Microcontrollers

Programming Methods and the Arduino Uno

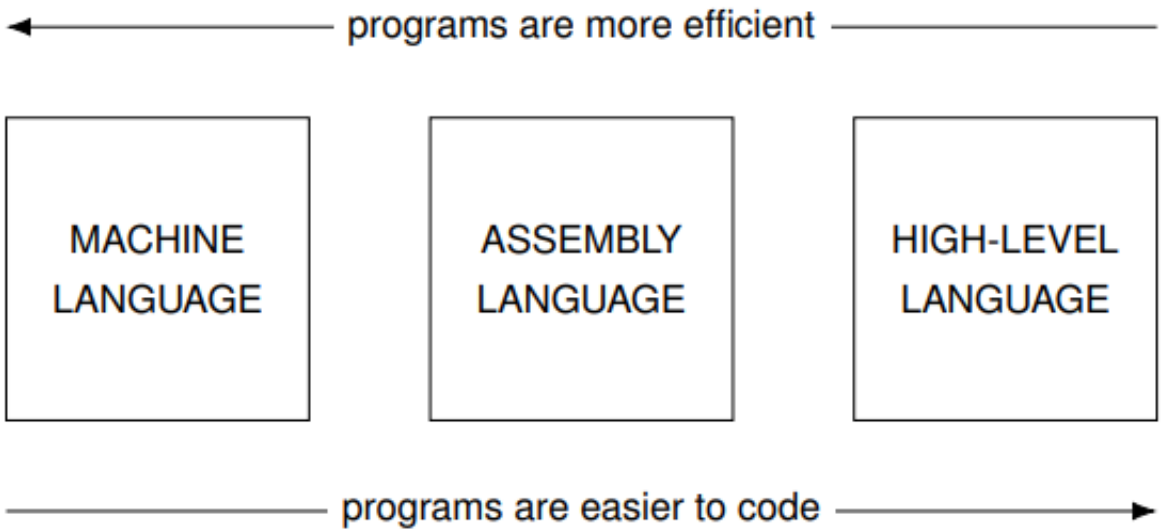
Developed and written by: Simon Hudson

Learning Objectives for this lesson

- Overview of programming methods.
- Introduction to the Arduino Uno microcontroller board.
- Installing and using software for programming the Arduino Uno.
- Uploading and executing simple Arduino Uno Sketches.
- Introduction to the “C” programming language.
- Writing, uploading and executing simple “C” programs developed in the Arduino IDE.

Programming methods – Machine Language

Programming of microcontrollers can be undertaken in 3 ways:



Using Machine Language:

This involves writing the actual Hex (or Binary) data as the instructions into the programming memory. It is very coding efficient but rarely used. It is confusing to read (and understand), difficult to troubleshoot for errors and is susceptible to the programmer making errors during coding.

Programming methods – Assembly Language

Using Assembly Language:

This involves using special assembly instructions, called ‘mnemonics’ as the instructions to be placed into the programming memory.

In our last lesson we looked at the 131 assembly instructions for the ATmega328 microcontroller.

A program called an **Assembler** converts the mnemonic instructions into the actual machine language code for placing in the programming memory of a specific microcontroller.

Assembly language programming is often used but it is not portable to other microcontrollers. This is a major limitation if you want to use your same program on a different microcontroller board – such as the Raspberry Pi Pico.

Programming methods – High Level Language

Using High Level Language:

This involves using a high-level programming language.

The high-level language instructions are much easier to understand and troubleshooting for any programming errors is a much easier task.

A program called a **Compiler** then converts the high-level instructions into the actual machine language code targeted to that specific microcontroller.

High level language programming is often used as it is portable to other microcontrollers – all you need to do is recompile your high-level program using a compiler targeted at the new microcontroller.

For example, we can write a high-level program to flash led (“Blink”). This same program can be used on both an Arduino Uno and a Raspberry Pi Pico. We just need to make sure we compile the program with the correct compiler.

Assembly Language versus High Level Language

Assembly Language:

- More efficient in coding
- Executes faster on the microcontroller
- Can be difficult to understand and troubleshoot for errors
- Not portable to other microcontrollers

High Level Language:

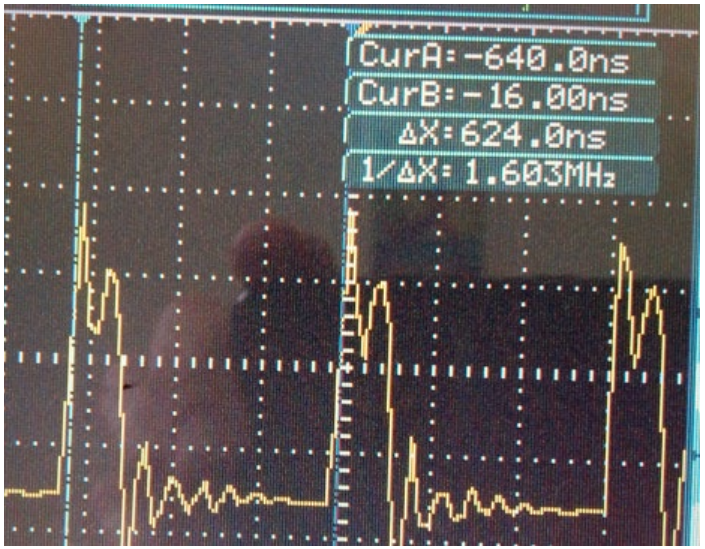
- Executes slower on the microcontroller
- Less efficient in coding
- Much easier to understand and troubleshoot for errors
- Portable to other microcontrollers

Execution speed – Assembler versus C high-level language

Here are two images showing the fastest speed at which the Arduino Uno can flash the on-board Led.

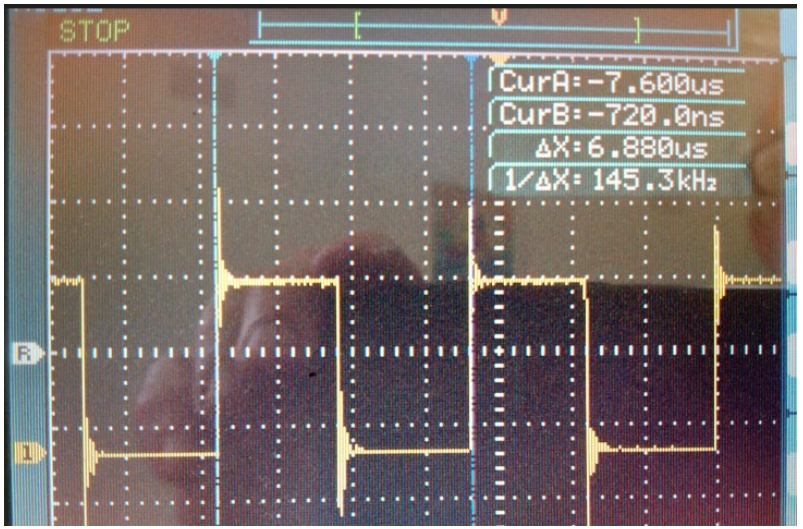
Assembler version

(cycle time = 624 ns)



High-level version

(cycle time = 6.9 us)



Assembler is 11 x faster

Code readability – Assembler version (fastest Blink rate)

```
1   /* Assembler version to Blink Led
2    * at the fastest possible rate
3   */
4
5   void setup() {
6     asm volatile (
7       "sbi %0, %1 \n\t"           //pinMode(13, OUTPUT);
8       :: "I" (_SFR_IO_ADDR(DDRB)), "I" (DDB5)
9     );
10  }
11
12  void loop() {
13    asm volatile (
14      "sbi %0, %1 \n\t"           //LED on
15      "cbi %0, %1 \n\t"           //LED off
16      "rjmp 4f \n\t"              //exit
17
18    "4: \n\t"                     //exit
19
20      :: "I" (_SFR_IO_ADDR(PORTB)), "I" (PORTB5)
21      : "r18", "r20", "r21", "r24", "r25"
22
23       );
24  }
```

Code readability – C High-level Language version

Blink_Fastest_In_C.ino

```
1   /*
2    *   Turns the on-board LED ON then immediatley turns the LED off
3    *   (with no delays!)
4
5   */
6
7   void setup() {
8     // initialize digital pin LED_BUILTIN as an output.
9     pinMode(LED_BUILTIN, OUTPUT);
10  }
11
12  void loop() {
13    digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is ON)
14    digitalWrite(LED_BUILTIN, LOW);   // turn the LED off (LOW os OFF)
15    }
16
```

Flashing the Led at 1 second

- But what about if we want to flash (“Blink”) the on-board Led at a specific rate – say 1 second ON and 1 second OFF ?
- Now, the code becomes far more complex for Assembler versus C.
- First let us look at the C version…

Code readability – C Language version for 1 second ON

- The only change to the original code is the call to the routine **delay(1000).**

  (1000 milliseconds = 1 second delay)

```
/*
  Blink
  Turns an LED on for one second, then off for one second, repeatedly.

*/

// the setup function runs once when you press reset or power the board
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 (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

Code readability – Assembly Language version for 1 second ON

```
/* Assembler version to Blink Led
* at a rate of 1 second */

void setup() {
  asm volatile (
    "sbi %0, %1 \n\t"           //pinMode(13, OUTPUT);
    :: "I" (_SFR_IO_ADDR(DDRB)), "I" (DDB5)
  );
}
void loop() {
  asm volatile (
      "sbi %0, %1 \n\t"         //LED on
      "call OneSecondDelay \n\t" //delay
      "cbi %0, %1 \n\t"         //LED off
      "call OneSecondDelay \n\t" //delay
      "rjmp 4f \n\t"            //exit
   "OneSecondDelay: \n\t"
      "ldi r18, 0 \n\t"         //delay 1 second
      "ldi r20, 0 \n\t"
      "ldi r21, 0 \n\t"
    "1: ldi r24, lo8(400) \n\t"
      "ldi r25, hi8(400) \n\t"
    "2: sbiw r24, 1 \n\t"       //10x around this loop = 1ms
      "brne 2b \n\t"
      "inc r18 \n\t"
      "cpi r18, 10 \n\t"
      "brne 1b \n\t"
       "subi r20, 0xff \n\t"     //1000 x 1ms = 1 second
      "sbci r21, 0xff \n\t"
      "ldi r24, hi8(1000) \n\t"
      "cpi r20, lo8(1000) \n\t"
      "cpc r21, r24 \n\t"
      "breq 3f \n\t"
       "ldi r18, 0 \n\t"
      "rjmp 1b \n\t"
    "3: \n\t"
      "ret \n\t"
    "4: \n\t"                   //exit

      :: "I" (_SFR_IO_ADDR(PORTB)), "I" (PORTB5)
      : "r18", "r20", "r21", "r24", "r25"
    );
}
```

Programming methods – High Level Languages

There are many types of high-level programming languages. Here are some examples:

- Java
- C
- C++
- PHP
- Python
- JavaScript
- Perl
- Ruby
- Objective C

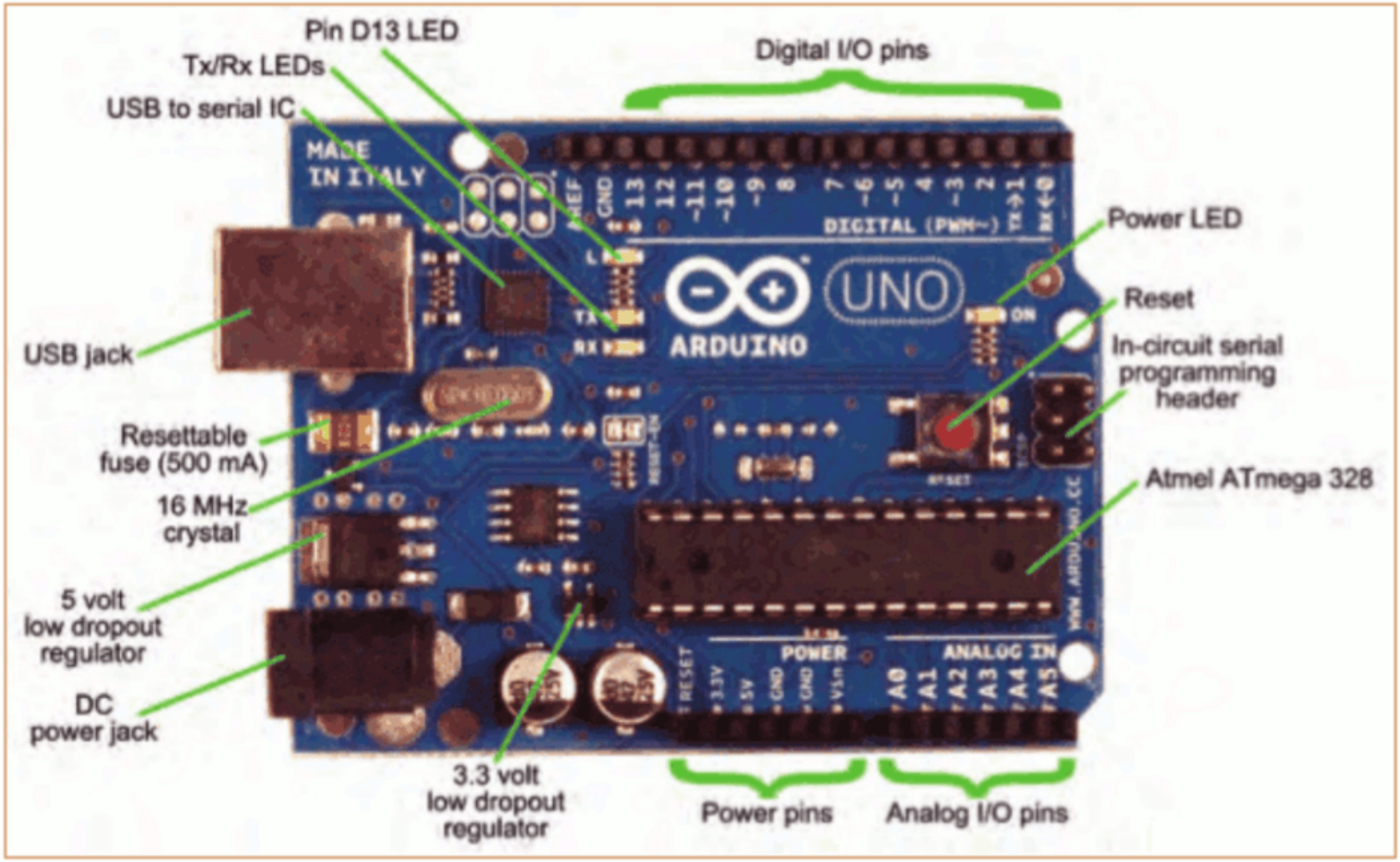
All have advantages and disadvantages. The most common high-level languages used in programming microcontrollers is C and Python.

In this course we will focus on using **C** to program the ATmega328P microcontroller that is installed on Arduino Uno microcontroller board.

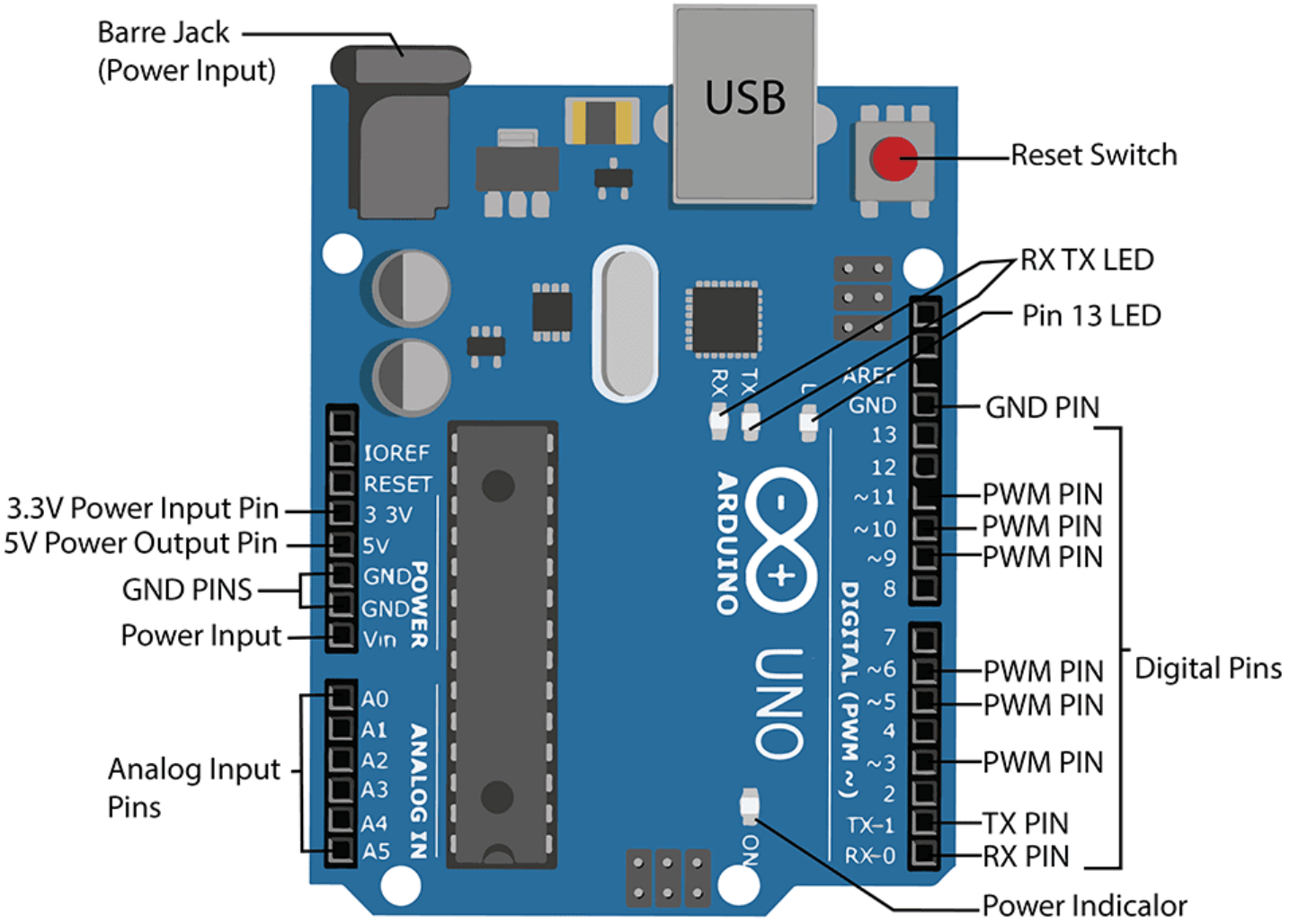
Introducing the Arduino Uno

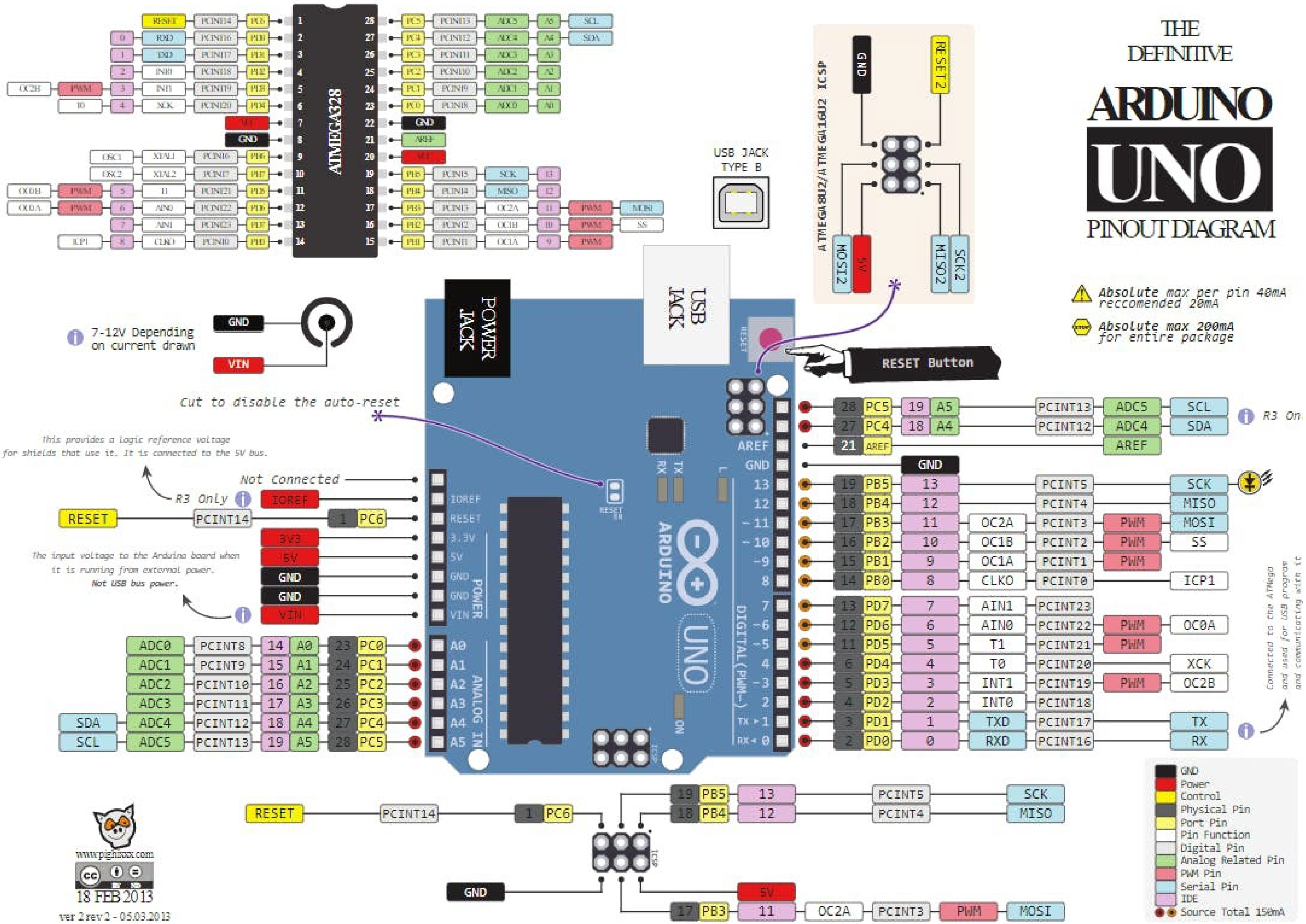
- The Arduino Uno is a microcontroller board based on the Atmel ATmega328P microcontroller chip – clocked at 16 MHz.
- The board provides all the features (peripherals) associated with ATmega328 microcontroller.
- Digital I/O (and PWM) pins and Analogue Input pins are available to interface to the outside world via header connectors.
- The board also contains a USB to serial IC to allow easy interface to a PC via a USB connector.
- Power is supplied to the Arduino Uno either via the USB connector or via a barrel DC power jack
- A power on Led and a dedicated on-board Led assigned to pin D13 is also included. The on-board Led is typically used in the “Blink” program.

Arduino Uno


Pin D13 LED
Tx/Rx LEDs
USB to serial IC
Digital I/O pins
Power LED
Reset
In-circuit serial programming header
USB jack
Resettable fuse (500 mA)
16 MHz crystal
Atmel ATmega 328
5 volt low dropout regulator
DC power jack
3.3 volt low dropout regulator
Power pins
Analog I/O pins
MADE IN ITALY
DIGITAL (PWM~)
ARDUINO
UNO
POWER
ANALOG IN

Arduino Uno


Barre Jack
(Power Input)
USB
Reset Switch
RX TX LED
Pin 13 LED
GND PIN
PWM PIN
PWM PIN
PWM PIN
PWM PIN
PWM PIN
PWM PIN
TX PIN
RX PIN
Digital Pins
3.3V Power Input Pin
5V Power Output Pin
GND PINS
Power Input
Analog Input
Pins
Power Indicalor
IOREF
RESET
3 3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
ARDUINO
UNO
DIGITAL (PWM ~)
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
TX
RX
L
ON


THE DEFINITIVE
ARDUINO
UNO
PINOUT DIAGRAM
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
7-12V Depending on current drawn
GND
VIN
Cut to disable the auto-reset
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
IOREF
RESET
PCINT14
PC6
3V3
5V
GND
VIN
The input voltage to the Arduino board when it is running from external power. Not USB bus power.
USB JACK TYPE B
ATMEGA328
ATMEGA16U2 ICSP
RESET2
MOSI2
MISO2
SCK2
RESET Button
POWER JACK
USB JACK
R3 On
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
SDA
SCL
PCINT8
PCINT9
PCINT10
PCINT11
PCINT12
PCINT13
AREF
SCK
MISO
MOSI
SS
OC2A
OC1B
OC1A
CLKO
ICP1
AIN1
AIN0
T1
T0
INT1
INT0
TXD
RXD
OC0A
XCK
OC2B
TX
RX
PWM
Connected to the ATmega and used for USB programming and communicating with it
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA
www.pighixxx.com
18 FEB 2013
ver 2 rev 2 - 05.03.2013

Software for programming the Arduino Uno

- The Arduino board uses a dedicated Integrated Development Environment (**IDE**) which can be downloaded from the Arduino website (www.arduino.cc).

- The IDE is based on the high-level language C (although there are sometimes some subtle differences to the C standard)

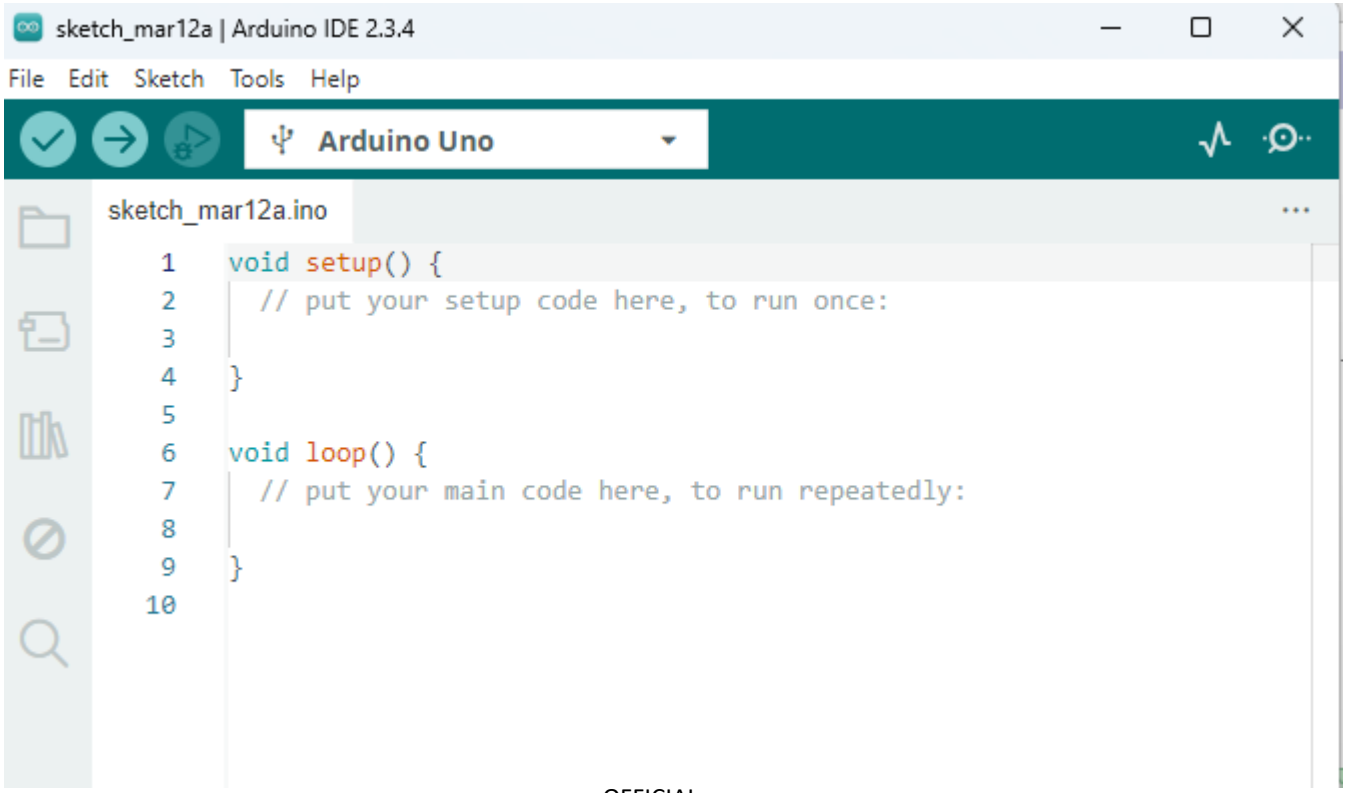
Click on this link to find the software:

Software | Arduino

- Let us now install the Arduino IDE on to our laptops….

The Arduino IDE

- When you first run the IDE you will get a screen view as shown below..
- Programs you write in the IDE are called “Sketches”.



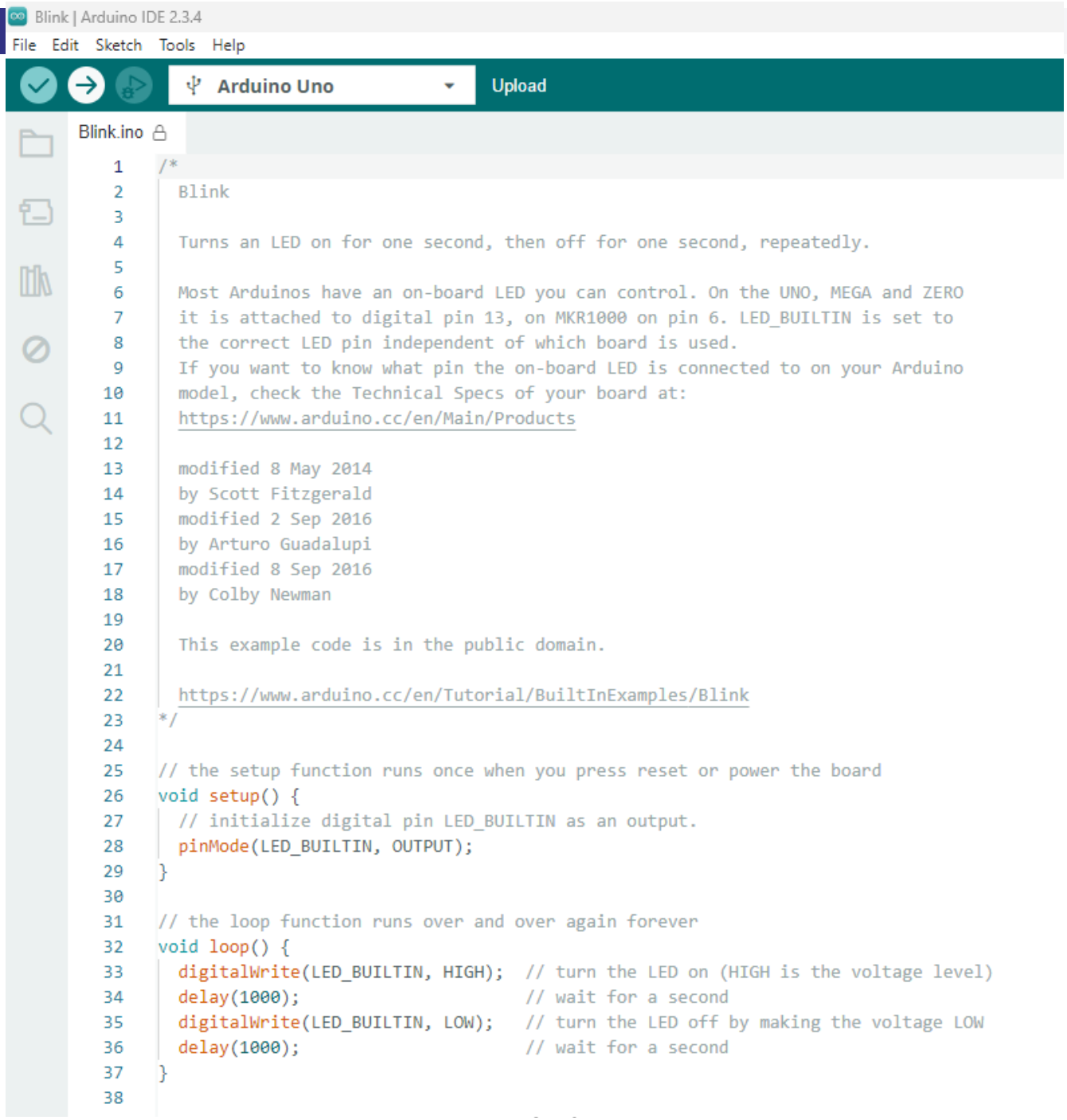
- **Verify / Upload** - compile and upload your code to your Arduino Board.
- **Select Board & Port** - detected Arduino boards automatically show up here, along with the port number.
- **Sketchbook** - here you will find all of your sketches locally stored on your computer. Additionally, you can sync with the Arduino Cloud, and also obtain your sketches from the online environment.
- **Boards Manager** - browse through Arduino & third party packages that can be installed. For example, using a MKR WiFi 1010 board requires the `Arduino SAMD Boards` package installed.
- **Library Manager** - browse through thousands of Arduino libraries, made by Arduino & its community.
- **Debugger** - test and debug programs in real time.
- **Search** - search for keywords in your code.
- **Open Serial Monitor** - opens the Serial Monitor tool, as a new tab in the console.

Student Activity 1 – Running “Blink”

- Start the Arduino IDE by clicking on the Arduino ICON
- Now do the following commands..

  File -> Examples -> 01.Basics -> Blink

- A new IDE page will open showing the Sketch associated with the program Blink


Blink | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Arduino Uno
Upload
Blink.ino
/*
Blink
Turns an LED on for one second, then off for one second, repeatedly.
Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
the correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your Arduino
model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products
modified 8 May 2014
by Scott Fitzgerald
modified 2 Sep 2016
by Arturo Guadalupi
modified 8 Sep 2016
by Colby Newman
This example code is in the public domain.
https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/
// the setup function runs once when you press reset or power the board
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 (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}

Verifying your Sketch

- To **verify** that the Sketch has no program errors click on the verify icon shown as: 
- The IDE will now Compile the program
- Assuming there are no errors, then a global variables message will now be displayed on a separate Output panel as shown:

```
26  void setup() {
27    // initialize digital pin LED_BUILTIN as an output.
28    pinMode(LED_BUILTIN, OUTPUT);
```

Output

```
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

Uploading your Sketch

- To **Upload** the Sketch program to the Arduino Uno board, click on the upload icon shown as: 
- If you have **NOT** connected the Uno to your PC or **not configured** the comms port correctly then you will get the following error message:

```
13  modified 8 May 2014
14  by Scott Fitzgerald
15  modified 2 Sep 2016
```

Output

```
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
avrdude: ser_open(): can't open device "\\.\COM6": The system cannot find the file specified.


Failed uploading: uploading error: exit status 1
```

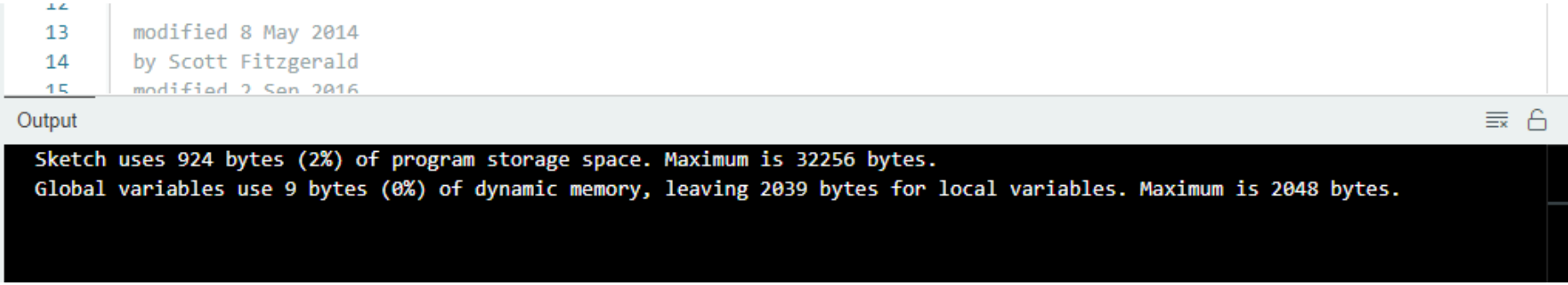
Upload error: Failed uploading: uploading error: exit status 1

COPY ERROR MESSAGES

Ln 1, Col 1 Arduino Uno on COM6 [not connected] 2

Uploading your Sketch

- Ok, let us now connect the Uno board to our PC and upload again.
- **Upload** the Sketch program to the Arduino Uno board by clicking on 
- You will now get no red error messages in the Output panel and your Arduino Uno board will start flashing the on-board Led.

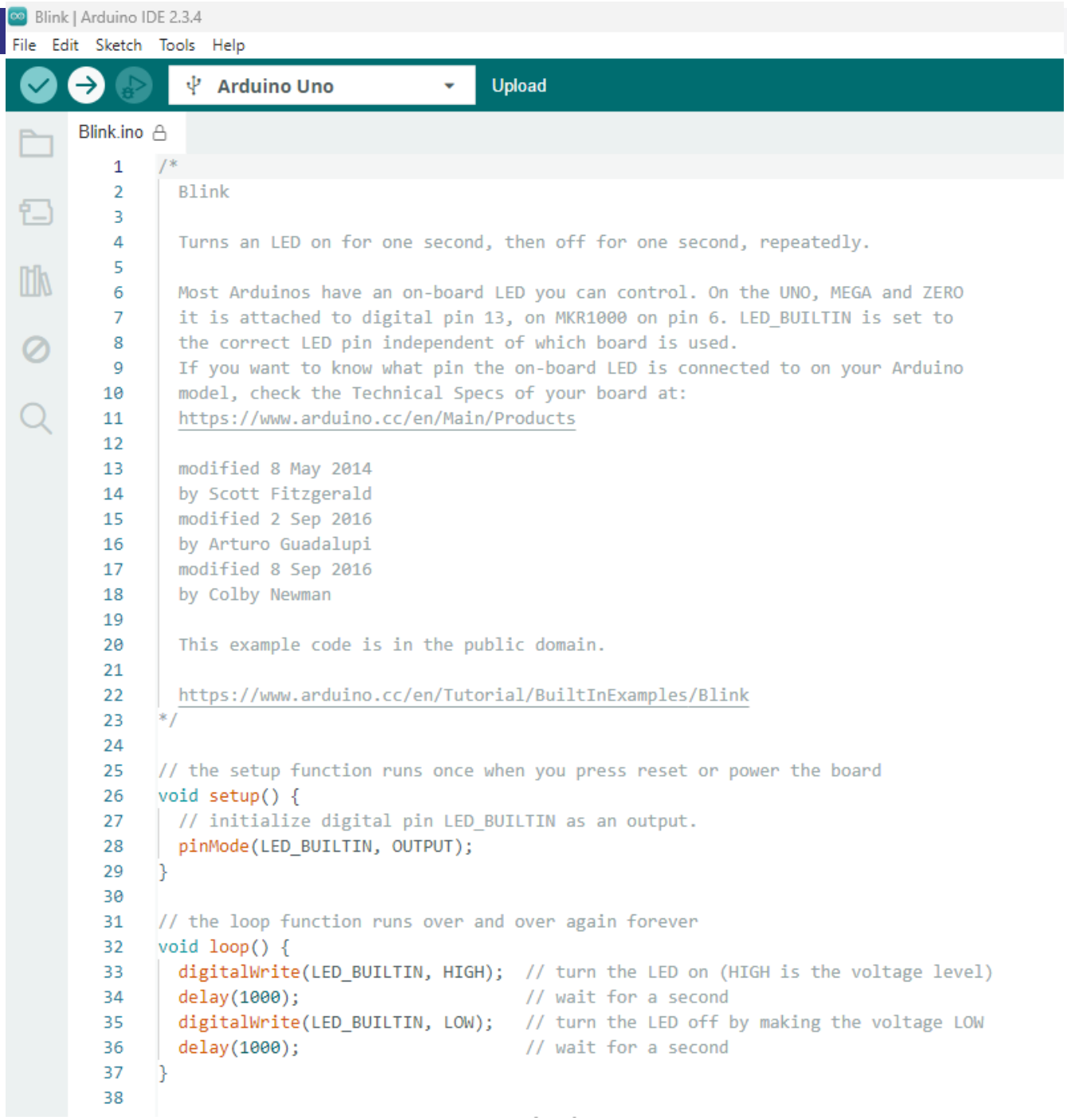


Student Activity 2 – Changing “Blink”

- Start the Arduino IDE by clicking on the Arduino ICON
- Now do the following commands..

  File -> Examples -> 01.Basics -> Blink

- A new IDE page will open showing the Sketch associated with the program Blink

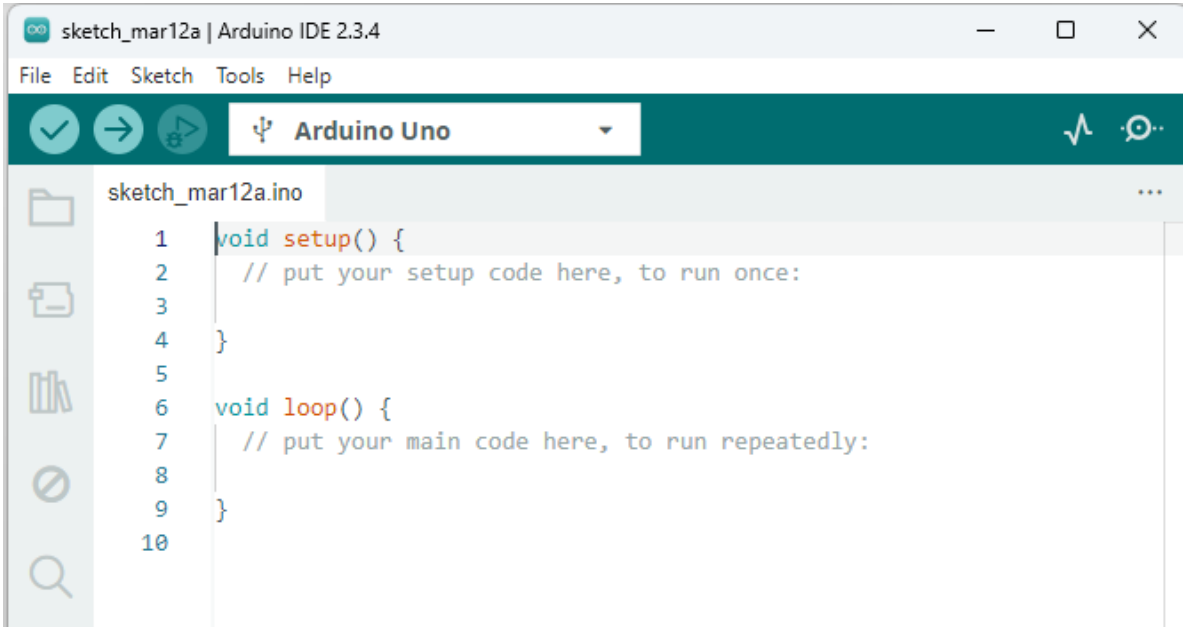

Blink | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Arduino Uno
Upload
Blink.ino
/*
Blink
Turns an LED on for one second, then off for one second, repeatedly.
Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
the correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your Arduino
model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products
modified 8 May 2014
by Scott Fitzgerald
modified 2 Sep 2016
by Arturo Guadalupi
modified 8 Sep 2016
by Colby Newman
This example code is in the public domain.
https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/
// the setup function runs once when you press reset or power the board
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 (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}

Changing “Blink”

- Now do these commands to create a new Sketch

  File -> New Sketch

- A new screen will open as shown:



- On this new screen save your Sketch to a new name using the commands: File -> Save As.. MyFirstSketch

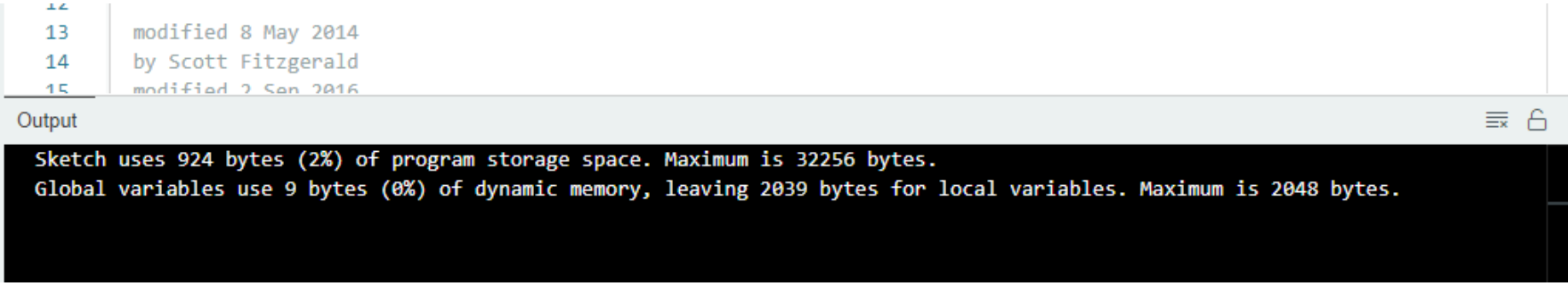
Changing “Blink”

- Use your mouse to select and copy all of the Sketch code in the Blink screen.
- Paste your selection into the Sketch called “MyFirstSketch”
- In your “MyFirstSketch” screen, now change both the “delay” code lines to read delay(100); as shown below:

```
// the loop function runs over and
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}
```

Verifying and Uploading your new Sketch

- **Verify** the new Sketch by clicking on. 
- **Upload** the new Sketch by clicking on. 
- You should get no red error messages in the Output panel and your Arduino Uno board will start flashing the on-board Led at a much faster rate.



Student Activity 3

- Change your MyFirstSketch to do the following..
  - First, perform two flashes of 1 second each.
  - Then perform two flashes of 0.3 seconds each.
  - Then repeat the process.

- Verify and Upload your new Sketch and see if it performs as expected.

Coding Hint:

In your sketch you can easily insert and copy lines of code and paste where needed.