



Introduction to Arduino and C Programming

Presentation by:
Shamim Ahmed

Assumptions and Goals

Assumptions

- You have taken at least one programming course prior

Goal

- Be able to create applications on arduino using the following methods
 - Adding Shields (Accessories; Attachments)
 - Adding Libraries
 - Using control flows (If-Else, Do While, While loop)
 - Using functions (Methods: e.g. getVariable, setVariable, changeVariables)
 - Reading connection diagrams
 - Debugging applications

Why

- 495 will not be a success if the hardware does not complete its mission objectives. Faculty are eager to see a working model. To improve something and make it work, you first must understand it



Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor
Control Flow
Loops
Arrays & Strings
Functions, Structs, and Unions
Libraries/IO/Connection Diagrams/EEPROM
Group Activities, Post-490 Planning

Arduino Models

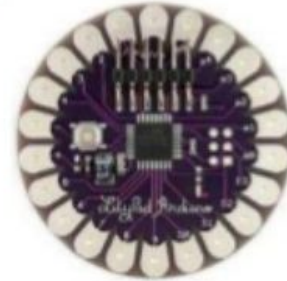
Arduino's are microcontrollers: mini-computers with their own memory, disk, and processor



UNO



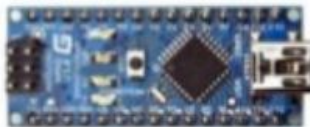
Mega



LilyPad



Arduino BT



Arduino Nano



Arduino Mini

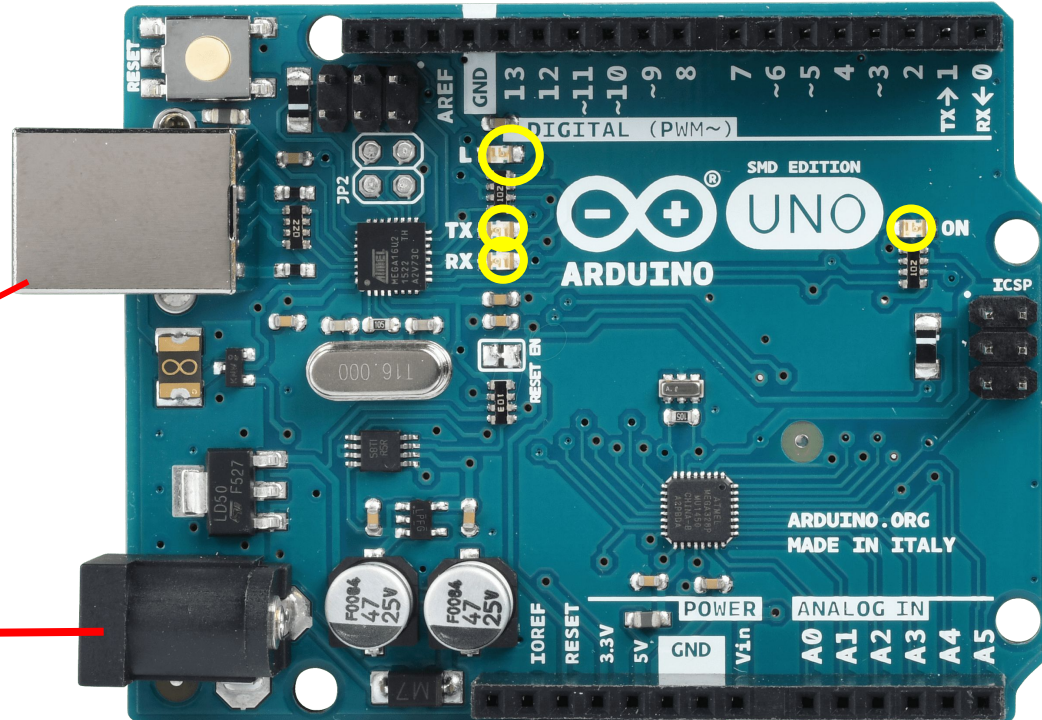
Lights on an Arduino (Arduino UNO)

4 Lights on an Arduino

1. ON Light - shows that it is powered on
2. TX Light - shows that data is being transmitted
3. RX Light - shows that data is being received
4. L Light - an LED you are able to control in your program

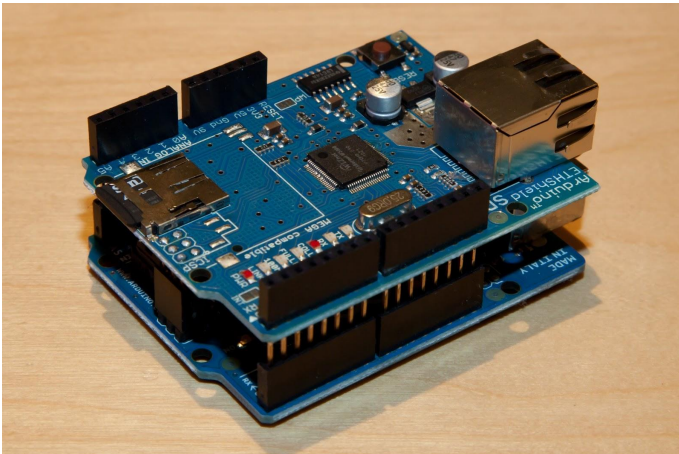
Serial Port (USB)

External Power Source (AA, AAA Batteries)



Arduino Accessories & Shields

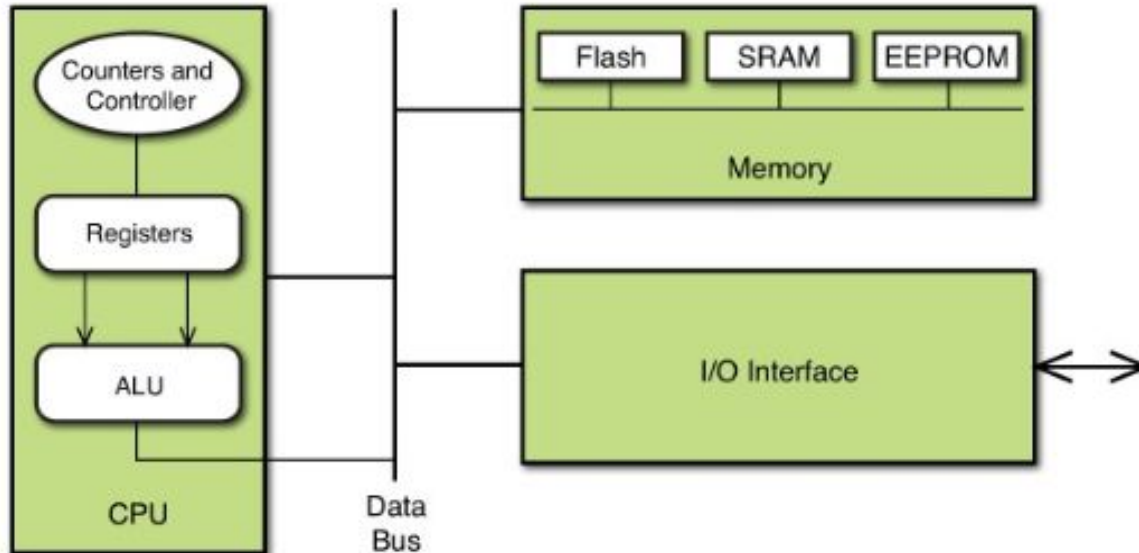
- Accessories include USB A-B cable, external power source, breadboard, resistors, variable resistors (potentiometers), switches, wires, sensors, motors
- Shields” are add ons or accessories that extend the functionality of Arduino. The code is already written for them
 - Ethernet Shields
 - LCD Shields
 - Motor Shields extends the number of motors you can use on an arduino from 6
 - Prototype Shield - for circuit development rather than soldering
 - Use breadboard as an alternative to this shield
 - There are many more shields including bluetooth, wireless, etc.



battery or AC/DC converter to run without being connected to a computer



Basic components of the Arduino Microcontroller





Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor
Control Flow
Loops
Arrays & Strings
Functions, Structs, and Unions
Libraries/IO/Connection Diagrams/EEPROM
Group Activities, Post-490 Planning

The Arduino IDE

- You can retrieve the IDE from the main arduino website (arduino.cc)
- The IDE is written in Java; however, the arduino only accepts programs written in C. Therefore you must program in C. The IDE acts as a C Compiler.

Tip:
1. Use Auto-Format to clean your code spacing
2. Use Serial Plotter to see Arduino Output

Verify: Checks if your program compiles (syntax check)

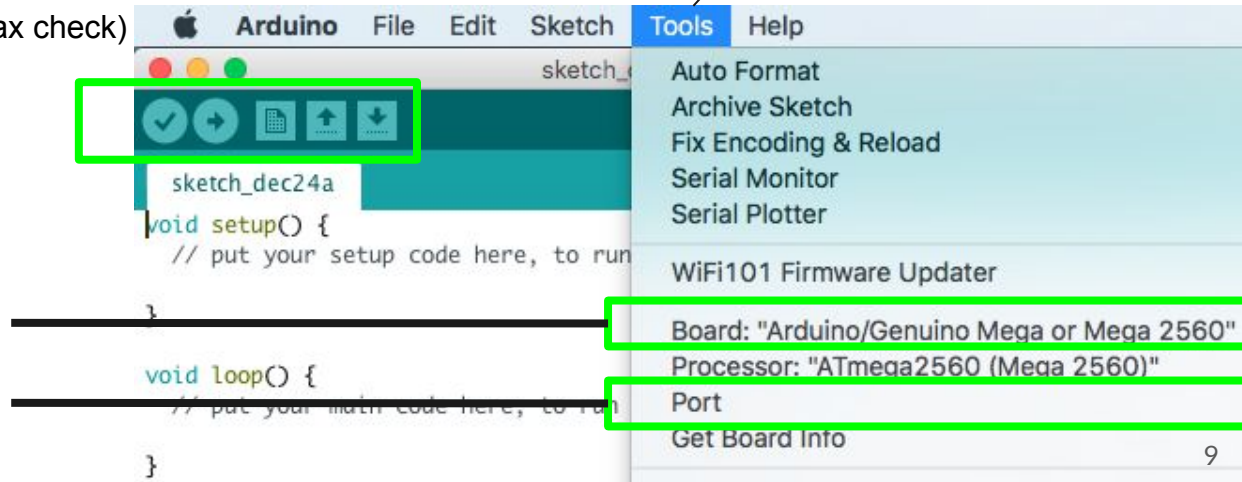
Upload: Uploads your program to the Arduino.

New: Creates a new program file

Open: Open an existing arduino program file

Save: Allows you to save the current program

Must Choose Appropriate Arduino Board before uploading programs & choose the port on the *computer* the arduino is connected to



How are Arduino Programs Structured

```
void setup() {  
Code to run once  
}
```

```
void loop(){  
Code to run repeatedly  
}
```

Title of
Program



```
sketch_dec24a | Ardui  
✓ → [Icons] [Icons]  
sketch_dec24a  
void setup() {  
  // put your setup code here, to run once:  
  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  
}
```

- Programs get saved in Documents/Arduino on your workstation

Declaring and Instantiating Variables

Declaring a Variable

dataType *variableName*;

Example:

```
int year;
```

Instantiating a Variable

Add equals sign

Example:

```
int year;  
year = 2017;
```

Data Type	Size (Bytes)	Value Range
boolean	1	Logic true or false
char	1	-128 to +127
byte	1	0 to 255
int	2	-32,768 to 32,767
word	2	0 to 65,535
long	4	-2,147,483,648 to 2,147,483,647
float	4	-3.4028235E+38 to 3.4028235E+38
double	4	-3.4028235E+38 to 3.4028235E+38

Declaring and Instantiating Simultaneously

Example:

```
int year = 2017;
```

For Constants

Add 'const' before dataType

Example:

```
const float pi = 3.14;
```

Scope of Variables

- Variable scope determines where the variable can be used in the sketch. There are two variable scopes
 - Local Variables
 - Can only be used inside a function, and only appear inside that function block.
 - You won't see a local variable from the setup function in the loop function
 - Global Variables (Static Variables)
 - Can be used in ALL functions
 - Common to see them at the start of the sketch before the setup function

Math Operators

Standard Operators are built-in for use and can be used with variables. Two examples below:

```
int x;  
float y;  
int z;  
x = 5;  
y = 2.7;  
z = x+y;  
What is z equal to above?
```

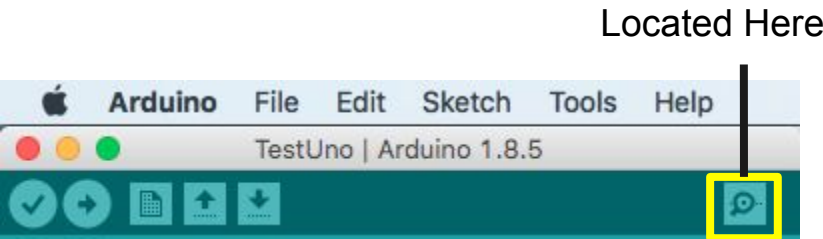
```
int x = 5;  
float y = 2.7;  
float z = x+y;  
What is z equal to above?
```

Tip: instead of `x = x + 1;` you can write `x += 1;`

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
!	Logical NOT
&&	Logical AND
	Logical OR
&	Bitwise AND
	Bitwise OR
<<	Left shift
>>	Right shift

Using the Serial Monitor

The serial monitor allows you to see the output from the program



Located Here

```
void setup() {  
  Serial.begin(9600);  
}
```

```
void loop() {  
  int x = 5;  
  float y = 2.56;  
  float sum = x+y;  
  Serial.println(sum);  
  delay(1000);  
}
```

1. insert `Serial.begin(baudRate);` to initialize the serial port

baudRate = Speed of Connection (higher is faster; must match workstation baud). // i want that baud

default baud rate is 9600;

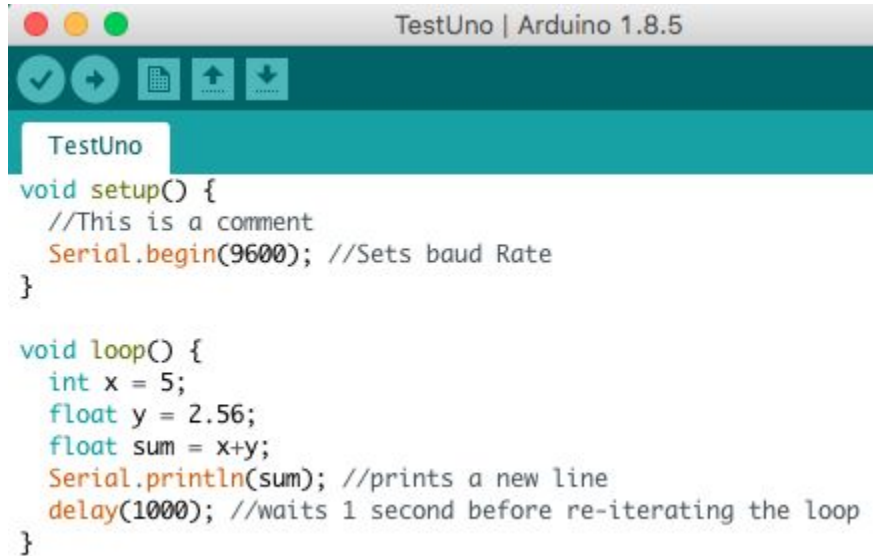
2. Printing to the serial monitor:

`Serial.print(sum)` // does not start a new line
`Serial.println(sum)` //starts a new line


3. Working with time

- `delay(x)`: pauses the sketch for x milliseconds
- `delayMicroseconds(x)`: pauses the sketch for x microseconds
- `micros()`: Returns the number of microseconds since the arduino was reset
- `millis()`: returns the number of milliseconds since the Arduino was reset

Output on the Serial Monitor



```
TestUno | Arduino 1.8.5  
TestUno  
void setup() {  
  //This is a comment  
  Serial.begin(9600); //Sets baud Rate  
}  
  
void loop() {  
  int x = 5;  
  float y = 2.56;  
  float sum = x+y;  
  Serial.println(sum); //prints a new line  
  delay(1000); //waits 1 second before re-iterating the loop  
}
```



```
/dev/cu.usbmodemFA1  
7.56  
7.56  
7.56  
7.56  
7.56  
7.56  
7.56  
7.56  
7.56  
7.56
```

Tip: use `//` to place a comment. Examples above

Note: You cannot concatenate the Serial output

Ex.

`// this won't work`

`Serial.println("The sum is" + sum);`

Advanced Math Functions

Example

```
int Temperature = -7;  
int value = abs(temperature);  
Serial.println(value);
```

What does it print above?

Function	Description
<code>abs(x)</code>	Returns the absolute value of x
<code>constrain(x, a, b)</code>	Returns x if x is between a and b (otherwise returns a if x is lower than a , or b if x is higher than b)
<code>cos(x)</code>	Returns the cosine of x (specified in radians)
<code>map(x, fromLow, fromHigh, toLow, toHigh)</code>	Remaps the value x from the range $fromLow$ to $fromHigh$ to the range $toLow$ to $toHigh$
<code>max(x, y)</code>	Returns the larger value of x or y
<code>min(x, y)</code>	Returns the smaller value of x or y
<code>pow(x, y)</code>	Returns the value of x raised to the power of y
<code>sin(x)</code>	Returns the sine of x (specified in radians)
<code>sqrt(x)</code>	Returns the square root of x
<code>tan(x)</code>	Returns the tangent of x (specified in radians)

Note: The `map()` and `constrain()` functions are mostly used with sensors. They allow you to keep values returned by the sensors within a specific range that your sketch can manage

Generating Random Numbers

Two functions are available for working with random numbers. `random()` and `randomSeed()`

`random(min, max)` : returns a random number between min and max -1

`random(max)` : returns a random number between 0 and max -1

`randomSeed(seed)`: Initializes the random number generator, causing it to restart at an arbitrary point in random sequence.

```
//this variable will hold a random number generated by the random() function
long randomNumber;

//Set up - this is where you get things "set-up". It will only run once
void setup() {

  //setup serial communications through the USB
  Serial.begin(9600);

  //Let's make it more random
  randomSeed(42);

} //close setup

void loop() {

  //generate a random number
  randomNumber = random(2,5);

  //display the random number on the serial monitor
  Serial.print("The Random Number is = ");
  Serial.println(randomNumber);

}
```



Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor

Control Flow

Loops

Arrays & Strings

Functions, Structs, and Unions

Libraries/IO/Connection Diagrams/EEPROM

Group Activities, Post-490 Planning

if control

```
if (condition) {  
    Statement 1;  
    Statement 2;  
    etc.  
}
```

Example

```
if (temperature > 100) {  
    Serial.println("wow!")  
    Serial.println("that's hot!")  
}
```

if/else control

```
if (condition) {  
    Statement 1;  
    Statement 2;  
    etc.  
}  
else {  
    Statements;  
}
```

Example

```
if (grade > 92) {  
    myGrade = 'A';  
} else {  
    myGrade = 'F';  
}
```

if/else if control

```
if (condition)  
    Statement;  
else if (condition)  
    Statement;  
else if (condition)  
    Statement;  
else  
    Statement;
```

Example

```
if (grade > 92) {  
    myGrade = 'A';  
} else if (grade > 83)  
    myGrade = 'B';  
else  
    myGrade = 'C';
```

Numeric Comparisons

Operator	Description
==	Equal
!=	Not equal
<>	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

Compound Conditions

Example 1

```
If ((a == 1) || (b == 1)){  
    statements;  
}
```

Example 2

```
If ((a == 1) && (b == 2)) {  
    statements;  
}
```

Negating a condition check

```
int a == 1;  
If (!(a == 1))  
    Serial.println("The 'a' variable is not  
equal to 1");  
if(!(a ==2))  
    Serial.println("The 'a' variable is not  
equal to 2");
```

Or just use
!= in the
condition 20

Using a switch statement

Instead of doing a lot of if and else statement, it may be useful to do a switch

Format

```
switch (var) {  
  case 23:  
    //do something when var equals 23  
    break;  
  case 64:  
    //do something when var equals 64  
    break;  
  default:  
    // if nothing else matches, do the default  
    // default is optional  
    break;  
}
```

Example

```
switch (grade) {  
  case 'A':  
    Serial.println("you got higher than a 92");  
    break;  
  case 'B':  
    Serial.println("You got higher than a 80");  
    break;  
  default:  
    Serial.println("You have dishonored the family");  
    break;  
}
```



Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor
Control Flow

Loops

Arrays & Strings

Functions, Structs, and Unions

Libraries/IO/Connection Diagrams/EEPROM

Group Activities, Post-490 Planning

for

```
for (statement1; condition; statement 2){  
    Code statements  
}
```

executes until condition is met

Example

```
int values[5] = {10, 20, 30, 40, 50};  
for (int counter = 0; counter < 5; counter++){  
    Serial.print("one value in the array is ");  
    Serial.println(values[counter]);  
}
```

while

```
while (condition){  
    Code statements  
}
```

executes until condition is met

Example

```
int i = 0;  
while (i<3) { //hearts for days  
    Serial.println(" i is : " i);  
    i++;  
}
```

do while

```
do {  
    Code statements  
} while (condition);
```

executes at least once, and
until condition is met

Example

```
int i = 0;  
Serial.println("Hi");  
do {  
    Serial.println("my name is");  
    if(i==0) Serial.println("What");  
    if(i==1) Serial.println("Who");  
    i++;  
} while (i < 2);  
Serial.println("slicka chika slim shady");
```

Loops Continued

Using Multiple Variables

You can initialize multiples variables in a for statement

Example

```
int a,b;
For (a = 0, b = 0; a < 10; a++, b++){
    Serial.print("One value is ");
    Serial.println(a);
    Serial.print(" and the other value is ");
    Serial.println(b);
}
```

Nesting Loops

You can place loops inside of another loop. The trick to using inner loop sis that you must complete the inner loop before you complete the outer loop.

Example

```
int a,b;
for (a = 0; a < 10; a++){
    for (b = 0; b < 10; b++){
        Serial.print("one value is ");
        Serial.print(a);
        Serial.print("and the other value is ");
        Serial.println(b);
    }
}
```


Controlling Loops

Break Statement

- You can use the break statement when you need to break out of a loop before the condition would normally stop the loop

Example:

```
int i;
for (i = 0; i <= 20; i++) {
    if (i == 15)
        Break;
    Serial.print("currently on iteration:");
    Serial.println(i);
}
Serial.println("This is the end of the test");
}
```

Continue Statement

- You can use the continue statement to control loops. Instead of telling the Arduino to jump out of a loop, it tells the Arduino to stop processing code inside the loop, but still jumps back to the start of the loop

Example

```
int i;
for (i = 0; i <= 10; i++){
    If ((i > 5) && (i < 10))
        Continue;

    Serial.print("The value of the counter at");
    Serial.println(i);
}
Serial.println("this is the end of the test");
```



Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor
Control Flow
Loops
Arrays & Strings
Functions, Structs, and Unions
Libraries/IO/Connection Diagrams/EEPROM
Group Activities, Post-490 Planning

Arrays

An array stores multiple data values of the same data type in a block of memory, allowing you to reference the variables using the same variable name. It does it through an index value.

Format

datatype variableName[size];

Example 1:

```
int myarray[10]; //able to store 10 values  
myarray[0] = 20; //stores values in index 0  
myarray[1] = 30; // stores values in index 1
```

Example 2:

```
// assigns values to first 5 data locations (index 0-4)  
int myarray[10] = {20, 30, 40, 50, 100};
```

Example 3

```
int myarray[ ] = {20, 30, 40, 50 100};
```

Using Loops with Arrays

```
int values[5] = {10, 20, 30, 40, 50};  
for (int counter = 0; counter < 5; counter++){  
    Serial.print("one value in the array is " );  
    Serial.print(values[counter]);  
}
```

Arrays Continued

Determining the size of an Array

- You may not remember how many data points are in your array
- You can use the handy `sizeof` function

```
size = sizeof(myArray) / sizeof(int);
```

Example:

```
for (counter = 0; counter < (sizeof(value)/sizeof(int)); counter++){  
//This will only iterate through number of points in an array  
statements;  
}
```

Challenge Question 1: What is the array 'myArray' look like in the program below?

```
int myArray[3][4];  
for (int i = 0; counter < 3; i++){  
    for (int j = 0; j < 4; i++){  
        myArray[ i ][ j ] = 1;  
    }  
}
```

Challenge Question 2: What is syntactically wrong with the array below? How do i fix it?

```
char myArray[ ] = { "mon", "tue", "wed", "thu", "fri"};
```

Strings

A string value is a series of characters put together to create a word or sentence

Format

```
String name = "my text here";
```

Example

```
String myName = "Jackie Chan";
```

Manipulating Strings

```
String myName = "Jackie Chan";  
myName.toUpperCase();
```

Output: JACKIE CHAN

Method	Description
<code>charAt(<i>n</i>)</code>	Returns the character at the <i>n</i> th position in the string.
<code>compareTo(<i>string2</i>)</code>	Returns 0 if the string is equal to <i>string2</i> , a negative number if the string is less than <i>string2</i> , or a positive number if the string is greater than <i>string2</i> .
<code>concat(<i>string1</i>, <i>string2</i>)</code>	Appends the <i>string2</i> value to the end of <i>string1</i> , and creates a new string value.
<code>endsWith(<i>string2</i>)</code>	Returns <code>true</code> if the string ends with the <i>string2</i> value.
<code>equals(<i>string2</i>)</code>	Returns <code>true</code> if the string is equal to <i>string2</i> .
<code>equalsIgnoreCase(<i>string2</i>)</code>	Returns <code>true</code> if the string is equal to <i>string2</i> , ignoring character case.
<code>getBytes(<i>buf</i>, <i>len</i>)</code>	Copies <i>len</i> string characters into the <i>buf</i> variable.
<code>indexOf(<i>val</i>, [<i>from</i>])</code>	Returns the index location where the string <i>val</i> starts in the string. By default, it starts at index 0, or you can specify a starting location using the <i>from</i> parameter. Returns <code>-1</code> if <i>val</i> is not found in the string.
<code>lastIndexOf(<i>val</i> [, <i>from</i>])</code>	Returns the index location where the string <i>val</i> starts in a string. By default, it starts at the end of the string, working toward the front of the string, or you can specify a starting location using the <i>from</i> parameter. Returns <code>-1</code> if <i>val</i> is not found in the string.

Although you can just create a char array, Strings are much easier to work with! They have many more supported functions

More String Functions

`length()`

Returns the number of characters in the string (not counting the terminating null character).

`replace(substring1, substring2)`

Returns a new string with the `substring1` value with `substring2` in the original string value.

`reserve(n)`

Reserves a space of `n` characters in memory for a string value.

`startsWith(string2)`

Returns `true` if the string starts with `string2`.

`substring(from [, to])`

Returns a substring of the original string value, starting at the `from` index location. By default, it returns the rest of the string from that location, or you can specify the `to` index value.

`toCharArray(buf, len)`

Copies `len` characters in the string to the character array variable `buf`.

`toInt()`

Returns an integer value created from the string value. The string must start with a numeric character, and the conversion stops at the first non-numeric character in the string.

`setCharAt(index, c)`

Replaces the character at `index` with the character `c`.

`toLowerCase()`

Converts the string value to all lowercase letters.

`toUpperCase()`

Converts the string value to all uppercase letters.

`trim()`

Removes any leading and trailing space or tab characters from the string value. 30



Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor
Control Flow
Loops
Arrays & Strings
Functions, Structs, and Unions
Libraries/IO/Connection Diagrams/EEPROM
Group Activities, Post-490 Planning

Functions

You'll often find yourself using the same code in multiple locations. Doing large chunks of these is a hassle. However, functions make these a lot easier. You can encapsulate your C code into a function and then use it as many times as you want.

Structure

```
datatype functionName() {  
    // code statements  
}
```

To create a function that does not return any data values to the calling program, you use the **void** data type for the function definition

```
Void myFunction() {  
    Serial.println("This is my first function");  
}
```

TIP: Make sure you define your function outside of the setup and loop functions in your arduino code sketch. If you define a function inside another function, the inner function becomes a local function, and you can't use it outside the outer function

Using the function/Returning a value

Using the Function

To use a function you defined in your sketch, just reference it by the function name you assigned, followed by parentheses

```
void setup() {  
  Serial.begin(9600)  
  MyFunction();  
  Serial.println("Now we're back to the main program");  
}
```

Returning a Value

To return a value from the function back to the main program, you end the function with a return statement

```
return value;
```

The value can either a constant numeric or string value, or a variable that contains a numeric or string value. However, in either case, the data type of the returned value **must match** the data type that you use to define the function

```
int myFunction2() {  
  int value = 10*20;  
  return (value);  
}
```

Passing Values to Functions

You will most likely want to pass values into function. In the main program code, you specify the values passed to a function in what are called arguments, specific inside the function parenthesis

```
returnValue = area(10,20);
```

The 10 and 20 are value arguments separated by a comma. To retrieve the arguments passed to a function, the function definition must declare what are called parameters. You do that in the main function declaration line

```
void setup() {
```

```
    int returnValue;
```

```
    Serial.begin(9600);
```

```
    Serial.print("The area of a 10 x 20
```

```
size room is ");
```

```
    returnValue = area(10,20);
```

```
    Serial.println(returnValue);
```

```
}
```

Arguments

```
void loop() {
```

```
}
```

Parameters

```
int area (int width, int height) {
```

```
int result = width * height;
```

```
Return result;
```

```
}
```



Handling Variables inside Functions

One thing that causes problem for beginning sketch writers is the scope of a variable. The scope is where the variable can be referenced within the sketch. Variables defined in function can have a different scope than regular variables. That is, they can be hidden from the rest of the sketch. Functions use two types of variables:

Global Variables

Local Variables

Defining Global Variables

Write them before the `setup()` loop. Ex:

```
const float pi = 3.14;
```

Be careful in modifying global variables.

Declaring local variables

Local variables are declared in the function code itself, separate from the rest of the sketch code. What's interesting is that a local variable can override a global variable (but not good practice)

Calling Functions Recursively

Recursion is when the function calls itself to reach an answer. Usually a recursion function has a base value that it eventually iterates down to. Many advanced algorithms use recursion to reduce a complex equation down one level repeatedly until they get to the level defined by the base value.

```
int factorial (int x) {  
    if (x <=1) return 1;  
    else return x * factorial(x-1);  
}
```

Remember you are allowed to call other functions from inside a function

Structs

Data structures allow us to define custom data types that group related data elements together into a single object.

Before you can use a data structure in your sketch, you need to define it. To define a data structure in the Arduino, you can use the *struct* statement. Here is the generic format for declaration

Format

```
struct name {  
    variable list  
};
```

Example of declaration

```
struct sensorinfo {  
    char date[9];  
    int indoortemp;  
    int outdoortemp;  
}morningTemps, noonTemps, eveningTemps;
```

Full Example: Declaring and Calling

```
struct sensorinfo {  
    char date[9];  
    int indoortemp;  
    int outdoortemp;  
}morningTemps
```

```
void setup() {  
    // put your setup code here, to run once:  
    Serial.begin(9600);  
    strcpy(morningTemps.date, "01/01/14");  
    morningTemps.indoortemp = 72;  
    morningTemps.outdoortemp = 25;  
    Serial.print ("Today's date is ");  
    Serial.println(morningTemps.date);  
    Serial.print("The morning outdoor temperate  
is ");  
    Serial.println(morningTemps.outdoortemp);  
}
```

Unions

Unions allow you to have a variable take on different data types

Format

```
union {  
    //variable list  
};
```

Example

```
Union {  
    float analogInput;  
    int digitalInput;  
} sensorInput;
```

//Full Example:

```
Union{  
    float analogInput;  
    Int digitalInput;  
}sensorInput;
```

//Saving a value

```
sensorInput.analogInput = myFunction1();  
sensorInput.digitalInput = myFunction2();
```

//Calling a value;

```
Serial.println(sensorInput.analogInput);  
Serial.println(sensorInput.DigitalInput);
```

Using Libraries

Libraries allow you to bundle related functions into a single file that the Arduino IDE can compile into your sketches. Instead of having to rewrite the functions in your code, you just reference the library file from your code, and all the library functions become available. This is handy for Arduino Shields

Defining the library in your sketch

```
#include 'libraryheader'  
#include 'EEPROM'
```

Referencing the Library Functions

```
Library.function()
```

Ex. for the EEPROM library
EEPROM.read(0);

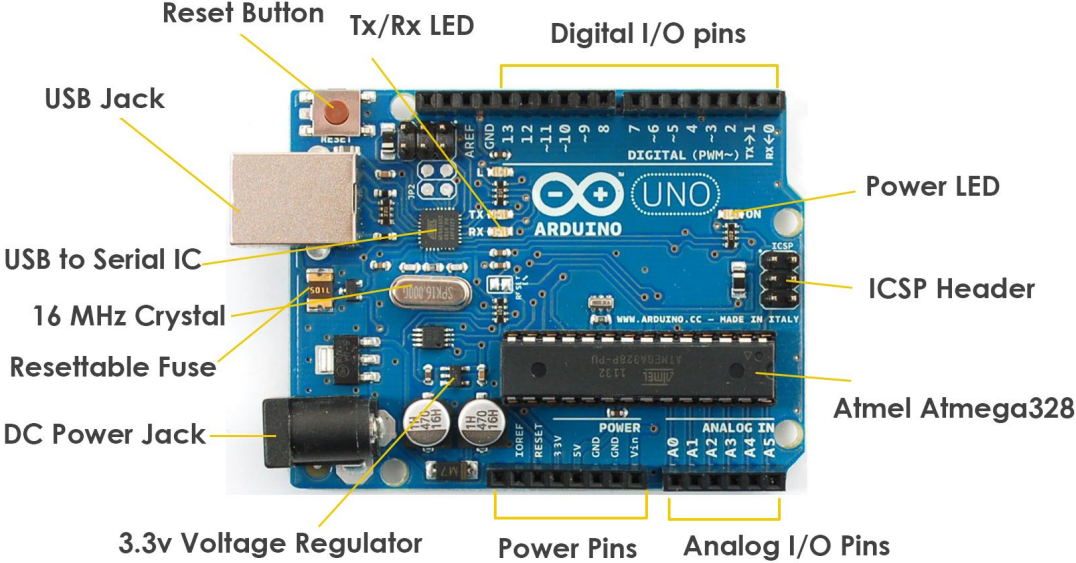
Installing your library

1. Open the Arduino IDE
2. Select the sketch from the menu bar
3. Select Import libraries from the submenu
4. Select Add library from the list of menu options

Library	Description
EEPROM	Functions to read and write data to EEPROM memory.
Esplora	Functions for using the game features of the Esplora unit.
Ethernet	Functions for accessing networks using the Ethernet shield.
Firmata	Functions for communicating with a host computer.
GSM	Functions for connecting to mobile phone networks using the GSM shield.
LiquidCrystal	Functions for writing text to an LCD display.
Robot_Control	Functions for the Arduino Robot.
SD	Functions for reading and writing data on an SD card.
Servo	Functions for controlling a servo motor.
SoftwareSerial	Functions for communicating using a serial port.
SPI	Functions for communicating across the SPI port.
Stepper	Functions for using a stepper motor.
TFT	Functions for drawing using a TFT screen.
Wifi	Functions for accessing a wireless network interface.
Wire	Functions for communicating using the TWI or I2C interfaces.

Sample libraries
already installed
for call and use

Input/Output Layout on Arduino



Digital & Analog I/O

Format

```
pinMode(pin, MODE);
```

The pinMode function requires two parameters. The pin parameter determines the digital interface number to set. The mode parameter determines whether the pin operates input or output mode. There are 3 values for interface mode setting:

INPUT - To set an interface for normal input mode

OUTPUT - To set an interface for output mode

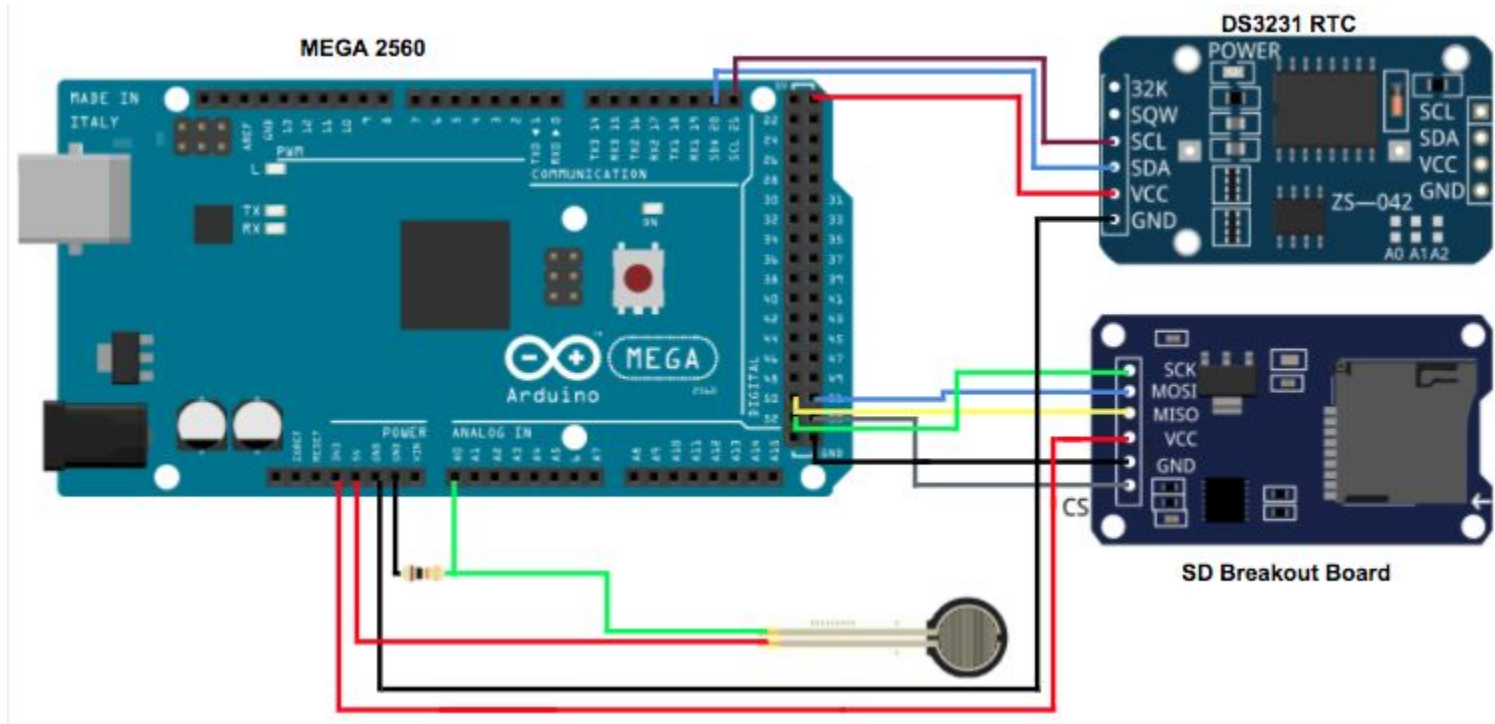
Getting a reading from an input device:

```
analogRead(pin)
```

Complete Example

```
void setup (){  
    Serial.begin(9600);  
    pinMode(A1, INPUT);  
}  
  
void loop() {  
    float reading = analogRead(A1);  
}
```

Reading Connection Diagrams



Writing to EEPROM

EEPROM is the long term memory in the Arduino. It keeps data stored even when powered off, like a USB flash drive.

Important Note: EEPROM becomes unreliable after 100,000 writes

You'll first need to include a library file in your sketch

```
#include <EEPROM.h>
```

After you include the standard EEPROM library file, you can use the two standard EEPROM functions in your code:

read (address) - to read the data value stored at the EEPROM location specified by address

write (address, value) - to read values to the EEPROM location specified by address

```
#include <EEPROM.h>
```

```
void setup()
```

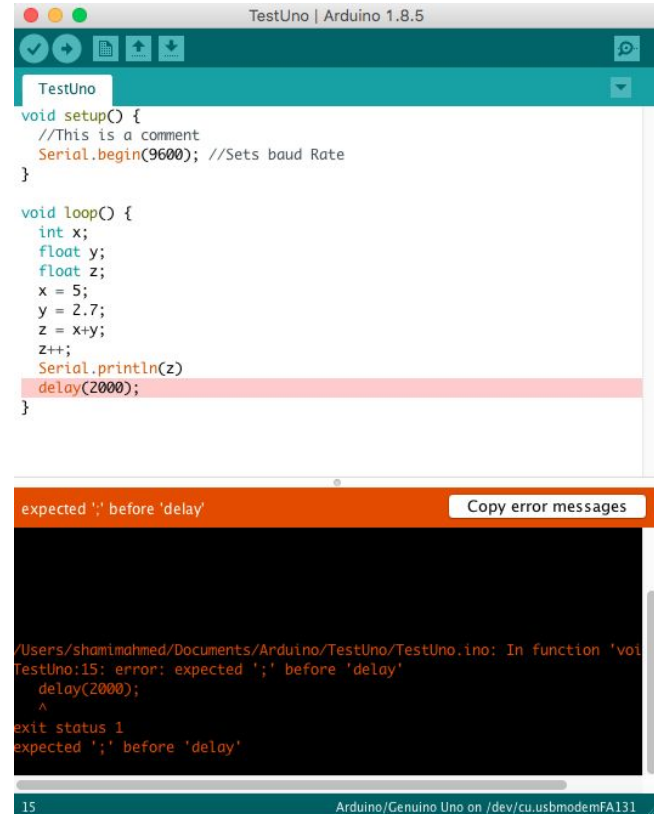
```
{  
  for (int i = 0; i < 255; i++)  
    EEPROM.write(i, i);  
}
```

```
void loop()
```

```
{  
}
```

Debugging Applications

- Compiler will give you the line code(s)
- Compiler will give you the error
- The line causing problems may get highlighted
- Look up errors online



```
TestUno | Arduino 1.8.5
TestUno
void setup() {
  //This is a comment
  Serial.begin(9600); //Sets baud Rate
}

void loop() {
  int x;
  float y;
  float z;
  x = 5;
  y = 2.7;
  z = x+y;
  z++;
  Serial.println(z)
  delay(2000);
}

expected ';' before 'delay' Copy error messages

/Users/shamimahmed/Documents/Arduino/TestUno/TestUno.ino: In function 'void
TestUno:15: error: expected ';' before 'delay'
  delay(2000);
  ^
exit status 1
expected ';' before 'delay'
```

15 Arduino/Genuino Uno on /dev/cu.usbmodemFA131



Agenda

Introduction to the Arduino Hardware
IDE, Variables, Operands, and the Serial Monitor
Control Flow
Loops
Arrays & Strings
Functions, Structs, and Unions
Libraries/IO/Connection Diagrams/EEPROM
Group Activities, Post-490 Planning

Group Practice Problems

1. Have the Arduino to create and save an excel sheet onto an SD card. Make the first row have these cells in order:

"Sequence", "Day", "Month", "Date", "Year", "Hour", "Minute", "Second", "Temp F", "Sensor 1", "Sensor 2";

Tip: Look into the library SD.h in your IDE. There are examples of how to use each library given by the authors (IDE>Files>Examples). Save the file as demo.csv. Use this [link](#) for more information on connection setup.

2. Configure the time on the Arduino DS3231 board to the current time and date and save it. Then, create a program that gets the current month, day, year, hour, minute, second, and temperature. Set the delay to 3000 milliseconds

Tip: Search for and download the Library Sodaq_DS3231.h **from your IDE**. It is not a pre-installed library but you can easily find it by searching it in your IDE library manager. There are examples of how to use each library given by the authors (IDE>FILES>Examples).