Arduino UNO board
Arduino UNO Features

- 14 digital I/O pins, of which 6 can be analog (PWM) output pins, and 6 analog input pins
- multiple serial ports (at the expense of digital I/O pins)
- 2 external interrupt pins
- 1KB EEPROM, easily accessible from code
- a very useful debugging LED
- USB-to-serial connection
- power from USB or external power supply
Write a program in the Arduino programming environment
1. Write a program in the Arduino programming environment
2. Connect the board to your computer via USB
1. Write a program in the Arduino programming environment
2. Connect the board to your computer via USB
3. Click the 'upload' button to send your program to the board.
Uploading Code to the Arduino

1. Write a program in the Arduino programming environment
2. connect the board to your computer via USB
3. click the 'upload' button to send your program to the board
4. after uploading, the code runs automatically!
1. Write a program in the Arduino programming environment
2. Connect the board to your computer via USB
3. Click the 'upload' button to send your program to the board
4. After uploading, the code runs automatically!
5. If power comes from an external power supply (like a 9V battery), you can disconnect the USB cable and the board keeps running your program.
Uploading Code to the Arduino

1. Write a program in the Arduino programming environment
2. Connect the board to your computer via USB
3. Click the ‘upload’ button to send your program to the board
4. After uploading, the code runs automatically!
5. If power comes from an external power supply (like a 9V battery), you can disconnect the USB cable and the board keeps running your program.
6. Even after all power was switched off, the board retains your program in memory. When you reconnect the power, it starts running automatically.
Uploading Code to the Arduino

1. Write a program in the Arduino programming environment.
2. Connect the board to your computer via USB.
3. Click the 'upload' button to send your program to the board.
4. After uploading, the code runs automatically!
5. If power comes from an external power supply (like a 9V battery), you can disconnect the USB cable and the board keeps running your program.
6. Even after all power was switched off, the board retains your program in memory. When you reconnect the power, it starts running automatically.
First steps

Check out the URL http://arduino.cc

Our board is the UNO model.
First steps

Check out the URL http://arduino.cc

Our board is the UNO model.

Download the IDE by clicking the Download button.
First steps

Check out the URL http://arduino.cc

Our board is the UNO model.

Download the IDE by clicking the Download button.

Read the page corresponding to the Learning button.
First steps

Check out the URL http://arduino.cc

Our board is the UNO model.

Download the IDE by clicking the Download button.

Read the page corresponding to the Learning button.

The language definition is available by clicking the Reference button.
First steps

Check out the URL http://arduino.cc

Our board is the UNO model.

Download the IDE by clicking the **Download** button.

Read the page corresponding to the **Learning** button.

The language definition is available by clicking the **Reference** button.
IDE – Integrated Development Environment

- Compile (verify program)
- Stop execution
- New
- Open
- Save
- Send program to board
- Display serial
- Code area
- Status area
All code is written around two important functions: `setup()` and `loop()`. When you place code inside these functions, it executes as follows:

- `setup()` is executed once, on Arduino power-up and reset, after this,
- `loop()` is repeatedly executed *ad infinitum*.

Example:

```c
int n;

void setup() {
    n = 28;
}

void loop() {
    n++;
    n = n % 256;
}
```
LEDs can be connected in any of the 14 digital I/O pins.

When in OUTPUT mode, digital pins can supply either 0V (LOW) or 5V (HIGH).

How to configure pin 7 as output? `pinMode(7, OUTPUT);`

How assign it a Low? `digitalWrite(7, LOW);`

How assign it a High? `digitalWrite(7, HIGH);`
Example program to blink LEDs

```cpp
int ledPin = 8; // LED connected to pin 8

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

void loop()
{
    digitalWrite(ledPin, HIGH); // set the LED on
    delay(1000); // wait for a second
    digitalWrite(ledPin, LOW); // set the LED off
    delay(1000); // wait for a second
}
```
How to connect a LED

NOTE: The LED short leg goes to ground
Digital Input

A digital input signal can be either 0V (LOW) or 5V (HIGH).

In order for a pin to be read, it must be configured as an INPUT.
Example – read a switch

```c
int switchPin = 8; // digital pin to attach the switch
int LED = 7;

void setup()
{
    pinMode(switchPin, INPUT); // pin 8 as input
    pinMode(LED, OUTPUT); // pin 7 as output
}

void loop()
{
    if (digitalRead(switchPin) == HIGH) // if the switch is pressed
        digitalWrite(LED, HIGH); // turns the LED ON
    else // if the switch is not pressed
        digitalWrite(LED, LOW); // turns the LED OFF
}
```
How to connect a switch
Analog input and output signals can have many values, not just 0V (LOW) and 5V (HIGH).

Voltage levels between 0 - 5V can be read via the analog input pins (A0 through A5 in the UNO board).

How to read an analogue voltage? `int v = analogRead(pin);`

The value of `v` will be between 0 and 1023 because the analog to digital converter inside the microcontroller has an output of 10 bits.
How to connect a switch

- Connect a potentiometer between +5V and GND.
- To analog input pin.

- Connect a photoresistor between GND and +5V.
- To analog input pin.

- Wiring diagram for Arduino.

- Use analog input PIN 0.
Analog Output through PWM

Analog output pins can generate voltage levels between 0 - 5V, using a method called Pulse Width Modulation.

PWM is basically a way of faking different voltages by very quickly alternating between 0V and 5V – the longer the 5V spikes take, the higher the output voltage appears. So PWM does not generate pure analog waves.

How to generate a PWM: `analogWrite(pin, value);`

After a call to `analogWrite()`, the pin (pin) will generate a steady square wave of the specified duty cycle (value) until the next call to `analogWrite()`. The duty cycle value varies from 0 (always off, or 0%) to 255 (always on, or 100%). The frequency of the PWM signal is approximately 490 Hz.
On most Arduino boards (those with the ATmega168 or ATmega328, the case of UNO), this function works on pins 3, 5, 6, 9, 10, and 11.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

The analogWrite function has nothing to do with the analog pins or the analogRead function.
You may be interested in consulting the `analogReference(type)` function, which sets the reference voltage used for analog input (i.e. the value used as the top of the input range).
Serial Communication

The board can communicate with other devices via the RS232 protocol, e.g., a computer.

Serial communication can act in two directions simultaneously – duplex or full-duplex.

RS232 is emulated over USB.

Most computers don’t have a RS232 port. So to use this type of communication, you will need an USB-to-serial adapter.
Communication is performed through pins 0 (RX) and 1 (TX), thus they cannot be used for digital input or output.

The Arduino IDE has a built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to `begin()`.

To communicate with an external TTL serial device, connect the TX pin to your device’s RX pin, the RX to your device’s TX pin, and connect the grounds of both devices.

Don’t connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage the Arduino board.
```cpp
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available() > 0) {
        int p = Serial.read();
        Serial.write(p);
    }
}
```
print()

Prints data to the serial port as human-readable ASCII text.

Numbers are printed using an ASCII character for each digit.

Floats are printed as ASCII digits, defaulting to two decimal places.

Bytes are sent as a single character.

Characters and strings are sent as is.
Examples

Serial.print(78) gives “78”
Serial.print(1.23456) gives “1.23”
Serial.print(’N’) gives “N”
Serial.print("Hello world.") gives “Hello world.”
There is an optional second parameter, which specifies the base (format) to use. Examples:

Serial.print(78, BIN) gives “1001110”

Serial.print(78, OCT) gives “116”

Serial.print(78, DEC) gives “78”

Serial.print(78, HEX) gives “4E”

Serial.println(1.23456, 0) gives “1”

Serial.println(1.23456, 2) gives “1.23”

Serial.println(1.23456, 4) gives “1.2346”

To send a single byte, use Serial.write().
available() 

Gets the number of bytes (characters) available for reading from the serial port.

The serial receive buffer holds a maximum of 64 bytes.

The function returns the number of bytes available to read.
Interrupts

interrupts() re-enables interrupts (after they’ve been disabled by noInterrupts()).

Interrupts allow certain important tasks to happen in the background and are enabled by default.

Some functions will not work while interrupts are disabled, and incoming communication may be ignored.

Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.
Example

```c
void setup() {}

void loop()
{
    noInterrupts();
    // critical, time-sensitive code here
    interrupts();
    // other code here
}
```
Using External Interrupts

attachInterrupt() specifies a function to call when an external interrupt occurs.

The UNO board has two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3).

Syntax: attachInterrupt(interrupt, function, mode), where interrupt represents the number of the interrupt, function represents the function to call when the interrupt occurs (this function must take no parameters and return nothing), and mode represents when the interrupt should be triggered.

Four constants are 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; and FALLING for when the pin goes from high to low.
int pin = 13;
volatile int state = LOW;
void setup()
{
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE);
}
void loop()
{
    digitalWrite(pin, state);
}
void blink()
{
    state = !state;
}
EEPROM (Electrically Erasable Programmable Read-Only Memory) is memory that retains the values stored in it when power is disconnected.

When you upload code into the board, the values in the EEPROM are not changed.

Arduino typically has 512-1024 bytes of EEPROM memory available to store values in.
Example

```c
#include <EEPROM.h> // import eeprom functions

byte b;

void setup() {
    Serial.begin(9600); // start serial communication
}

void loop() {
    b = EEPROM.read(28); // read byte value from eeprom address 28
    Serial.println(b, DEC); // print that value via the serial port
    b = (b+1) % 256; // cyclically increment the value
    EEPROM.write(28, b); // write new value to eeprom address 28
    delay(1000); // waste one second
}
```
millis() returns the number of milliseconds (unsigned long$^1$) since the Arduino board began running the program.

This number will overflow (go back to zero), after approximately 50 days.

delay() pauses the program for the amount of time (in milliseconds) specified as parameter.

$^1$0 to 4,294,967,295 ($2^{32} - 1$).
unsigned long startime;
unsigned int secs = 0;

void setup() {
  Serial.begin(9600);
  startime = millis();
}

void loop() {
  int timepassed = millis()-startime;
  if ( timepassed>1000 ) {
    secs++;
    Serial.print(secs);
    Serial.println(" segs");
    startime = millis();
  }
}
#include "TimerOne.h"

void setup() {
    Serial.begin(9600);
    pinMode(13, OUTPUT);
    Timer1.initialize(100000); // timer1’s period is 100msecs
    Timer1.attachInterrupt(callback);
}

void callback() {
    static int state=0;
    static int count=0;
    static unsigned long secs=0;
    if(++count==10) {
        Serial.print(++secs);
        Serial.println(" secs");
        digitalWrite(13,HIGH);
        count=0;
    }
    else
        digitalWrite(13,LOW);
}

void loop() {}
Interrupts with the serial port (not quite)

String inputString = "";  // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete

void setup() {
    Serial.begin(9600);
    inputString.reserve(200);
}

void loop() {
    // print the string when a newline arrives:
    if (stringComplete) {
        Serial.println(inputString);
        // clear the string:
        inputString = "";
        stringComplete = false;
    }
}
/* SerialEvent() occurs whenever a new data comes in the hardware serial RX. This routine is run between each time loop() runs, so using delay inside loop can delay response. */

void serialEvent()
{
while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
        stringComplete = true;
    }
}
}
Bibliography

http://mediatechnology.leiden.edu/openaccess/arduino
http://arduino.cc/