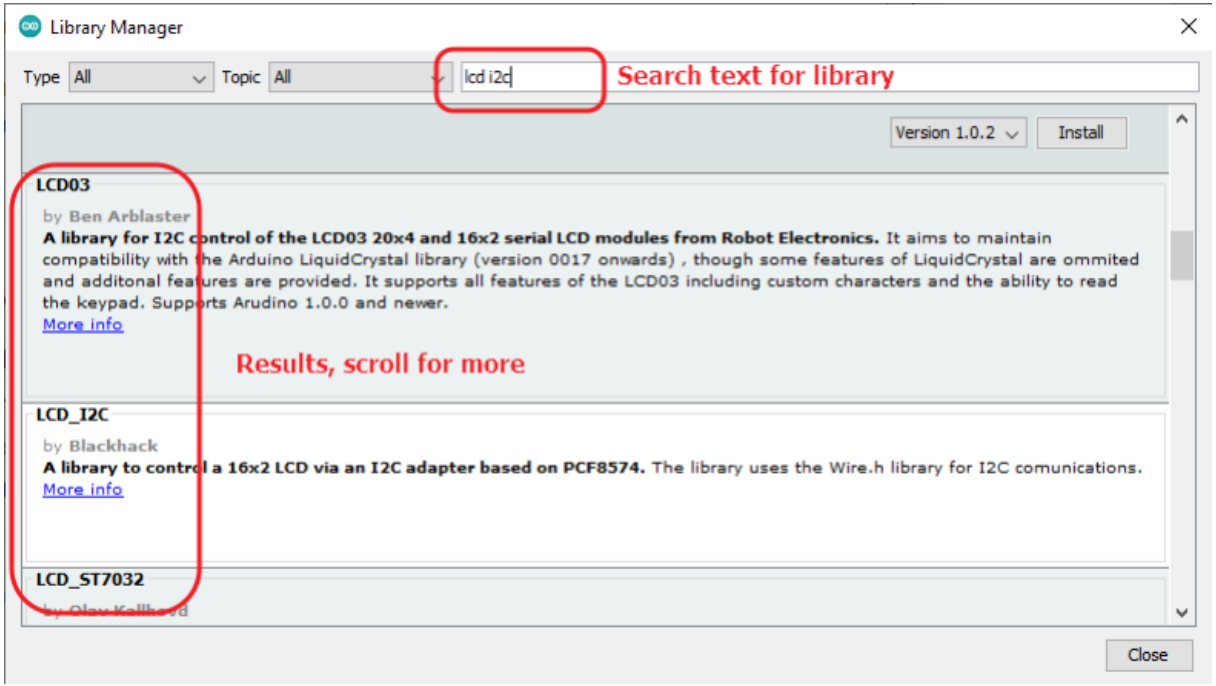
Arduino IDE managed libraries

- Simplest method of loading a library
- Recommended by Arduino, for most used libraries
- IDE > Tools > Manage Libraries



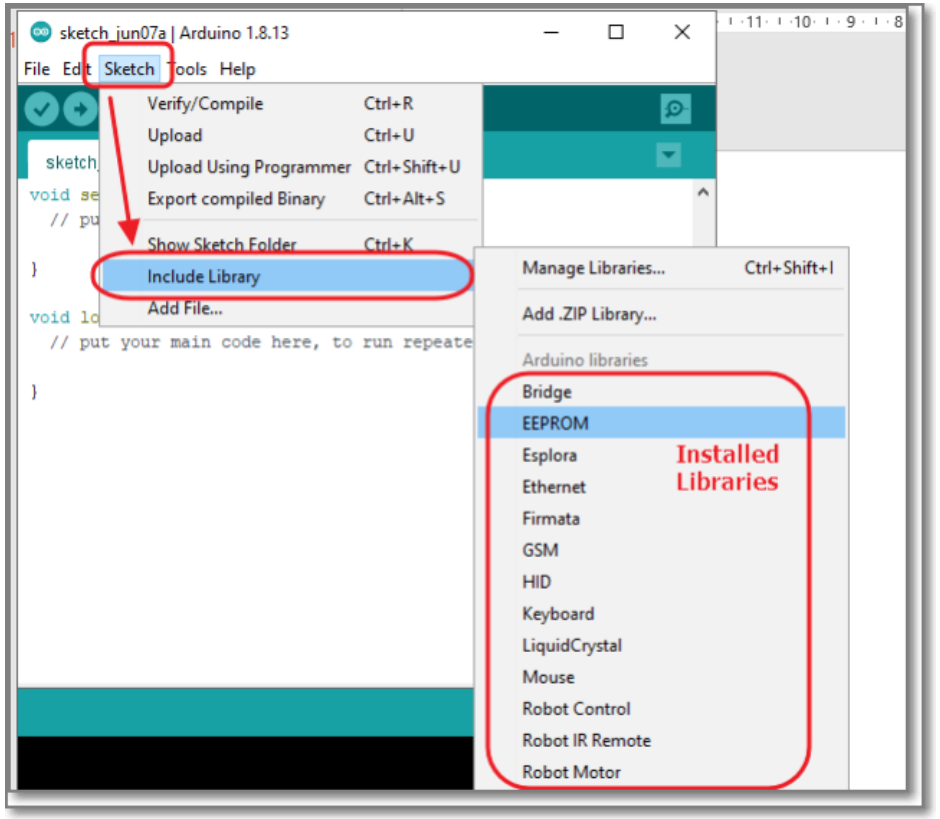
- Search for the library you require
- Select the library
- To use the library: IDE > Sketch > Include library
- Sometimes examples are included with the library for you to test the functions.

IDE managed library example



Including the header file

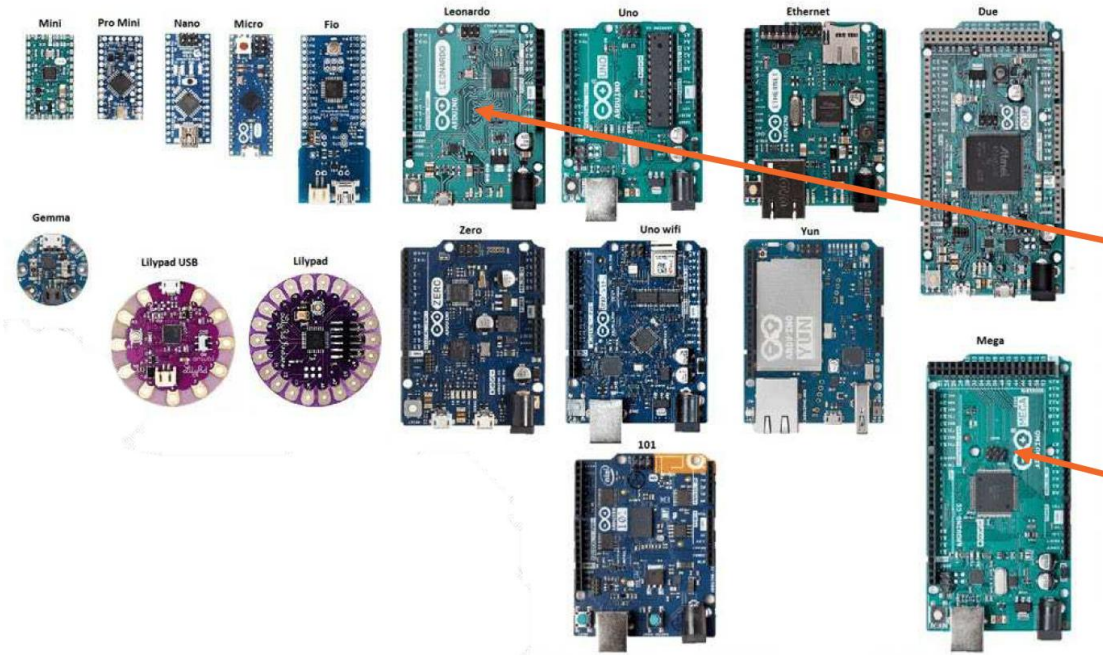
- Drop in the appropriate include statement for your library header file via Sketch > Include Library
  - #include <device.h> for system libraries
  - #include “device.h” for user libraries



Arduino shields

- Most Arduino boards have very basic capabilities
- Additional hardware implementing advanced features can be connected by using shields
  - Ethernet, Bluetooth, 2G/3G, WiFi, …
  - Real-time clock
  - SD card
  - Audio, display
  - High power relay
  - GPS
- Most of the addons use I2C or SPI buses

Choosing other Arduino boards



An example:
UNO has 13 digital pins

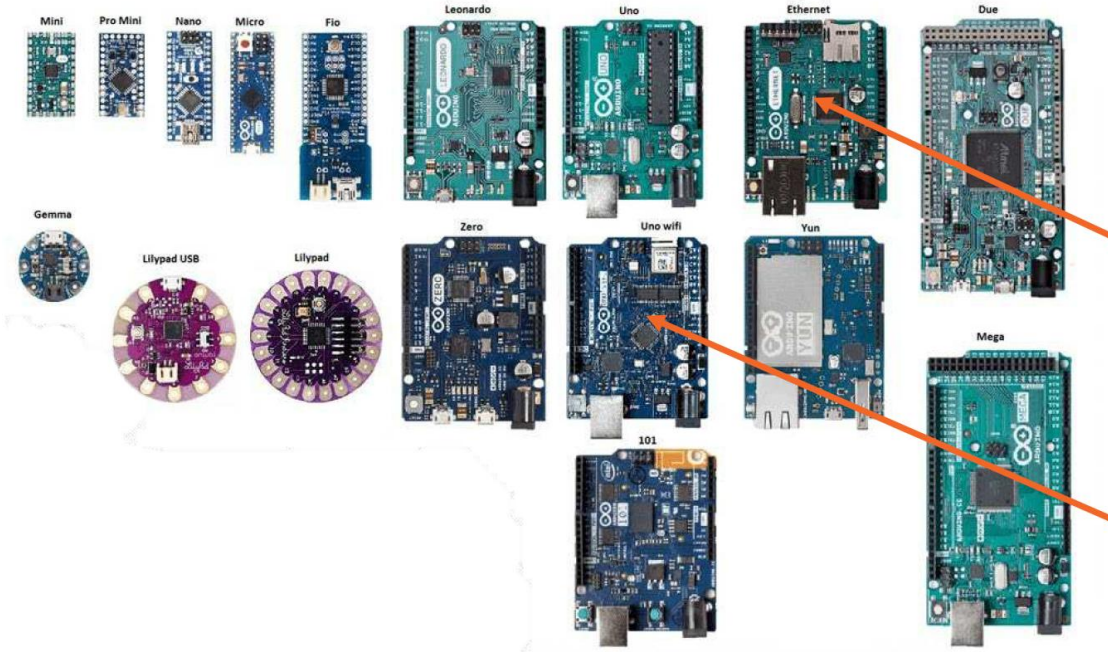
More pins?

Leonardo has
20 digital IO pins

More Pins?

Mega has 54 Digital IO pins!

Choosing other Arduino boards



Another example:
UNO can be connected to the PC only via USB (serial port)

Other connection?

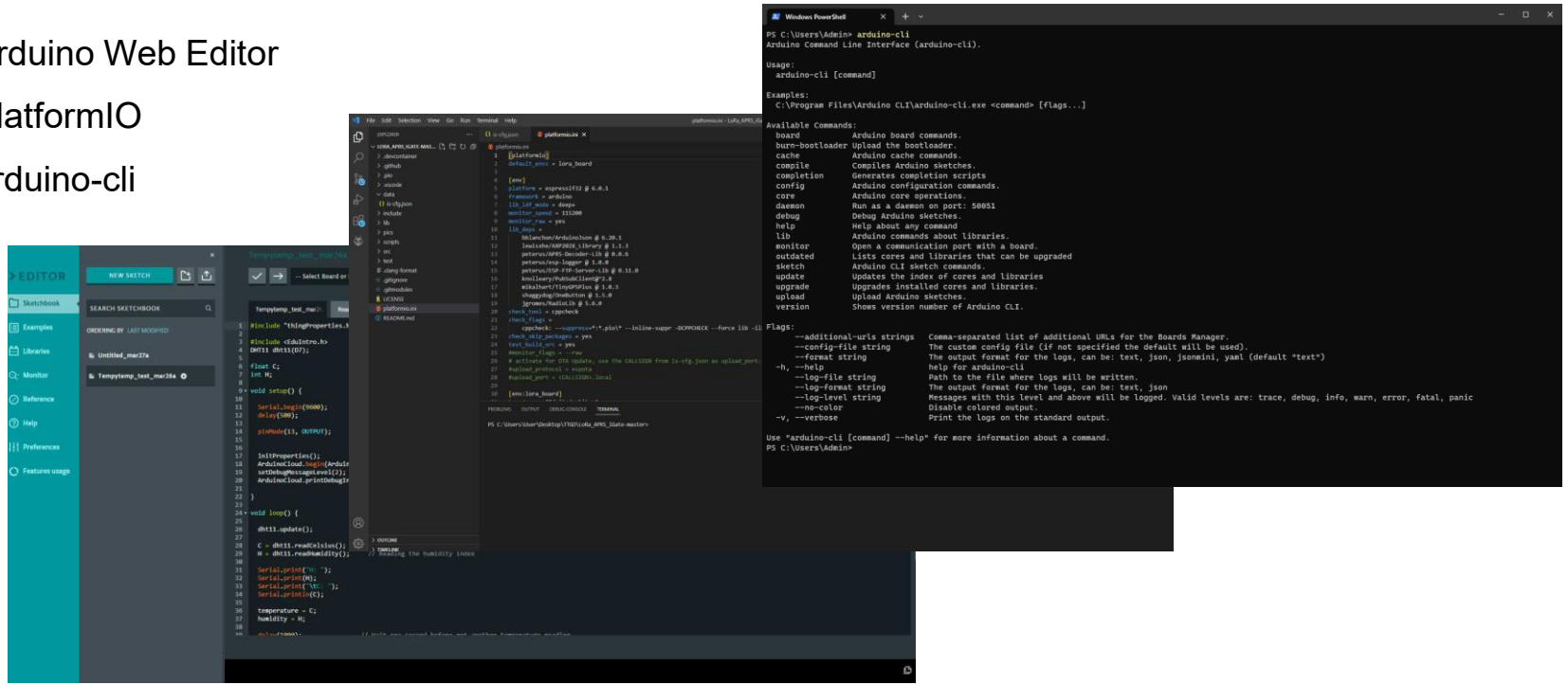
Ethernet!

Cannot run the cable?

Uno WIFI!

Other Arduino programming platforms

- Arduino Web Editor
- PlatformIO
- arduino-cli



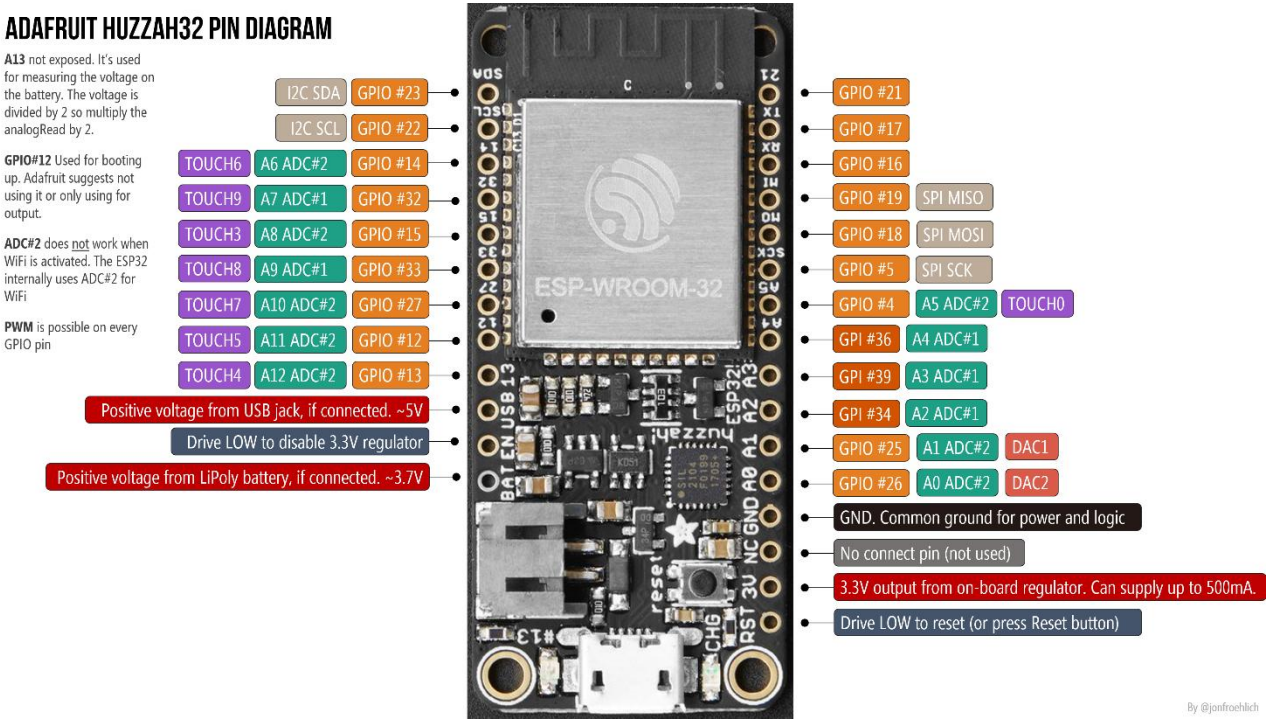
Arduino competitors

- Nowadays there are a lot of competitors with different specifications
- One major competitor is Espressif Systems
  - MCU boards:
    - ESP32
    - ESP8266

ESP32

- 32-bit Tensilica Xtensa LX6 processor-core MCU, 240 MHz
- 2/4 MB Flash, 448 KB ROM, 520 KB SRAM
- Wi-Fi + Bluetooth + BLE
- 34 GPIO pins, timers, 12-bit ADCs
- 3.3V operating voltage
- Arduino IDE support via ESP32 board core
- Good for projects needing both sensor I/O and networking
- $6 - $12 cost

ESP32 pin diagram



ESP8266

- 32-bit Tensilica L106 processor-core MCU, 80MHz
- 8 MB Flash, 16 KB ROM, 160 KB SRAM
- Wi-Fi
- 17 GPIO pins, 10-bit ADC: typically A0 only (range depends on board design)
- 3.3V operating voltage
- Arduino IDE support via ESP8266 board core
- Great for simple Wi-Fi sensor nodes
- $4 - $6 cost

Why use ESP32/ESP8266

- Low cost
- Low power consumption
- WiFi capabilities
- ESP32 adds Bluetooth (and BLE) support and more peripherals than Uno
- Arduino compatibility
- Robust design

Large ecosystem, still growing…