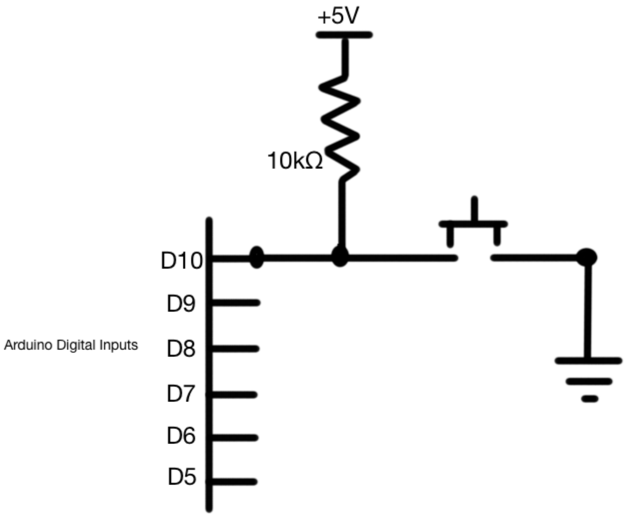
# PAT 451 INTERACTIVE MEDIA DESIGN I

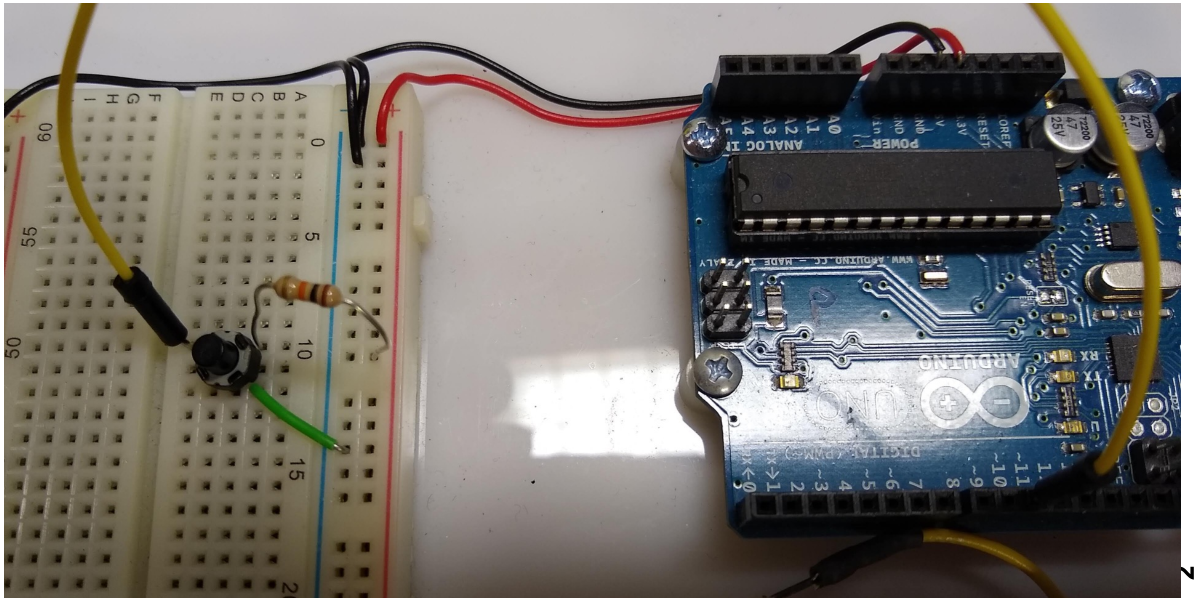
ARDUINO_FUNCTIONS

# READING AN INPUT PIN

Recall:

```
pinMode(10,INPUT);
.
.
.
int reading = digitalRead(10);
       // returns the value applied to an input pin
       // either HIGH or LOW
```





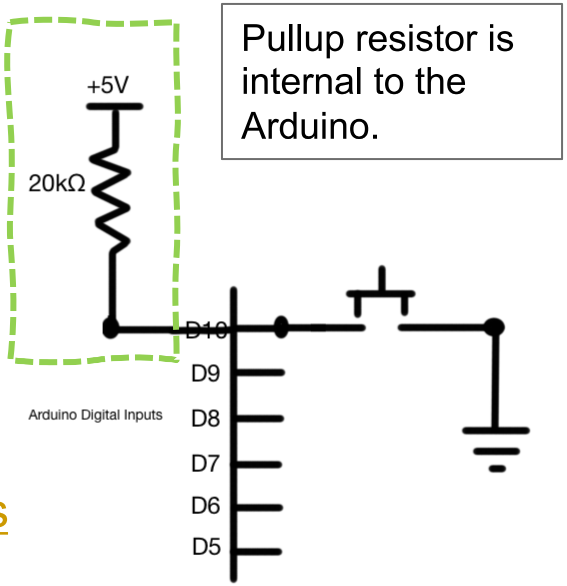
# THERE'S AN EASIER WAY!

The need to prevent a 'floating' input pin, where there is no voltage applied to the pin in its default state, is a fairly common situation with a microcontroller.

So to avoid having to connect that resistor to +5V, the Arduino's microcontroller (and many others) have "**pull-up resistors**" connected to their digital inputs.

**Pull-up resistors** are *internal* to the microcontroller, and can be activated on any digital input.

```
pinMode(10,INPUT_PULLUP);
// activates internal 20kΩ resistor
// no resistor required in your
button circuit
```

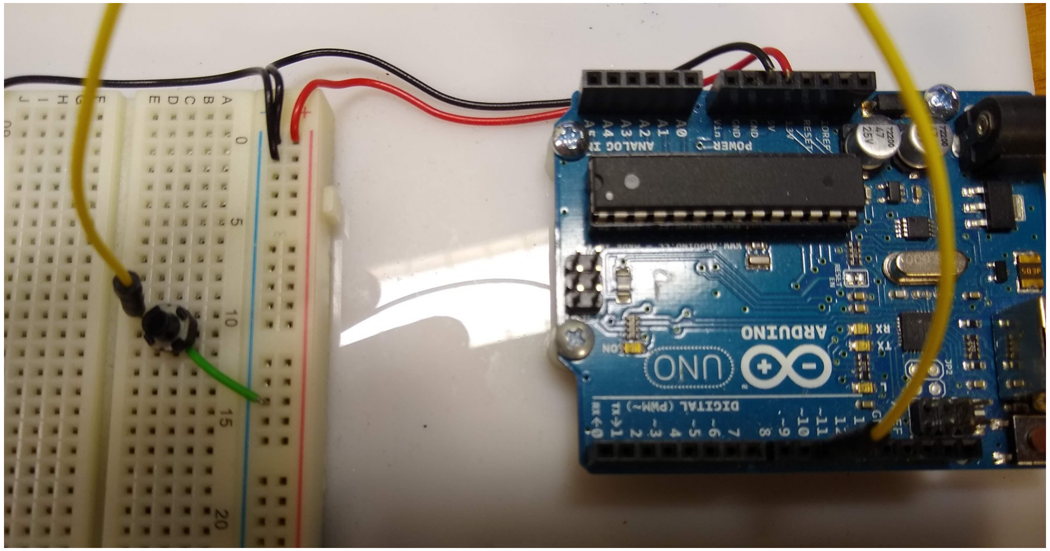


Pullup resistor is internal to the Arduino.

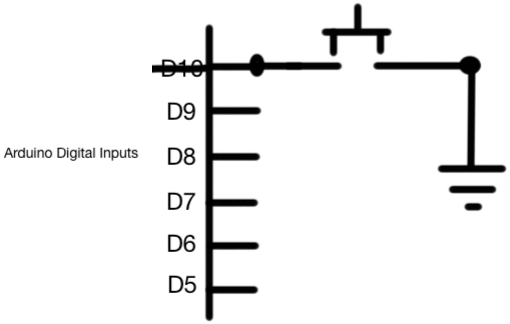
See: https://www.arduino.cc/en/Tutorial/DigitalPins

# NO EXTERNAL RESISTOR REQUIRED!

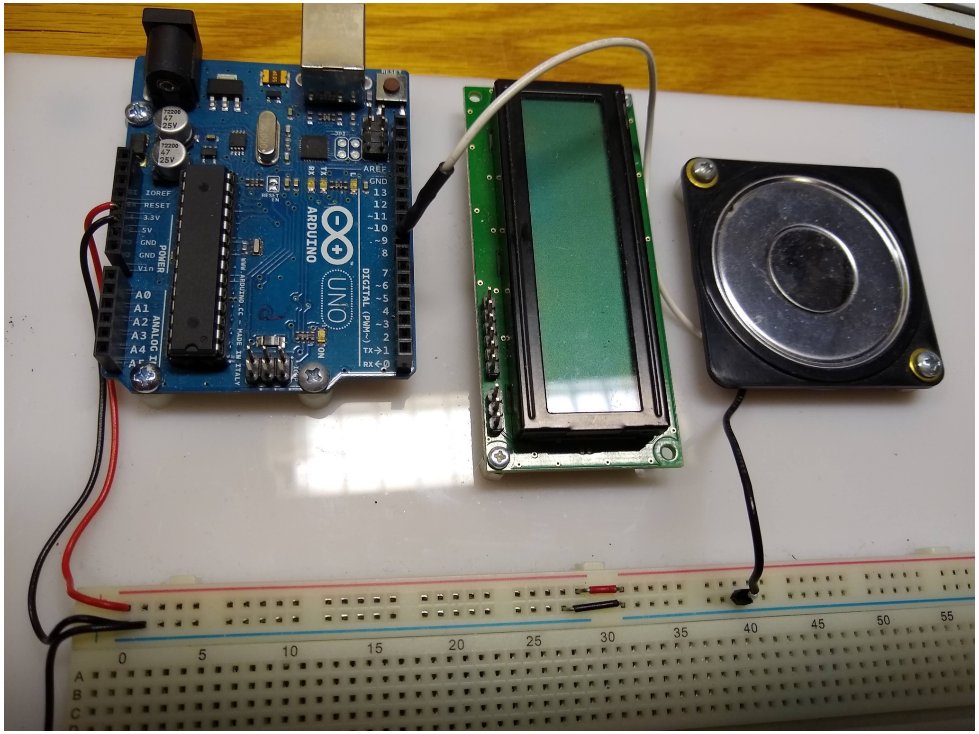
With `INPUT_PULLUP` mode, there is no need for an external resistor for your button circuit.



This is now the button circuit if you use INPUT_PULLUP mode.



# PLAYING A TONE FROM YOUR SPEAKER



Connect the white wire of your speaker to a digital output pin. Connect the black wire to ground.

Caution!

Be careful not to bend/break the pins connected to the speaker. They can be a bit fragile. When removing them, pull them straight upward.

# ARDUINO TONE FUNCTION

https://www.arduino.cc/en/Reference/Tone

```
tone(pin,f);
       // plays a square wave at frequency f on pin pin
       // plays until noTone() is called
noTone();
       // stops a tone if one is playing
```

```
tone(pin,f,dur);
       // plays a square wave at frequency f on pin pin
       // and duration dur, in milliseconds
```

See **tune.ino** example

# USING THE ARDUINO SERIAL MONITOR

An important capability of the Arduino is to **send digital information** to other devices.

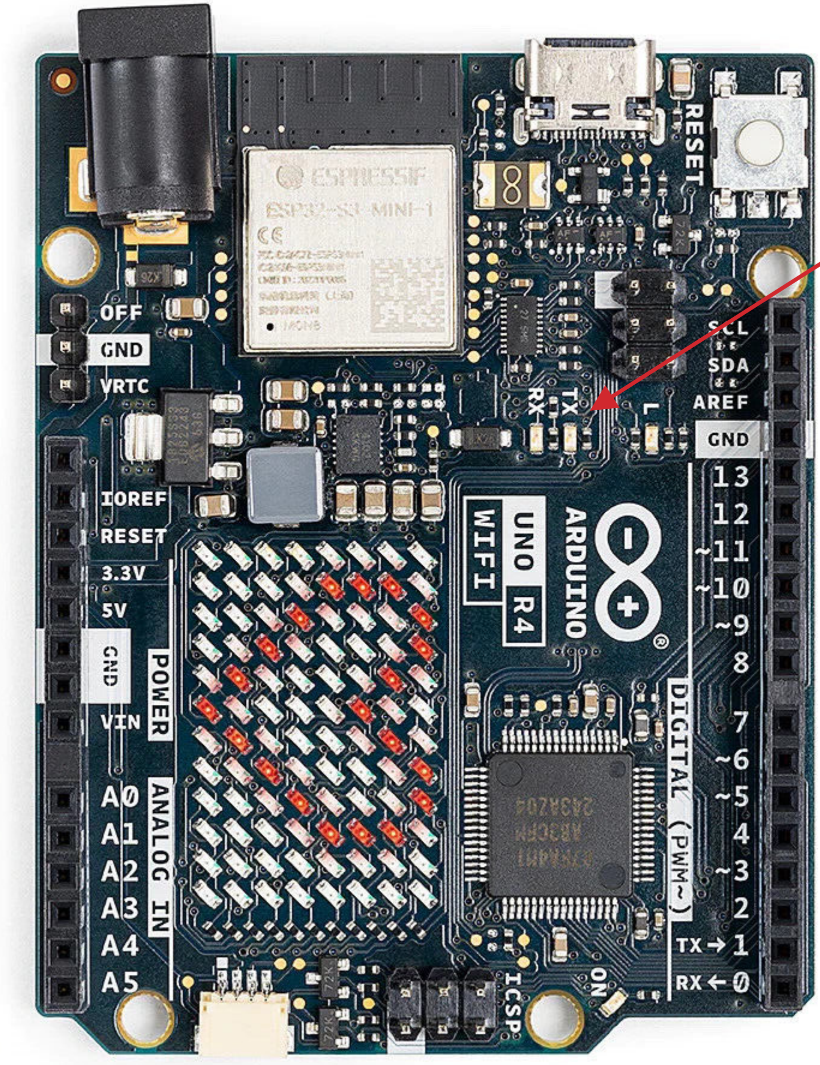
This information is sent over a “**serial port**”: an electronic connection that uses high/low voltages to represent that information.

”**Serial**” means that the information is transmitted 1 bit at a time. (Vs. “Parallel” where more connections are used to send more than one bit at a time.)

The microcontroller’s serial port is connected to a chip on the Arduino that translates this data to work over USB to be received on your computer.

We receive this information on your computer using the Arduino Serial Monitor.

# PHYSICAL IMPLICATIONS



Serial activity LEDs

Digital Pins 1 and 0 are labeled

TX-> “Serial Transmit”
RX<- “Serial Receive”

Try to avoid using these pins, especially if you are sending/receiving Serial data in your program.

# INITIALIZING THE SERIAL PORT

```
void setup() {
   Serial.begin(int baudrate);
}
// initialize the serial port before using it
// must be in setup()
// only do this ONCE in your program
```

Baud rate is the speed in bits per second that data is sent over the serial port

There is a standard set of baud rates available, and you must use one of these. We most commonly use: **9600** and **115200**

# SENDING SERIAL DATA

```
Serial.print(something);
// print something as ASCII-formatted text to the Serial port
// ‘something’ can be a string, int, char, or float

Serial.println(something);
// same as Serial.print, but adds a line break after the
characters to be displayed
```

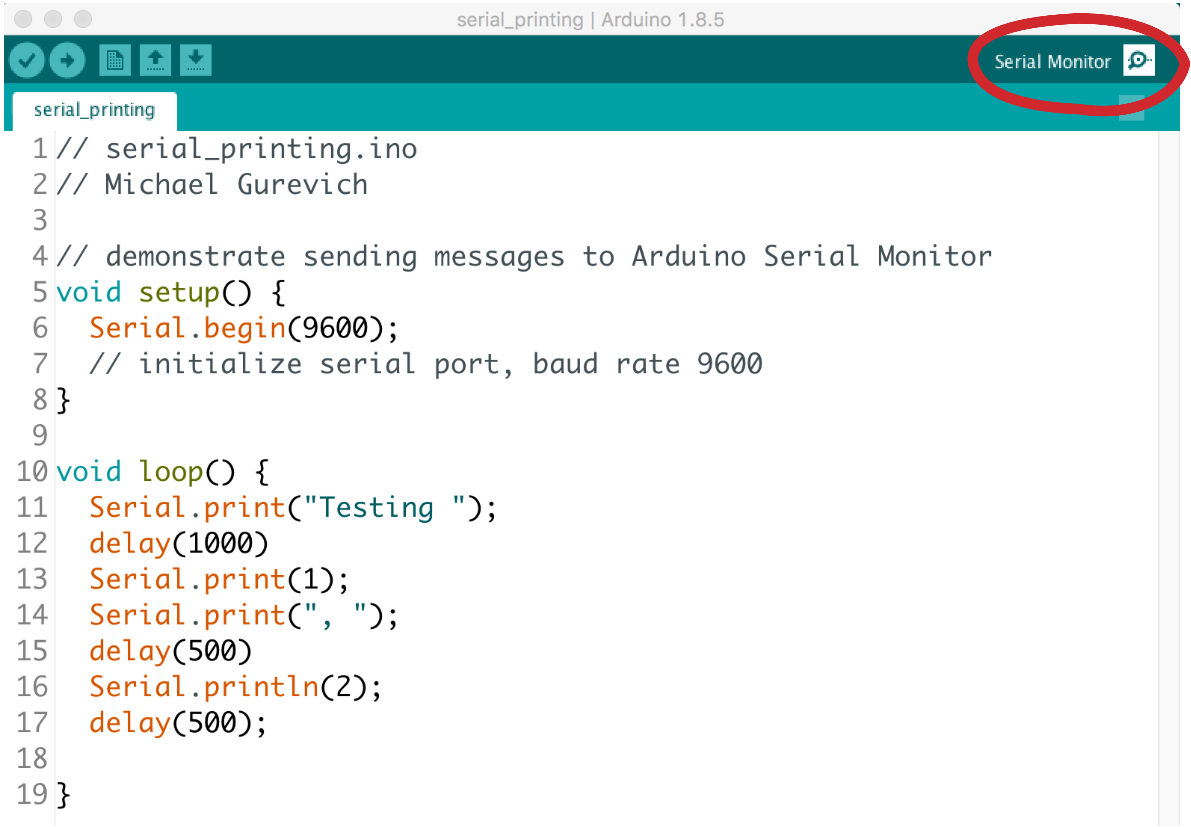
| Code | outcome |
|---|---|
| `Serial.print(42);` | 42 |
| `Serial.print(5.4321);` | 5.43 |
| `Serial.print(’A’);` | A |
| `Serial.print(“bananas”);` | bananas |

**For more advanced examples, see:**

https://www.arduino.cc/reference/en/language/functions/communication/serial/print/

https://www.arduino.cc/reference/en/language/functions/communication/serial/println/

# USING THE ARDUINO SERIAL MONITOR



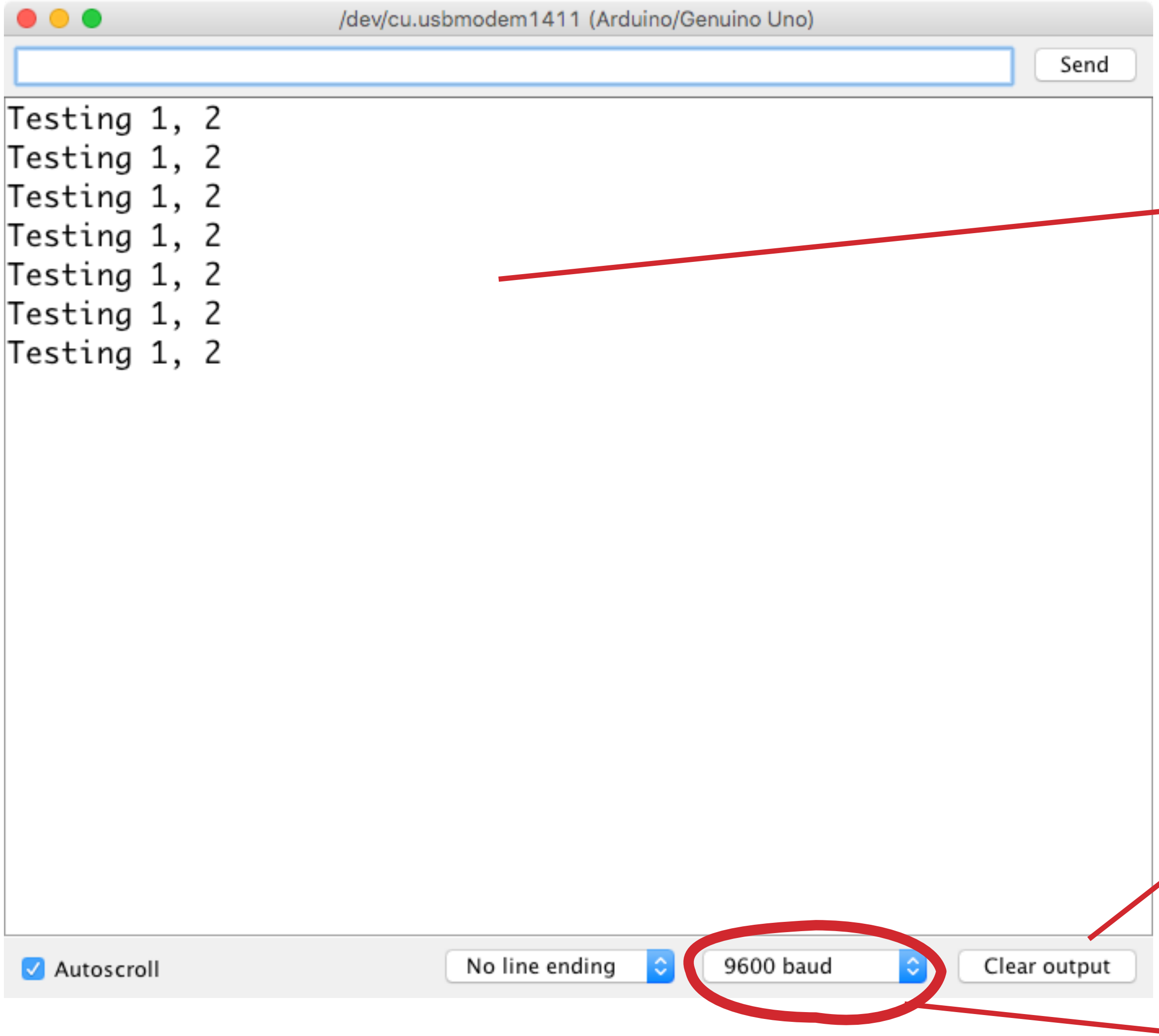
```
// serial_printing.ino
// Michael Gurevich

// demonstrate sending messages to Arduino Serial Monitor
void setup() {
  Serial.begin(9600);
  // initialize serial port, baud rate 9600
}

void loop() {
  Serial.print("Testing ");
  delay(1000)
  Serial.print(1);
  Serial.print(", ");
  delay(500)
  Serial.println(2);
  delay(500);

}
```

Click here to open Serial Monitor.

Or Tools->Serial Monitor

Or cmd+shift+M

# SERIAL MONITOR WINDOW



```
Testing 1, 2
Testing 1, 2
Testing 1, 2
Testing 1, 2
Testing 1, 2
Testing 1, 2
Testing 1, 2
```

Incoming serial messages are displayed here.

You can clear the display window if you want

Make sure baud rate matches what you are using in your program

# EXERCISE 1

Write a program that:

- Detects a button press (use `INPUT_PULLUP`, no external resistor)
- Lights an LED for as long as the button is held down
- Plays a tone for as long as the button is held down
- Uses `Serial.println`() to send a number each time the button is pressed. It should start at 0 and count up, only sending a single new value at the onset of each press.

# RANDOMNESS

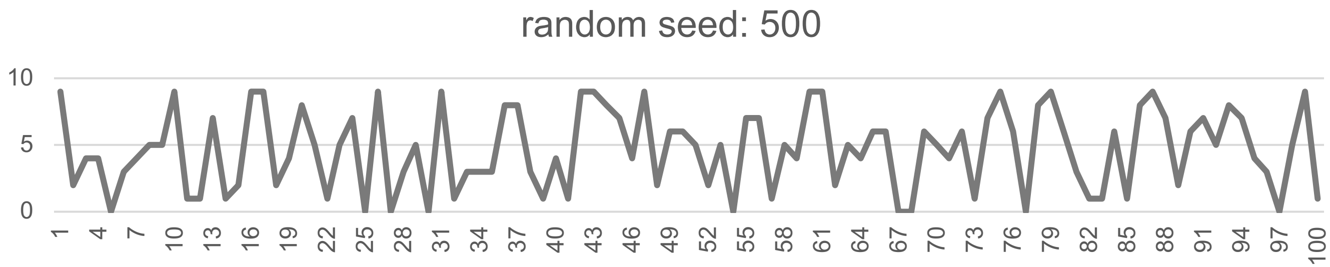
```
unsigned long random(max)
// returns a random integer between 0 and max-1

Unsigned long random(min,max)
// returns a random integer between min and max-1
```

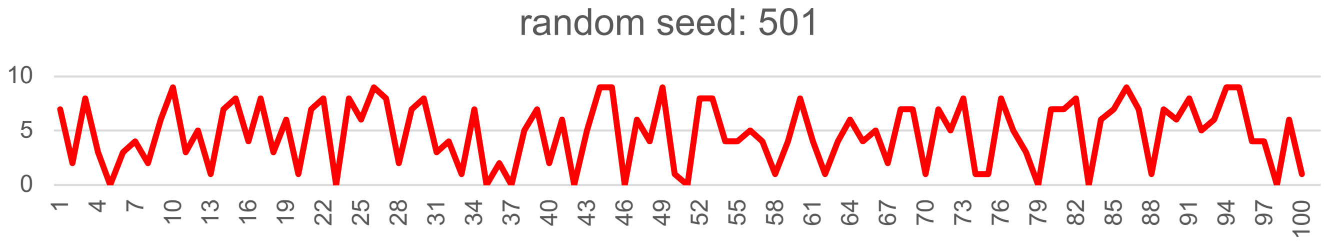
See: https://www.arduino.cc/reference/en/language/functions/random-numbers/random/

# RANDOM SEED

Random number generator is a complex function that takes an argument. That argument is known as the **random seed.**

For a given random seed, the sequence of numbers generated by successive calls to `random()` will always be the same.


random seed: 500


random seed: 501

*For your sequence of random numbers to be different each time you run a program, you need to choose a random value for the seed each time.*

# SELECTING A RANDOM SEED

```
randomSeed(unsigned long seed)
// sets the random seed to be the value seed
```

You can see one technique for selecting the seed here: https://www.arduino.cc/reference/en/language/functions/random-numbers/random/

- A much better way is to capitalize on the fact that user input happens at arbitrary times:
- Use the elapsed time since the program began to a point where the user triggers an input to generate the random seed.
- *Question: How do you prompt the user to trigger an input?*

# SELECTING A RANDOM SEED

# MEASURING ELAPSED TIME WITH MILLIS()

```
long millis()
// returns the number of milliseconds elapsed since a
// program began running
```

# ARDUINO DATA TYPES

| type | description | size | range |
|---|---|---|---|
| bool | Boolean, true/false | 8 bits? | true, false |
| byte<br>uint8_t | Positive integer (incl 0) | 8 bits, unsigned | 0 to 255 |
| int8_t<br>signed char | Signed integer | 8 bits, signed | -128 to 127 |
| char | Character; ASCII code | 8 bits, ambiguous | 0 to 255 or<br>-128 to 127 |
| int<br>int16_t | Signed integer | 16 bits, signed | -32768 to 32767 |
| unsigned int<br>uint16_t | Positive integer (incl 0) | 16 bits, unsigned | 0 to 65535 |
| long<br>int32_t | Signed integer | 32 bits, signed | -2,147,483,648 to 2,147,483,647 |
| unsigned long<br>uint32_t | Positive integer (incl 0) | 32 bits, unsigned | O to 4,294,967,295 |
| float | Decimal number | 32 bits, signed | -3.4028235E38 to 3.4028235E38 |

# EXERCISE 2

Write a program that uses `millis()` to generate a random seed when a user presses a button.

Exactly one second after the user presses the button, print a sequence of 10 random numbers between 0 and 99 to the Arduino Serial Monitor. Make sure the numbers are readable.

Run your program multiple times to prove that the sequence is different each time.

# ARRAYS OF BUTTON/LED PINS

See multiflash.ino example