# Crash course in C for Arduinos

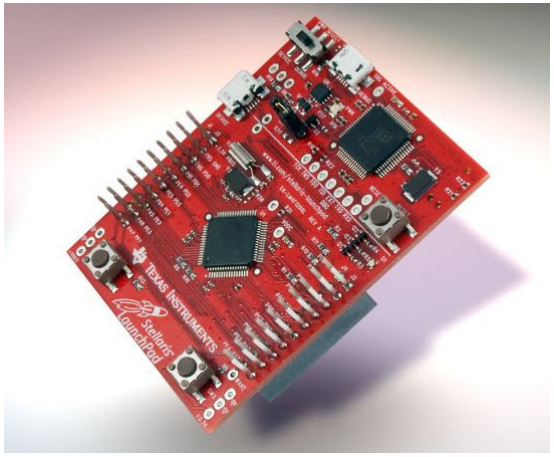
Dan Gastler

2016-03-14

# Microcontrollers

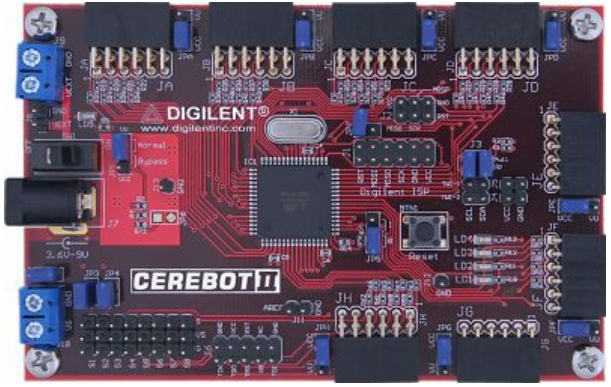
**What are microcontrollers (uCs)?**

- Basically simple computers
- They have a processor, memory, and input/output ports.
- They run a program designed to control systems all around you.
- Examples include phones, mp3 players, computer peripherals, kitchen appliances, remote controls, etc.

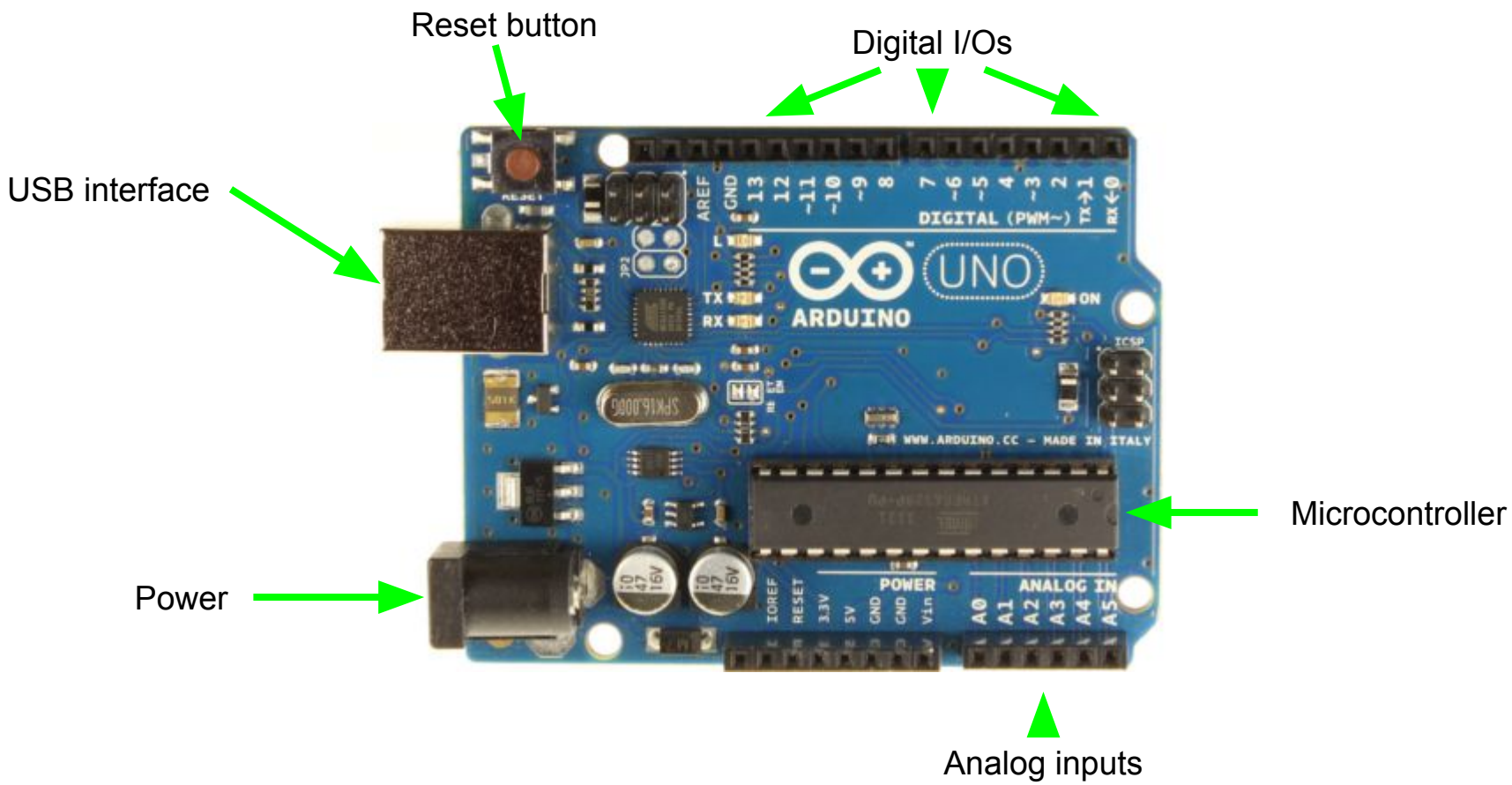



**Why are we learning about these?**

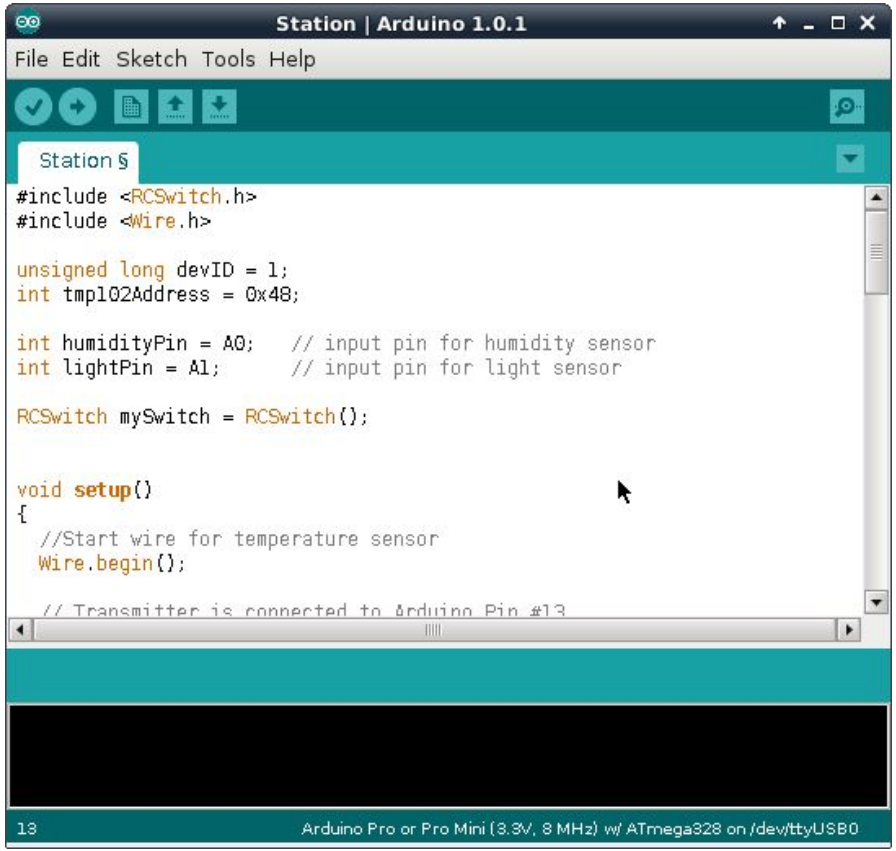
- Microcontrollers allow you to use the analog and digital circuits you've learned to control and monitor the real world.
- When you walk into a research lab, you will be able to take data from your equipment and send it to a computer for you to process.

# Arduinos (hardware)



# Arduinos (software)




**How do we program our arduinos?**

- Free IDE (win/OSX/Linux): http://arduino.cc/en/Main/Software
- You can use this software to write your program, build it into something the arduino can use, and upload your program to the arduino.
- Programs are written in a language that is similar to C.
- The IDE also contains a serial port monitor that gives you an input/output terminal to your arduino.
- Contains many example programs showing you how to use all the features of your arduino.

# What goes into a program?

**Flow control and loops:**

- These alter what the program does based on the input given.
- Give us a structure to repeat a set of instructions.

**Data:**

- Data in programs is stored in variables and constants.
- These include integers, real numbers, characters and strings.

**Functions:**

- These are blocks of code that have well defined inputs and outputs.
- They are used to compartmentalize and organize your code, making it easier to understand.

**Libraries:**

- These are groups of functions and data that have been put together to accomplish a task.
- Using existing code makes life easier.

**Comments/Tabbing:**

- Comments tell everyone looking at your code what it does.
- Good comments and tabbing help you and others use and modify your code.
- Choose a tabbing and stick with it!

# Data types

Data in digital electronics:

- Data in any computer is stored as binary bits.
- The hardware groups these bits in groups of 8 bits (1 byte) and in groups of bytes (words).
- We can interpret these bits as integers, real numbers, or characters.

Integers:

- Integers are whole numbers (no decimal point) and can be signed (positive or negative) or unsigned (only positive).
- We represent integers in three ways:
  - decimal (count to 10):
  - hex (count to 16):
  - binary (count to 2):

| decimal | binary | hex |
|---|---|---|
| 1 | 0b1 | 0x1 |
| 43 | 0b101011 | 0x2B |
| 193 | 0b11000001 | 0xC1 |

Real numbers:

- Numbers like 3.4, 2.7182, and 1.602*10^-19.
- Only an approximation!
- Real numbers need a decimal point (4.0 for 4) to tell the compiler that it is really a float.

Characters:

- An ASCII character is a code that uses a number to represent a character to print to the screen.
- The character “E” is stored as 69dec, “8” is stored as 56dec.

# C/Arduino data types

| type | size (bytes) | min value | | max value | |
|---|---|---|---|---|---|
| | | dec | hex | dec | hex |
| char | 1 | -128 | 0x80 | 127 | 0x7F |
| byte | 1 | 0 | 0x00 | 255 | 0xFF |
| int | 2 | -32768 | 0x8000 | 32767 | 0x7FFF |
| unsigned int | 2 | 0 | 0x0000 | 65535 | 0xFFFF |
| long | 4 | -2,147,483,648L | 0x80000000 | 2,147,483,647L | 0x7FFFFFFF |
| unsigned long | 4 | 0 | 0x00000000 | 4,294,967,295 | 0xFFFFFFFF |
| float (*double) | 4 | (+/-)3.4 x 10^38 | ~6 decimal | (+/-)1.2 x 10^-38 | |

* On most computers and some arduinos, doubles are 8

# ASCII cheat sheet

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

# Arrays

Strings:

- Strings are collections of characters that allow you to build messages
- Strings are NULL terminated which means that the last character is ‘\0’, the NULL character.
- A string for “Hello world!” would be,

| ‘H’ | ‘e’ | ‘l’ | ‘l’ | ‘o’ | ‘ ‘ | ‘w’ | ‘o’ | ‘r’ | ‘l’ | ‘d’ | ‘!’ | ‘\n’ | ‘\0’ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x72 | 0x65 | 0x6c | 0x6c | 0x6f | 0x20 | 0x77 | 0x6f | 0x72 | 0x6c | 0x64 | 0x21 | 0x0A | 0x00 |

- Example on the arduino

```
void loop()
{
  //Create an array of chars and load it with "Hello world?"
  char str[20] = "Hello world?";  //NULL character added automatically
  Serial.println(str); // prints "Hello world?" and a line feed

  str[11] = '!';  //change the '?' to a '!'
  str[12] = '\n';  //Add the '\n' char to add line feed ourselves
  //We just overwrote the \0 char, oops.

  //Add the NULL character since we just deleted it.
  str[13] = '\0';
  Serial.print(str); //prints "Hello world!!"
```

# Mathematical operations

- Assignment (=)
  - This operation takes what is on right of it and puts it in the variable to the left of it.
  - d = 2 + 3:  loads 5 into the variable d
- Basic math (+,-,*,/,%)
  - These are the add/subtract, multiply/divide and modulus operations you are use to.
- Compound operations (++,--,+=,-=,*=,/=)
  - These combine math operations with the assignment operator as a shorthand
  - a++  means  a = a + 1
  - a+=b means  a = a + b
  - a/=3.0  means  a = a / 3.0
- Example of mixed operations

```
float a,b,c;
a = 12;
b = 3.1;
c = (a * b) + 4; // 40.2
```

```
int d = 5;
d = d + 3; // d = 8
d++;       // d = 9
d *= 2;    // d = 18
d = d % 4;  // d = 2
```

**Gotchas on the next slide!**

## Floats are truncated to integers

```
float x = 0.1;

int i = x;          // 0
int i = x*9         // still 0
int i = x*10        // 1
```

## Floats are truncated to integers

```
float x = 10 * (50 / 500);   // 0
float y = 10.0 * (50/500);   // 0!
float y = 10 * (50.0/500);   // 1!
```

You must add ".0" to force arithmetic to be floating point.

By default, all integer arithmetic is truncated to 16 bits (-32768 to 32767)
If any intermediate result will overflow this range you must force a "long" (32-bit) calculation by appending "L"

```
float x = 0.1;

int i = 500 * 5000 / 1024;    // 9??
long j = 500 * 5000 / 1024;   // still 9??
int I = 500L * 5000 / 1024;   // 244 - yes!
```

# Logical operations

Logical operations:

- We can use integers to represent boolean values with the integer **0x0** being **false** and all other integers being **true**.
- There are shorthands in the arduino C of “true” and “false” for these boolean ints.
- Examples with a = true and b = false:

| symbol | meaning | example | result |
|---|---|---|---|
| && | logical and | a && b | false |
| \|\| | logical or | a \|\| b | true |
| ! | logical not | !a | false |
| == | logical equals | a == b | false |

**Common mistake! ( “=” is not the same as “==” )!!!!!!!!**

b == a compares b and a **returns false**

b = a means b is assigned the value of a, which is the value true **returns true!**

# Control Flow

From **IfStatementConditional** example

```
void loop() {
  // read the value of the potentiometer:
  int analogValue = analogRead(analogPin);

  // if the analog value is high enough, turn on the LED:
  if (analogValue > threshold) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin,LOW);
  }
```

From **switchCase** example

```
  // do something different depending on the
  // range value:
  switch (range) {
  case 0:    // your hand is on the sensor
    Serial.println("dark");
    break;
  case 1:    // your hand is close to the sensor
    Serial.println("dim");
    break;
  case 2:    // your hand is a few inches from the sensor
    Serial.println("medium");
    break;
  case 3:    // your hand is nowhere near the sensor
    Serial.println("bright");
    break;
  }
```

These types of statements tell our program how to change its behavior based on data.

**if - else if - else:**

- An **if** statement executes a block of code if a logical condition is true.
- If we want to execute another block of code if that statement isn't true, we add in an **else**.
- If we have multiple conditions, we can use the **else if** statement to organize the blocks.

**Switches:**

- A **switch** statement chooses a block of code to run based on the value of an integer.
- Each **case** should be followed by a **break** statement to end the block of code.
- Cases can be grouped together only one break if they share the same block of code.
- It is good practice to have a **default:** case to catch missing cases.

# Loops

From **ForLoopIteration** example

```
// loop from the lowest pin to the highest:
for (int thisPin = 2; thisPin < 8; thisPin++) {
  // turn the pin on:
  digitalWrite(thisPin, HIGH);
  delay(timer);
  // turn the pin off:
  digitalWrite(thisPin, LOW);
}
```

From **WhileStateConditional** example

```
// while the button is pressed, take calibration readings:
while (digitalRead(buttonPin) == HIGH) {
  calibrate();
}
```

**For loops:**

- Use these when you have a set number of things that need to be operated on in the same way.
  - processing arrays of data
  - processing groups of objects (left)
- Control uses 3 parts
  - initialization (how we start)
  - test condition (how we stop)
  - step (how we move forward)
- The code is placed between “{“ and “}”.

**While and do-while loops:**

- These loops continue until a condition is met.
- The condition is checked
  - before code (while loop)
  - after code (do-while loop)

# Functions

From **BarometricPressureSensor** example

```
void writeRegister(byte thisRegister, byte thisValue) {

  // SCP1000 expects the register address in the upper 6 bits
  // of the byte. So shift the bits left by two bits:
  thisRegister = thisRegister << 2;
  // now combine the register address and the command into one byte
  byte dataToSend = thisRegister | WRITE;

  // take the chip select low to select the device:
  digitalWrite(chipSelectPin, LOW);

  SPI.transfer(dataToSend); //Send register location
  SPI.transfer(thisValue);  //Send value to record into register

  // take the chip select high to de-select:
  digitalWrite(chipSelectPin, HIGH);
}
```

From **Ping** example

```
long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Why do we use functions?

- Functions allow us to break up our code into logical chunks.
- Organizing often used code in functions saves us from copy/pasting it

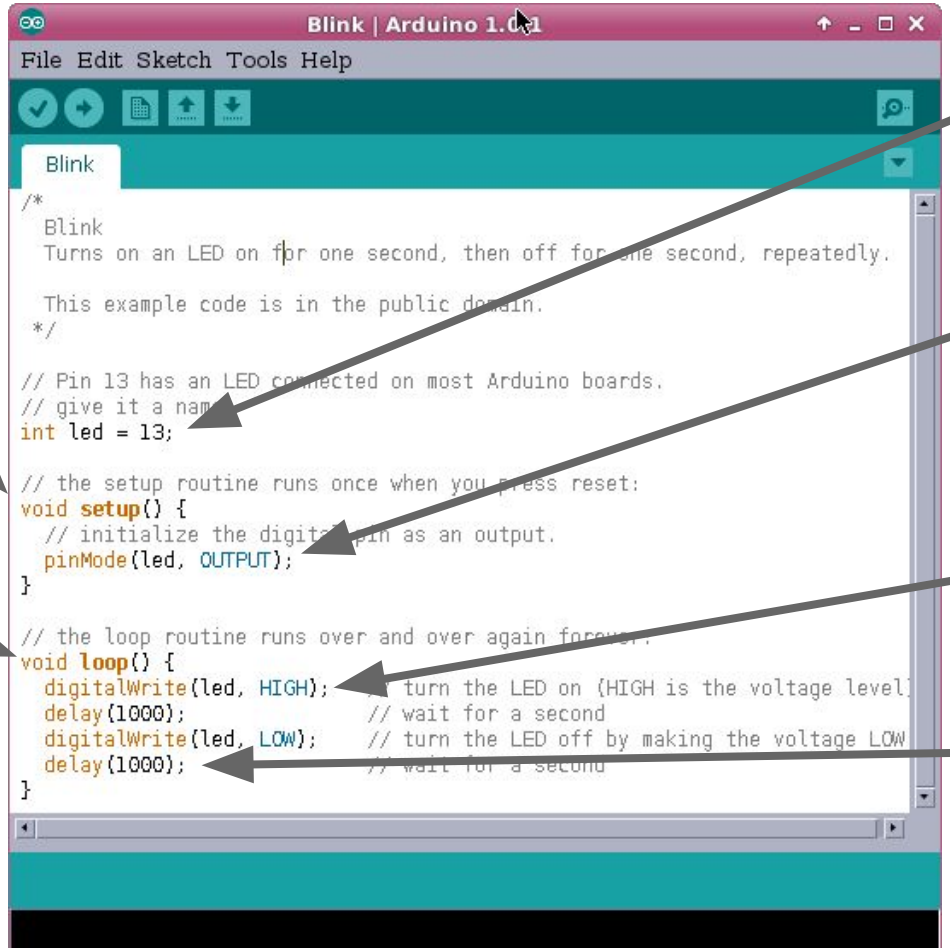
Syntax:

- Functions have:
  - A name
  - A list of values they expect
  - Code that manipulates those values
  - A value returned to the caller
- The code in the function is put between the “{“ and “}” characters.
- The function ends with a return statement or when it gets to the end of the code.
- This code has access to global variables and variables created inside the function.

# A basic program for the Arduino

Special **setup** function that is called once when the arduino starts.

Special **loop** function that is called over and over again forever.




Global variable

Library function that configures a I/O pin for output mode.

Library function that writes a ‘1’ to the led pin

Library function that causes the arduino to sleep for 1000 ms

# Arduino specific functions & libraries

**Special Arduino functions**

void setup(): This function is run once when the Arduino is powered up (or reset) and sets up the initial conditions for your code.

void loop(): This function is called after setup() has finished and is called over and over again forever.

**I/O port control:**

Digital:

- pinMode(pin,mode): This sets the **mode** (INPUT/OUTPUT) of the **pin** on the arduino.
- digitalWrite(pin, value): This writes the **value**(HIGH/LOW) to **pin** on the arduino.
- digitalRead(pin): This returns the value of the **pin** on the arduino.

Analog:

- analogRead(pin): Read the analog value on **pin** and return the value (0 - 1024).
- analogWrite(pin,value):Write (PWM) the **value** to **pin**.

**Useful libraries:**

Serial: Used to communicate with your computer over a USB cable and a terminal program. See the SerialCallResponse example under communication.

# Bitwise Operations

- Remember, data types are fundamentally made out of bits.
- There are special commands to do operations directly on these bits.

| symbol | meaning | example | result |
|---|---|---|---|
| & | and | 0b0111 & 0b0100 | 0b0100 |
| \| | or | 0b0011 \| 0b1010 | 0b1011 |
| ^ | xor | 0b0011 ^ 0b1010 | 0b1001 |
| ~ | complement | ~0b0101 | 0b1010 |
| << | left shift | 0b0010 << 2 | 0b1000 |
| >> | right shift | 0b1000 >> 1 | 0b0100 |

- These are used when  we want to assemble read bits into a byte or get a bit out of a byte for writing.

# 10hz Binary counter with LED outputs

```
// pins 8, 9, 10, 11 are LED outputs

void setup() {
    for( int i=8; i<=11; i++)
        pinMode( i, OUTPUT);
}

int count = 0;
void loop() {
    count = count + 1;                  // increment our counter
    for( int b=0; b<4; b++) {           // loop over bits
        int mask = 1<<b;                // mask = 1, 2, 4, 8
        if( count & mask)               // test one bit in count
            digitalWrite( 8+b, HIGH);
        else
            digitalWrite( 8+b, LOW);
    }
    delay(100);                         // delay 100ms
}
```

# Variable scope

```
int n = 12;           // global variable

void loop() {
  int z = 3;          // "automatic" variable

  static int cnt;     // static variable

  for( int i=0; i<5; i++)
    …

}
```

Visible in any function
Initialized to zero by default.
Retains value indefinitely.

Visible only inside { }
where it is declared.
*Not Initialized by default!*
*Does not retain value*
*(created anew on each function call)*

Visible only inside { }
where it is declared.
Initialized to zero by default
*Retains value* across function calls

# Go have fun with your Arduinos!

- If you haven't already, please pick up an Arduino pack from Prof. Sulak, one of the TA's, or me.
- Your homework over break is to install the Arduino software (http://arduino.cc/en/Main/Software) and get the blinky example working on your Arduino.
- You can find it along with many other examples by going to File->Examples in the Arduino IDE.
- Please change the frequency of your blinky program to make sure you have everything working.
- If you need help, feel free to talk to the TAs or myself or check the arduino reference website: http://arduino.cc/en/Reference/HomePage .

# Homework

| dec (unsigned) | hex | binary | val >>3 | val &0x3 | val \| 0x80 | final dec |
|---|---|---|---|---|---|---|
| 123 | 0x7B | 0b01111011 | 0x0F | 0x03 | 0x83 | 131 |
| 255 | | | | | | |
| | 0x80A5 | | | | | |
| | | 0b11100111 | | | | |
| -128 *(signed int) | | | | | | |

**Work these out on paper first and then write code to check them.**

*hint: ints are 2 bytes long and http://en.wikipedia.org/wiki/Two%27s_complement