Day1:

Circuit 1: Spaceship Interface	
Digital Ouput:	**Digital Input:**
Output: Sends current out to attached devices. **Digital Output** can only be either on all the way or off all the way (HIGH or LOW). Examples: LED lights, motors	**Input**: Data is entered into the Arduino through input devices (keyword, sensors, etc). **Digital Input** can only be either on or off; no range of values. Examples: switches, buttons
Setup Pins for Output:	**Setup Pins for Input:**

void setup() {
  pinMode(pinNumber, OUTPUT);
}
	void setup() {
	  pinMode(pinNumber, INPUT);
	}

Pin Modes:		
OUTPUT	INPUT	INPUT_PULLUP

pinMode(): a function that sets a pin's mode. OUTPUT, INPUT, and INPUT_PULLUP have predefined values understood to represent their respective modes.

Turning on LED light:	**Taking Digital Input**:

void loop() {
  digitalWrite(ledPin, HIGH);
}
	void loop() {
	  int sensorState = digitalRead(sensorPin);
	}

Turning off LED light:	**Digital Input for Buttons**:

void loop() {
  digitalWrite(ledPin, LOW);
}
	LOW when pressed. (connected to ground)
	HIGH when not pressed. (connected to 5 Volts)

Digital: translation of information is into binary format (zero or one) where each bit is representative of two distinct amplitudes.

Analog: information is translated into electric pulses of varying amplitude.

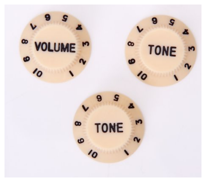
INPUT_PULLUP	

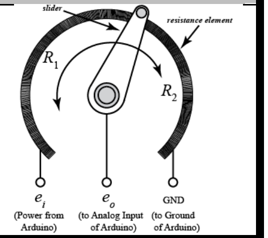
Similar to INPUT except it only records the input once per press (once it has been switched off)	**Example**:
	setup() {
	pinMode(9, INPUT_PULLUP);
	}

Circuit 2: Potentiometer

Potentiometer (dial):

Inputs a value between the range of 0 – 1023 based on the position of the dial. (range of 0 – 5 Volts)





Examples of Uses:

Heat dials on ovens and stoves,
Volume and tone knobs on electric instruments and amplifiers

Variables in Code:

A variable is a container that holds a number and has an address.

Data Types:

int	float	double	bool	char

Declaring Variables: (creating a variable)

```
dataType variableName;
```

Initializing Variables: (set value while declaring)

```
dataType variableName = value;
```

Taking Analog Input:

```
void loop()
{
  int sensorValue = analogRead(sensorPin);
}
```

Example:

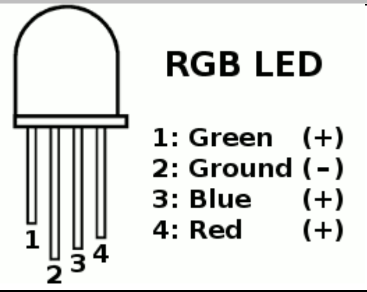
```
dialPin = 0; //variable holding the pin number

void loop() {
  int sensorValue = analogRead(dialPin);
  }                          //reads in from pin 0
```

Circuit 3: RGB LED

RGB light (Red Green Blue light):

Send power to any of the colors to activate them
(set pin value to HIGH), Common connects to ground



Secondary Colors:

Green + Blue:	**Green + Red**:	**Red + Blue**:
Cyan	Yellow	Magenta

Examples of Uses:

Computer screens, sign/exhibit backlighting, household light strip décor

Constants:

Use the keyword const before initializing a variable to create a constant. The value of the variable must be set when it is declared and cannot be changed later.

Example:

```
const int toes = 10; //this value cannot be changed

toes = 20;  //ILLEGAL
```

Functions:

```
returnType functionName ( parameter1, parameter2, ...) { statements }
```

Example:

```
int square(int a) {return a * a;}
```

For Loops:

```
for (initialize; condition; increment){
  statements;
} //loops until the condition is met
```

Example:

```
for (int i = sizeof(servoList) / sizeof(servoList[0]); i  > -1; --i) {
  servoArray[i].attach(servoPinArray[i]);
}  //will attach all of the servos in the lists if both lists correspond
```

Day2:

Circuit 4: Multiple LEDs	
Arrays:	
Essentially a list of values. The values are identified by their indexes. Index begins at zero and count up by 1. Each item in the array is referred to as an element of the array. The programmer must set the length of the array when it is created (either by declaring it with a given length or by initializing it with all of its elements, or a combination thereof). The length of the array must be known at compile-time, and cannot be changed at run-time, (you can, however, at run-time, build a larger array, copy over all of the old elements, and then delete the old array, which accomplishes a similar task – requires dynamic memory allocation, may cause memory leaks). arrayType arrayName[numberOfElements];	**Examples**: (each will create an array of 4 integers called list)

	int list[4] = {1, 2, 3, 4};
	list[0] = 3; //changes the value of 1 to 3
	//list is now {3, 2, 3, 4}

	int list[] = {1, 2, 3, 4};
	int list[4]; //reserves 4 spaces the size of integers in memory
	int list[4] = {1, 2}; /*list[2] and list[3] (3rd and 4th elements) are currently random values, reserving space in memory*/

Circuit 5: Push Buttons				
Using Logic Operators:				
Comparison Operators are used in conjunction with conditional statements to compare 2 values. Logic operators used with comparison operators makes a compound conditional statement.	**Logic Operators**:			
	==	&&	||	!
	EQUAL	AND	OR	NOT
True can be seen as 1 (or anything except 0), and false as 0.				

Examples:	
==	!=

int length = 20;
//if length is 20, do statements()
if (length == 20) {statements();}
	int width = 10;
	//if width is not 0, do statements()
	if (width != 0) {statements();}

&&	||

int length = 20;
//if legth is >10 and < 50, do statements()
if (length > 10 && length < 50){statements();}
 	
/*Because length is greater than 10 and less than 50, the conditional statement will evaluate to true and statements() will execute.*/
	int width = 10;
	//if width is < 20 or width is > 50, do statements()
	if (width < 20 || width > 50){statements();}
	/*Because width is less than 20, the conditional statement will evaluate to true and statements() will execute. Because the computer checks width < 20 first and that evaluates to true, it won't even check if width > 50; it will automatically run statements();*/

	!

int length = 20;
//if length, do statements
if (length){statements();}
//Since length is not 0, statements() will execute
	int width = 10;
	//if not length, do statements
	if (!width){statements();}
	//Since width is not 0, statements() will not execute

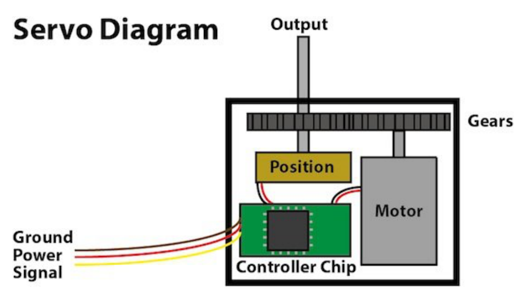
Circuit 6: A Single Servo

Servo Motor:

A motor that includes feedback circuitry allowing it to move specific positions.

Examples of Uses:

Radio-controlled airplanes, elevators, rudders, robotics, solar tracking systems



Example:

```
#include <Servo.h> //must have servo library
Servo servo1; //create the servo object
void setup() {
  servo1.attach(pinNumber);
}
void loop() {
  servo1.write(90); //servo goes to 90 degrees
}
```

Preprocessor Directives:

Preprocessor directives don’t need a semicolon. They are instructions for the compiler (they are executed at compile-time rather than run-time). They are marked with a #.

Common Directives:	
#include	#include “headerFile.h”
#define	#define PI 3.14

Using User-Created Objects:

A typedef or class that has it’s own members; the object becomes a type similar to how int, float, string, and char are their own types.

Examples:

```
Servo servo1; //Servo is a custom object
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
CapacitiveSensor cap = CapacitiveSensor(4,2);
```

Pulse Width Modulator (PWM):

Pulse Width Modulation is a technique for using a digital output as though it were analog. By using HIGH and LOW in a series broken up with delays, one can fake something between HIGH and LOW.

Slowing Down a Servo:

Use delays and smaller increments in position change to slow servo down.

Example:

```
for (int pos = 0; pos < 180; pos += 2) {
  servo1.write(pos);
  delay(20);
}
```

Day3:

Circuit 7: Keyboard Instrument

Serial Window:

Baud rate (usually 9600) is the speed of communication between the Arduino and the computer.

Example:

```
void setup() {
  Serial.begin(9600); //9600 is the baud rate
}
void loop() {
  Serial.print("hello");
  //every time loop runs hello is printed
}
```

Methods:

begin()	print()	println()	available()	parseInt()

Tone:

```
tone(pinNumber, frequency, duration);
tone(pinNumber, frequency);
noTone(pinNumber); //stops sound
```

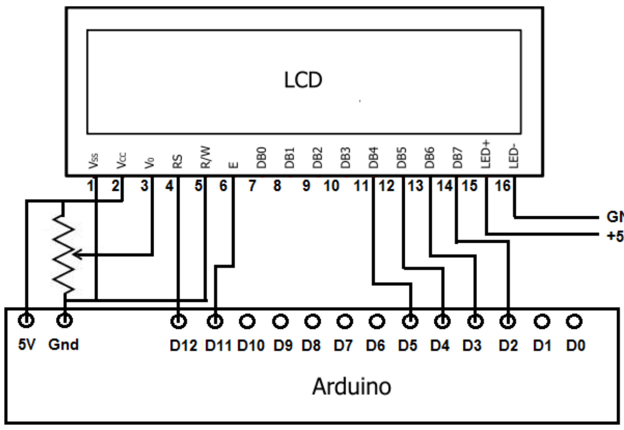
Example:

```
tone(5, 262, 4); //plays C note on pin 5 for 4 ms
```

Circuit 8: LCD

Liquid Crystal Display (LCD):

The LCD is a screen with a particular size used to display text. LCD monitors use a backlight and small filters controlled by electrical current to produce images.



LCD Methods:

begin()	clear()	print()	setCursor()

Setting up the Liquid Crystal Display:

LCD has its own library that must be included in order to access the LiquidCrystal object and all of its methods. Make a LiquidCrystal object into a global variable (at least for now) so it can be accessed by all the functions in the file. Use begin() and clear() methods in setup() to initialize the display and reset it.
*the data that you send to the display will remain there until either overwritten or cleared.

Example:

```
#include <LiquidCrystal.h> //must include library
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup()
{
  lcd.begin(16, 2); // 2 lines of 16 characters
  lcd.clear();
  lcd.print("hello, world!");
}
```

Circuit 9: Crystal Ball

Pseudo-Random Number Generator:

```
long random(long max);
//returns a number between 0 and (max-1)
long random(long min, long max);
//returns a number between min and (max-1)
```

```
int randomNum = random(7);
//randomNum is a number between 0 and 6
randomNum = random(7, 11);
//randomNum is now a number between 7 and 10
```

Switch Statement:

Allows a variable to be tested against a list of values, executing unique code for each case.
If the variable is not equivalent to any of the tested cases, default will run (default should be at the bottom of the list). Switch statements can be used in the place of using lots of if statements to check equality. Switch statements are actually somewhat faster as they use jump tables (algorithms to skip over some of the code for checking).

Syntax:

```
switch (variable) {
  case value1:
    statements1();
    break;
  case value2:
    statements2();
    break;
  default:
    break;
}
```

Example:

```
int cows = 10;
switch (cows)
{
  case 0:
    cows += 2; //cows should go from 0 to 2
    break;        //breaks out of switch statement
  case 2:
    cows *= 5; //cows should go from 2 to 10
    break;
  default:       //cows goes up by 1
    ++cows;
    break;
}

//Since cows is not 0 or 2, default will run.
//cows will become 11
```

Day4:

Circuit 10: Light Theremin

While Loop:

Enters loop if the condition is true. Breaks out of the loop if the condition is ever false (or break command is used). If condition is never false, the loop will repeat for eternity.

Syntax:

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

Example:

```
int i = 5;
while (i > 0){
  Serial.println(i); //prints value of i
  --i;              //decrease value of i by 1
} //repeat if i is greater than 0
```

Map Function:

map() converts one range into another range
map() can possibly return numbers beyond the range (if they had been beyond the previous range)

Example:

```
flexPos = analogRead(flexPin);
int servoPos = map(flexPos, 600, 900, 0, 180);
```

Constrain Function:

constrain() clips the value into the given range

Example:

```
servoPos = constrain(pos, 0, 180);
```

Circuit 11: Touchy-Feely Lamp

Capacitive Sensor:

The capacitive sensor will take in analog data through the method capacitiveSensor() rather than analogRead().

Taking fewer samples can result in a large range of variance and inaccuracy; taking too many can produce lag.

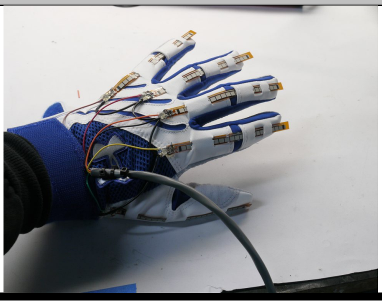
Example:

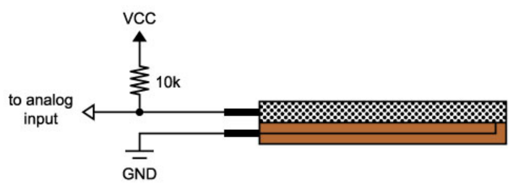
```
#include <CapacitiveSensor.h>
CapacitiveSensor capSensor = CapacitiveSensor(4,2);
void loop() {
  long sensorVal = capSensor.capacitiveSensor(30);
  //takes 30 samples – (more samples means more lag)
}
```

Circuit 12: Flex Sensor

Flex Sensor: (analog input)

A flex sensor is a plastic strip with conductive coating. When straight, the coating will have a particular resistance. When bent, the resistance will increase.




VCC
10k
to analog input
GND