Aalto University
School of Arts, Design
and Architecture

Programming 3: Debugging and more inputs

Wearable technology and functional wear

Antti Salovaara

What you'll learn today

Debugging

- = trouble-shooting errors (=bugs) in your code
- Using the console for tracing the logic of your program

Iterative programming style

- Writing programs in small pieces to verify their correctness

Pressure sensor as input

Using a timer instead of delay()

Debugging

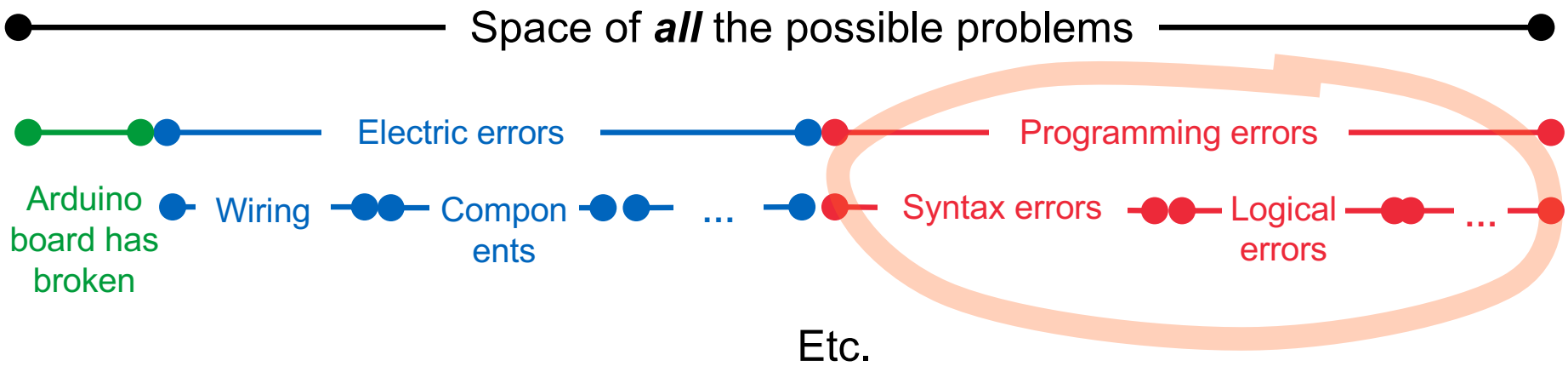
= trouble-shooting errors (=bugs) in your code

Using the console for tracing the logic of your program

When your project does not do what it should

Start narrowing down the reason for error

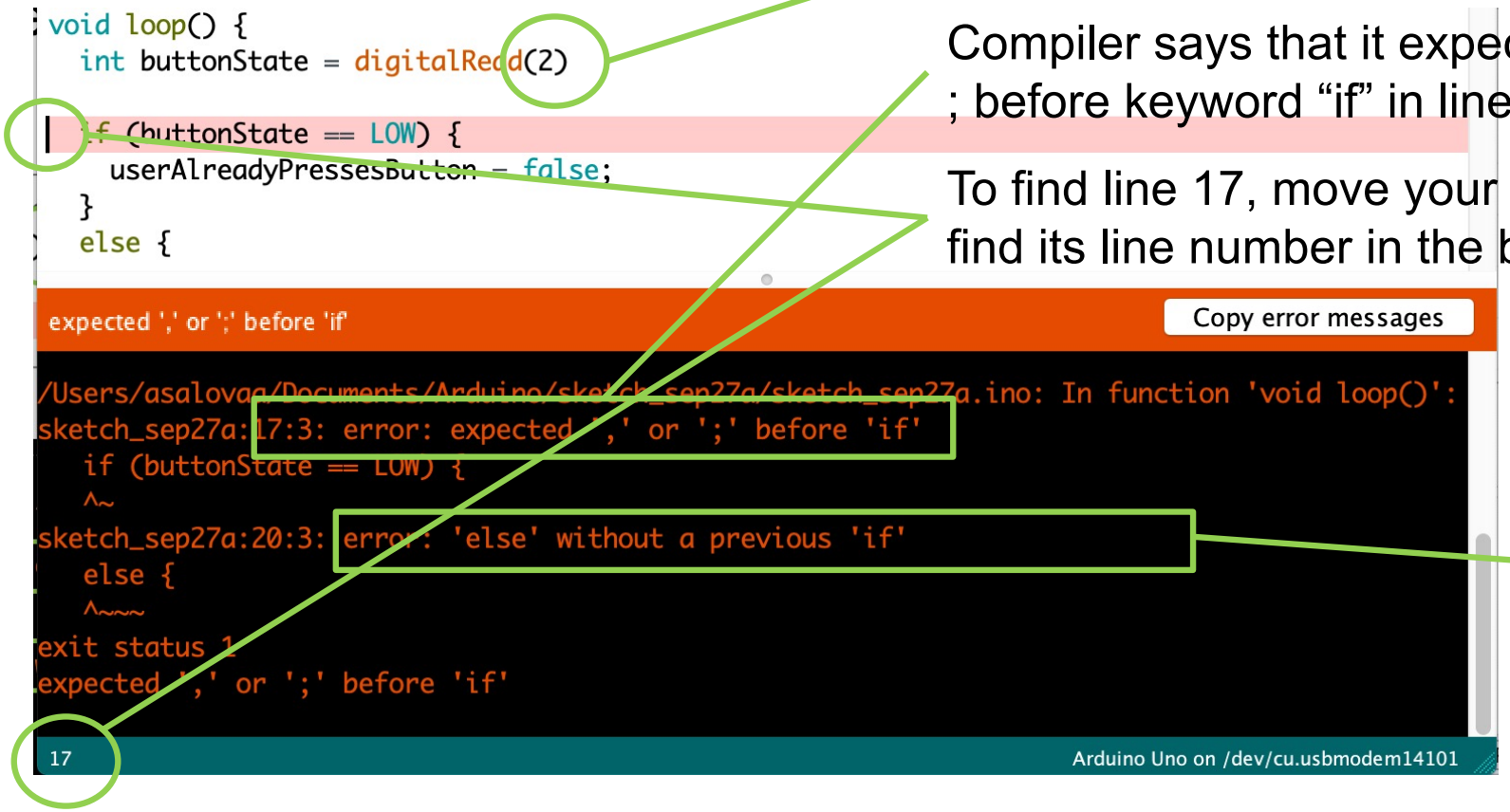
- Is the wiring correct and components working correctly?
- Does the code work correctly?



Syntax errors

Syntax errors are rather easy because they are not hidden:

compiler reports them and tries to give a helpful error message



Missing semicolon ;

Compiler says that it expected to see , or ; before keyword “if” in line 17

To find line 17, move your cursor and find its line number in the bottom row

This message is a side-effect of the earlier error

Logical errors

Logical errors are hidden:

- Compiler does not report errors, but its actions are not those that you want

Quick checks:

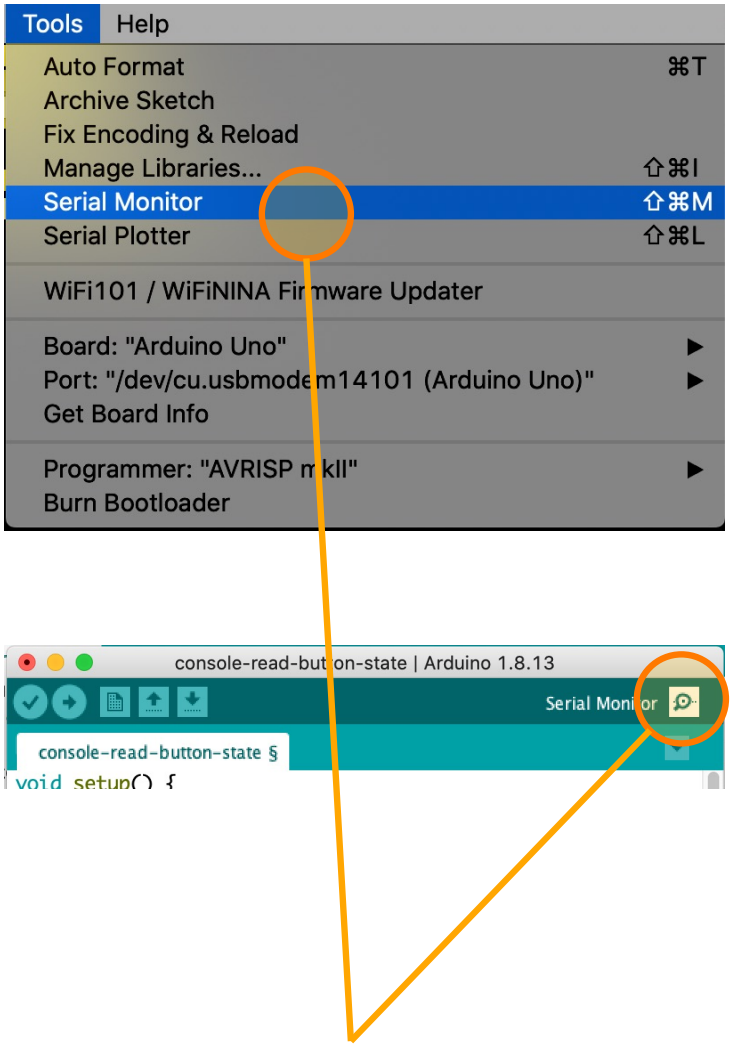
- Have you used = instead of == in `if()` tests?
- Are the curly brackets { } balanced correctly?

```
if (value = 0)
if (value == 0)
```

In harder cases:

- We can trace the actions of the program by printing information to a “console” (see next slide)
- This helps you find out where your program’s logic goes wrong

Printing to console



Two alternative ways to open the console

Console:

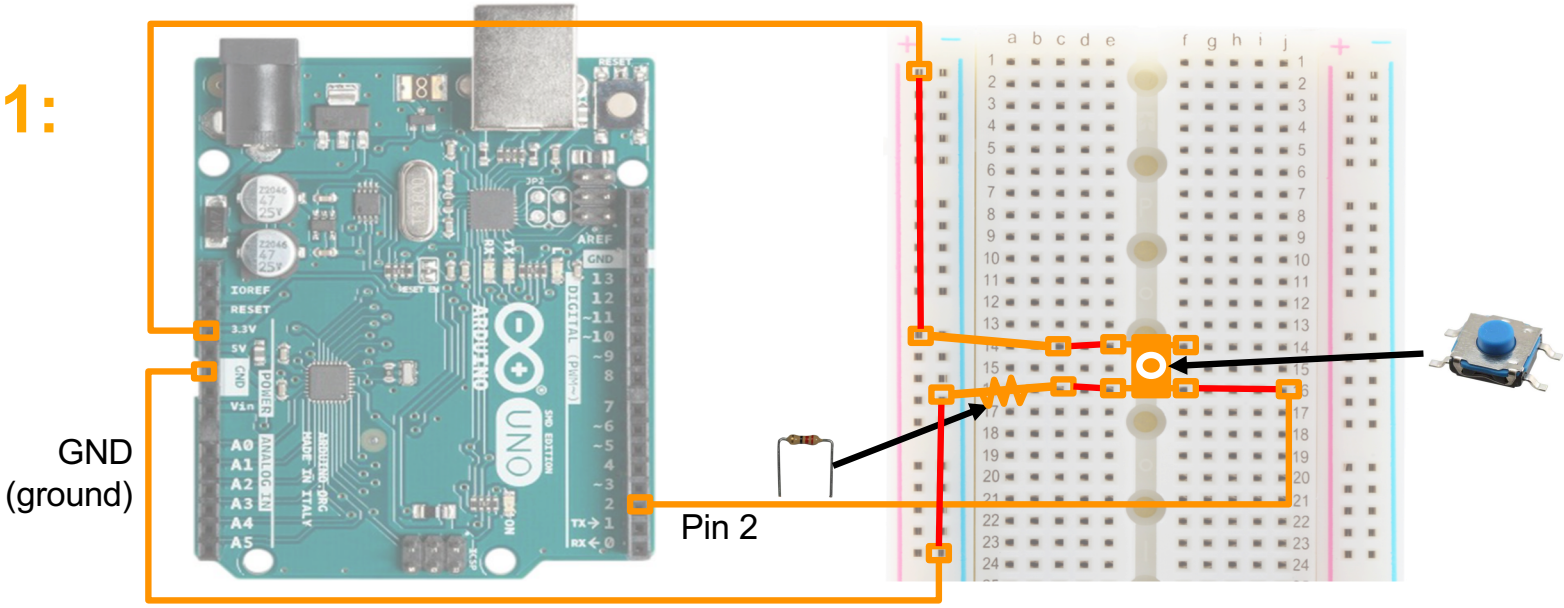
A text window in your computer to which Arduino can print

Printing:

Use Serial.print() to print information to the console

Let's try that out!

Example 1: Reading button state without LED



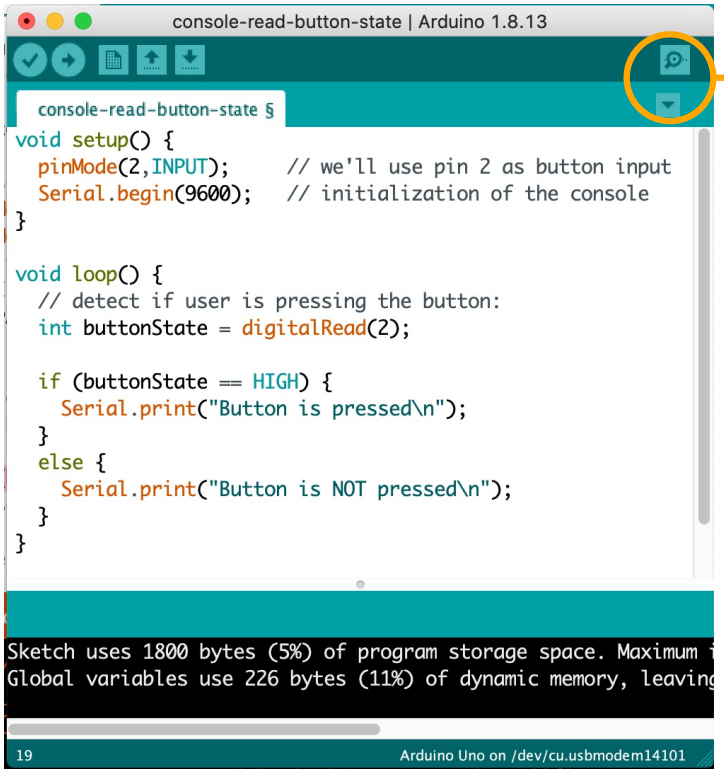

```
void setup() {
  pinMode(2,INPUT);     // we'll use pin 2 as button input
  Serial.begin(9600);   // initialization of the console
}

void loop() {
  // detect if user is pressing the button:
  int buttonState = digitalRead(2);

  if (buttonState == HIGH) {
    Serial.print("Button is pressed\n");
  }
  else {
    Serial.print("Button is NOT pressed\n");
  }
}
```

\n is a “new line” character

console-read-button-state.ino

Testing printing in our project

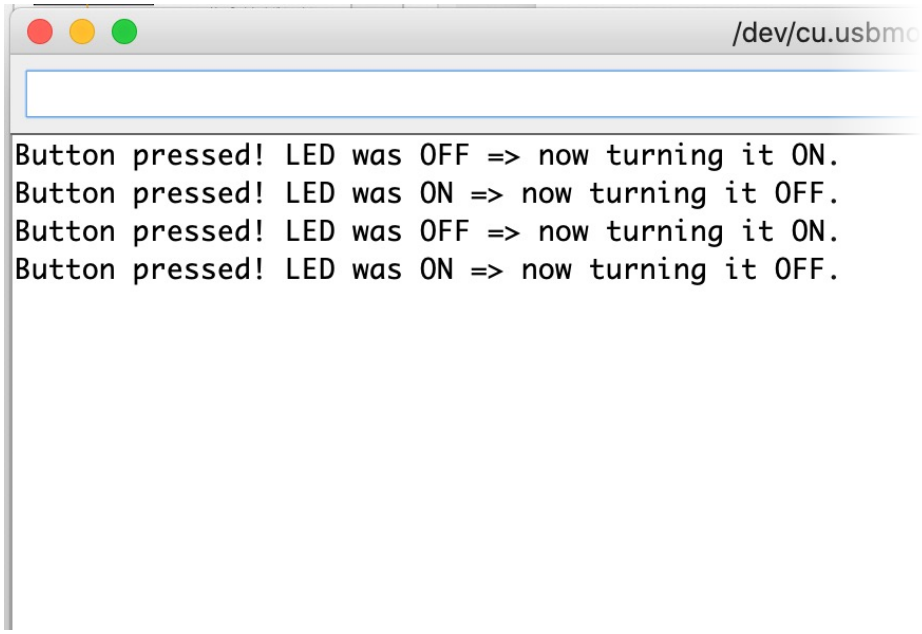
Once the console is open: what happens when you press the button?



Click!

Example 2: Tracing the actions of the simple (buggy) toggling project from lecture 2

```
bool ledIsOn;

void setup() {
  pinMode(2,INPUT);
  pinMode(13,OUTPUT);

  digitalWrite(13,LOW);
  ledIsOn = false;

  Serial.begin(9600);
}

void loop() {

  int buttonState = digitalRead(2);

  if (buttonState == HIGH) {
    Serial.print("Button pressed! ");
    if (ledIsOn == true) {
      Serial.print("LED was ON => now turning it OFF.\n");
      digitalWrite(13,LOW);
      ledIsOn = false;
    }
    else {
      Serial.print("LED was OFF => now turning it ON.\n");
      digitalWrite(13,HIGH);
      ledIsOn = true;
    }
  }
}
```

Observation: the LED turns on/off in a rapid sequence when the button is being pressed. This is why the simple solution did not work and we needed the new userIsPressingButton variable.

Lines that help us understand what is going on

How you can use the console

Add `Serial.print()` commands to those parts of the code that you want to track

Print variable values to the console too

You can use Serial.print() for printing values too

Code

```
Serial.print("User has now pushed the button, because buttonState = ");
Serial.print(buttonState);
Serial.print("\n");
```

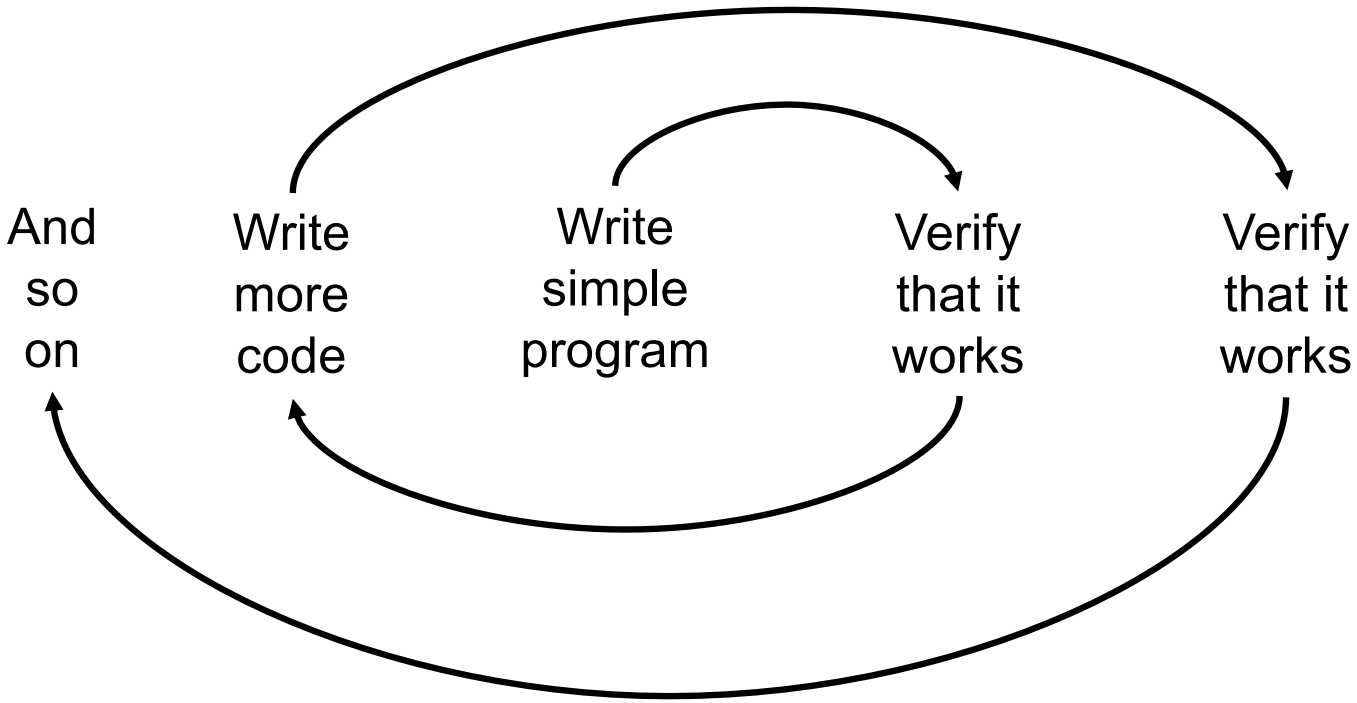
Console

```
User has now pushed the button, because buttonState = 1
```

Iterative programming style

Writing programs in small pieces to verify their correctness

What is “iterative” working style

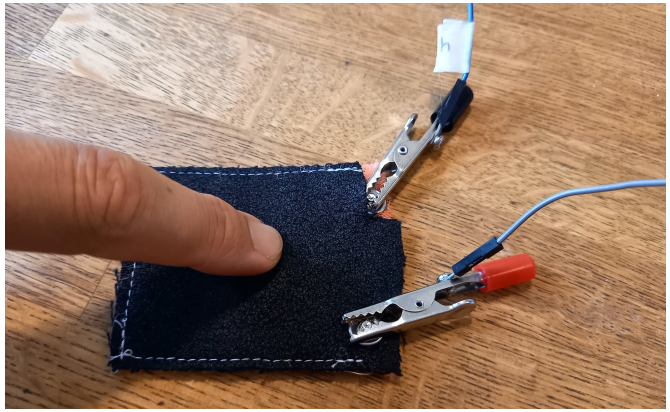


We’ll follow this style in our last project

Pressure sensor as input

Putting together all the things that we have learned

This is our goal:



A more sophisticated version

Pressure level indicator:

High

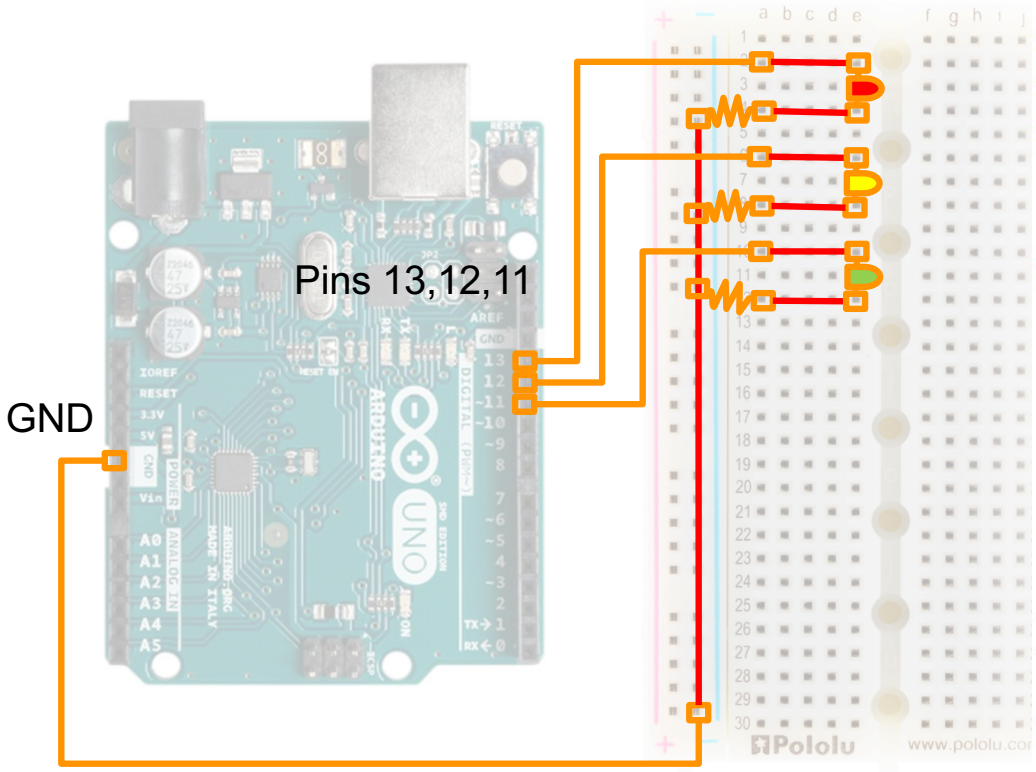
Zero

(all LEDs turned off)

Proceeding in stages

1. Creating LED traffic lights
    - Wiring them
    - Testing from Arduino that they work
2. Creating the pressure sensor
    - Creating the sensor
    - Wiring it
    - Testing from Arduino that they work

1. Creating LED traffic lights

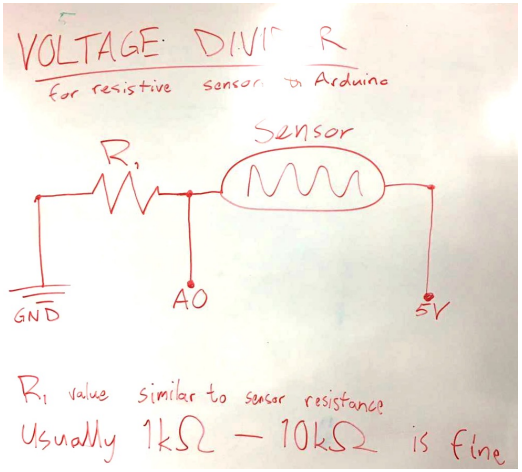



```
int redPin = 13;
int yellowPin = 12;
int greenPin = 11;


void setup() {
  pinMode(redPin,OUTPUT);
  pinMode(yellowPin,OUTPUT);
  pinMode(greenPin,OUTPUT);

  digitalWrite(greenPin,HIGH);
  digitalWrite(yellowPin,HIGH);
  digitalWrite(redPin,HIGH);
}

void loop() {
  // empty because we only test the lights
}
```

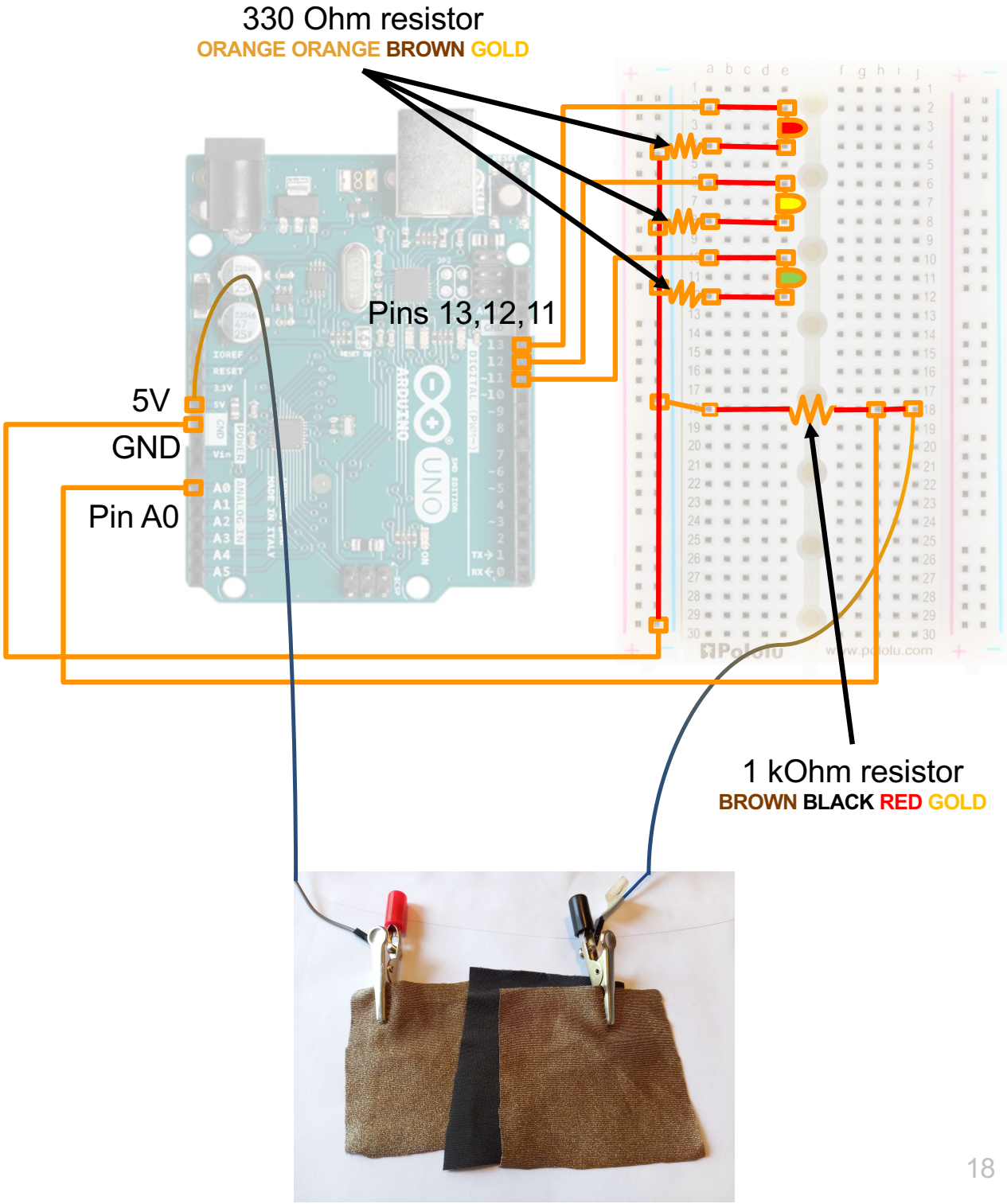
Write this code and upload it. If the lights turn on, then you know that their wiring does not have faults.

2. Creating (also) the pressure sensor

To have the pressure measurement on a suitable level, we split the voltage and measure only a small part of it:



Valtteri's drawing 17 Oct 2020
Thanks Valtteri!



Code for testing the pressure sensor

This code only prints measurements to the console

We can test the pressure value range

The LEDs are not used yet

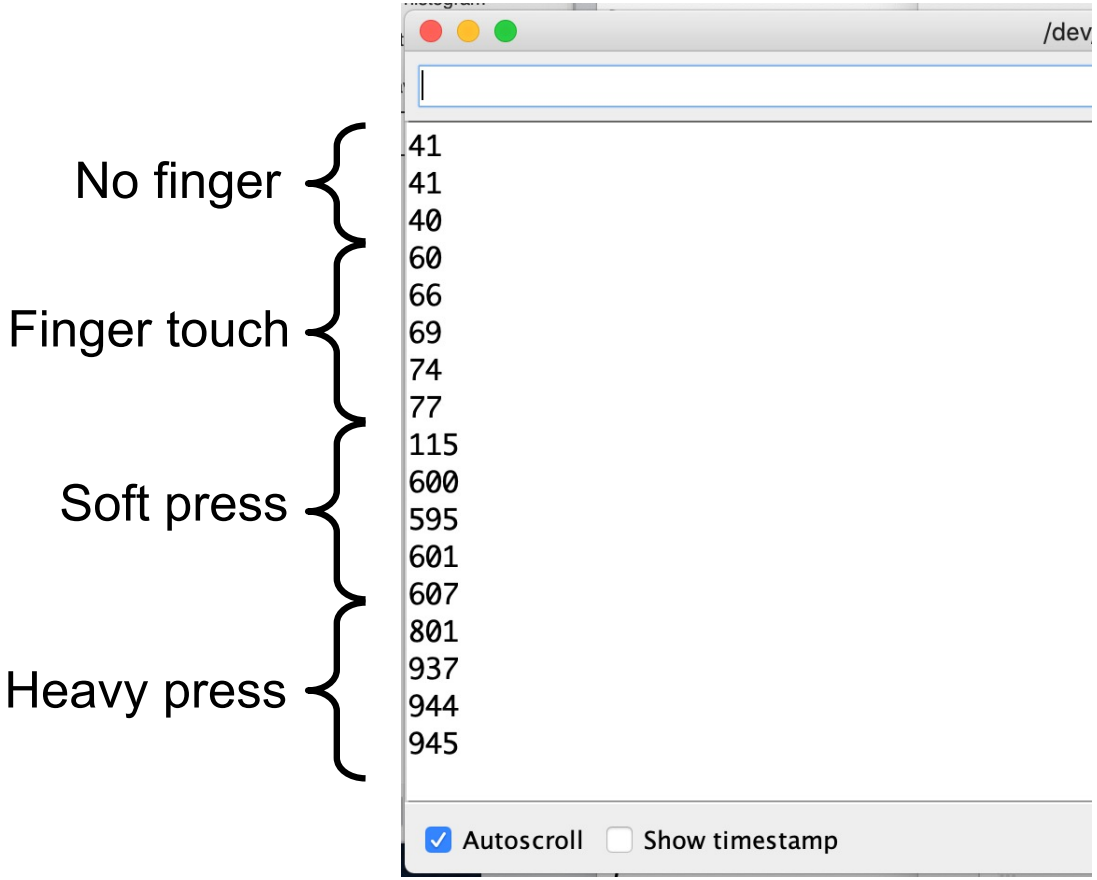
```
int redPin = 13;
int yellowPin = 12;
int greenPin = 11;

int pressurePin = 0;

void setup() {
  // Lights:
  pinMode(redPin,OUTPUT);
  pinMode(yellowPin,OUTPUT);
  pinMode(greenPin,OUTPUT);
  digitalWrite(redPin,LOW);
  digitalWrite(yellowPin,LOW);
  digitalWrite(greenPin,LOW);

  Serial.begin(9600);
  Serial.print("Start\n");
}

void loop() {
  int val = analogRead(pressurePin);
  Serial.print(val);
  Serial.print("\n");
  delay(500);
}
```

New lines

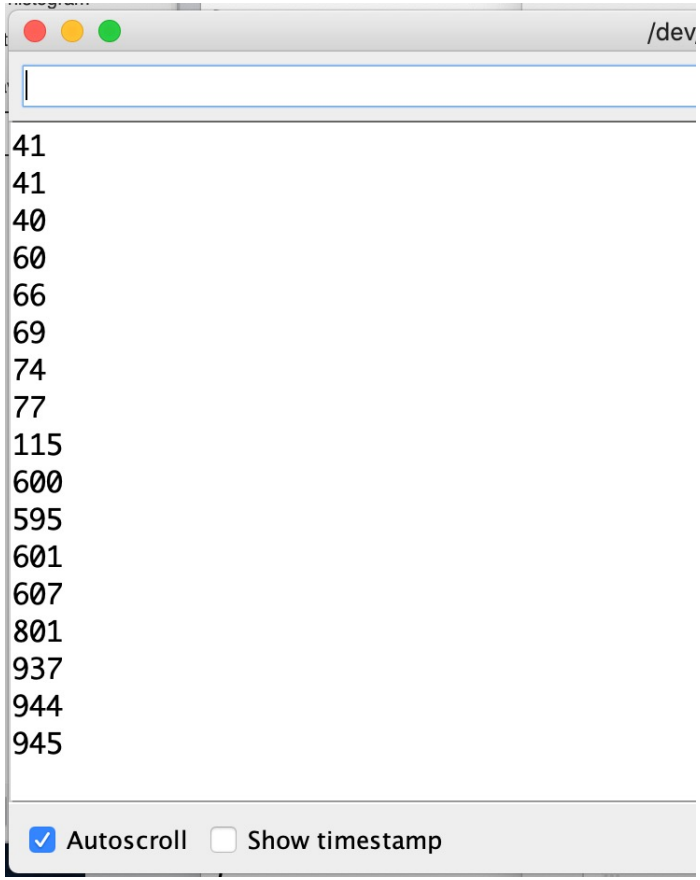
Measurements

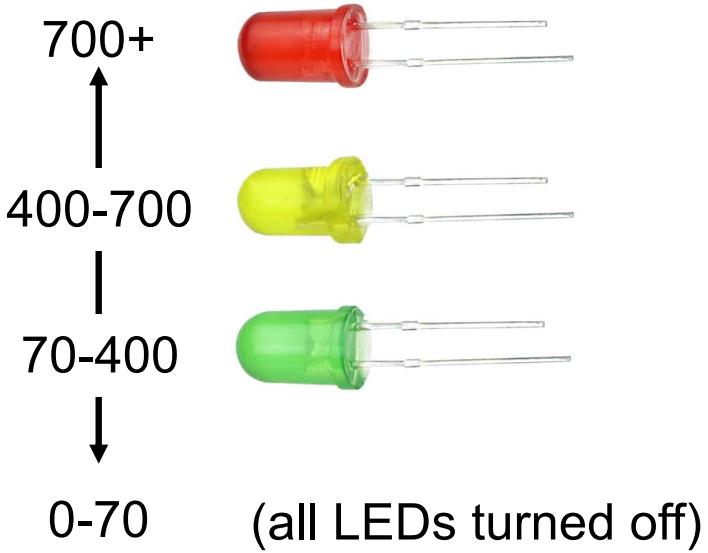
Now with steps 1 and 2 we have verified that the wiring for both the LEDs (step 1) and the pressure (step 2) work!

We only have to write the code that links suitable pressure levels to the lights.

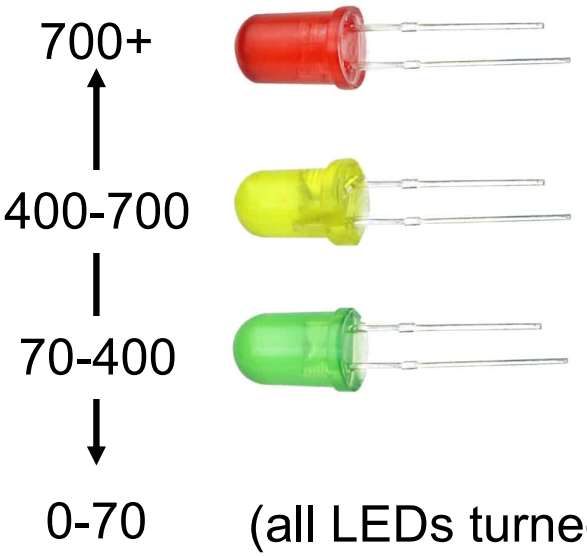


Combining the LEDs and pressure





Combining the LEDs and pressure




```
int redPin = 13;
int yellowPin = 12;
int greenPin = 11;

int pressurePin = 0;

void setup() {
  pinMode(redPin,OUTPUT);
  pinMode(yellowPin,OUTPUT);
  pinMode(greenPin,OUTPUT);

  digitalWrite(redPin,HIGH);
  digitalWrite(yellowPin,HIGH);
  digitalWrite(greenPin,HIGH);

  Serial.begin(9600);
  Serial.print("Start\n");
}

void loop() {
  int val = analogRead(pressurePin);
  Serial.print(val);
  Serial.print("\n");

  if (val > 700) {
    digitalWrite(redPin,HIGH);
    digitalWrite(yellowPin,HIGH);
    digitalWrite(greenPin,HIGH);
  }
  else if (val > 400) {
    digitalWrite(redPin,LOW);
    digitalWrite(yellowPin,HIGH);
    digitalWrite(greenPin,HIGH);
  }
  else if (val > 70) {
    digitalWrite(redPin,LOW);
    digitalWrite(yellowPin,LOW);
    digitalWrite(greenPin,HIGH);
  }
  else {
    digitalWrite(redPin,LOW);
    digitalWrite(yellowPin,LOW);
    digitalWrite(greenPin,LOW);
  }
}
```

New lines

Using a timer instead of delay()

= keeping your Arduino responsive (“alive”) all the time

Timers

What can you do with timers?

- “Do the following things 10 seconds from now”
- “Do the following things every 10 seconds”
- … While also doing other things in the meanwhile

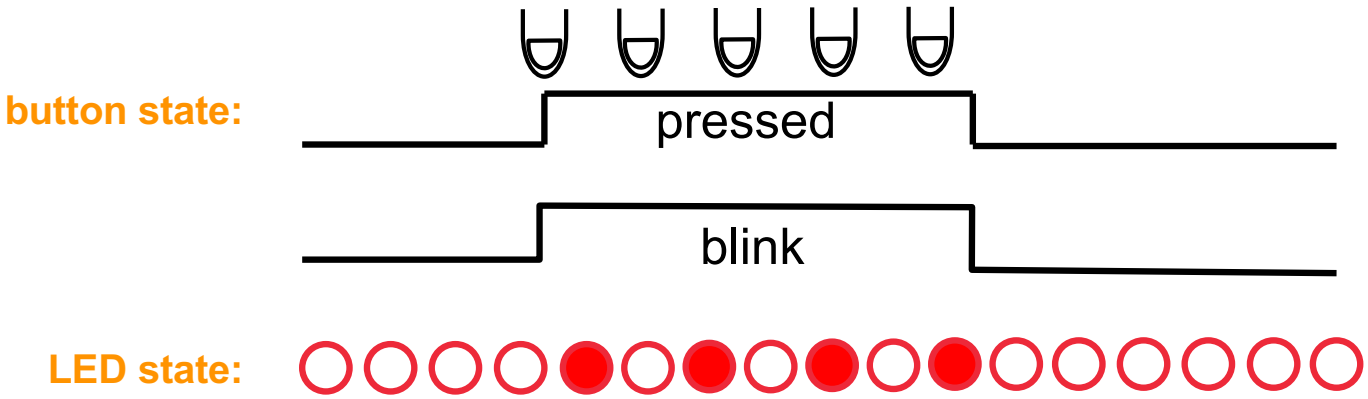
In other words, timers enable simple parallel processing:

- Wait for the right moment
- Do other things at the same time

Arduino community has created lots of code for everyone to use

- We’ll use a simple timer library (arduino-timer) that I found by Googling “arduino timer”
- It can be installed from Arduino community’s libraries

The logic

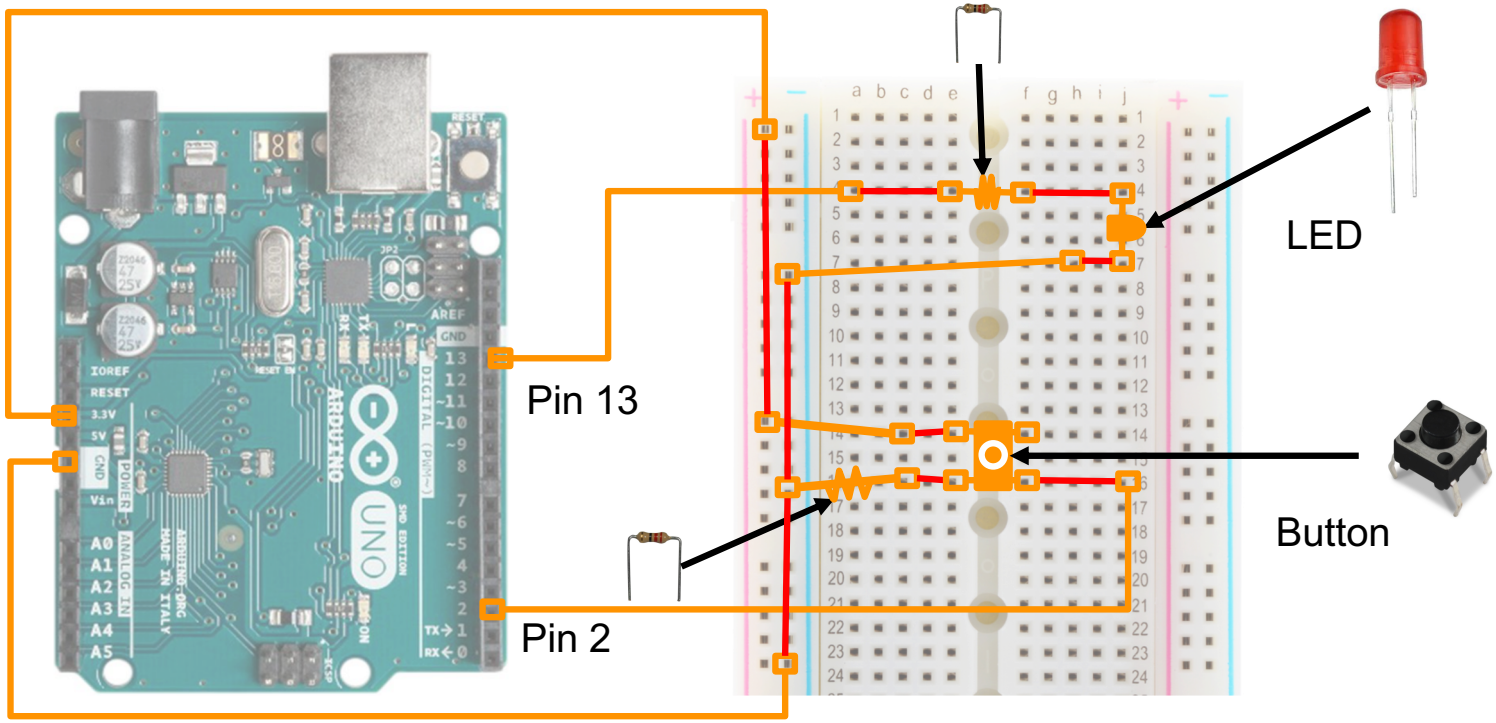


We work iteratively:

- Let’s first create a blinking LED without any button logic
- Only when that works, we add the button

Blinking a LED when a button is down

We use the same wiring as in our toggling LED project

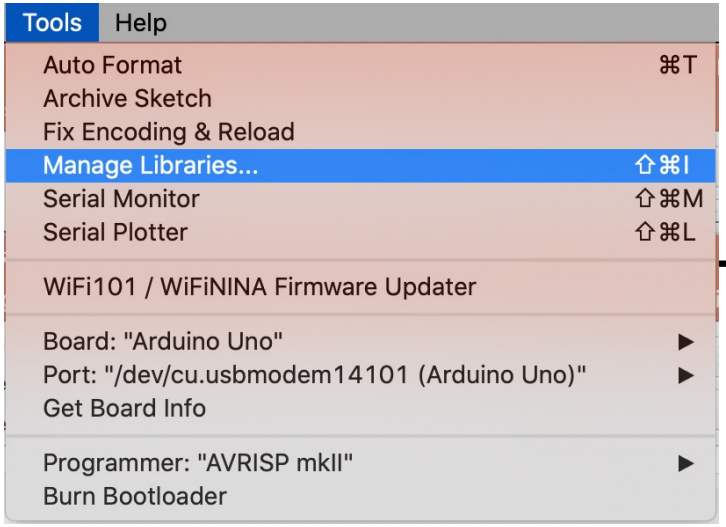


What we'll do:

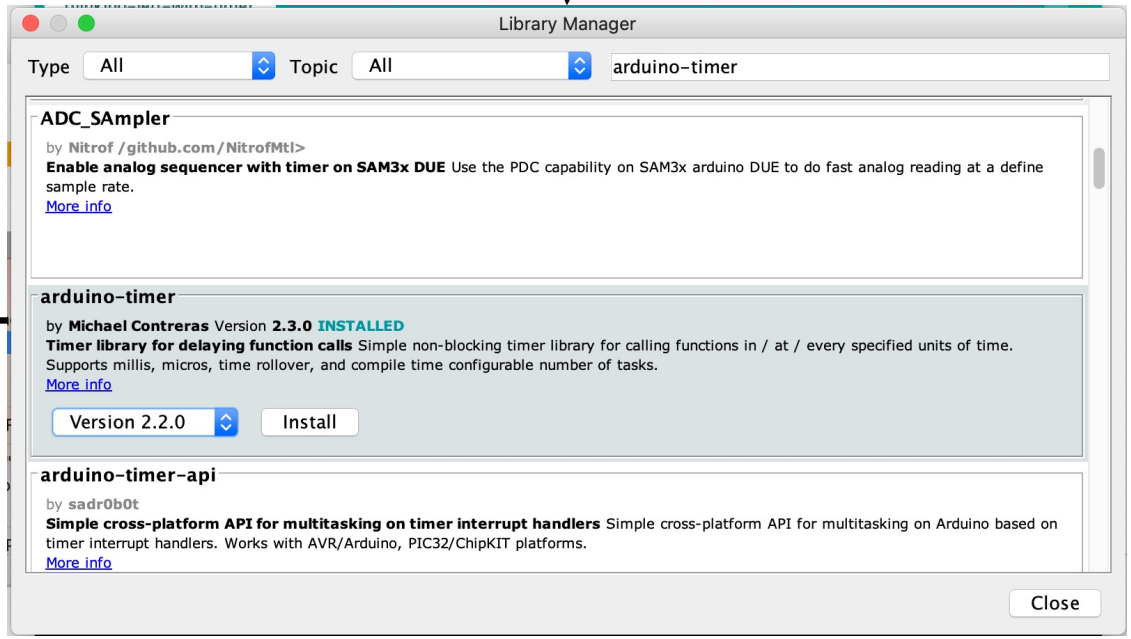
- Make a blinking LED that only blinks when user is pressing the button
- Solve the project one step at a time
- Program without hard-coding

How to install arduino-timer

1. Go to Manage Libraries in Arduino's top menu:



2. Write "arduino-timer" to search box
3. Scroll until you find the library



4. Select newest version.
5. Press Install

DONE!

```
// Preparations for the timer:
// https://www.arduino.cc/reference/en/libraries/arduino-timer/
#include <arduino-timer.h>                  // Include a timer library in our project
auto timer = timer_create_default();        // Create timer object
bool lightTogglerFunction(void*);           // Announcement of the function that the timer will call

// Our own soft-coded variables:
int lightState = LOW;
int lightPin = 13;
int blinkSpeed = 500;

void setup() {
  pinMode(lightPin,OUTPUT);
  digitalWrite(lightPin,LOW);

  // Create a timer that toggles the light in a given pin at a given interval:
  timer.every(blinkSpeed,lightTogglerFunction);
}

void loop() {
  timer.tick();                             // Update the timer
}

bool lightTogglerFunction(void*) {
  if (lightState == LOW) {
    digitalWrite(lightPin,HIGH);
    lightState = HIGH;
  }
  else {
    digitalWrite(lightPin,LOW);
    lightState = LOW;
  }

  // We keep the timer active by returning "true":
  return true;
}
```

Avoiding hard coding: We define the values for all the parameters here. Then we don't need to touch other parts of the code at all if we wish to change the pin or use a different blinking speed.

Step 1: LED that blinks forever using a timer

blinking-led-with-timer.ino

Step 2: LED that blinks only if button is pressed

```
// Preparations for the timer, based on these instructions:
// https://www.arduino.cc/reference/en/libraries/arduino-timer/

// Include a timer library:
#include <arduino-timer.h>
// Create a timer object:
auto timer = timer_create_default();
// Announce the function that the timer will call:
bool lightTogglerFunction(void*);

// Our own soft-coded variables:
int lightState = LOW;
int lightPin = 13;
int blinkSpeed = 500;
int buttonPin = 2;
bool doingBlinking;

void setup() {
  pinMode(lightPin,OUTPUT);
  digitalWrite(lightPin,LOW);

  pinMode(buttonPin,INPUT);
  doingBlinking = false;
}
```

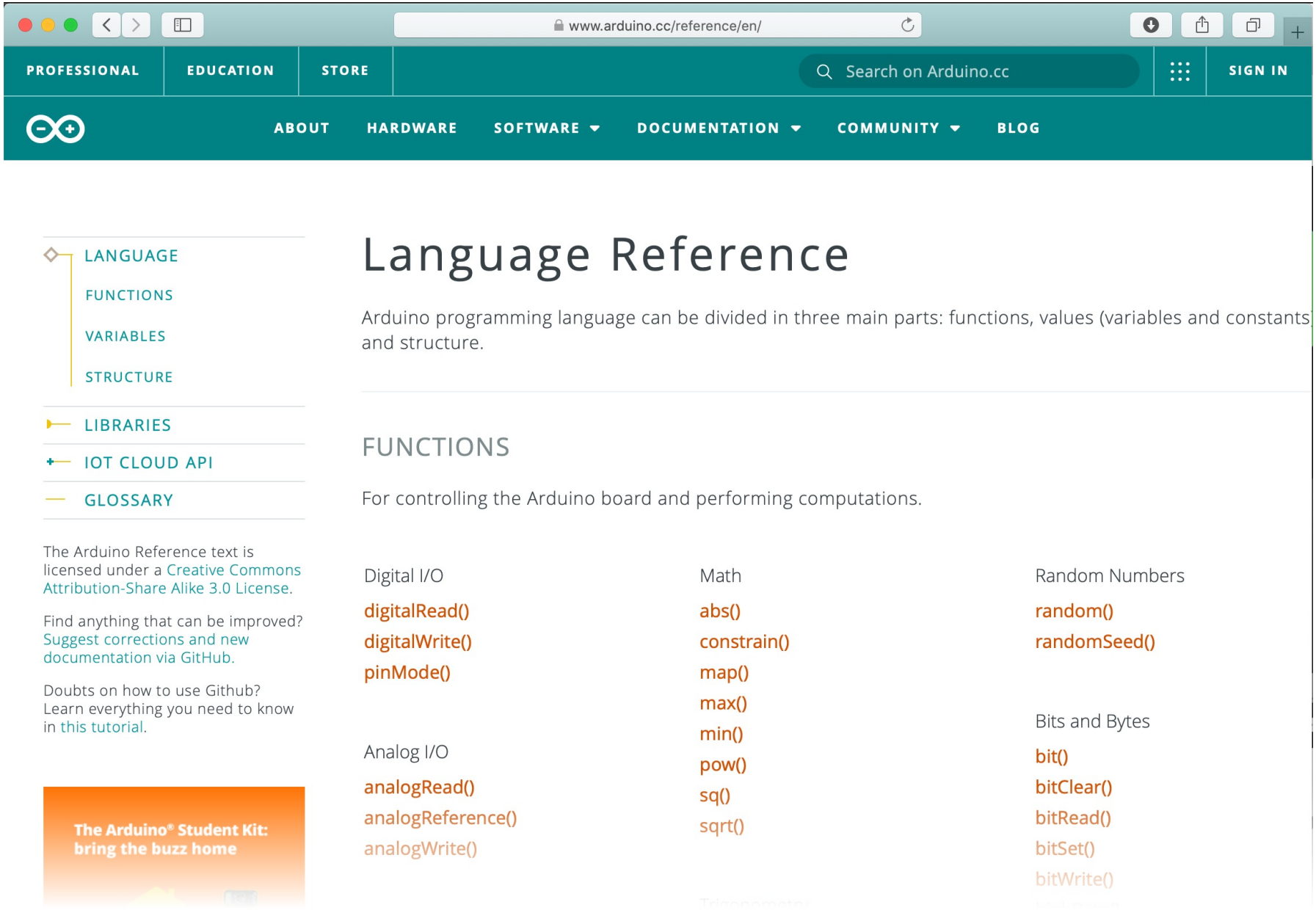
Done!

```
void loop() {
  timer.tick();                          // Update the timer

  int buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH) {
    if (doingBlinking == false) {
      // turn on blinking:
      doingBlinking = true;
      timer.every(blinkSpeed,lightTogglerFunction);
    }
    else {
      // If we are already blinking we don't need to change anything.
    }
  }
  else {
    // turn off blinking:
    doingBlinking = false;
    digitalWrite(lightPin,LOW);
  }
}

bool lightTogglerFunction(void*) {

  if (doingBlinking == false) {
    // If blinking has been turned off, stop this timer by returning "false":
    return false;
  }
  else {
    if (lightState == LOW) {
      digitalWrite(lightPin,HIGH);
      lightState = HIGH;
    }
    else {
      digitalWrite(lightPin,LOW);
      lightState = LOW;
    }

    // We keep the timer active by returning "true":
    return true;
  }
}
```

Where to learn more

https://www.arduino.cc/reference/en/

Arduino reference



Thank you!