

# Physical Computing

## Introduction

Owen Mundy | Spring 2012

# Overview

- Introduction
- Interaction design
- Physical computing
- Arduino
- Arduino vs. other platforms

# What is interaction design?

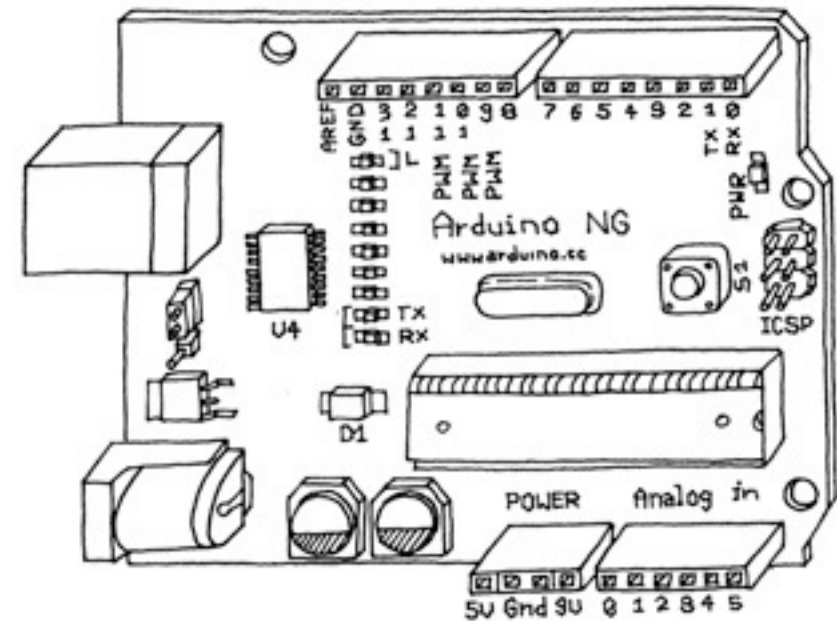
- Creating meaningful or useful experiences between humans and objects.
- Explore the beautiful, or even controversial, experiences between us and technology.
- It encourages art and design through an iterative process. Like conventional design, you develop and (re)develop prototypes to create outcomes. We will be prototyping with electronics.
- The specific field of Interaction Design involved with Arduino is Physical Computing (or Physical Interaction Design).

# What is physical computing?

- Designing interactive objects that can communicate with humans, using sensors and actuators, controlled by software running inside a microcontroller (a small computer on a single chip).
- Working with electronics used to be expensive and required an engineer.
- Thanks to the popularity of simple prototyping devices like the Arduino, creative folks have access electronics microcontrollers.
- With Arduino, a designer or artist can get to know the basics of electronics and sensors very quickly and can start building prototypes with very little investment.

# Arduino

- An open-source physical computing platform based on a simple input/output (I/O) board.
- Includes an open-source interface development environment (IDE) that implements the Processing [processing.org](http://processing.org) language and can be downloaded free from [arduino.cc](http://arduino.cc)
- Arduino can be used to develop stand-alone interactive objects or can be connected to software on your computer (e.g. Flash, Processing, Max/MSP).
- Boards can be assembled by hand or purchased preassemble



# Why choose Arduino over other platforms?

- It is multi-platform; it can run on Windows, Macintosh, and Linux.
- It is based on the Processing programming IDE, a free, easy-to-use development environment used by artists and designers.
- You program it via a USB cable, not a serial port. This feature is useful, because many modern computers don't have serial ports.
- It is open source hardware and software—if you wish, you can download the circuit diagram, buy all the components, and make your own, without paying anything to the makers of Arduino.

# Why choose Arduino over other platforms?

- The hardware is cheap. The USB board costs about \$35 and replacing a burnt-out chip on the board is easy and costs no more than \$4. So you can afford to make mistakes.
- There is an active community of users, so there are plenty of people who can help you.
- The Arduino Project was developed in an educational environment and is therefore great for newcomers to get things working quickly.

# Some philosophies for learning electronics

- Making designs rather than talk about them.
- Classic engineering relies on a strict process for getting from A to B; the Arduino Way delights in the possibility of getting lost on the way and finding C instead.
- Enjoy tinkering: Play with the medium in an open-ended way and allow yourself to find the unexpected.
- The next few sections present some philosophies, events, and pioneers that have inspired the Arduino Way.



# Creating your virtual sketchbook

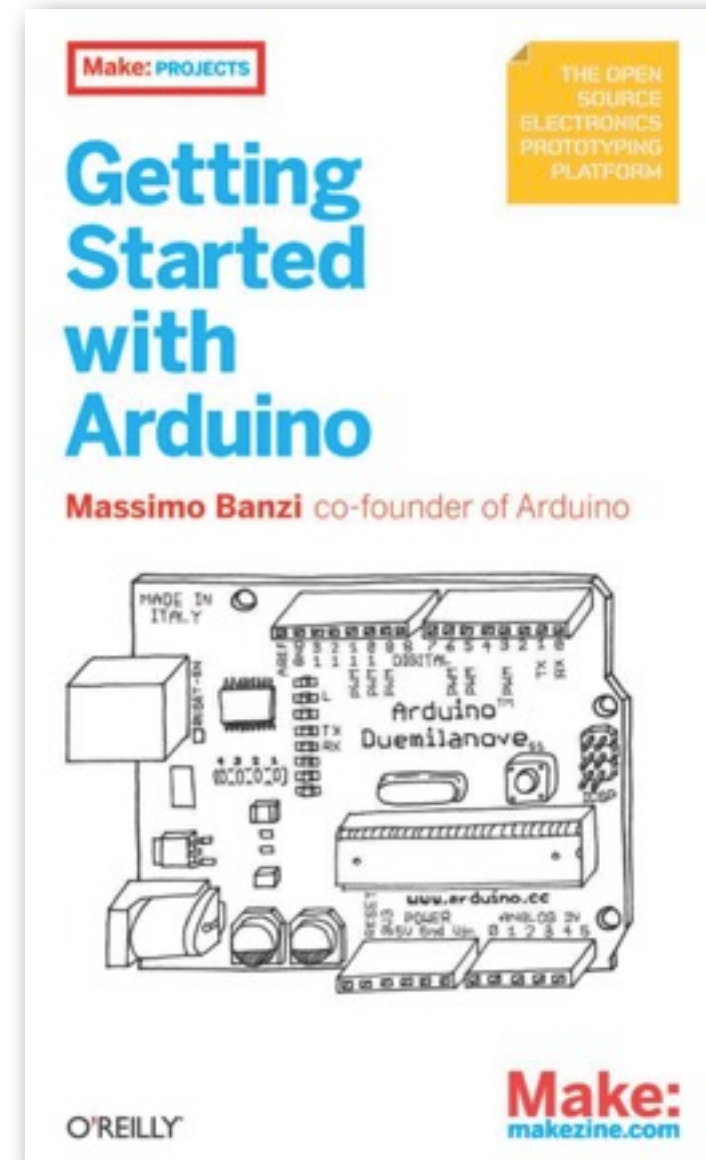
- [wordpress.com](http://wordpress.com)
- [blogger.com](http://blogger.com)
- HTML/CSS

# Purchasing parts

- In class demos require an Arduino compatible device + some basic electrical components such as:
  - LEDs
  - resistors
  - at least 3 different types of sensors
  - jumper wire
  - a push button
  - a USB A to: B cable
  - and a breadboard.
- Shopping for Arduinos is fairly easy. See [list of links](#) to devices on class site.
- Shopping for discreet electronic components is a pain. I recommend you start with a basic kit that includes these and an Arduino. See [list of starter kits](#) on class site.

# Credits

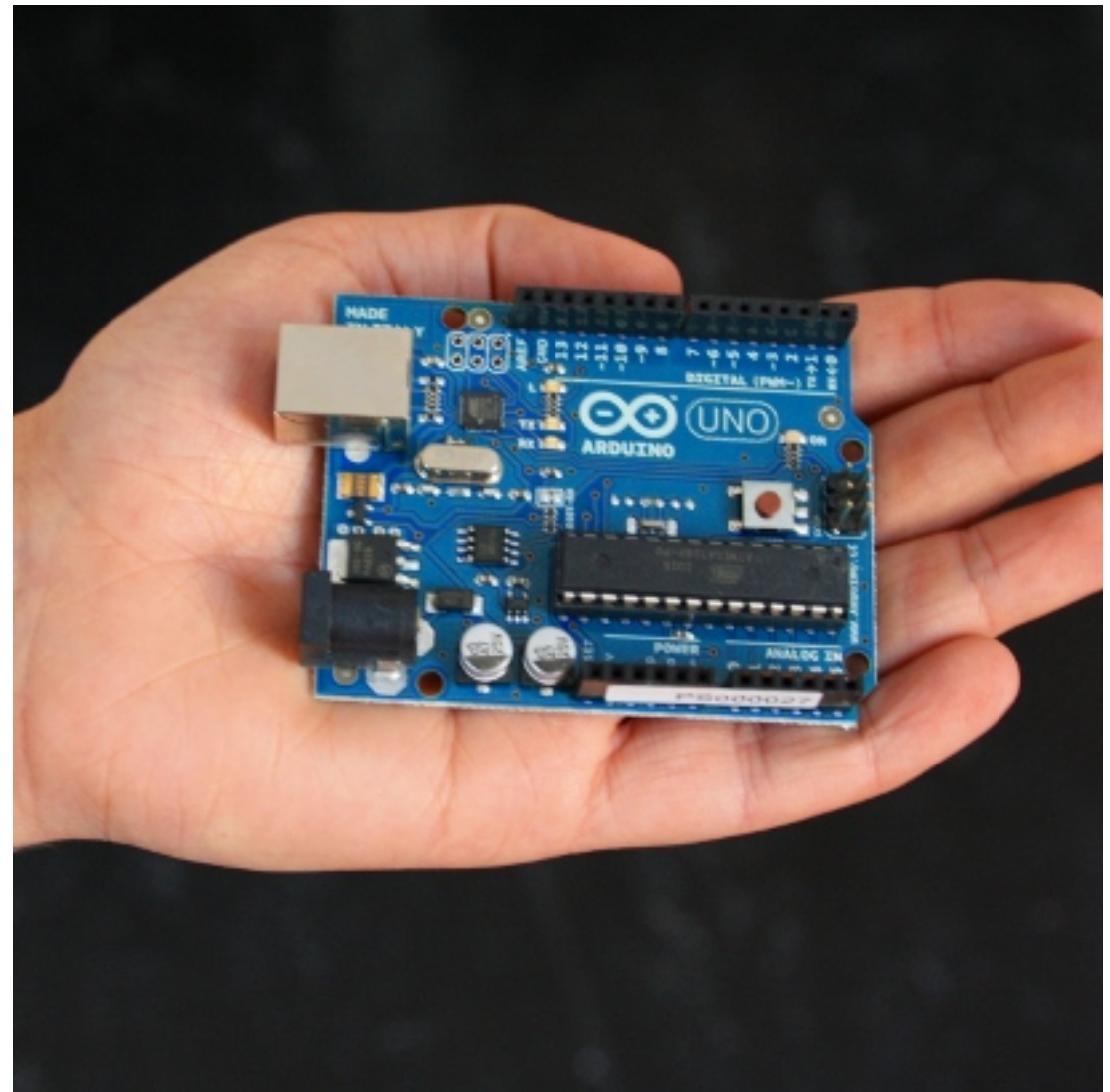
- Some content and images from this presentation come from, Getting Started with Arduino (First Edition) by Massimo Banzi (co-founder of Arduino) and published by O'Reilly and Make Magazine in 2009.



# Arduino 1

## Introduction + Blink!

Owen Mundy | Spring 2012



# Overview

- Overview of Arduino board (input/outputs, power, etc.)
- Powering your Arduino
- Powering components from the Arduino
- Intro to Arduino Interface Development Environment (IDE)
- Compiling and uploading programs to Arduino
- Blink tutorial

# Seeeduino v3.0

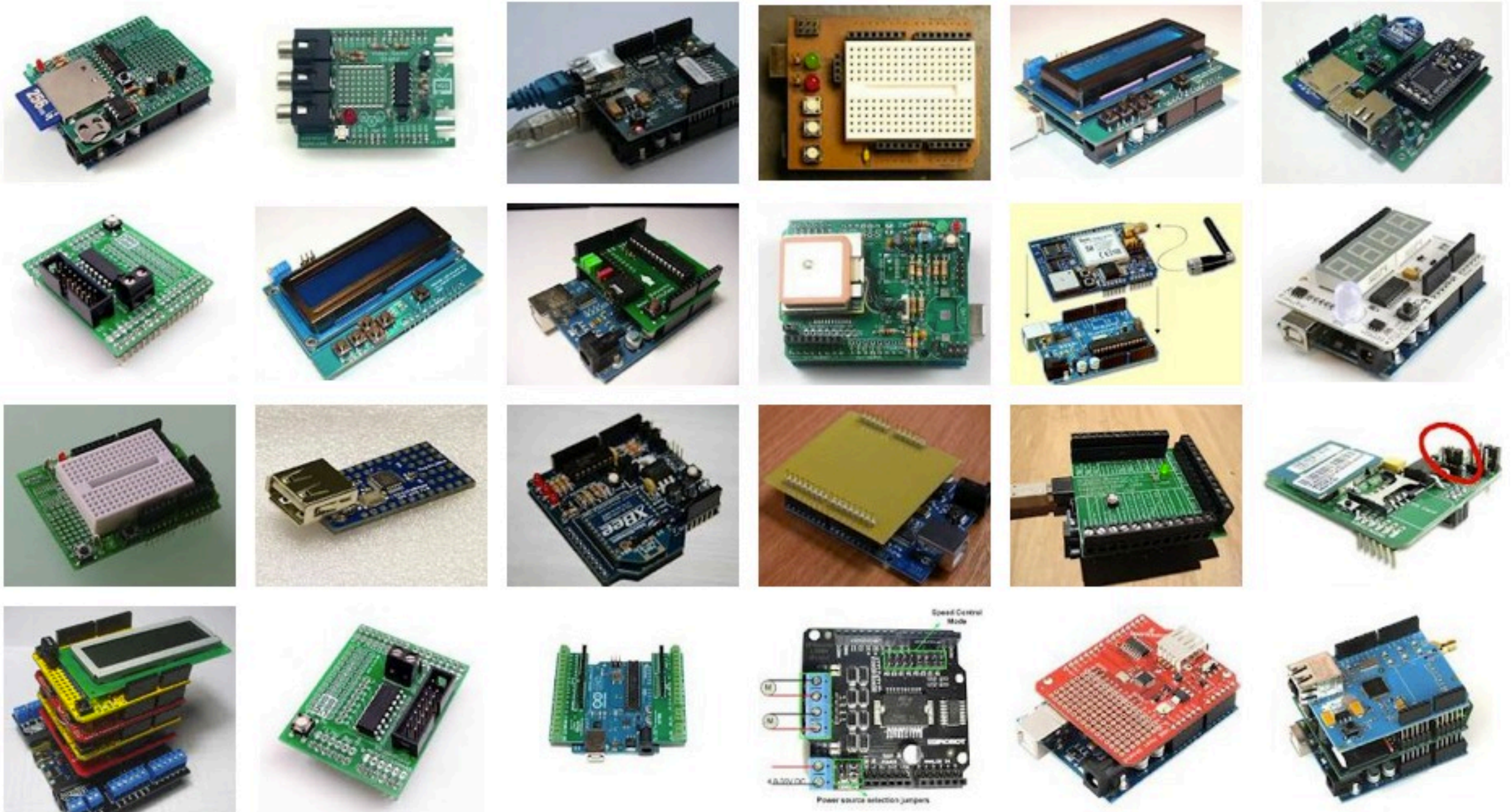
- In this class we're using the Seeeduino v3.0, an Arduino compatible clone created using Arduino's open source schematics.
- You can program it with the Arduino IDE and fit compatible shields onto it.





# Arduino shields

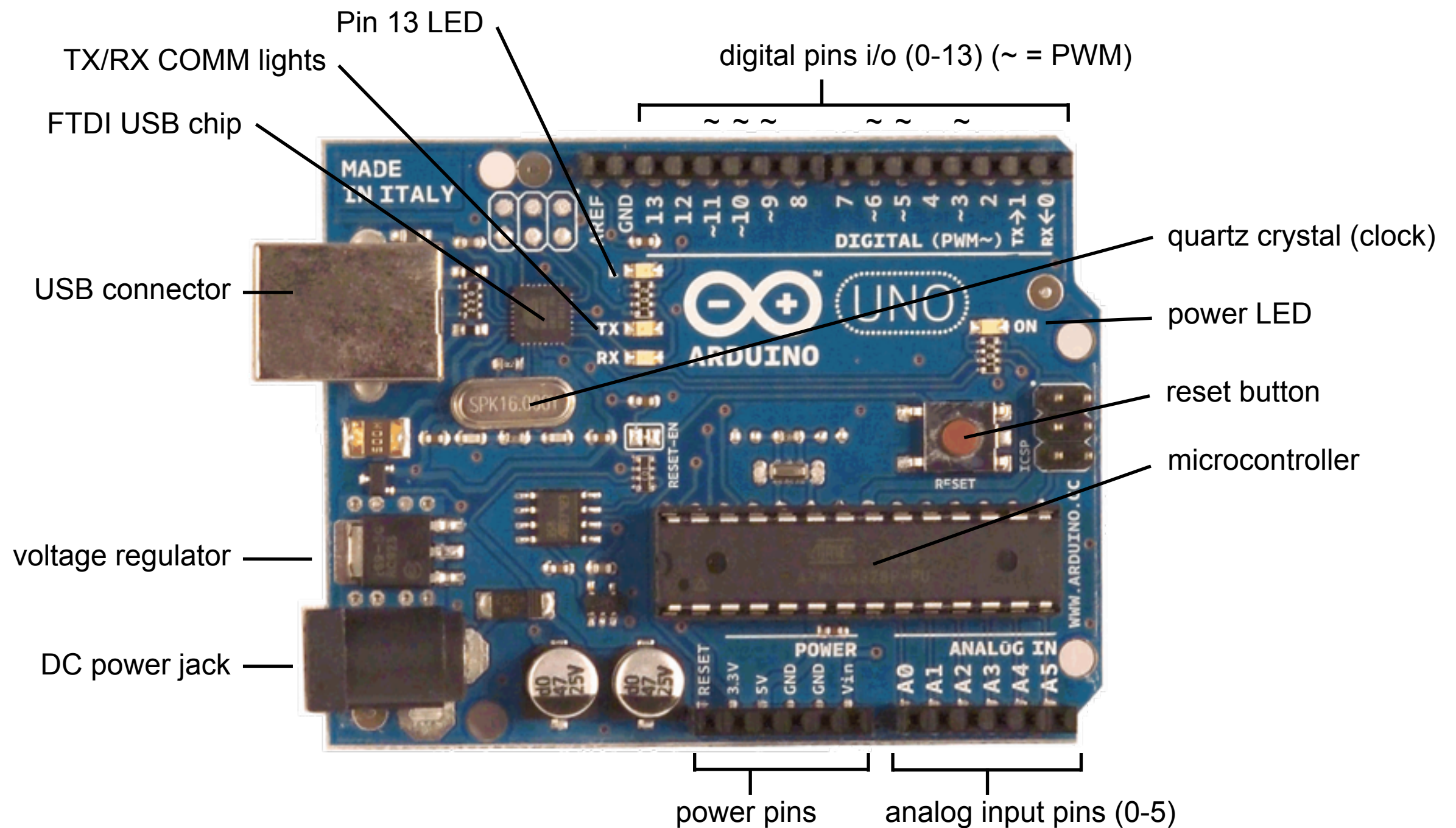
- Examples of Arduino shields: for plugging-in quickly to other devices like LCDs, SD cards, motors, ethernet, wireless, GPS, and more: [shieldlist.org](http://shieldlist.org)





# Arduino Uno

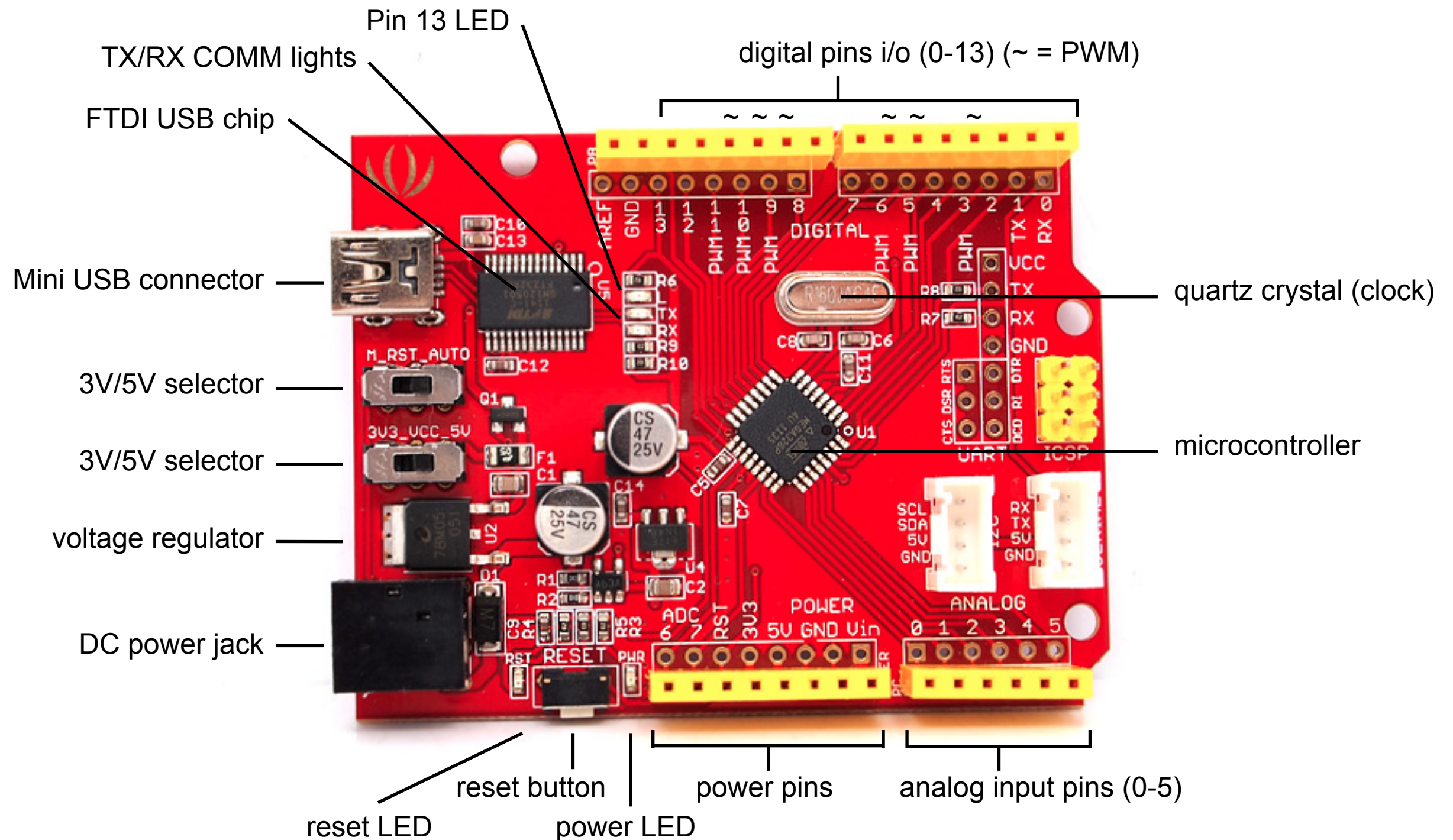
- The most recent popular version of the Arduino board.





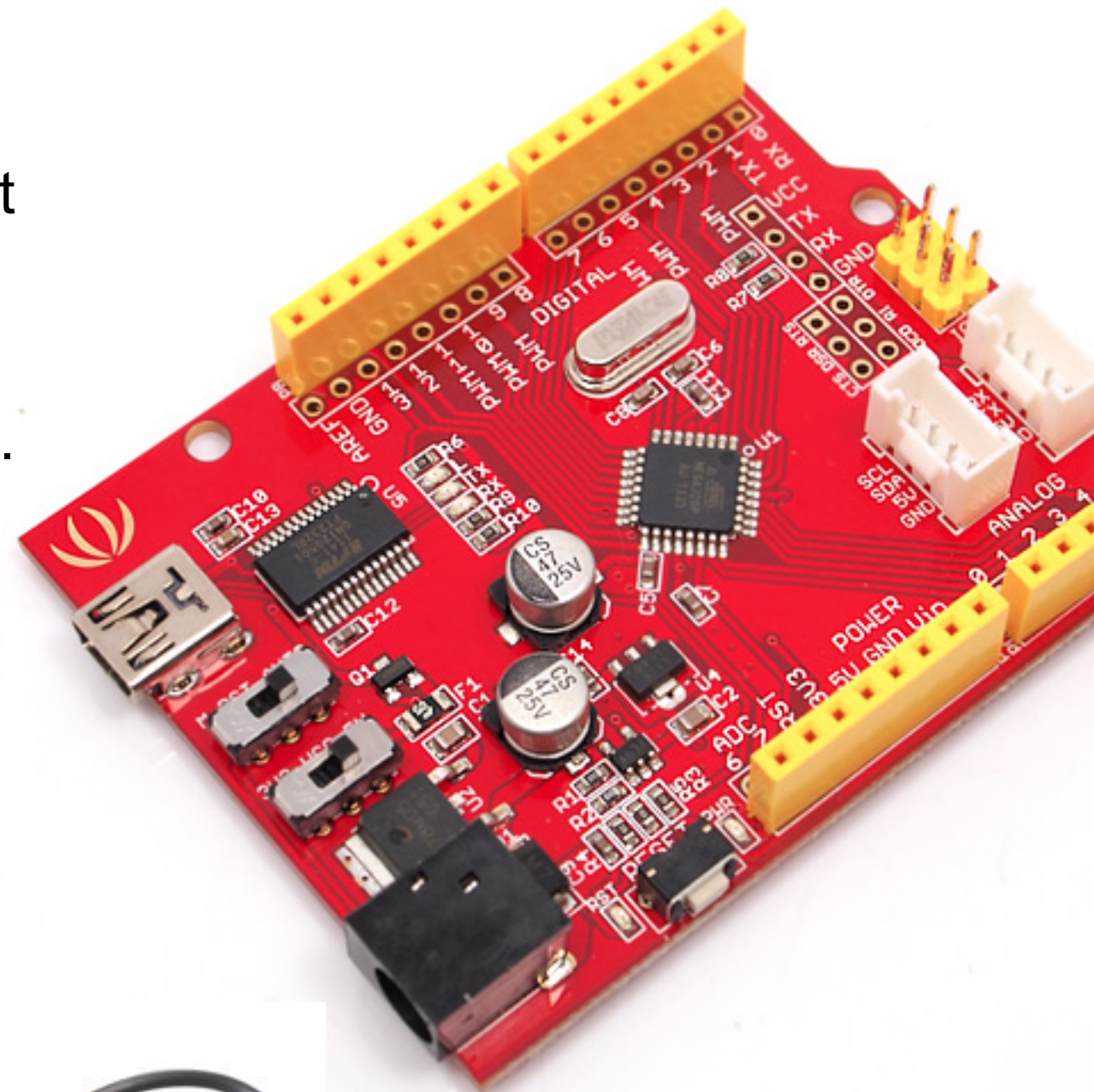
# Seeeduino v3.0

- Seeeduino is virtually identical to Arduino brand micros.



# Using the Seeeduino v3.0

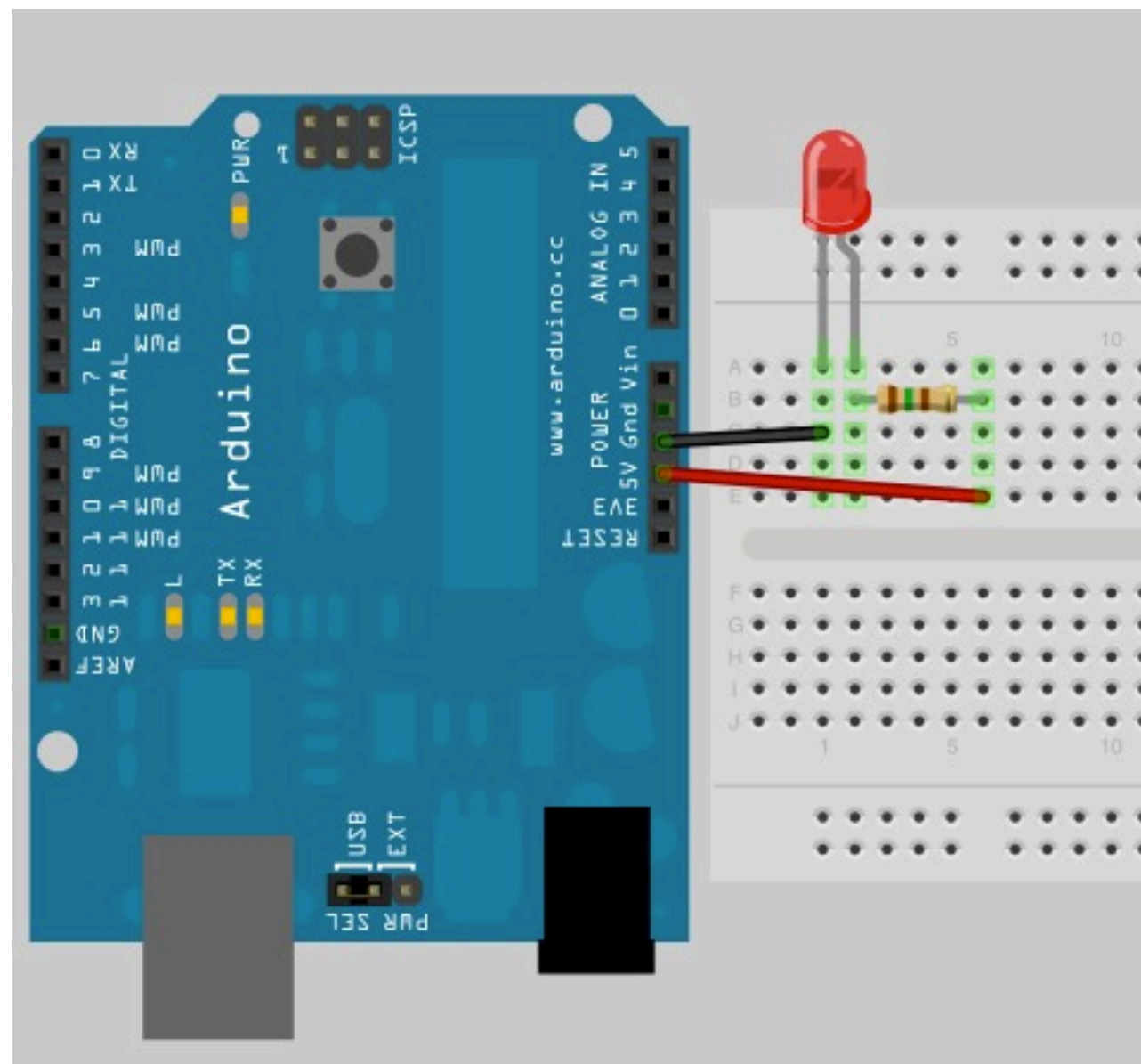
- Fairly rugged, but probably good to keep it out of water, away from dogs, etc.
- The 5V regulator means you can supply it with any voltage between 7V and 13V DC. The regulator will convert the V for you!
- For example, you can supply with with USB (5V) from your computer, or use the barrel jack with a 9V battery or an appropriate AC/DC converter (a.k.a wall wart). Be sure to check the rating on your wall wart before you plug it in!!
- Like all electronics, unplug everything ASAP if you smell burning.





# Arduino 5V+

- Let's create a simple LED circuit using the Arduino's 5V power (+) and Gnd (-).
- Mocking-up your circuit in Fritzing first can be very helpful. Its much easier to click+drag wires and other components from the library.



# Arduino 5V+

- The LED calculator tells us we need a 150 ohm resistor with our red LED (assuming the forward voltage is 2V and the forward current is 20mA).

**LED calculator: current limiting resistor value**

|    |                              |
|----|------------------------------|
| 5  | Source voltage ?             |
| 2  | diode forward voltage ?      |
| 20 | diode forward current (mA) ? |

Find R

The wizard recommends a 1/8W or greater 150 ohm resistor. The color code for 150 ohms is brown green brown.

5 V

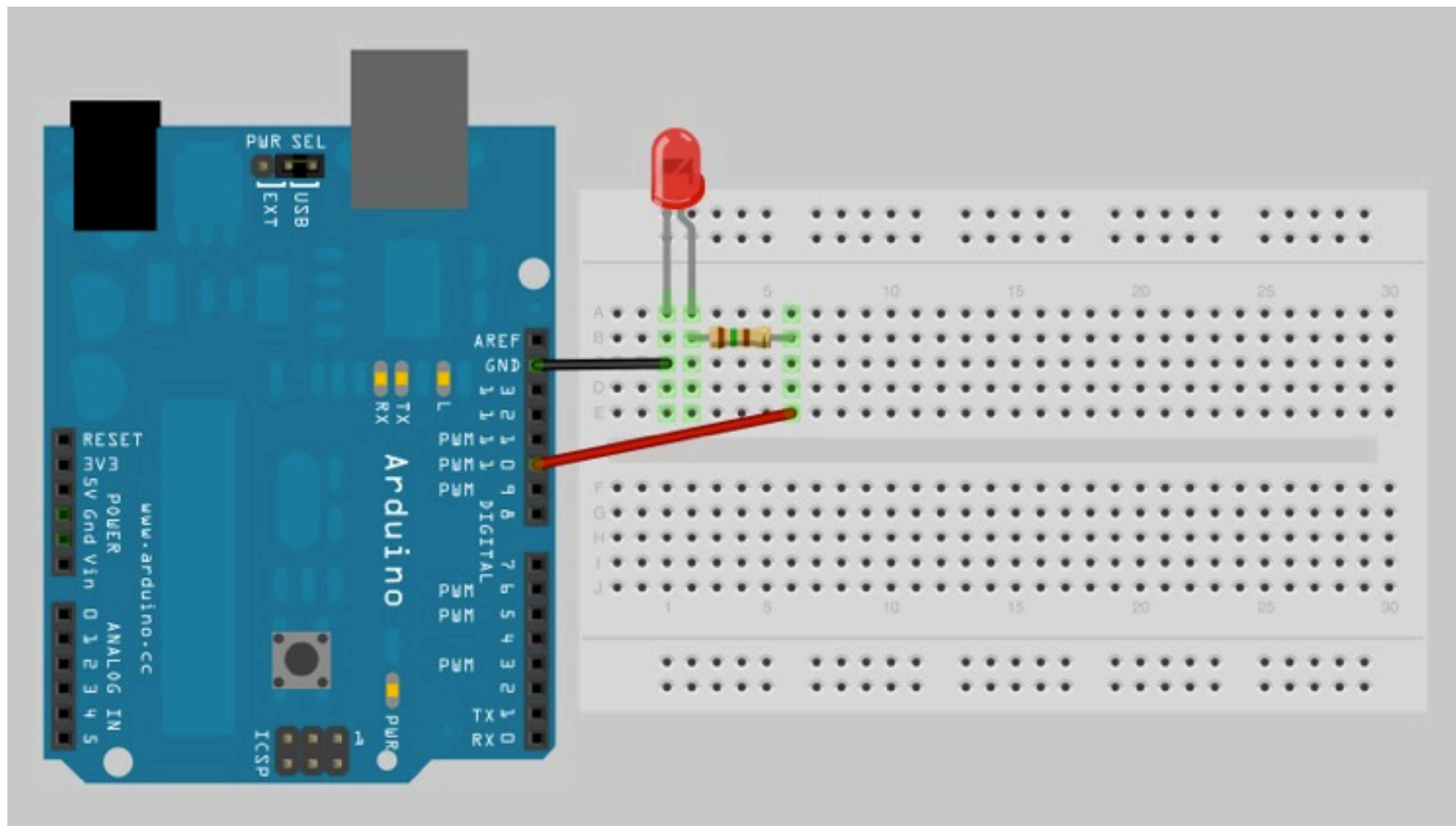
150 ohms, 1/8W

2V @ 20 mA

led.linear1.org

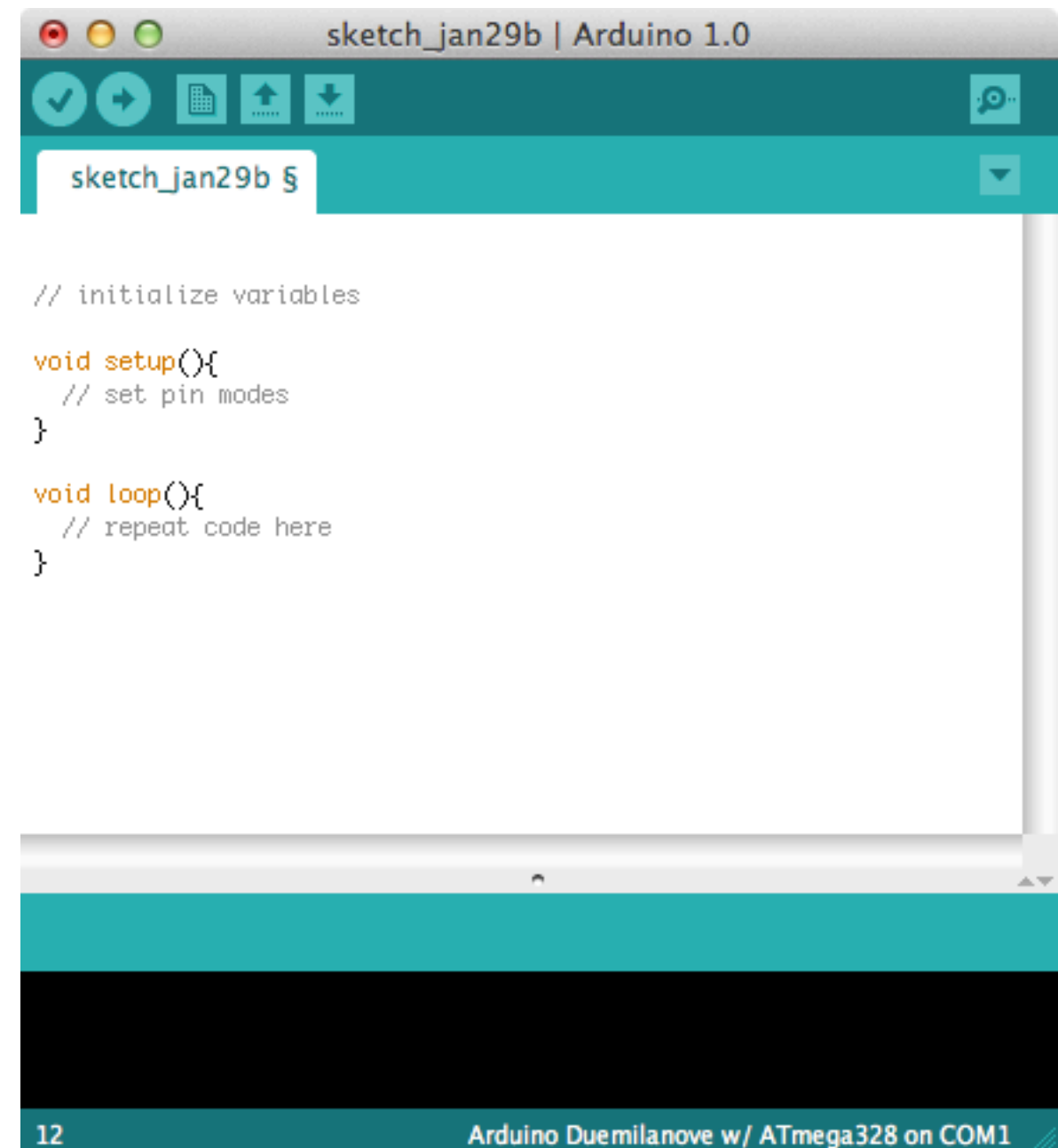
# Blink!

- Now let's write a program to blink the LED.
- Connect the black negative (-) wire to the Arduino's digital Gnd (on the other side of the board), and the red positive (+) wire to pin 10.
- Now connect the Arduino to your computer with your USB cable.



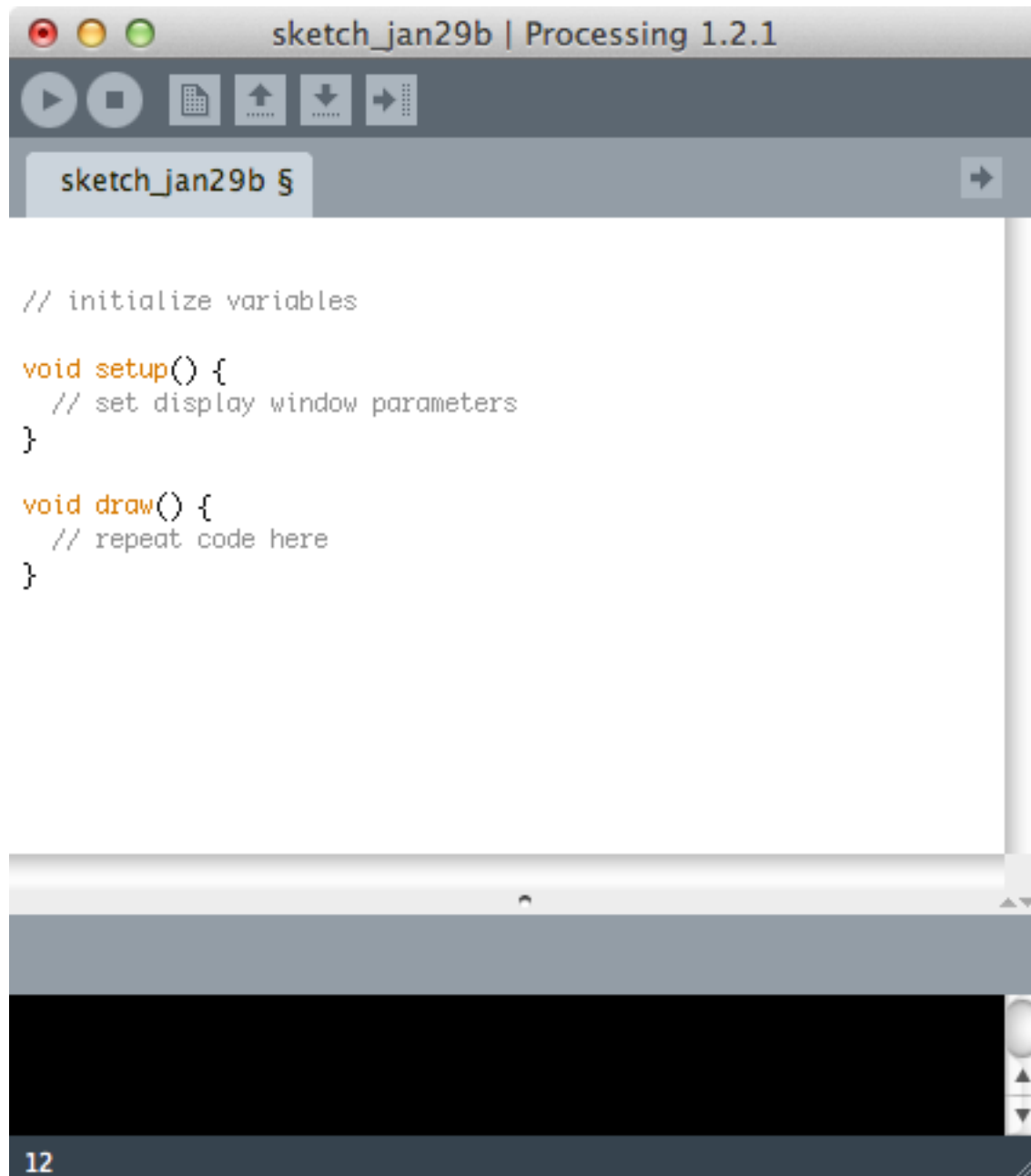
# The Arduino IDE

- Let's set the circuit aside and talk about the Arduino interface development environment (IDE).



# The Arduino IDE

- The Arduino IDE is based on the Processing IDE. Both are simple text editors and they use essentially the same interface...



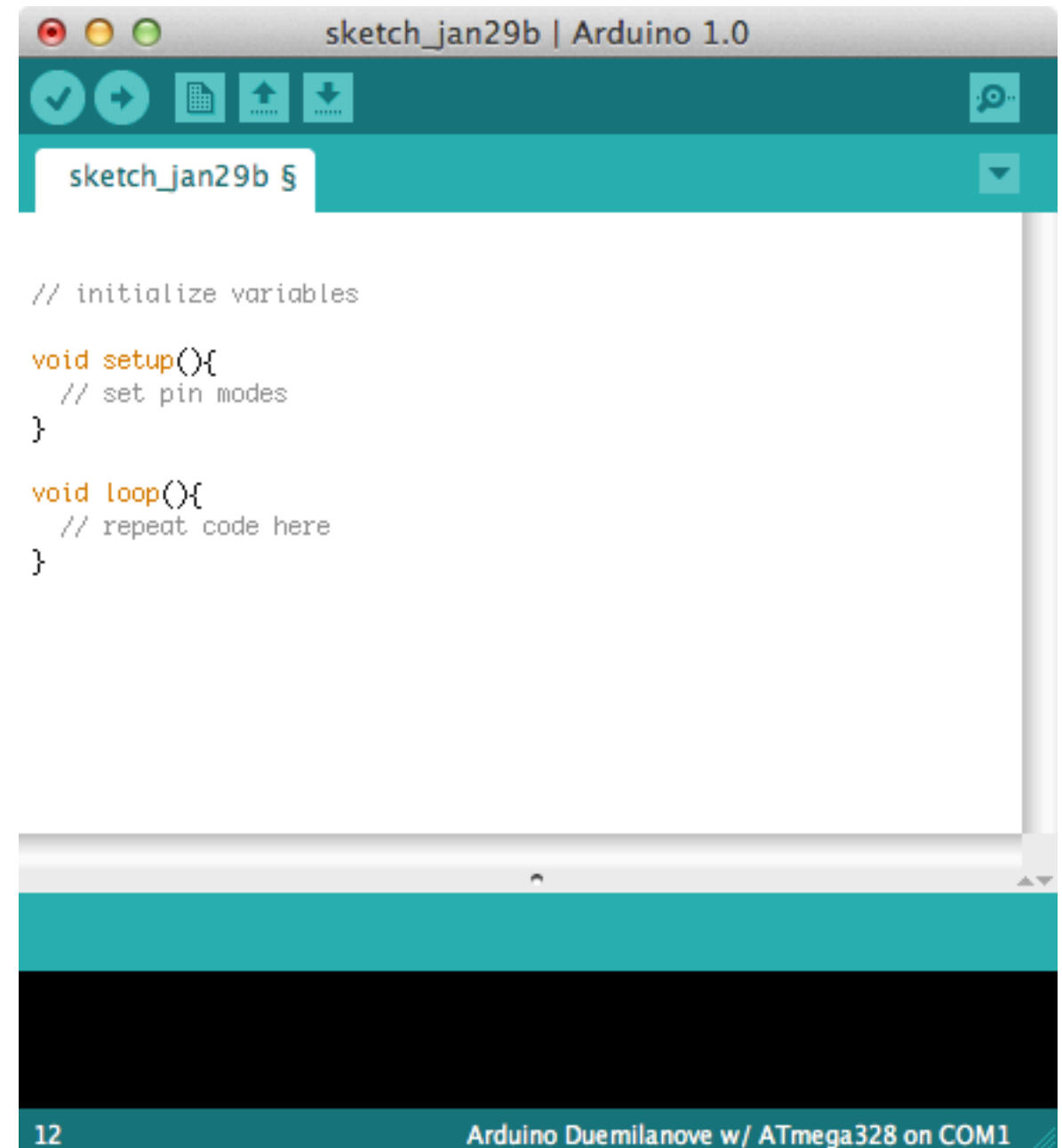
The screenshot shows the Processing IDE window titled "sketch\_jan29b | Processing 1.2.1". The interface has a grey title bar and a toolbar with icons for running, stopping, opening, saving, and zooming. Below the toolbar is a tab labeled "sketch\_jan29b §". The main text area contains the following code:

```
// initialize variables

void setup() {
  // set display window parameters
}

void draw() {
  // repeat code here
}
```

At the bottom, there is a status bar showing the number "12".



The screenshot shows the Arduino IDE window titled "sketch\_jan29b | Arduino 1.0". The interface has a teal title bar and a toolbar with icons for checking, running, opening, saving, and zooming. Below the toolbar is a tab labeled "sketch\_jan29b §". The main text area contains the following code:

```
// initialize variables

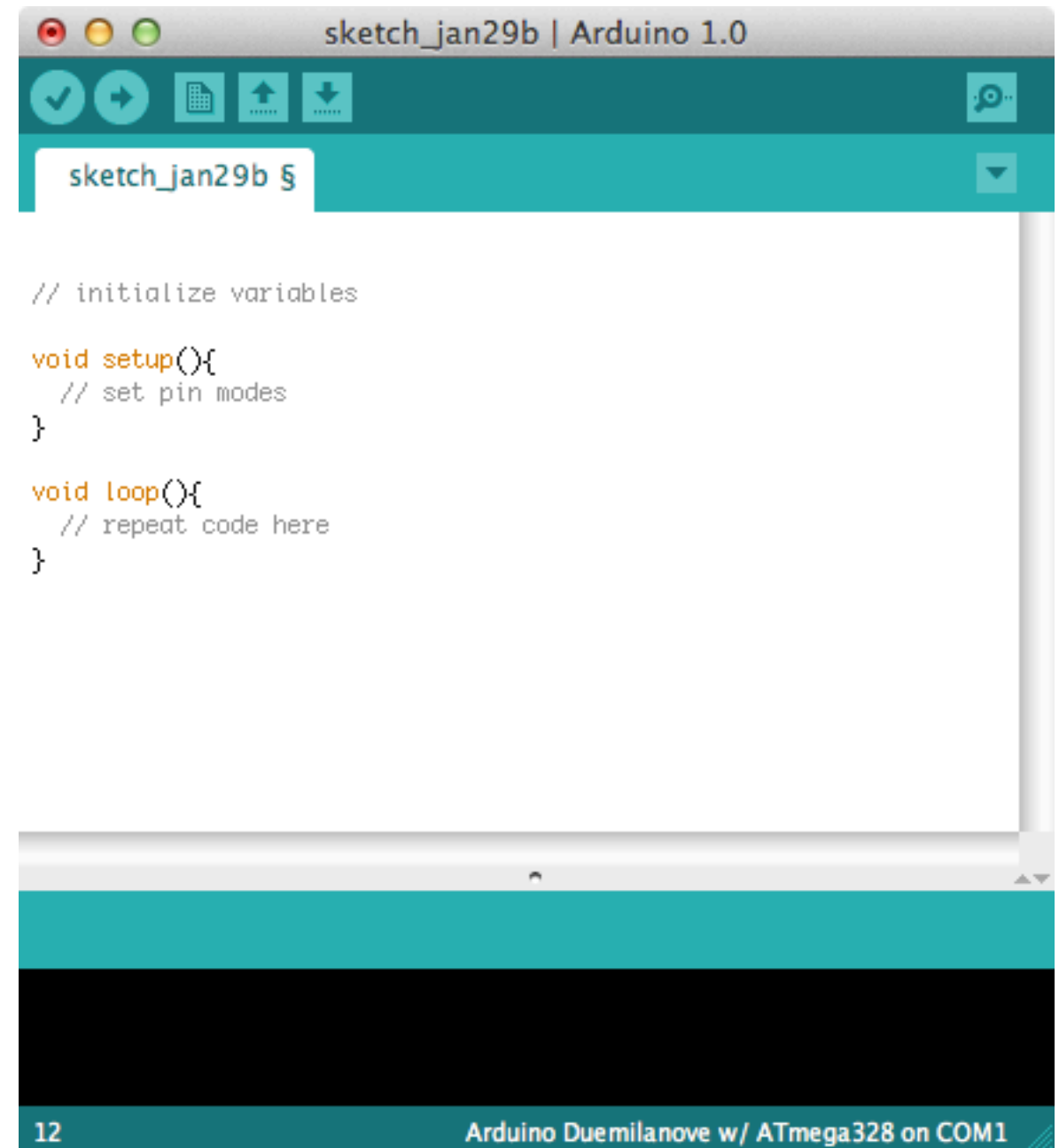
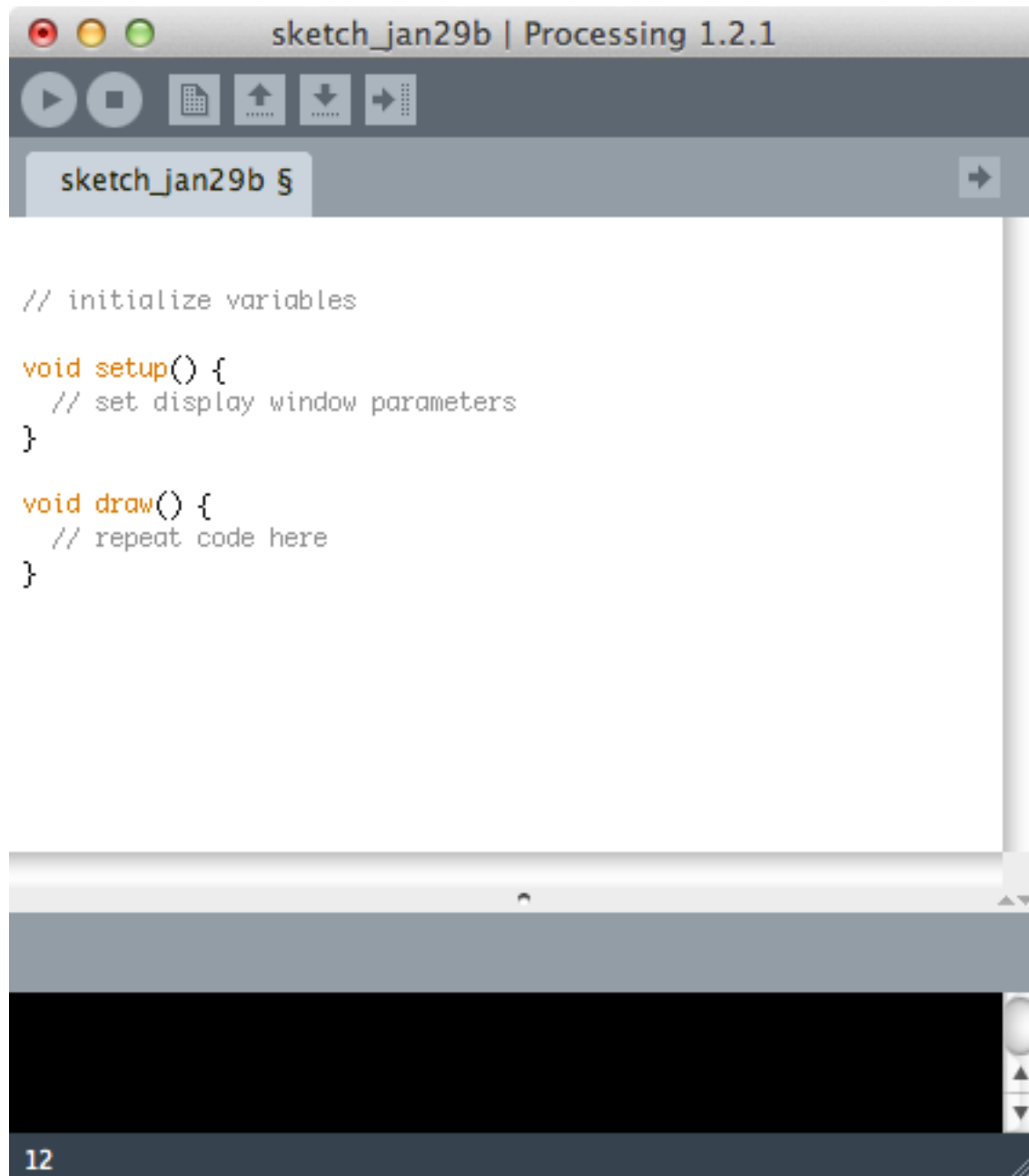
void setup(){
  // set pin modes
}

void loop(){
  // repeat code here
}
```

At the bottom, there is a status bar showing the number "12" and the text "Arduino Duemilanove w/ ATmega328 on COM1".

# The Arduino IDE

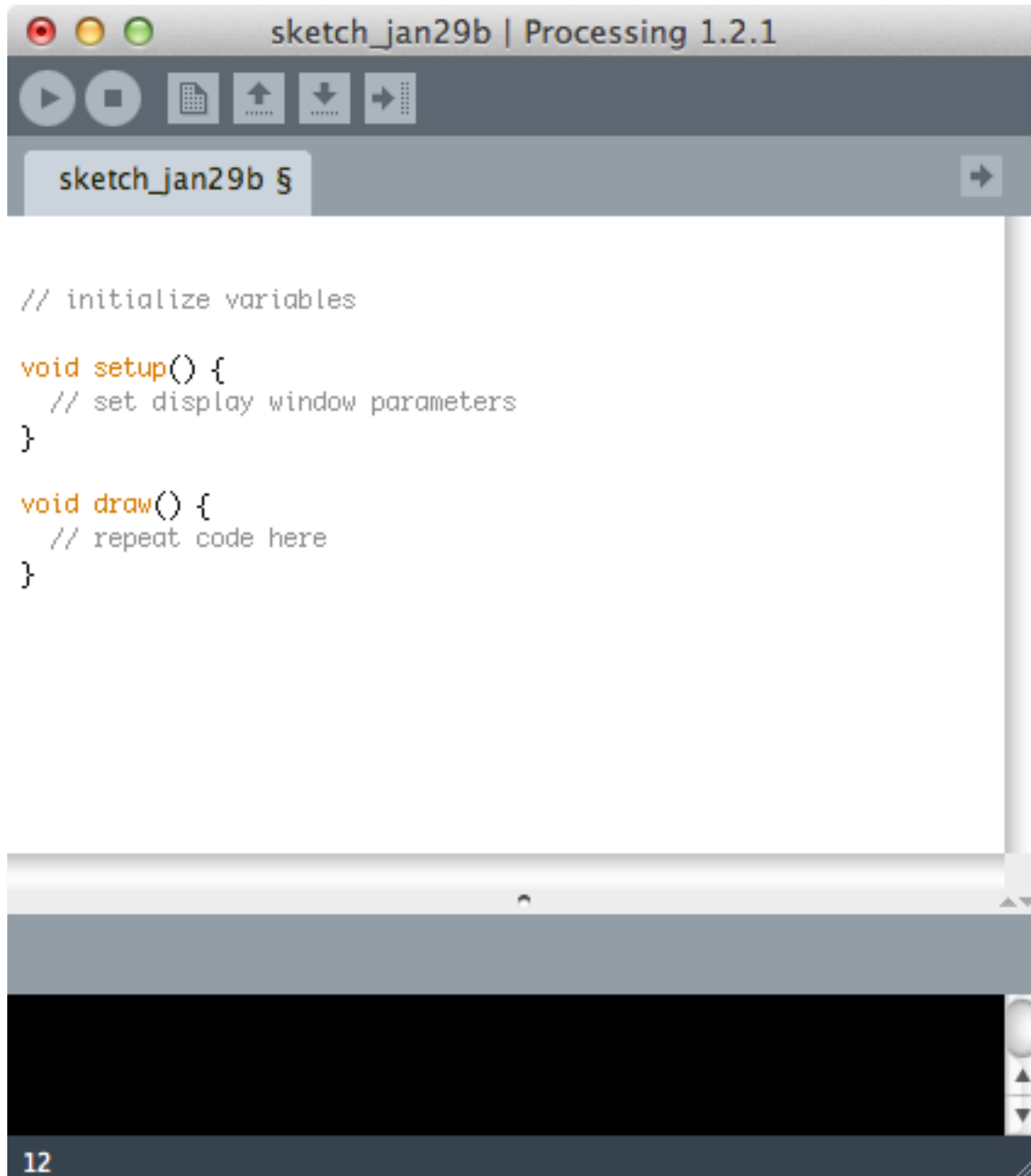
- A significant difference: while the Processing **Run** button compiles and runs your code, Arduino instead has a **Verify** button which checks for errors. The button to the right, **Upload**, sends your compiled code to your microcontroller.





# The Arduino IDE

- Both have functions specific to their environments, for example, you can't draw an ellipse with a microcontroller. They use the same code structure (except `loop()` replaces the `draw()` function).

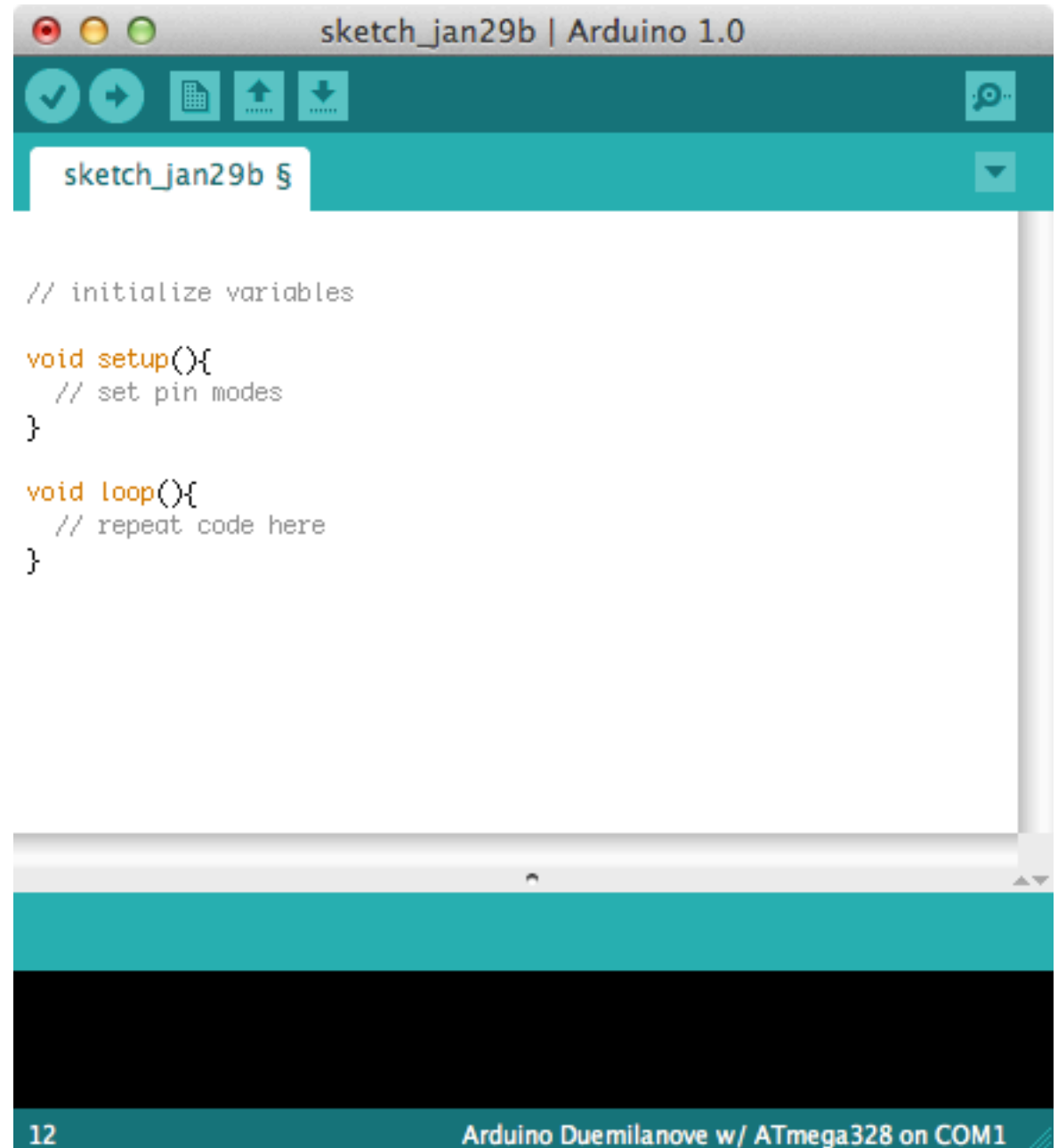


```
// initialize variables

void setup() {
  // set display window parameters
}

void draw() {
  // repeat code here
}
```

12



```
// initialize variables

void setup(){
  // set pin modes
}

void loop(){
  // repeat code here
}
```

12 Arduino Duemilanove w/ ATmega328 on COM1

# Blink! The code!

Programming the Arduino is essentially the same as programming in Processing except we have access to a whole new range of inputs.

The first thing we need to do in our Arduino sketch is to declare a variable to use for pin 10.

```
int ledpin = 10;
```

Now in our `setup()` function we write that the *pinMode* of pin 10 is for *output*, meaning, we can send digital signals to that pin from our program.

```
void setup() {  
  pinMode(ledpin, OUTPUT); // sets the digital pin as output  
}
```

# Blink! The code!

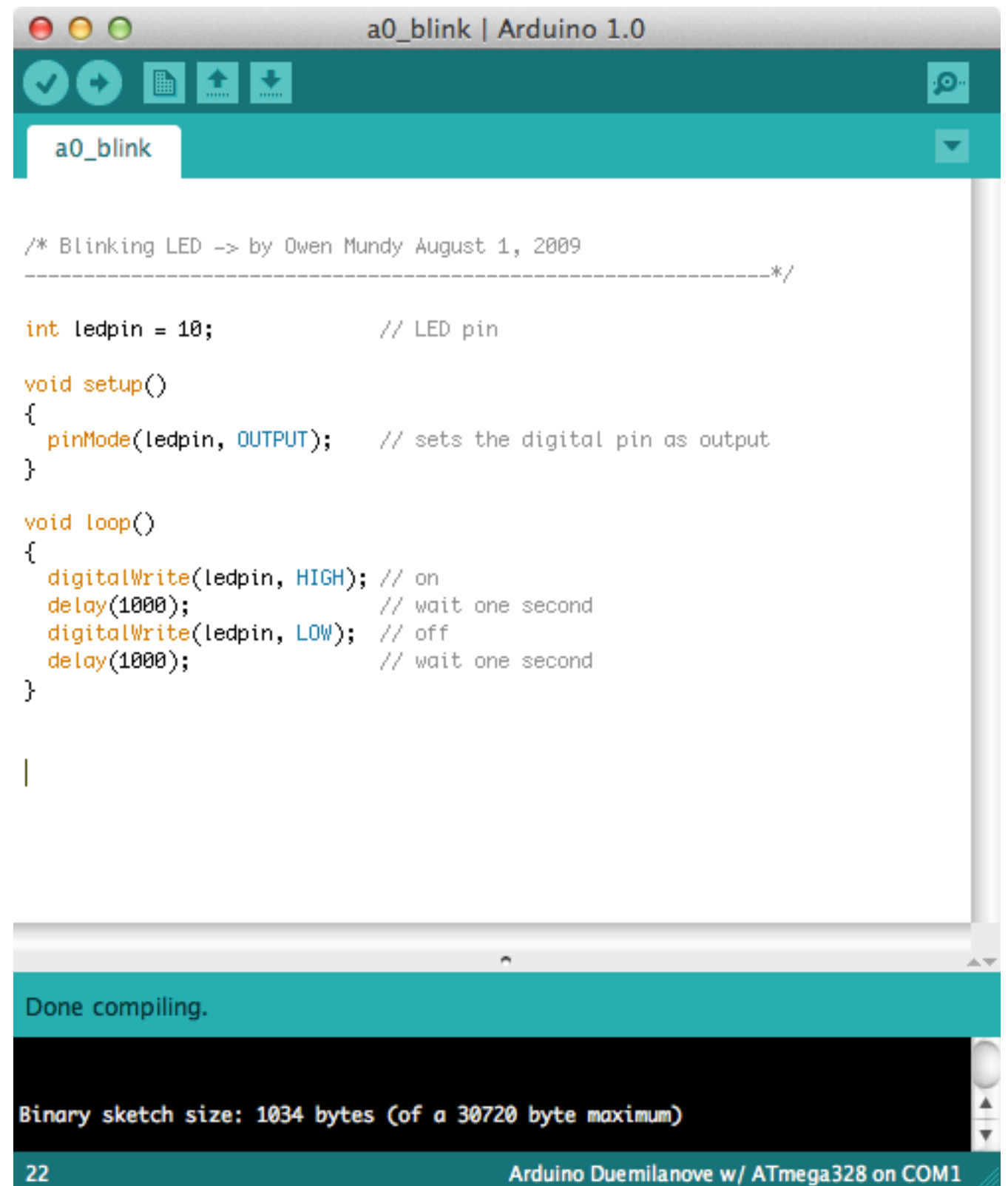
Next we create our loop, which will run infinitely. Here we use the *digitalWrite()* function to set the ledpin *HIGH* (LED is on) and then *LOW* (LED is off), over and over, with 1000 milliseconds (1 second) delay between each change.

```
void loop() {  
  digitalWrite(ledpin, HIGH); // on  
  delay(1000);                // wait one second  
  digitalWrite(ledpin, LOW);  // off  
  delay(1000);                // wait one second  
}
```

# Blink! Compiling

**Compiling** means that our written source code (what you see to the right) is checked for errors and translated into *machine code*, a lower-level computer language (e.g. binary code) that is optimized and comprehensible by our Arduino microcontroller.

So when we click **Verify**, the code is compiled and, if there are no errors, a success message and information about the final size of the compiled binary sketch appears in the Console.



The screenshot shows the Arduino IDE interface. The top bar indicates the sketch is named 'a0\_blink' and is for an 'Arduino 1.0'. The main editor area contains the following C++ code:

```
/* Blinking LED -> by Owen Mundy August 1, 2009
-----*/

int ledpin = 10;           // LED pin

void setup()
{
  pinMode(ledpin, OUTPUT); // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledpin, HIGH); // on
  delay(1000);                // wait one second
  digitalWrite(ledpin, LOW);  // off
  delay(1000);                // wait one second
}
```

Below the code editor, the console area shows the output of the compilation process:

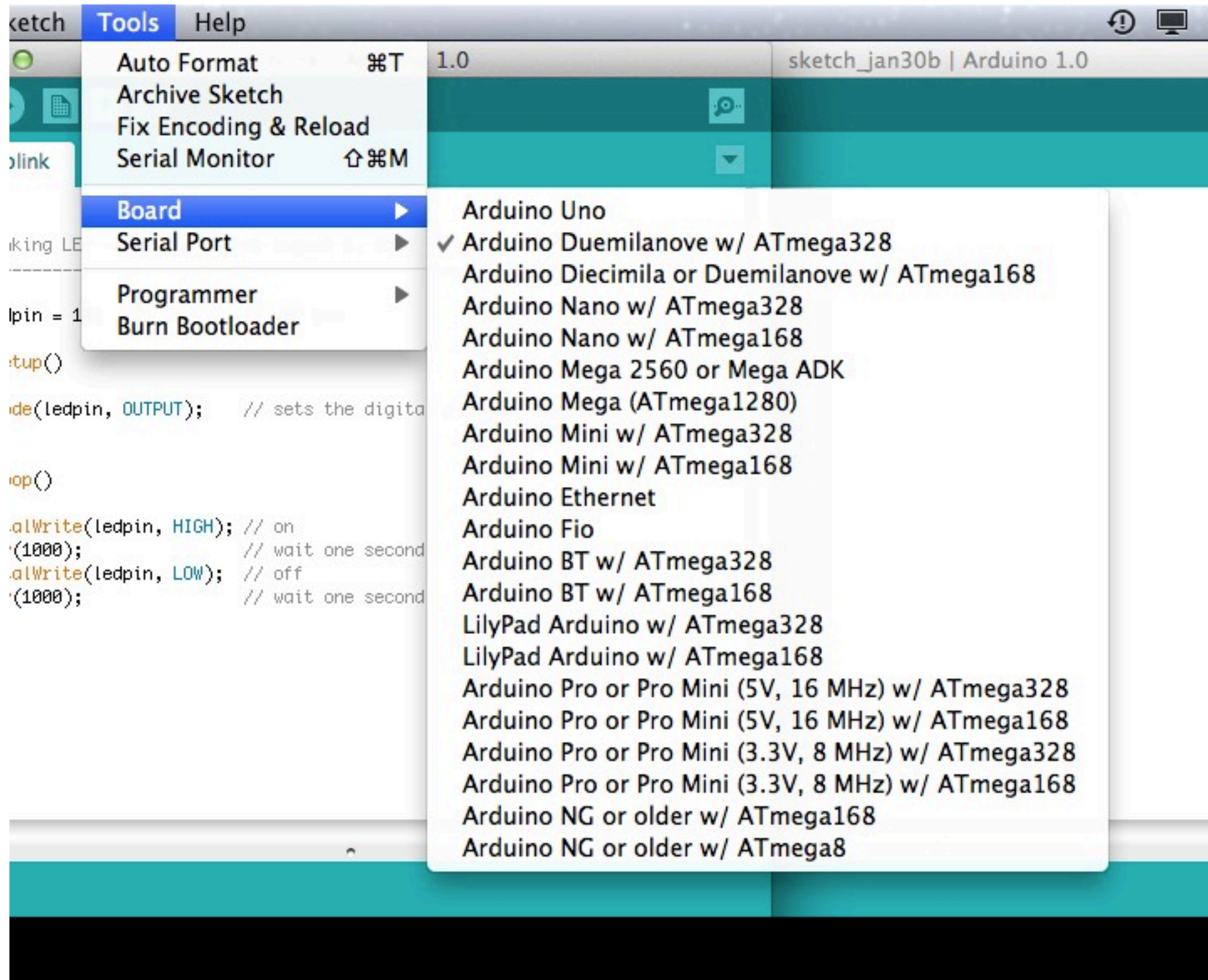
```
Done compiling.

Binary sketch size: 1034 bytes (of a 30720 byte maximum)
```

The status bar at the bottom indicates the board is 'Arduino Duemilanove w/ ATmega328 on COM1' and the line number is 22.

# Select your device

Go to Tools > Boards

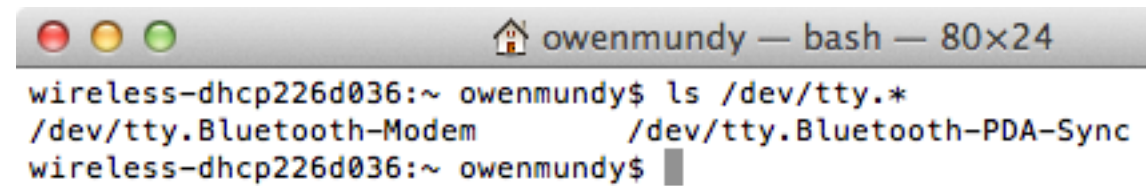


# Select your port

List the available ports on your computer

```
$ ls /dev/tty.*
```

Select this port in Arduino > Tools > Serial Port



```
owenmundy — bash — 80x24
wireless-dhcp226d036:~ owenmundy$ ls /dev/tty.*
/dev/tty.Bluetooth-Modem      /dev/tty.Bluetooth-PDA-Sync
wireless-dhcp226d036:~ owenmundy$
```

# Upload your sketch

Now click Upload and watch the magic happen!