كلية الهندسة
قسم هندسة الروبوت والأنظمة الذكية

تطبيقات ميكاترونيك1

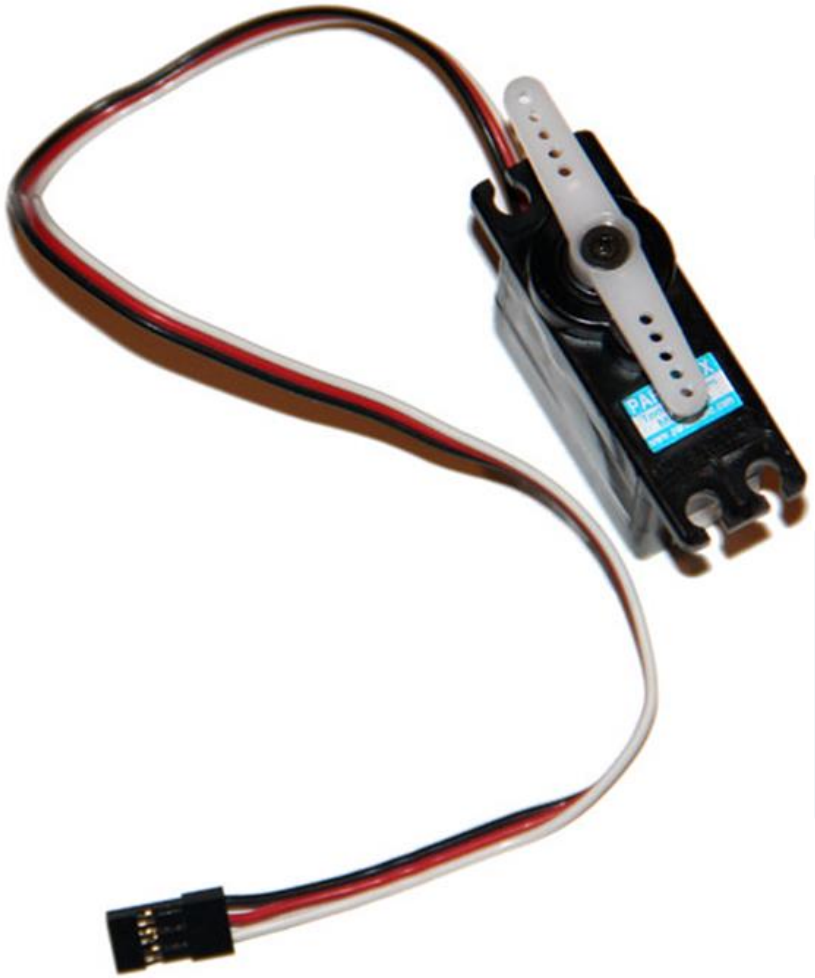
محاضرة 5

- Driving Servo Motors
- USB and Serial Communication

د. فادي متوج
د.عيسى الغنام

2024-2023

Driving Servo Motors

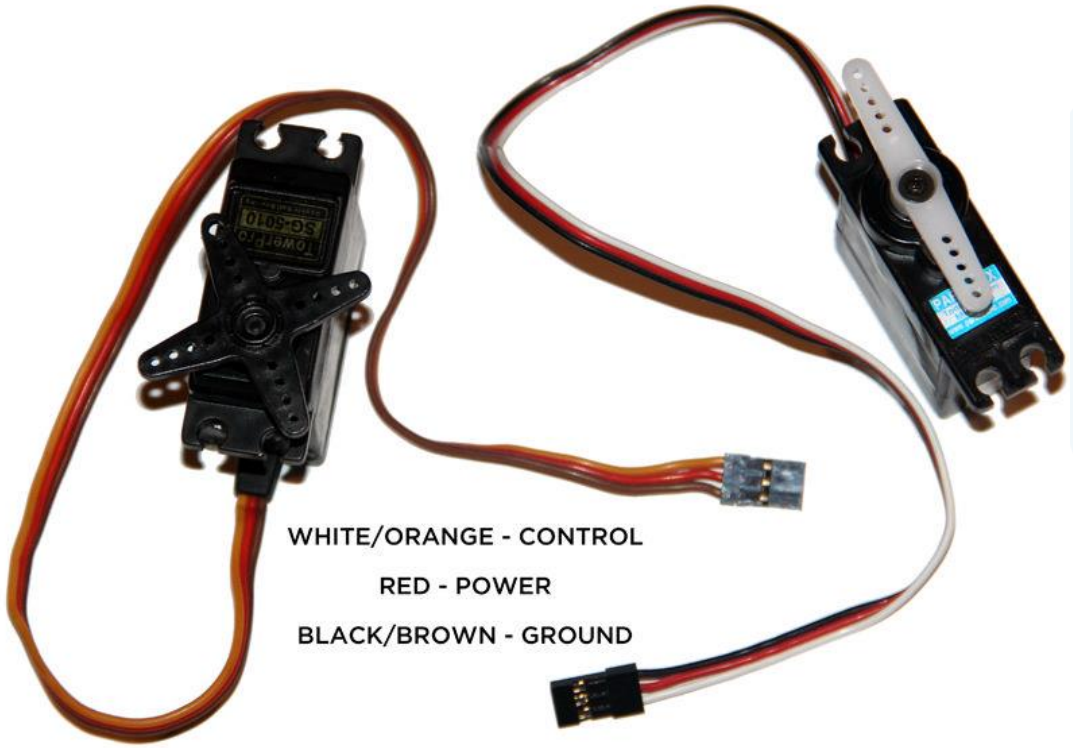


Servo Motors

- DC motors serve as excellent drive motors, but they are not as ideal for precision work because no feedback occurs. In other words, without using an external encoder of some kind, we will never know the absolute position of a DC motor.
- Servo motors, or servos, in contrast, are unique in that we command them to rotate to a particular angular position and they stay there until we tell them to move to a new position.
- This is important when we need to move our system to a known position.
- Examples include actuating door locks, precisely controlling the opening of an aperture.....

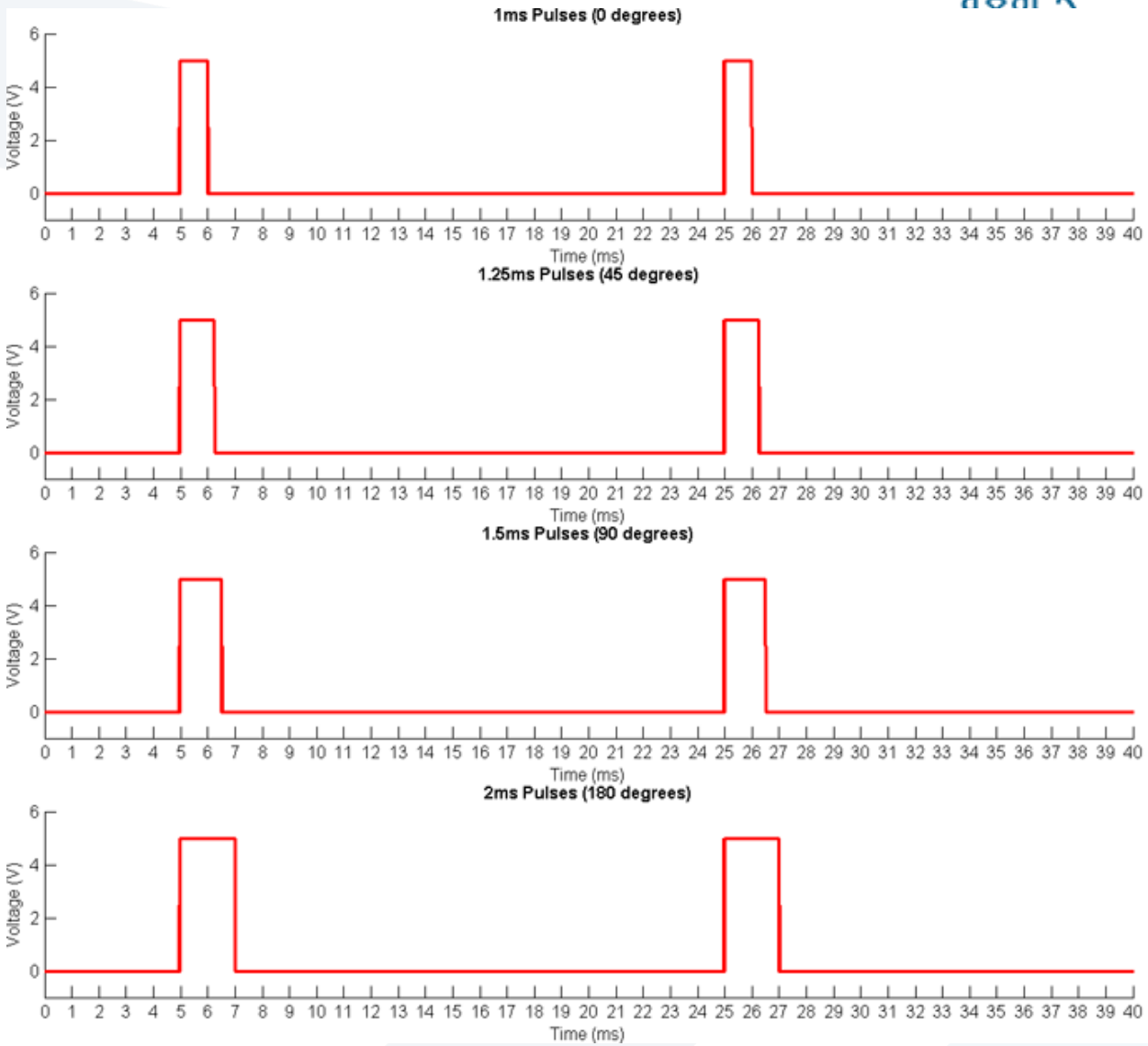
Understanding Servo Control

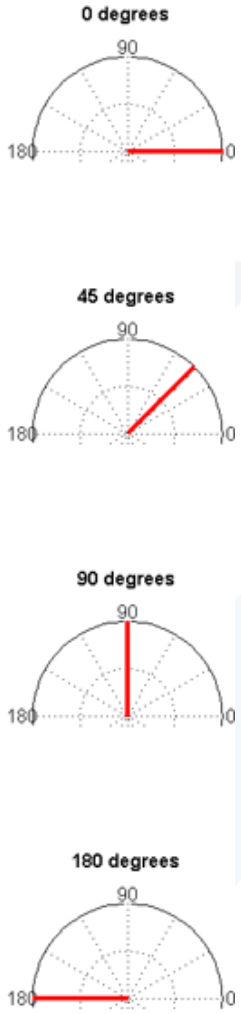
- Unlike their DC motor counterparts, servo motors have three pins: power (usually red), ground (usually brown or black), and signal (usually white or orange).



- Servos can draw more current than Arduino may be able to provide.
- The power and ground lines of a servo should always be connected to a **steady power source**.
- Servos are controlled using adjustable pulse widths on the signal line.
  - Sending a **1ms** 5V pulse turns the motor to **0 degrees**
  - sending a **2ms** 5V pulse turns the motor to **180 degrees**
  - Sending a **1.5ms** pulse turns the motor to **90 degrees**.
- Once a pulse has been sent, the servo turns to that position and stays there until another pulse instruction is received.
- However, if we want a servo to "hold" its position (resist being pushed on and try to maintain the exact position), we just resend the command once every **20ms**.

Servo motor timing diagram



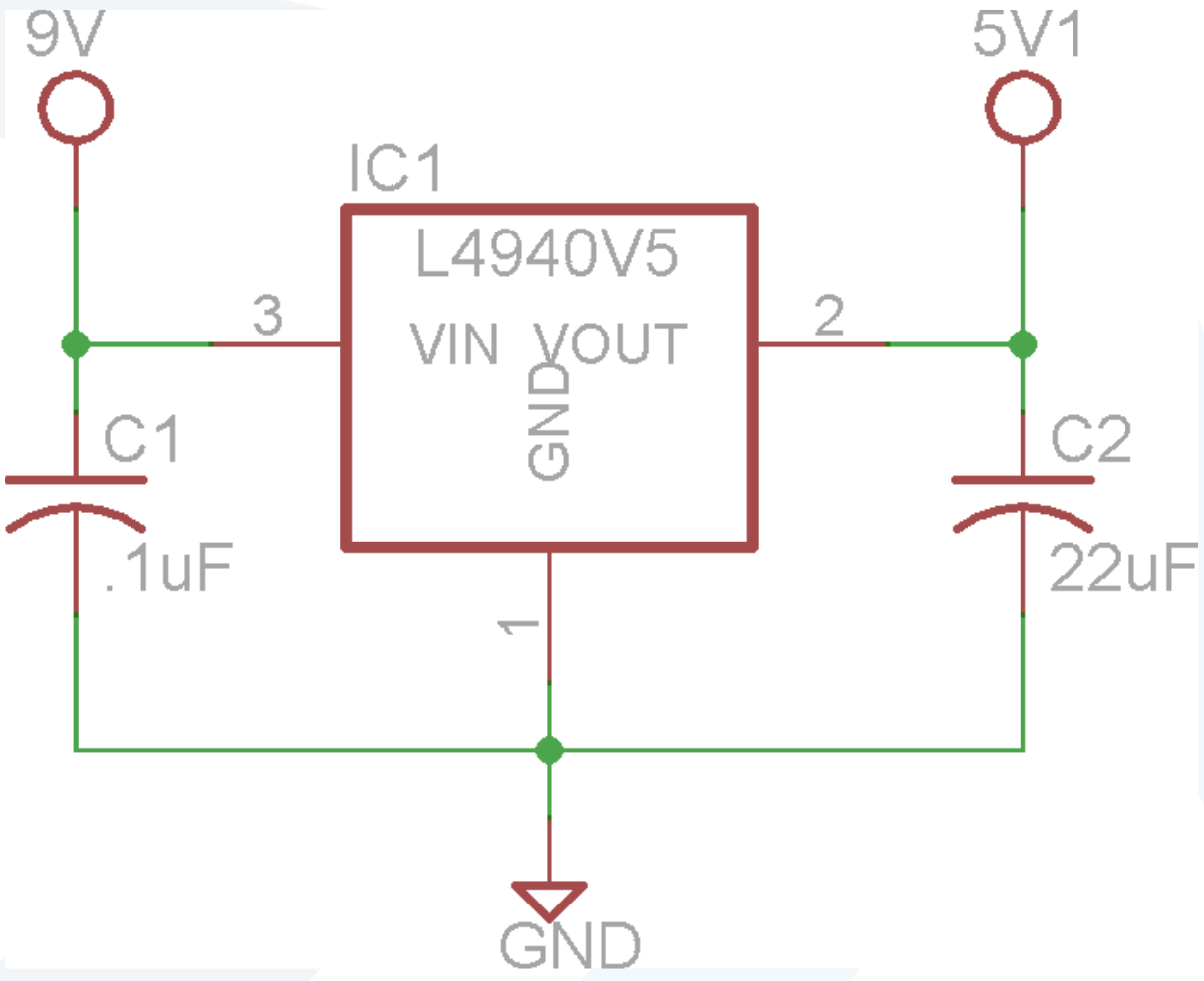


- Servos can draw more current than Arduino may be able to provide. However, most servos are designed to run at 5V, not 9V or 12V like a DC motor.
- Even though the voltage is the same as that of an Arduino, we use a separate power source that can supply more current.
- To do this ,we use a 9V battery and a linear regulator to generate a 5V supply from our 9V battery.

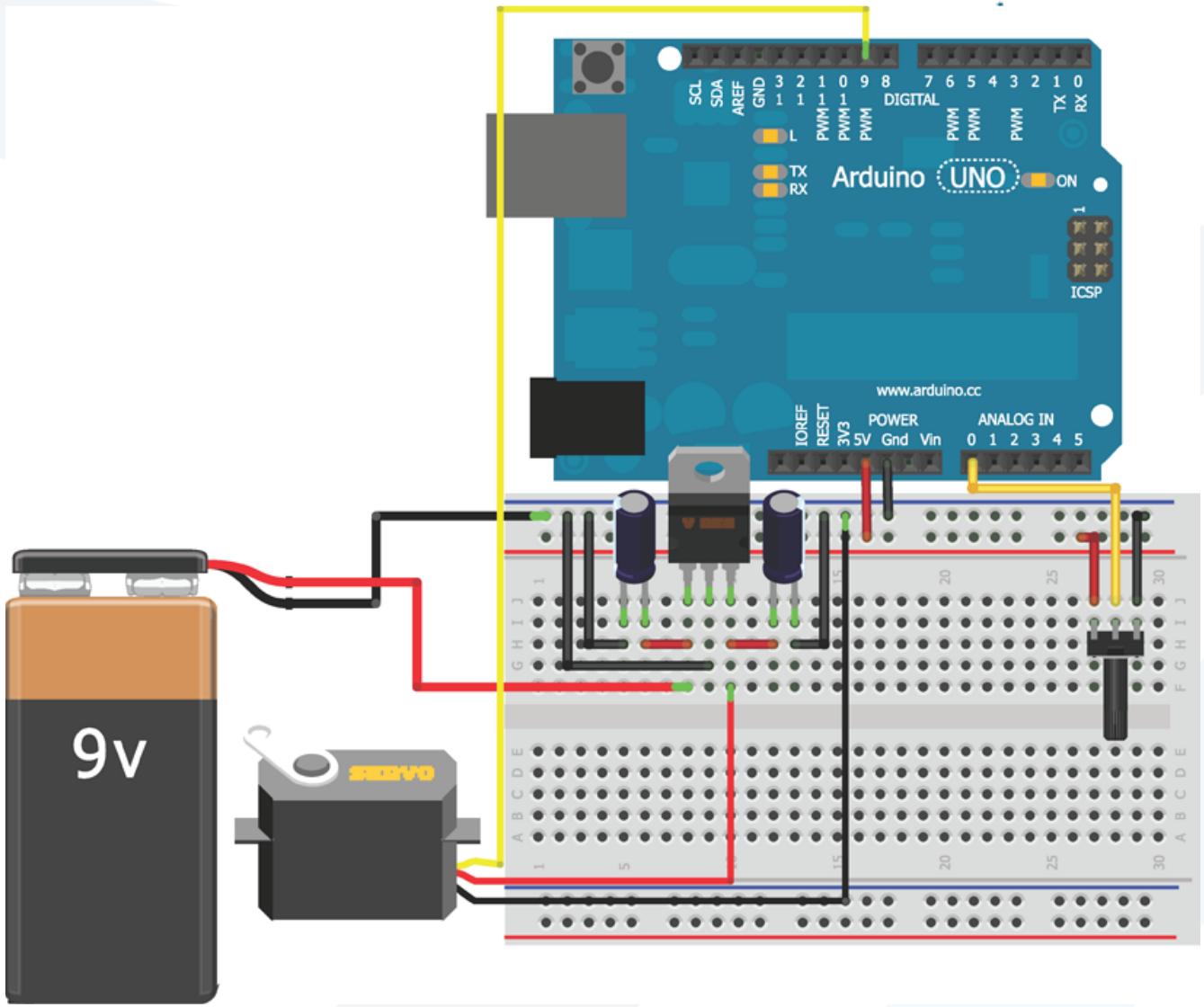
The voltage regulator

- A linear regulator is an simple device that generally has three pins: input voltage, output voltage, and ground.
- The ground pin is connected to both the ground of the input supply and to the ground of the output
- In the case of linear-voltage regulators, the input voltage always must be higher than the output voltage, and the output voltage is set at a fixed value depending on the regulator you use.
- The voltage drop between the input and the output is burned off as heat, and the regulator takes care of ensuring that the output always remains the same, even as the voltage of the input drops (in the case of a battery discharging over time).
- L4940V5 is a 5V voltage regulator that is capable of supplying up to 1.5 amps at 5V.

L4940V5 5V voltage regulator



Controlling a Servo



- The Arduino IDE includes a built-in **library** that makes controlling servos a breeze.
- A library is a collection of code that is useful, but not always needed in sketches.
- All we have to do is attach a servo "object" to a particular pin and give it an angle to rotate to.
- The library takes care of the rest, even setting the pin as an output.
- The simplest way to test out the functionality of our servo is to map the potentiometer directly to servo positions.
  - Turning the potentiometer to 0 moves the servo to **0 degrees**.
  - Turning the potentiometer to 1023 moves the servo to **180 degrees**

```
//Servo Potentiometer Control
#include <Servo.h>
const int SERVO=9;     //Servo on Pin 9
const int POT=0;    //POT on Analog Pin 0
Servo myServo;
int val = 0;        //for storing the reading from the POT
void setup() {
myServo.attach(SERVO);
}
void loop() {
val = analogRead(POT);          //Read Pot
val = map(val, 0, 1023, 0, 179);    //scale it to servo range
myServo.write(val); //sets the servo
delay(15); //waits for the servo
}
```

USB and Serial Communication

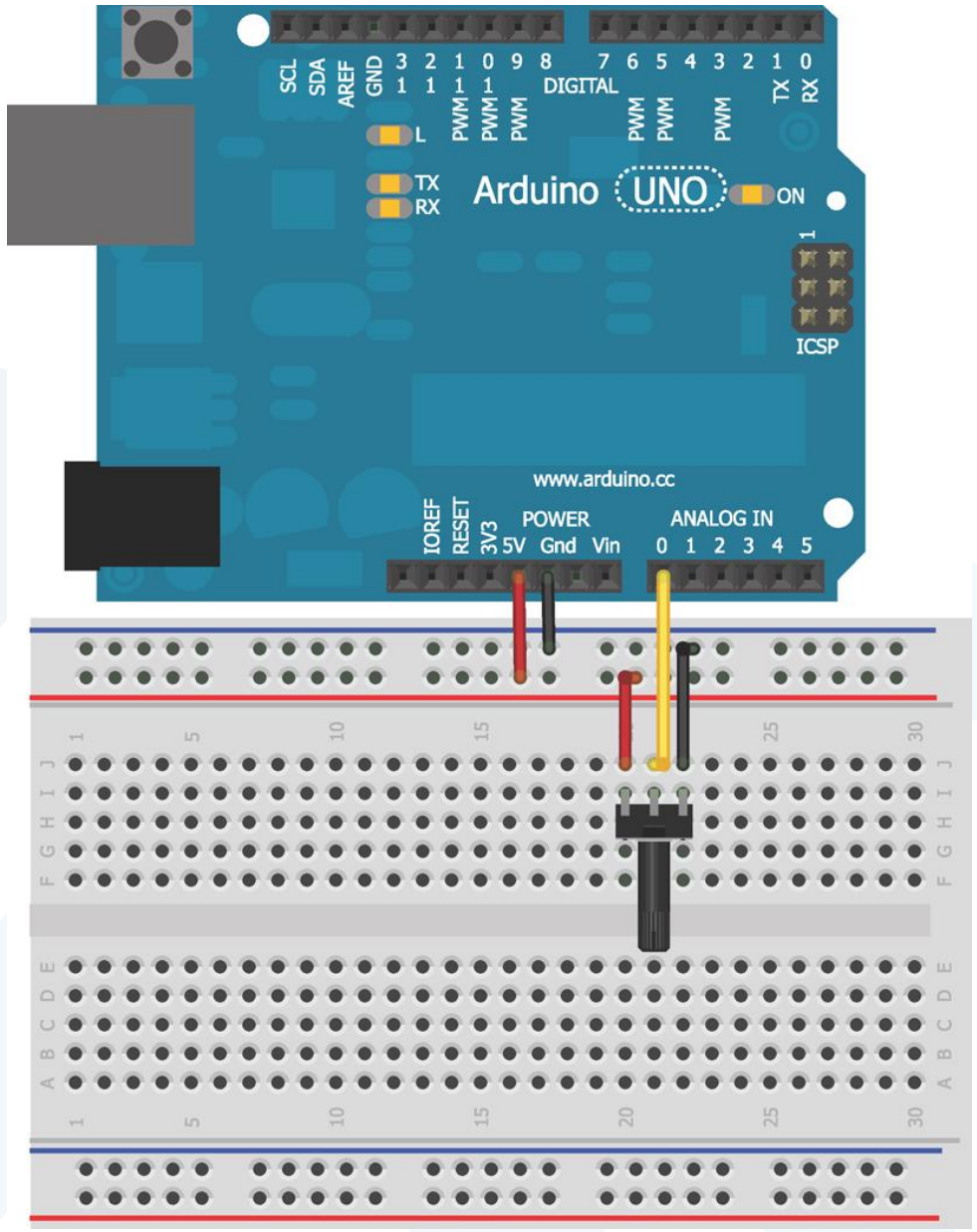
- **ATMega328P** that we find on the Arduino Uno have **one hardware serial port**.
- It includes a transmit (**TX**) and receive (**RX**) pin that can be accessed on digital pins 0 and 1.
- Arduino is equipped with a **bootloader** that allows us to program it over this serial interface.
- To facilitate this, those pins are "**multiplexed**" (meaning that they are connected to more than one function); they connect, **indirectly**, to the transmit and receive lines of our USB cable.
- However, **serial** and **USB** are not directly compatible, so a secondary integrated circuit (IC) is used to facilitate the conversion between the two.
- This is the type of interface present on an Uno, where an intermediary IC facilitates USB-to-serial communication.

Listening to the Arduino

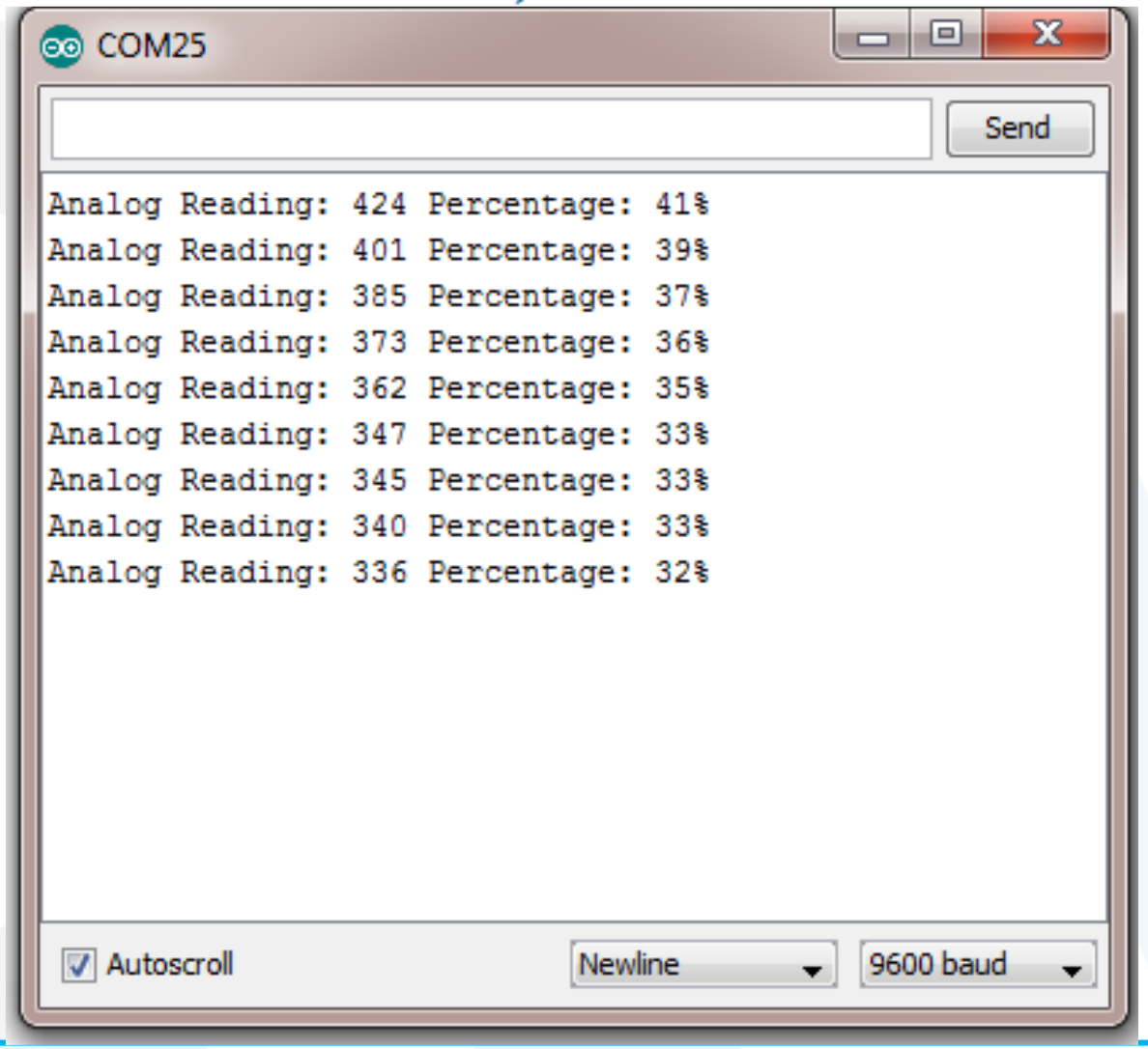
- The most basic serial function that we can do with an Arduino is to print to the computer's **serial terminal**.
- To print data to the terminal, we only need to utilize three functions:
  - Serial.begin(baud_rate)
  - Serial.print("Message")
  - Serial.println("Message")

Serial Printing Test with a Potentiometer

- To experiment with this functionality, we wire up a simple circuit with a potentiometer connected to pin A0 on the Arduino.
- We will read the value of the potentiometer and print it as both a raw value and a percentage value.


SCL
SDA
AREF
GND
3 2 1 0 9 8 7 6 5 4 3 2 1 0
1 1 1 1
PWM PWM PWM PWM PWM PWM
DIGITAL
TX RX
L
TX
RX
Arduino UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V
POWER
Gnd
Vin
ANALOG IN
0 1 2 3 4 5

```
//Simple Serial Printing Test with a Potentiometer
const int POT=0;          //Pot on analog pin 0
void setup() {
Serial.begin(9600);       //Start serial port with baud = 9600
}
void loop() {
int val = analogRead(POT);                //Read potentiometer
int per = map(val, 0, 1023, 0, 100);      //Convert to percentage
Serial.print("Analog Reading: ");
Serial.print(val);                        //Print raw analog value
Serial.print(" Percentage: ");
Serial.print(per);                        //Print percentage analog value
Serial.println("%");                      //Print % sign and newline
delay(1000);                              //Wait 1 second, then repeat
}
```


COM25
Send
Analog Reading: 424 Percentage: 41%
Analog Reading: 401 Percentage: 39%
Analog Reading: 385 Percentage: 37%
Analog Reading: 373 Percentage: 36%
Analog Reading: 362 Percentage: 35%
Analog Reading: 347 Percentage: 33%
Analog Reading: 345 Percentage: 33%
Analog Reading: 340 Percentage: 33%
Analog Reading: 336 Percentage: 32%
Autoscroll
Newline
9600 baud

- Serial.begin()must be called once at the start of the program in setup() to prepare the serial port for communication.
- After we have done this, we can use Serial.print() and Serial.println() functions to write data to the serial port.
- The only difference between the two is that Serial.println() adds a carriage return at the end of the line (so that the next thing printed will appear on the following line).

Using Special Characters

- We can transmit a variety of "**special characters**" over serial, which allow us to **change the formatting** of the serial data we are printing.
- We indicate these special characters with a slash escape character (\ ) followed by a command character.
- There are a variety of these special characters, but the two of greatest interest are the **tab** and **newline** characters.
- To insert a tab character, we add a **\t** to the string.
- To insert a newline character, we add a **\n** to the string.
- This is particularly useful if we want a newline to be inserted at the beginning of a string, instead of at the end as the Serial.println() function does.
- If, for some reason, we actually want to print \n or \t in the string, we can do so by printing \\n or \\t , respectively.

Tabular serial printing test with a potentiometer

```
//Tabular serial printing test with a potentiometer
const int POT=0;                    //Pot on analog pin 0
void setup()
{
Serial.begin(9600);                 //Start Serial Port with Baud = 9600
}
void loop()
{
Serial.println("\nAnalog Pin\tRaw Value\tPercentage");
Serial.println("----------------------------------------");
for (int i = 0; i < 10; i++)
{
int val = analogRead(POT);                  //Read potentiometer
int per = map(val, 0, 1023, 0, 100);        //Convert to percentage
Serial.print("A0\t\t");
Serial.print(val);
Serial.print("\t\t");
Serial.print(per);                  //Print percentage analog value
Serial.println("%");                //Print % sign and newline
delay(1000);                        //Wait 1 second, then repeat
}
}
```

COM9

Send

```
A0              87              8%
A0              0               0%
A0              0               0%

Analog Pin      Raw Value       Percentage
----------------------------------------
A0              2               0%
A0              43              4%
A0              90              8%
A0              141             13%
A0              163             15%
A0              238             23%
A0              311             30%
A0              376             36%
A0              422             41%
A0              470             45%

Analog Pin      Raw Value       Percentage
----------------------------------------
A0              514             50%
A0              572             55%
A0              720             70%
A0              1021            99%
A0              1023            100%
```

Autoscroll Newline 9600 baud

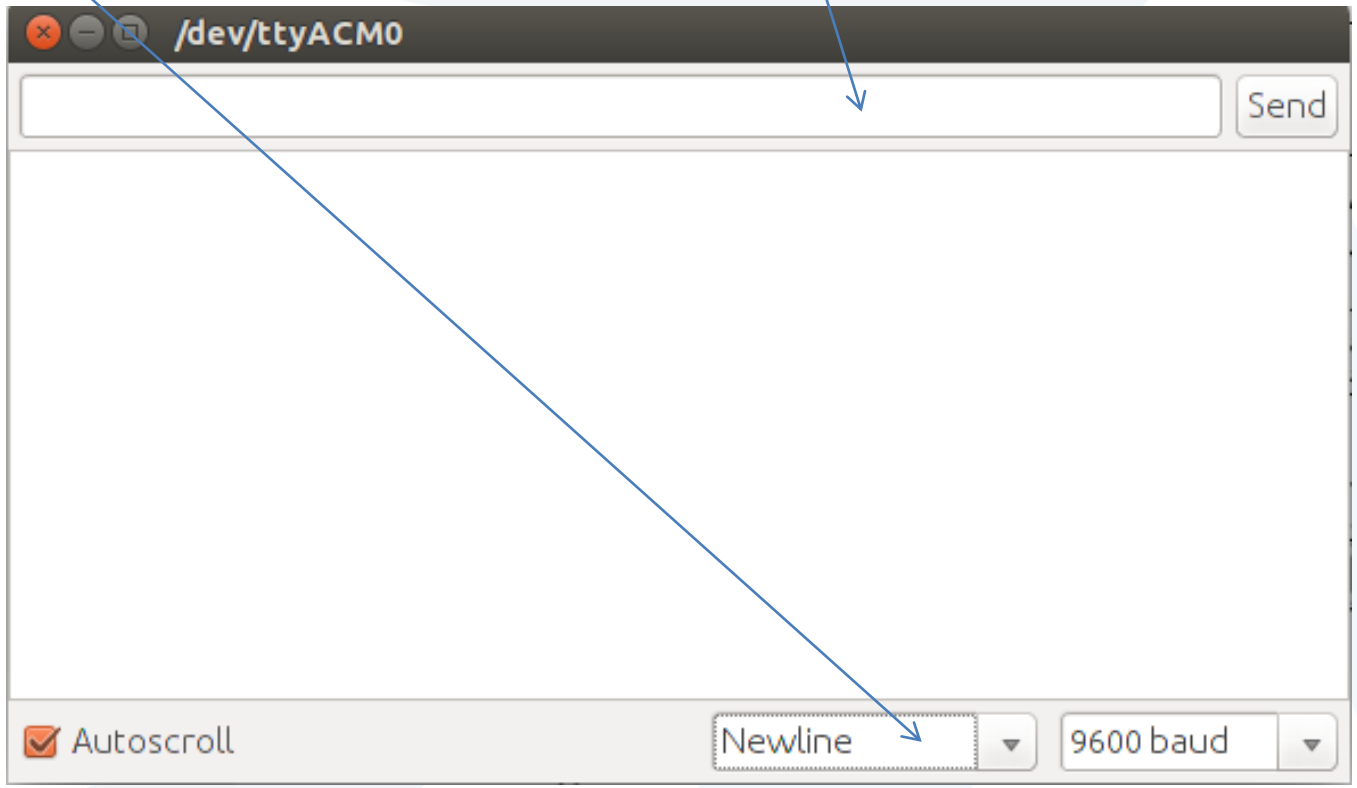
Serial Data Type Options

DATA TYPE	EXAMPLE CODE	SERIAL OUTPUT
Decimal	`Serial.println(23);`	23
Hexadecimal	`Serial.println(23, HEX);`	17
Octal	`Serial.println(23, OCT)`	27
Binary	`Serial.println(23, BIN)`	00010111

Talking to the Arduino

- What good is a conversation with our Arduino if it's only going in one direction?
- Now that we understand how the Arduino sends data to our computer, let's discuss how to send commands from our computer to the Arduino.

Arduino IDE **serial monitor** has a text entry field at the top, and a drop-down menu at the bottom.



- The **drop-down menu** determines what, if anything, is appended to end of our commands when we send them to the Arduino.
- When we select **Newline**, which just appends a **\n** to the end of anything that we send from the text entry field at the top of the serial monitor window.
- Unlike with some other terminal programs, the Arduino IDE serial monitor sends our whole command string at one time when we press the Enter key or the Send button.
- This is in contrast to other serial terminals like PuTTy that send characters as we type them.

Reading Information from a Computer or Other Serial Device

- How to use the Arduino IDE serial monitor to send commands manually to the Arduino?
- How to send multiple command values at once ?

- Arduino's serial port has a **buffer**. In other words, we can send several bytes of data at once and the Arduino will queue them up and process them in order based on the content of our sketch.
- We do not need to worry about sending data faster than our loop time, but we do need to worry about sending so much data that it overflows the buffer and information is lost.

Telling the Arduino to Echo Incoming Data

- The simplest thing we can do is to **make the Arduino echo back everything that we send it**.
- To accomplish this, the Arduino basically just needs to monitor its serial input buffer and print any character that it receives.
- To do this, we need to use two commands :
  - **Serial.available()** : returns the number of characters (or bytes) that are currently stored in the Arduino's incoming serial buffer. Whenever it's more than zero, we will read the characters and echo them back to the computer.
  - **Serial.read()** : reads and returns the next character that is available in the buffer.
- Each call to Serial.read()will only return 1 byte, so we need to run it for as long as Serial.available() is returning a value greater than zero.
- Each time Serial.read() grabs a byte, that byte is removed from the buffer, so the next byte is ready to be read.

```
//Echo every character

char data;    //Holds incoming character
void setup()
{
Serial.begin(9600);    //Serial Port at 9600 baud
}
void loop()
{
      //Only print when data is received
if (Serial.available() > 0)
   {
     data = Serial.read();     //Read byte of data
     Serial.print(data);       //Print byte of data
   }
}
```

- When we send an alphanumeric character via the serial monitor, we aren't actually passing a "5", or an "A". We are sending a **byte** that the computer interprets as a character.
- In the case of serial communication, the **ASCII** character set is used to represent all the letters, number, symbols, and special commands that we might want to send.

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

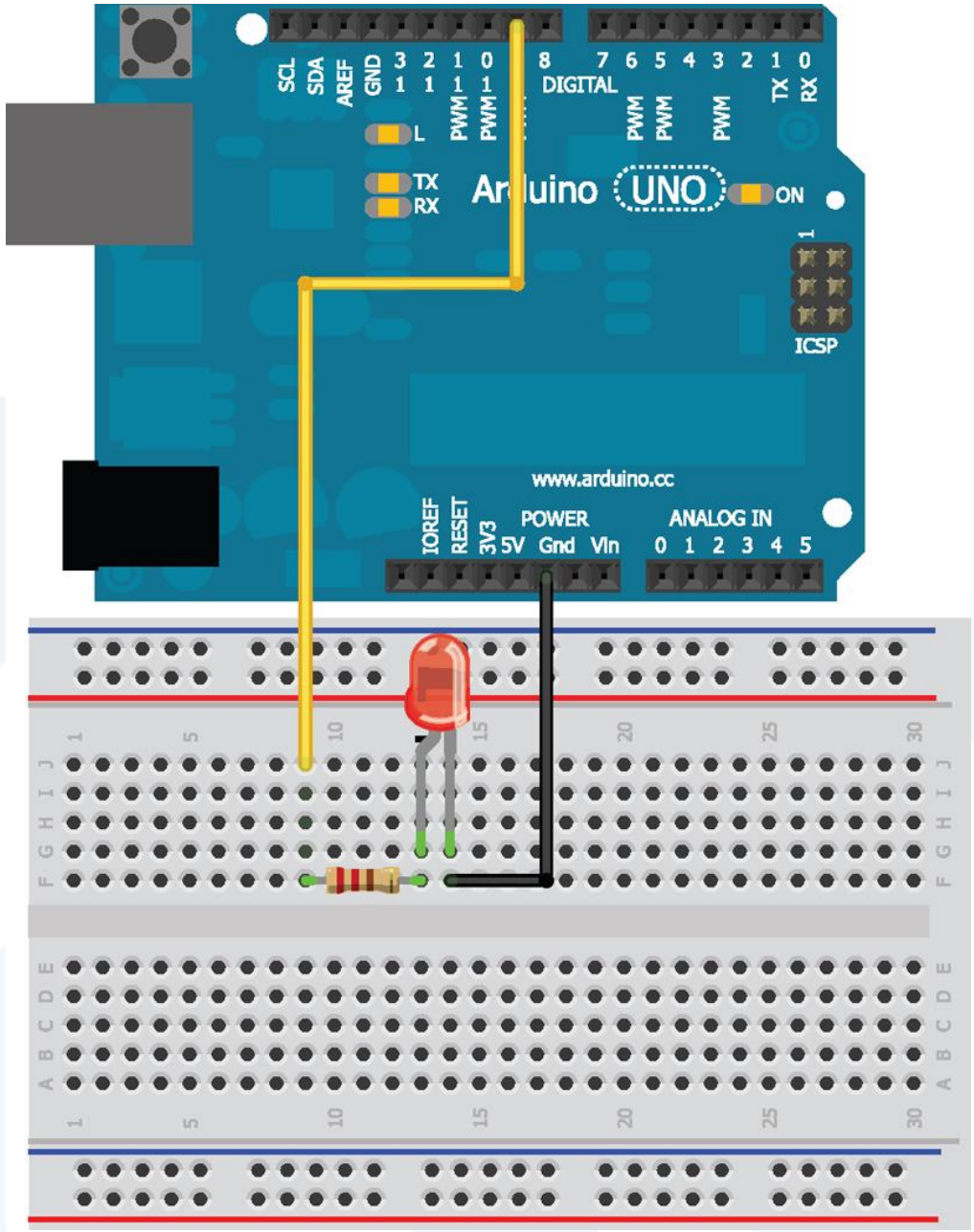
ASCII table

- When reading a value that we have sent from the computer, the data must be read as a **char type**.
- Even if we are only expecting to send numbers from the serial terminal, we need to read values as a character first, and then convert as necessary.
- For example, if we modify the code to declare data as type **int** , sending a value of **5** would return **53** to the serial monitor because the decimal representation of the character 5 is the number 53.

- We often want to send **numeric values** to the Arduino. *So how do we do that?*
- We can do so in a few ways:
  - First, we can simply compare the characters directly. If we want to turn an LED on when we send a 1 , we can compare the character values like this: **if (Serial.read() == '1')** .
  - A second option is to convert each incoming byte to an integer by subtracting the zero-valued character, like this: **int val = Serial.read() - '0'** However, this doesn't work very well if we intend to send numbers that are greater than 9, because they will be multiple digits. To deal with this, the Arduino IDE includes a handy function called **parseInt()**that attempts to extract integers from a serial data stream.

Example1 : Sending Single Characters to Control an LED

- In this example, we write a sketch that uses a simple character comparison to control an LED.
- When only sending a single character, the easier thing to do is to do a simple character comparison in an **if statement**.
- We will send a 1 to turn an LED on, and a 0 to turn it off.
- We wire an LED up to pin 9 of our Arduino.
- Each time a character is added to the buffer, it is compared to a '0' or a '1' , and the appropriate action is taken.

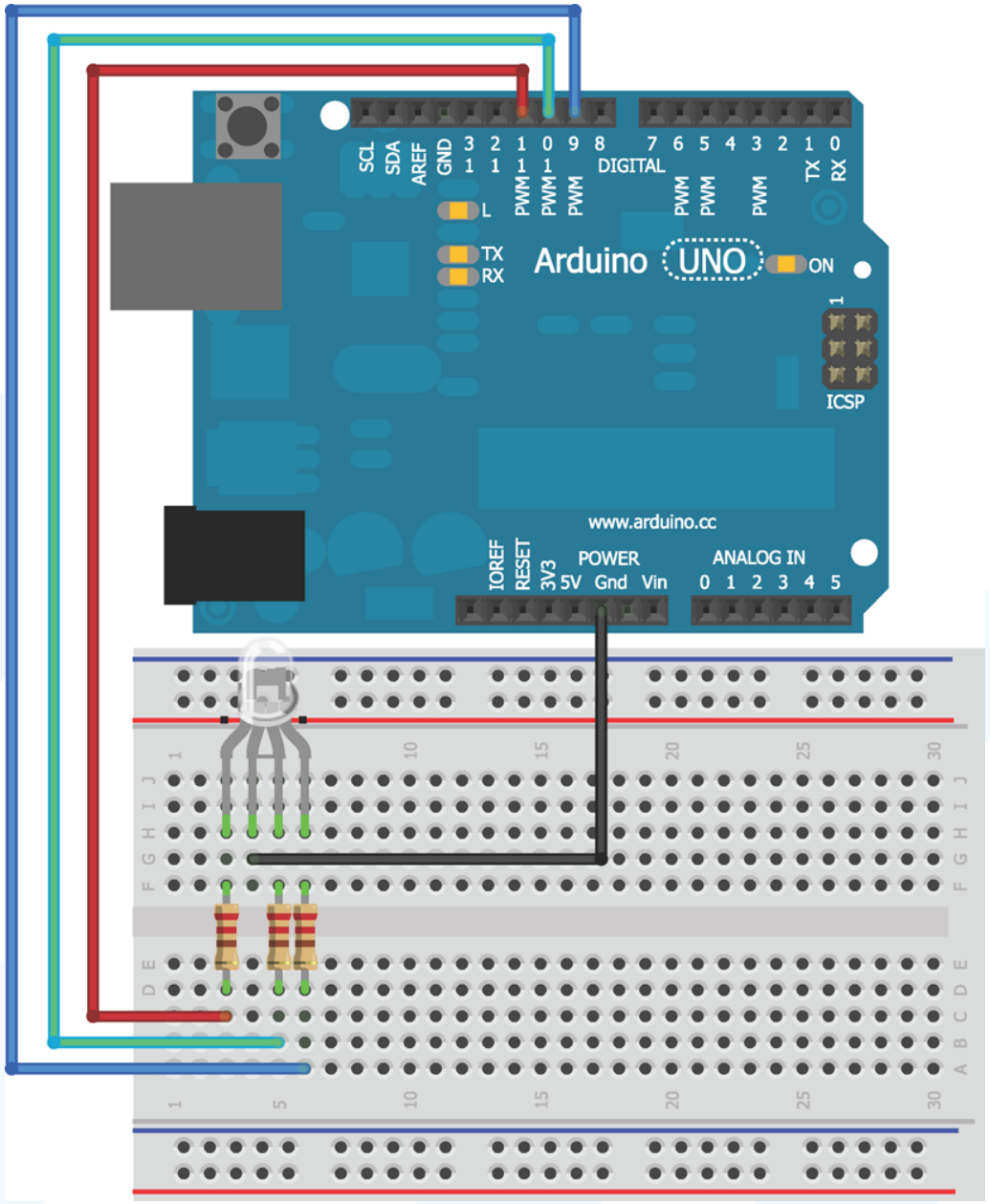

SCL
SDA
AREF
GND
3 2 1 0
1 1 1 1
PWM PWM
8 7 6 5 4 3 2 1 0
DIGITAL
PWM PWM PWM
TX RX
L
TX
RX
Arduino UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V
POWER
Gnd
Vin
ANALOG IN
0 1 2 3 4 5

```
//Single Character Control of an LED
const int LED=9;
char data; //Holds incoming character
void setup() {
Serial.begin(9600); //Serial Port at 9600 baud
pinMode(LED, OUTPUT);
}
void loop() {
//Only act when data is available in the buffer
if (Serial.available() > 0)
{
data = Serial.read(); //Read byte of data
//Turn LED on
if (data == '1')
{
digitalWrite(LED, HIGH);
Serial.println("LED ON");
}
//Turn LED off
else if (data == '0')
{
digitalWrite(LED, LOW);
Serial.println("LED OFF"); } } }
```

- In this example, an **else if** statement is used instead of a simple **else** statement.
- Because our terminal is also set to send a newline character with each transmission, it's critical to clear these from the buffer.
- Serial.read() will read in the newline character which is not equivalent to a '0' or a '1' , and it will be overwritten the next time Serial.read() is called.
- If just an else statement were used, both '0' and '\n' would trigger turning the LED off. Even when sending a '1' , the LED would immediately turn off again when the '\n' was received!

Example2 : Sending Lists of Values to Control an RGB LED

- Sending a single command character is fine for controlling a single digital pin, but **what if we want to command multiple devices ?**
- This example explores sending multiple comma-separate values to simultaneously command multiple devices.
- To facilitate testing this, we wire up a common cathode RGB LED


SCL
SDA
AREF
GND
3 2 1 0 9 8
1 1 1 1 1
PWM PWM PWM
DIGITAL
7 6 5 4 3 2 1 0
PWM PWM PWM
TX RX
L
TX
RX
Arduino UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V
POWER
Gnd Vin
ANALOG IN
0 1 2 3 4 5

- To control this RGB LED, we send three separate 8-bit values (0–255) to set the brightness of each LED color.
- For example, to set all the colors to full brightness, we send “255,255,255” .
- This presents a few challenges:
  - We need to differentiate between numbers and commas.
  - We need to turn this sequence of characters into integers that we can pass to analogWrite()functions.
  - We need to be able to handle the fact that values could be one, two, or three digits.

- The Arduino IDE implements a very handy function for identifying and extracting integers: **Serial.parseInt()**.
- Serial.parseInt()returns the first valid integer number from the serial buffer. Characters that are not integers are skipped.
- Serial.parseInt()is terminated by the first character that is not a digit.
- Each call to this function waits until a non-numeric value enters the serial buffer, and converts the previous digits into an integer.
- The first two values are read when the commas are detected, and the last value is read when the newline is detected.

```
//Define LED pins
const int RED =11;
const int GREEN =10;
const int BLUE =9;
//Variables for RGB levels
int rval = 0;
int gval = 0;
int bval = 0;
void setup()
{
Serial.begin(9600); //Serial Port at 9600 baud
//Set pins as outputs
pinMode(RED, OUTPUT);
pinMode(GREEN, OUTPUT);
pinMode(BLUE, OUTPUT);
}
```

```
void loop()
{
//Keep working as long as data is in the buffer
while (Serial.available() > 0)
{
rval = Serial.parseInt();     //First valid integer
gval = Serial.parseInt();     //Second valid integer
bval = Serial.parseInt();     //Third valid integer

if (Serial.read() == '\n') //Done transmitting
{
//set LED
analogWrite(RED, rval);
analogWrite(GREEN, gval);
analogWrite(BLUE, bval);
}}}
```

- The program keeps looking for the three integer values until a newline is detected. Once this happens, the values that were read are used to set the brightness of the LEDs.
- To use this, we open the serial monitor and enter three values between 0 and 255 separated by a comma, like "200,30,180" .
- We can try mixing all kinds of pretty colors!