Lecture 6
Arduino: Code Structure
Analog Input

IAT267 Introduction to Technological Systems

Organizational Items

- Assignment 1 marks – available on webct.

- Assignment 2 – due October, 20

- Project milestone 1 – due October 19
  - Teams (4 students/ team)
    - Each student should be in a team now
  - Project proposal

Quiz for Week 6

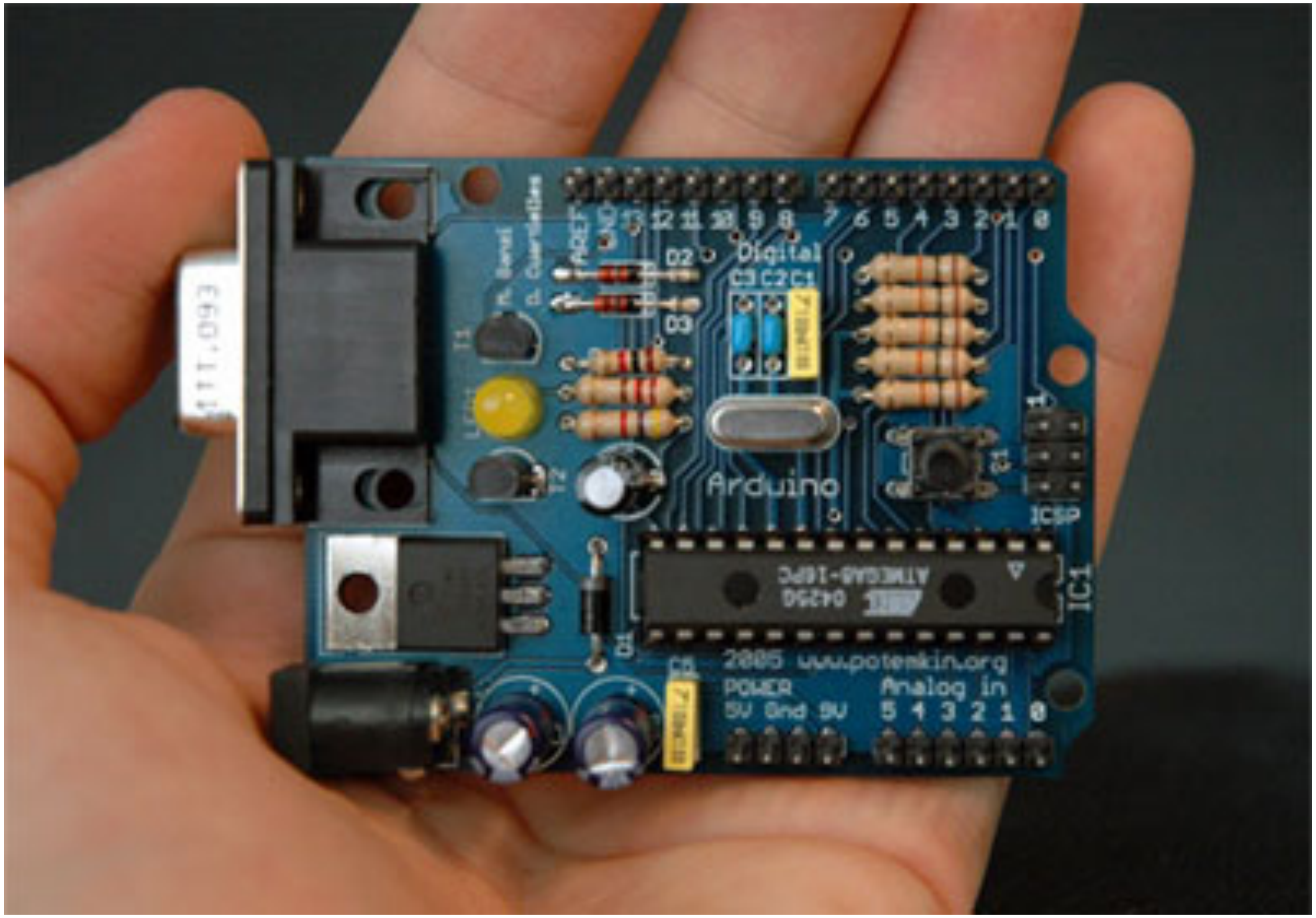
- Will be available starting this Friday until next Wednesday

- Can be done anytime in the availability interval

Lecture Topics for Today

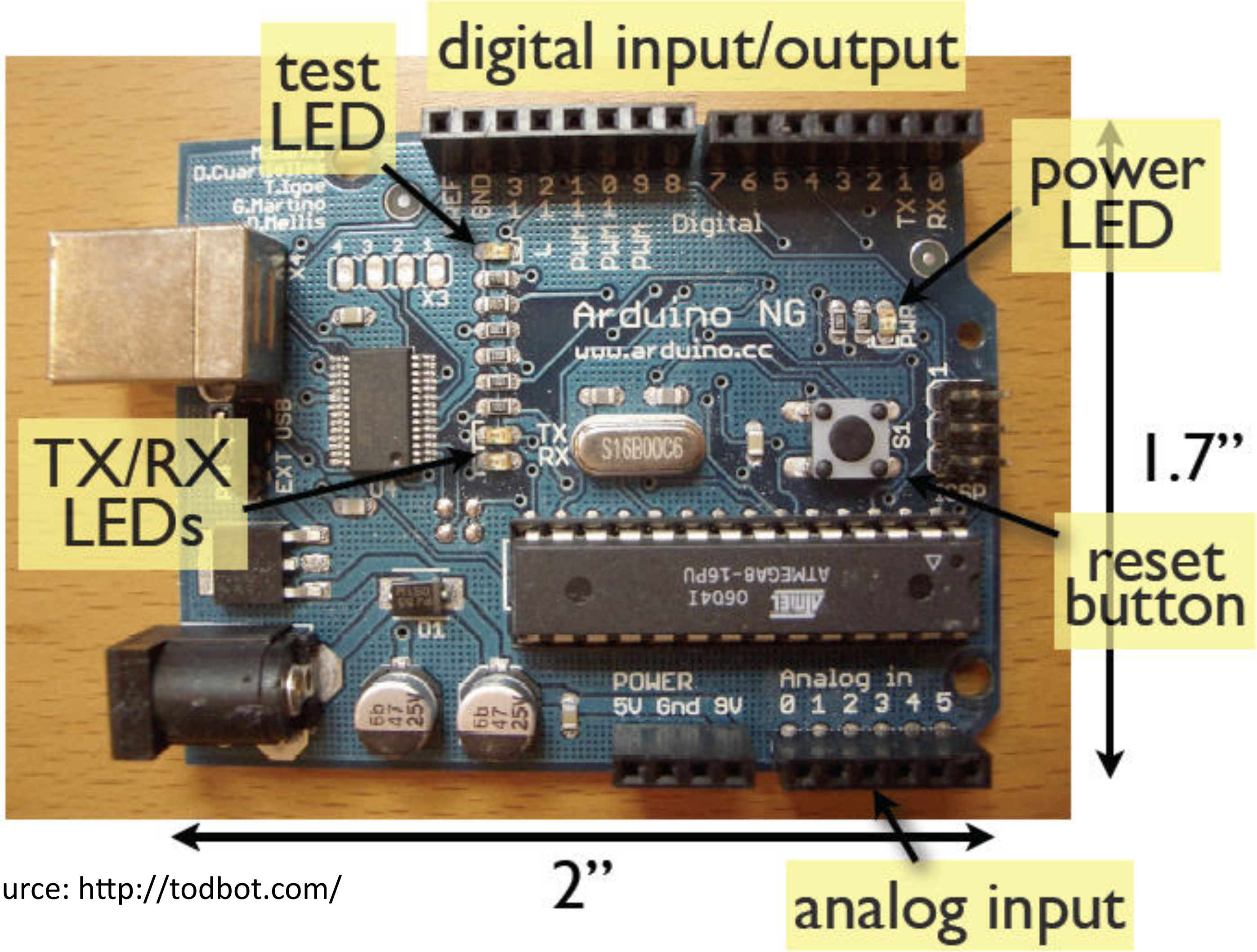
- Arduino – review from unit 5
- Arduino – code
  - General structure
- Arduino – analog input
  - General principles
  - Examples with circuits and code walk-through (these examples will be implemented in the workshops)

Arduino – Summary from Unit 5


Digital
Arduino
ICSP
IC1
2005 www.potemkin.org
POWER
5V Gnd 9V
Analog in
5 4 3 2 1 0

Arduino: 3 Separate Tools

- 1. Arduino hardware board
  - Several versions and forms
- 2. Arduino programming environment
  - Simple open source IDE
- 3. Language and compiler
  - Create code for the microcontroller



Source: http://todbot.com/

What is a Pin?

- A **pin** provides an **input** or **output** through which the microcontroller can communicate with components or computer

- Small wires can be inserted into the pin connectors

Digital vs. Analog Pins

- **Digital pins**:
  - Have two values that can be read or written to them: high and low
    - High: means that 5 V (Volts) is being sent either from the microcontroller or from a component
    - Low: means that the pin is at 0 Volts.
  - Any kind of binary information can be read or written to a digital pin.

Analog Pins

- Can have a wide range of information read or written to them.
- These pins are what we use to read and write information that has a range of values, e.g.:
  - The position of a dial
  - The distance of an object from an infrared sensor
  - The brightness of an LED light

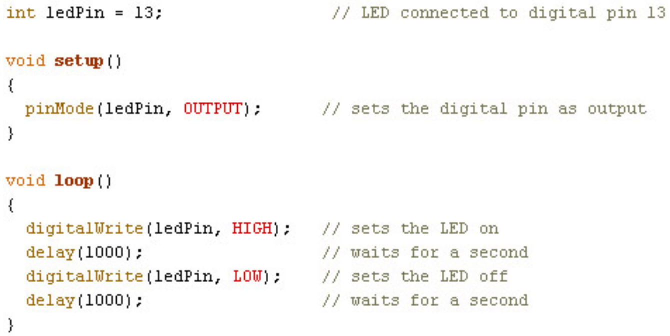
2. Arduino Programming Environment

How is Arduino Programmed?

- Write programs on your PC
- Download them into the Arduino board
- Arduino board can then be used by itself

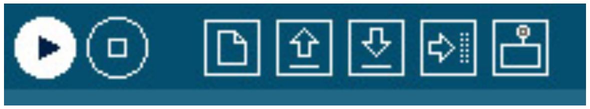
Development Cycle

- Edit code

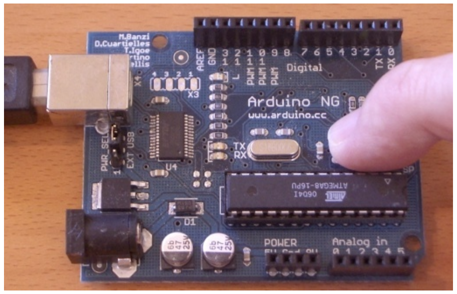


```
int ledPin = 13;                 // LED connected to digital pin 13

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

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

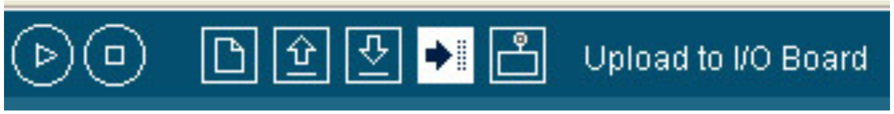
- Compile



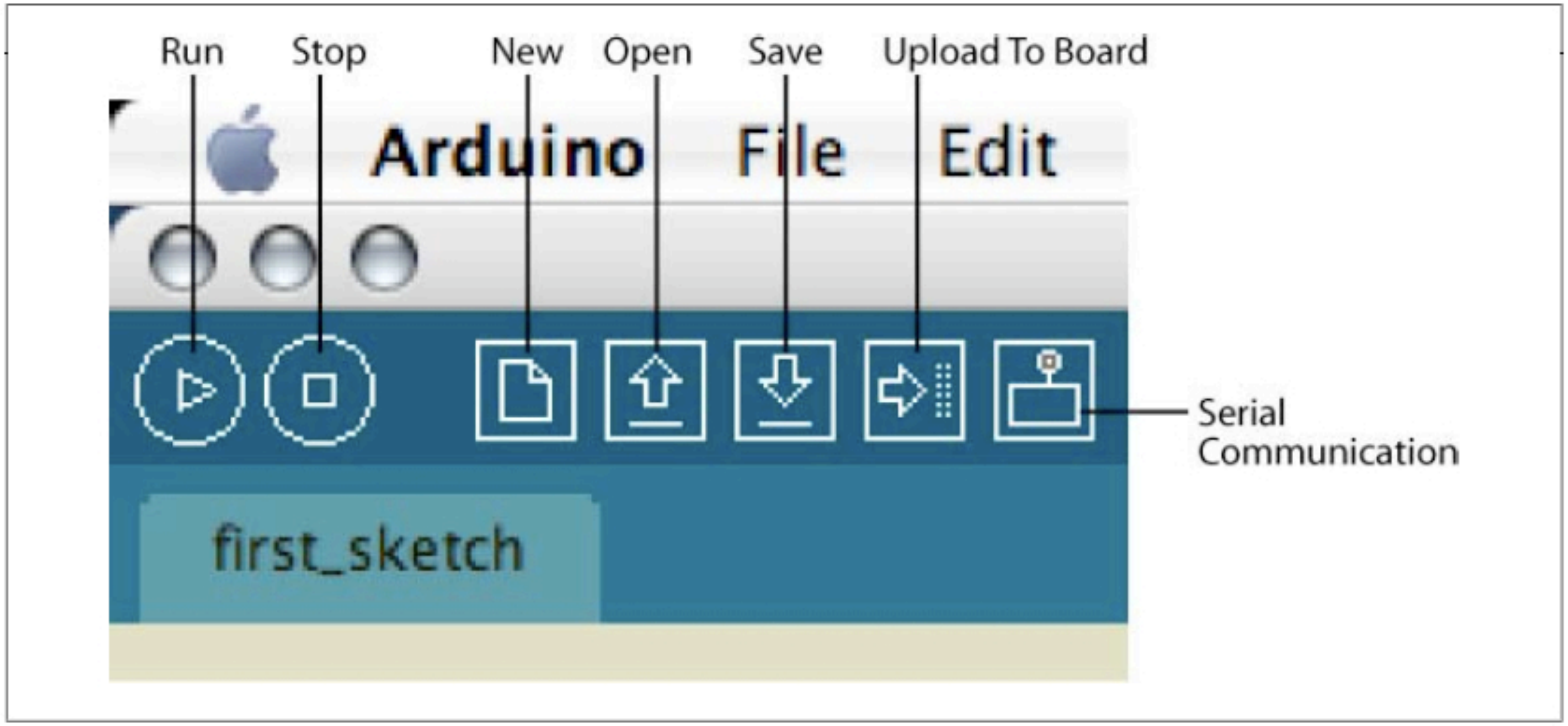
- Reset board



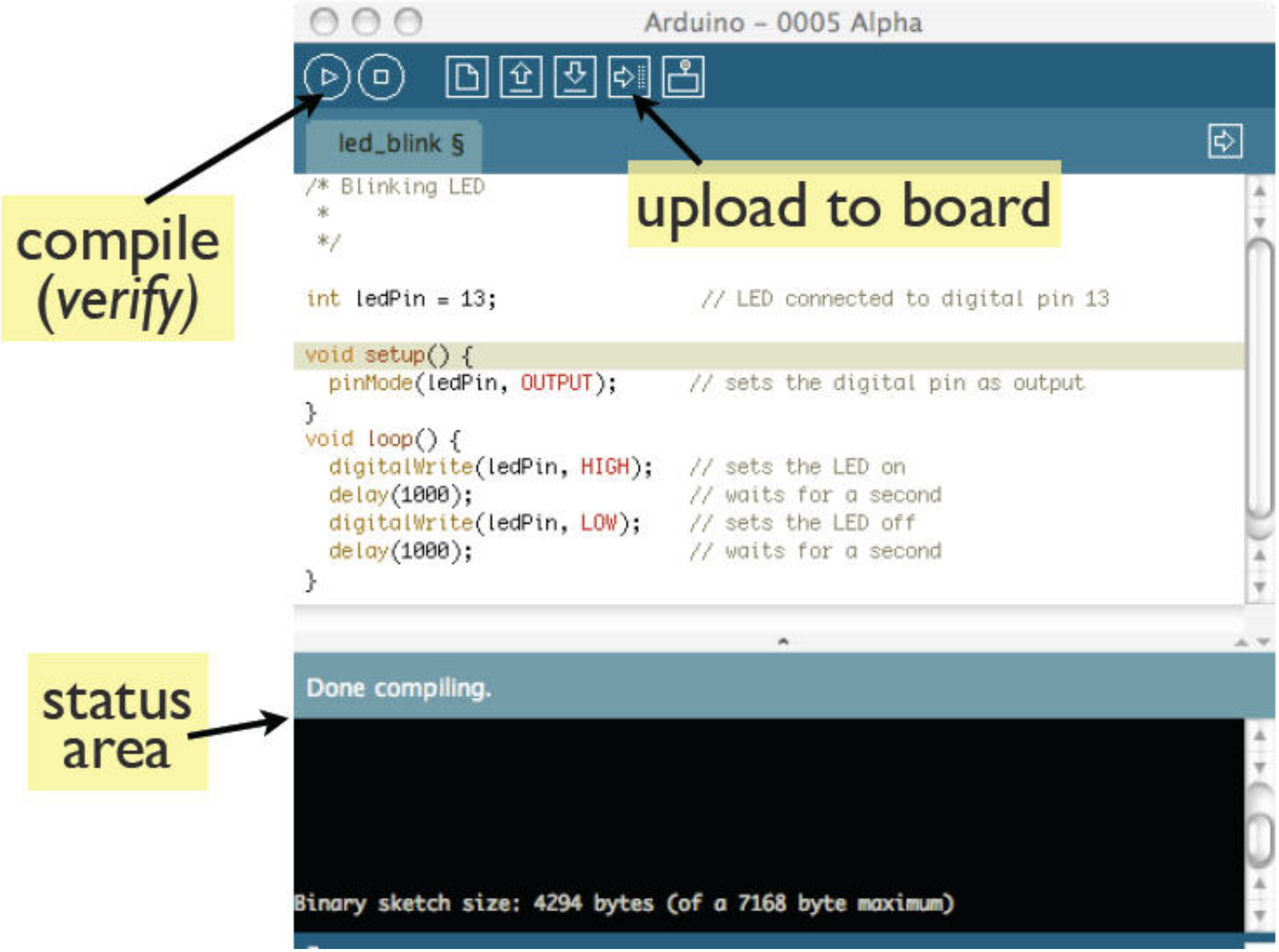
- Upload



The Arduino IDE



Arduino Software



compile (*verify*)

upload to board

status area

```
/* Blinking LED
 *
 */

int ledPin = 13;                  // LED connected to digital pin 13

void setup() {
  pinMode(ledPin, OUTPUT);        // sets the digital pin as output
}
void loop() {
  digitalWrite(ledPin, HIGH);     // sets the LED on
  delay(1000);                    // waits for a second
  digitalWrite(ledPin, LOW);      // sets the LED off
  delay(1000);                    // waits for a second
}
```

```
Done compiling.

Binary sketch size: 4294 bytes (of a 7168 byte maximum)
```

3. The Arduino Language

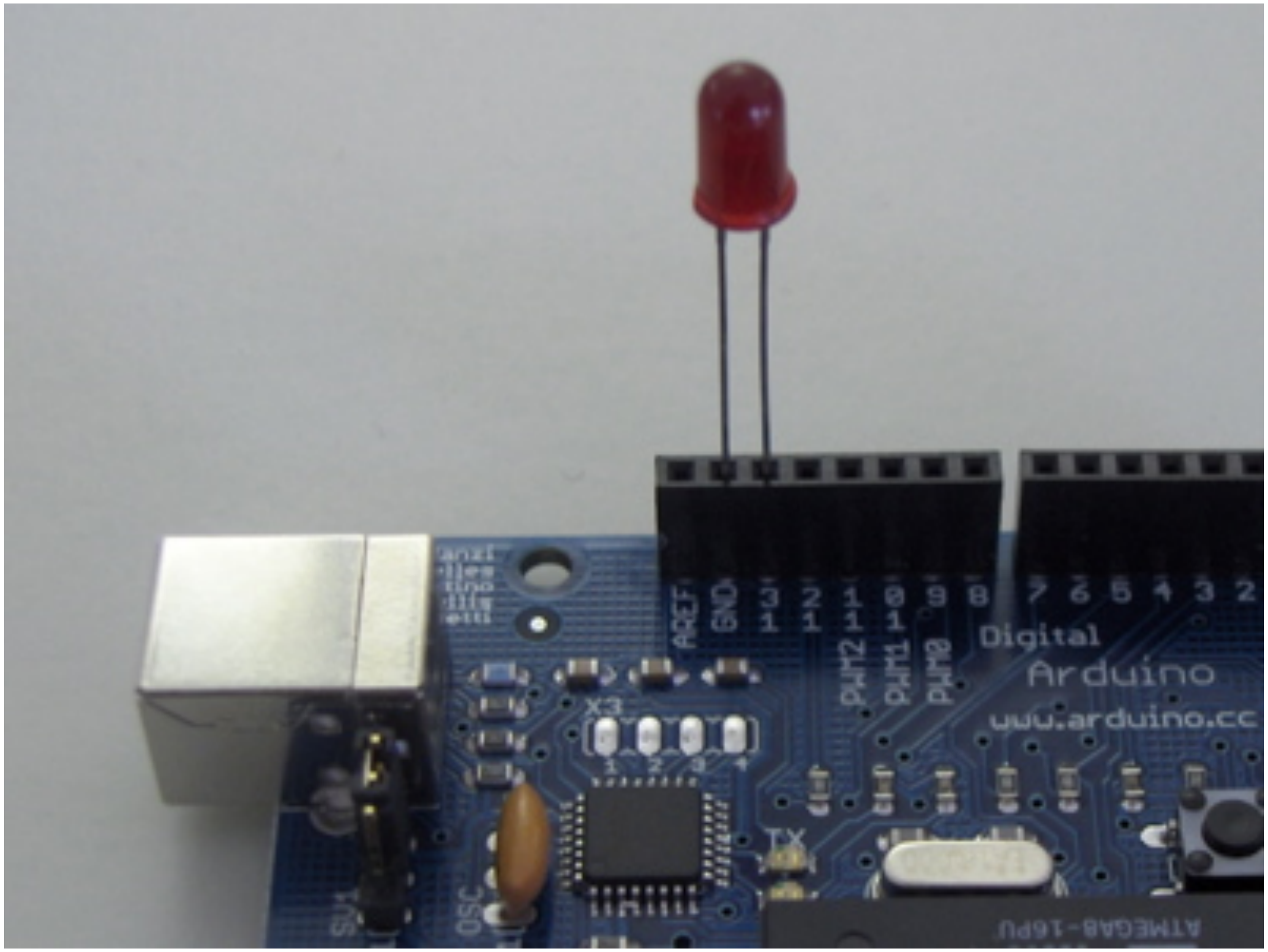
(Wiring)

Example Program: Blink

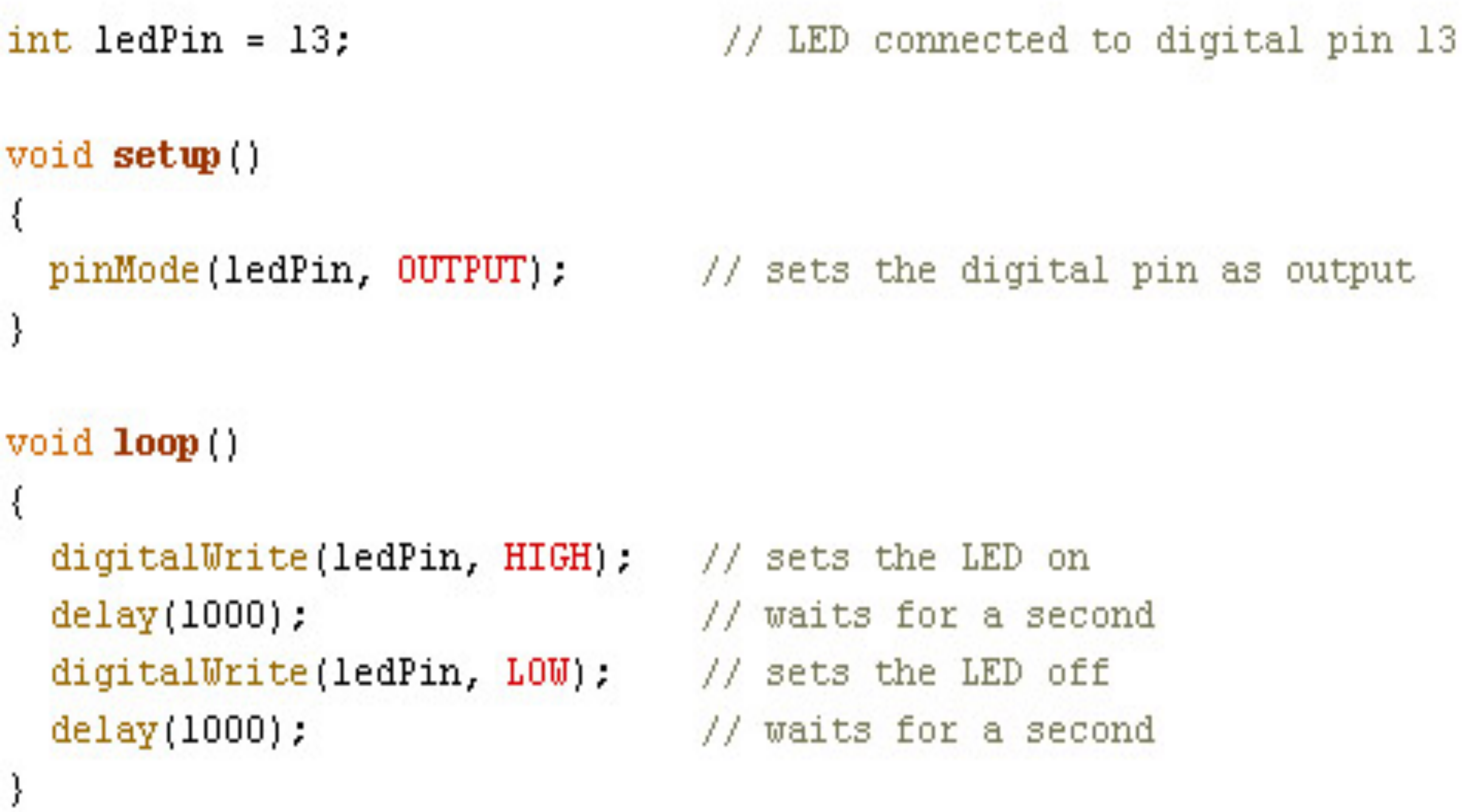
- LED connected to digital pin 13 (we choose pin 13 because depending on your Arduino board, it has either a built-in LED or a built-in resistor so that you need only an LED.

- LEDs have polarity, which means they will only light up if you orient the legs properly.

Circuit


AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
PWM2
PWM1
PWM0
Digital
Arduino
www.arduino.cc
X3
1
2
3
4
TX
OSC
ATMEGA8-16PU

The code



```
int ledPin = 13;                 // LED connected to digital pin 13

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

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Program Structure

- All Arduino program run in two parts:
  - void setup()
  - void loop()

- setup() is preparation
- loop() is execution

- In the setup section, always at the top of your program, you would set pin modes, initialize serial communication, etc.
- The loop section is the code to be executed -- reading inputs, triggering outputs, etc.

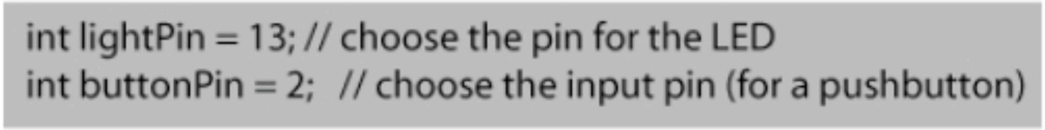
The ‘setup’ statement

- The first thing called in an Arduino application
- Some devices need to be initialized when the microcontroller starts up.
- All applications must have a setup() method, even if nothing is done in it.
  - The compiler will check for this method, and if it is not defined, an error will occur.

The ‘loop’ method

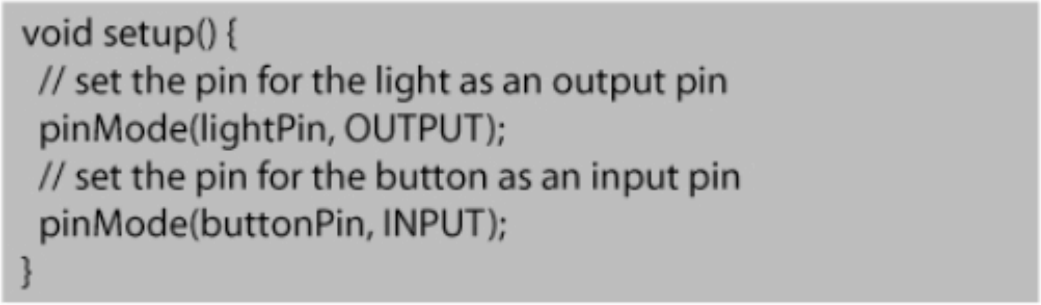
- Contains anything that needs to happen repeatedly in the application, e.g.:
    - Checking for a new value from an input
    - Sending a signal to a pin
    - Sending debug information
- Any instructions in this method will run repeatedly until the application is terminated

Organization of the code



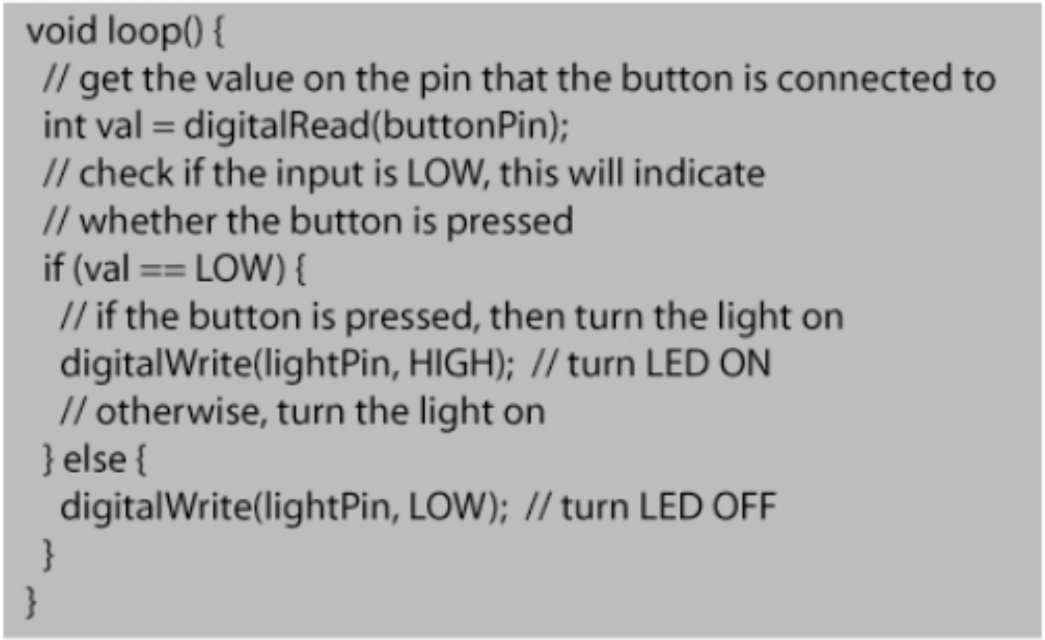
```
int lightPin = 13; // choose the pin for the LED
int buttonPin = 2;   // choose the input pin (for a pushbutton)
```

initialization - these variables will be available throughout the code



```
void setup() {
 // set the pin for the light as an output pin
 pinMode(lightPin, OUTPUT);
 // set the pin for the button as an input pin
 pinMode(buttonPin, INPUT);
}
```

setup() - get everything ready for the program to run



```
void loop() {
 // get the value on the pin that the button is connected to
 int val = digitalRead(buttonPin);
 // check if the input is LOW, this will indicate
 // whether the button is pressed
 if (val == LOW) {
  // if the button is pressed, then turn the light on
  digitalWrite(lightPin, HIGH);  // turn LED ON
  // otherwise, turn the light on
 } else {
  digitalWrite(lightPin, LOW);  // turn LED OFF
 }
}
```

loop() - check the pin and change the light

Initialization – setup - loop

- **Initialization**: contains all the variables and values that will be used throughout the program

- **Setup**: contains the code to configure the pin for the button to receive information and set the pin for the light to send information

- **Loop**: contains the code to check the value of the button.

Writing Programs for Arduino

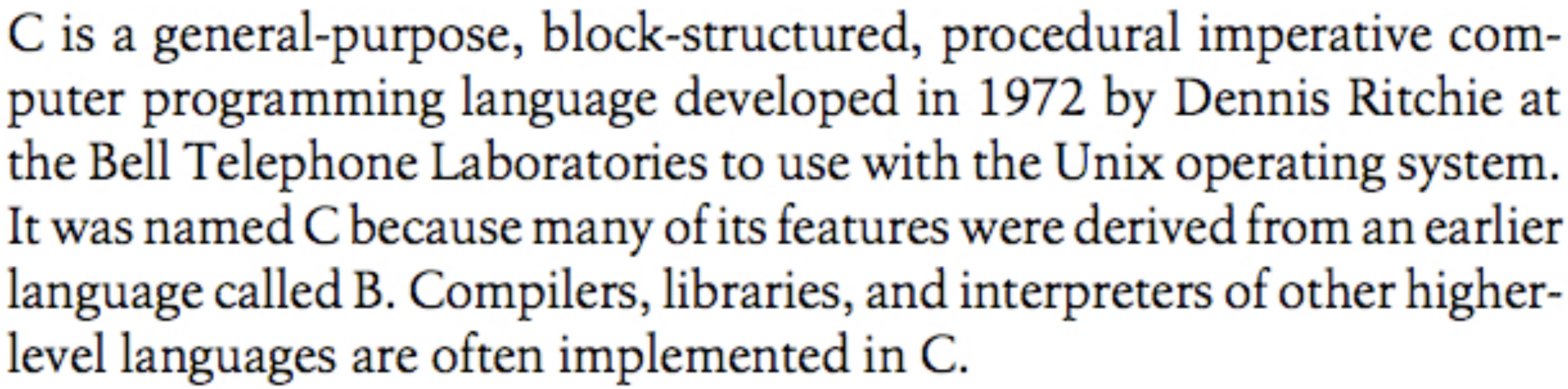
- Programs are called ‘sketches’.
- The sketch itself is in the text input area of the Arduino software.

Sketches

- Sketches are written in text, just like a document.
- When you select **Compile/Verify** from the menu, the Arduino software looks over the document and translates it to Arduino-machine-language - which is not human-readable but is easy for the Arduino to understand.

Features of the Arduino Language

- Built on the C language

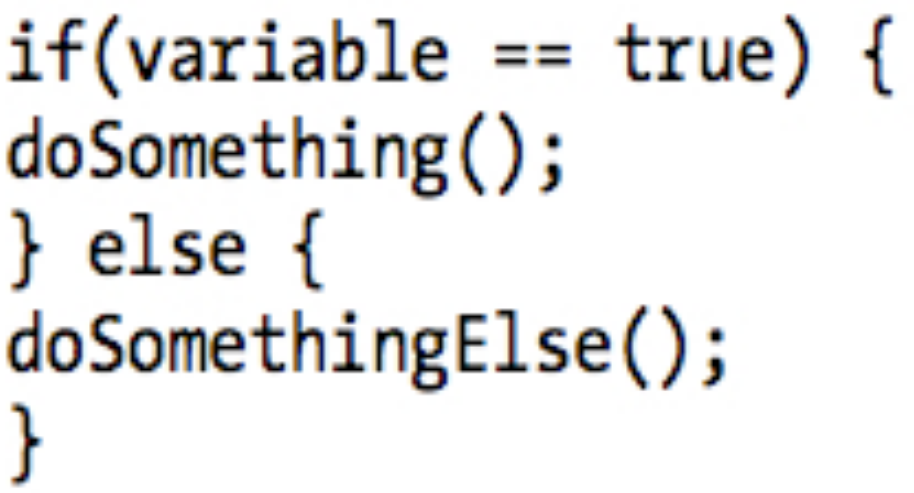


C is a general-purpose, block-structured, procedural imperative computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories to use with the Unix operating system. It was named C because many of its features were derived from an earlier language called B. Compilers, libraries, and interpreters of other higher-level languages are often implemented in C.

- Designed to support communication with electronic components

Constants

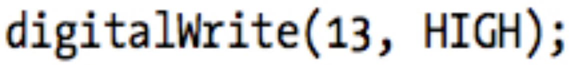
- true / false:



```
if(variable == true) {
doSomething();
} else {
doSomethingElse();
}
```

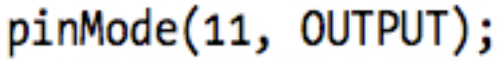
Constants

- HIGH / LOW: these define the voltage level on a digital pin, either 5V or 0V.
  - Make your code more readable



```
digitalWrite(13, HIGH);
```

- INPUT / OUTPUT: constants used or setting pins that can be used either for output or for input:



```
pinMode(11, OUTPUT);
```

Methods

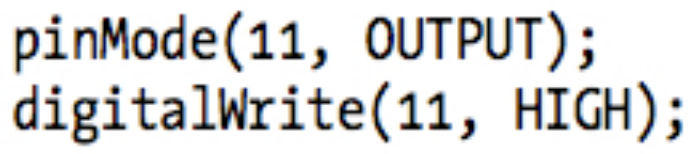
- pinMode() – set a pin as input or output
- digitalWrite() – set a digital pin high/low
- digitalRead() – read a digital pin’s state
- analogRead() – read an analog pin
- analogWrite() – write an “analog” PWM value
- delay() – wait an amount of time
- millis() – get the current time

pinMode(pinNumber, mode)

- The digital pins of Arduino can be set to either input or output
  - Send values or receive values from the microcontroller
- Before we use a digital pin, we need to establish in which direction the information will be flowing
- This is done in the setup() method

digitalWrite(value)

- Sets a digital pin to HIGH if value is high or LOW if value is low (meaning that it will send 5V or 0V through the pin).
- Works only on pins that have been set to OUTPUT using pinMode().
- Example:



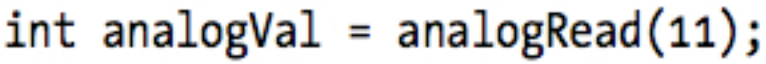
```
pinMode(11, OUTPUT);
digitalWrite(11, HIGH);
```

int digitalRead (pinNumber)

- Reads the state of a pin that is in input mode
- Can be either HIGH or LOW (5V or 0V) – no other value
- Used for reading buttons, switches, anything that has a simple on and off, and any control that returns a true/false or other type of binary value

analogRead (pin Number)

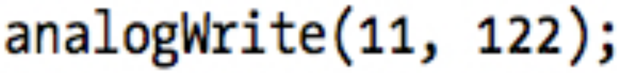
- Reads the value of an analog pin, returning a range from 0 to 1023, representing the voltage being passed into the pin
- It is important that any devices that you connect to Arduino send analog signals in the range between 0 and 5V because higher values will not be read and could damage the board.



```
int analogVal = analogRead(11);
```

analogWrite (pin, value)

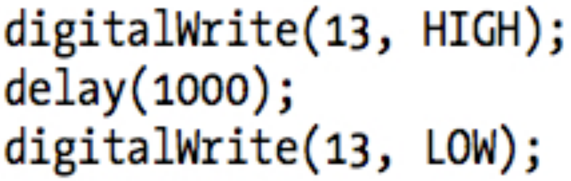
- Writes an analog value to a pin and can be any value between 0 and 255



```
analogWrite(11, 122);
```

delay (ms)

- Tells the program to wait for a given number f milliseconds before executing the next instruction.

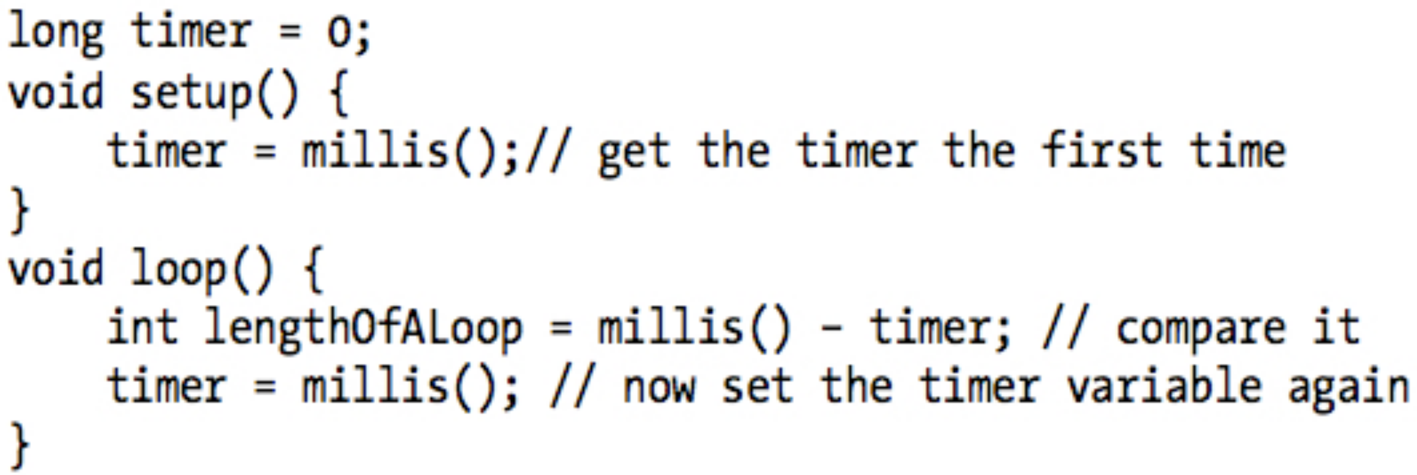


```
digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
```

- In practice this is used for timing, such as controlling how long a LED stays lit, for example.

millis ()

- Returns the number of milliseconds since the program started running.
- Can be useful when you need to keep track of time



```
long timer = 0;
void setup() {
    timer = millis();// get the timer the first time
}
void loop() {
    int lengthOfALoop = millis() - timer; // compare it
    timer = millis(); // now set the timer variable again
}
```

Analyze the code…

```
• /*
•   * Blink
•   *
•   * The basic Arduino example.  Turns on an LED on for one second,
•   * then off for one second, and so on...  We use pin 13 because,
•   * depending on your Arduino board, it has either a built-in LED
•   * or a built-in resistor so that you need only an LED.
•   *
•   * http://www.arduino.cc/en/Tutorial/Blink
•   */
```

Comments

- This is a **comment**, it is text that is not used by the Arduino, its only there to help humans like us understand whats going on.

- You can tell if something is a comment because there is a **/*** at the beginning and a ***/** at the end.

- Anything between the /* and */ is ignored by the Arduino.

- Comments are very useful and are strongly encouraged to be used: in every sketch you make have a comment in the beginning with information like who wrote it, when you wrote it and what its supposed to do.

Variables

- *int ledPin = 13; // LED connected to digital pin 13*
- This is the first line of actual instruction code.
- Ends with a semicolon.
- A sentence telling the computer that we would like it to create a variable named **ledPin** and to put the number 13 in that variable.
- The first part of this sentence is **int**, which is short for **integer.**
- The second part of this sentence is **ledPin** which is the name of the variable.

The other instructions

- *pinMode(ledPin, OUTPUT);   // sets the digital pin as output*

- *digitalWrite(ledPin, HIGH);   // sets the LED on*

- *digitalWrite(ledPin, LOW);   // sets the LED off*

Workshop 6 circuits

- Analog input

  - from potentiometer (or slider sensor, rotation sensor)

  - from light sensor

Arduino basic circuit

- Blink an LED connected to pin 13

- Digital pin 13: test pin, **because it has a resistance already connected, so no external resistance is needed**

- Digital pins can be configured as inputs or outputs → pin 13 should be configured as output because an LED is connected to it.

Analog input

- Potentiometer connected to analog pin 0
- Analog pins do not need to be configured (they are inputs by default)
- Functionality of the circuit: blink the LED with a rate which is a function of the value of the potentiometer

The code: variables, pins to be used

```
int potPin = 0;    // select the input pin for the
    potentiometer

int ledPin = 13;   // select the pin for the LED

int sensorVal = 0;       // variable to store the value
    coming from the sensor
```

Code: set up the digital pin

```
void setup()
{
    pinMode(ledPin, OUTPUT);  // declare the ledPin as an OUTPUT
}
```

Code: loop

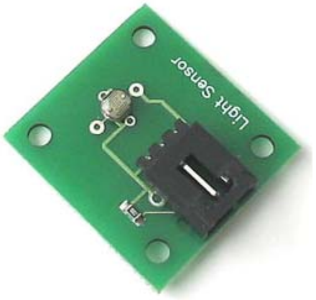
```
void loop()
{
    val = analogRead(potPin);    // read the value from the sensor
    digitalWrite(ledPin, HIGH);  // turn the ledPin on
    delay(sensorVal);            // stop the program for some time
    digitalWrite(ledPin, LOW);   // turn the ledPin off
    delay(sensorVal);            // stop the program for some time
}
```

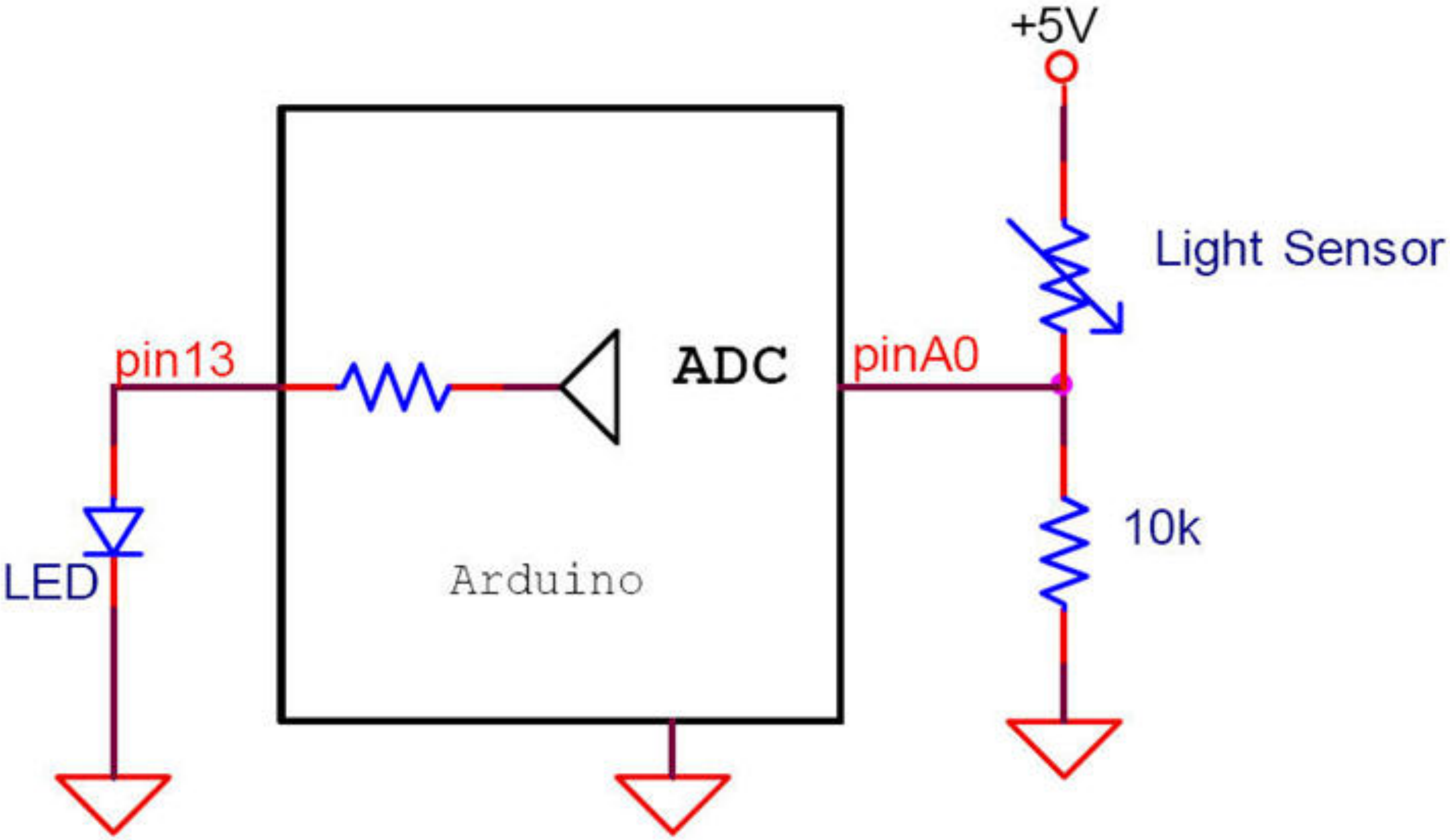
Using a light sensor

- Implement a circuit to sense light / dark

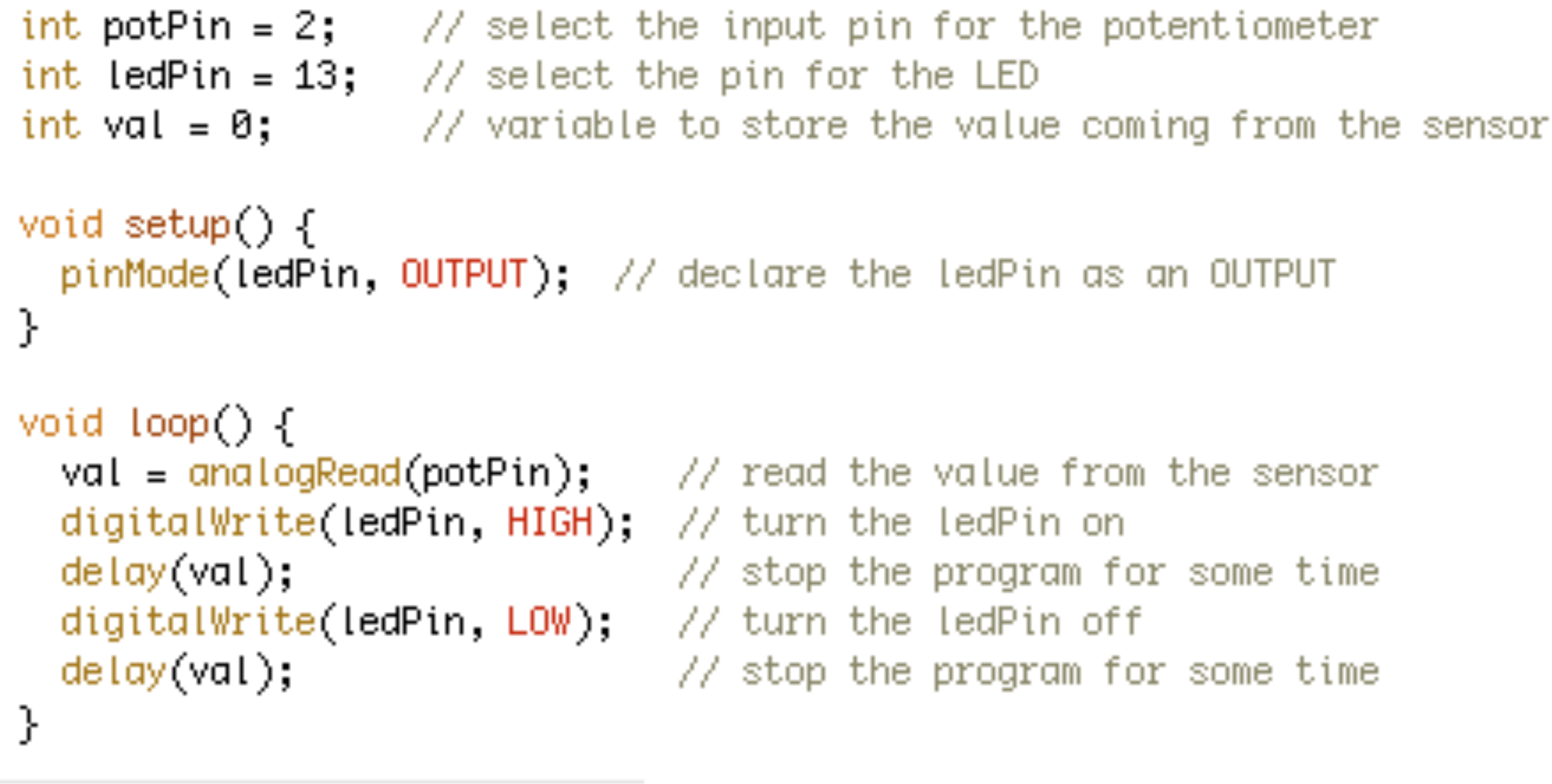
- Light sensor: light-dependent resistor
  - A variable resistor; output from the sensor is a variable resistance
  - Brighter light == lower resistance

- With no light the resistance of this sensor is 500 k ohm. At 10 lux the resistance falls to between 10 k and 5 k ohm.




+5V
Light Sensor
pin13
ADC
pinA0
10k
LED
Arduino

Code



```
int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT);  // declare the ledPin as an OUTPUT
}

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(val);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(val);                  // stop the program for some time
}
```

The circuit

- The light sensor (and the 10k resistor) are connected to analog pin 0
- **Voltage divider** circuit
- The LED is connected again to pin 13
- Delay (blink rate) is given by the voltage value from the voltage divider

Voltage Divider

- *Voltage = 10 K / (10K +* R_{LS}*)*5V*
  - – R_{LS} = 500K in condition of dark
  - – R_{LS} = 5K in condition of light

- The **Voltage Divider** structure is used frequently in cases when the output of the sensor is a variable resistance.

What happens:

- More light: light sensor has lower resistance, so the voltage from the voltage divider has a higher value → delay is larger → LED blinks slower
- Less light, darker: light sensor has higher resistance, less voltage on the analog input → delay has a smaller value → LED blinks faster

Thank you

Questions?