

Project 1 Lessons

Project 1 Lessons

Functions can be abstractions

- Hide details from their “callers”
- In our case: we are hiding the details of manipulating bits
- But: must only manipulate the relevant bits. Otherwise, the function could interfere with the activities of other functions

Project 1 Lessons

- Documentation is important
 - Three levels: file/project, function and in-line
 - Each has their own purpose
- Integrate code review feedback: your code will be used in subsequent projects (and points will be subtracted for persistent errors)

Loop() and Control

```
void loop() {  
    int val;  
    for(val = 0; val < 255; ++val) {  
        display_value(val);  
        delay(100);  
    }  
}
```

Loop() and Control

```
void loop() {  
    int val;  
    for(val = 0; val < 255; ++val) {  
        display_value(val);  
        delay(100);  
    }  
}
```

Nested loop design can be problematic...

- The Arduino environment performs other tasks outside of loop(). These can be time-critical.
- Therefore: we want to execute loop() and get out as quickly as possible

Loop() and Control

```
void loop() {  
    int val;  
    for(val = 0; val < 255; ++val) {  
        display_value(val);  
        delay(100);  
    }  
}
```

- As implemented, val is only a temporary variable: it disappears after we leave the loop() function
- How do we repair this?

One Solution: Global Variables

```
int val = 0;

void loop() {
    ++val;
    if(val > 255)
        val = 0;
    display_value(val);
    delay(100);
}
```


Alternative (and Cleaner) Solution: Static Variables

```
void loop() {  
    static int val = 0;  
  
    ++val;  
    if(val > 255)  
        val = 0;  
    display_value(val);  
    delay(100);  
}  
}
```

Alternative (and Cleaner) Solution: Static Variables

```
void loop() {  
    static int val = 0;  
  
    ++val;  
    if(val > 255)  
        val = 0;  
    display_value(val);  
    delay(100);  
}
```

- val is now persistent across calls to loop()
- The initialization of val only happens at the beginning of your program

Project 2: Analog Sensor Processing

Project 2: Analog Sensor Processing

- Each group has two Sharp distance sensors
- Connect to your circuit board & then to the Teensy
- Code: read the raw sensor state
- Collect data and analyze
- Model your sensors
- Write a function that returns calibrated distance values

Component 1: Circuit

Connect each sensor to circuit board:

- Power: +5V power: Vin on the Teensy)
- Ground
- Signal: analog input pin on the Teensy

Component 2: Test Function

Loop():

- Read the raw sensor values
- Print out the sensor values

Using the USB Serial Port

- In this context, *serial* refers to the exchange of character-based information
- Setup():
`Serial.begin(9600);`
- Loop():
`Serial.println("Foo");`
`Serial.print(42);`
- Viewing the output:
 - Use the *serial monitor* (upper right corner of the Arduino window)

Reading from an Analog Port

- Define the analog pin at the top of your INO file:

```
const int analogPin = 1;
```

- The “1” corresponds to analog input A1

- Read from the pin:

```
int val = analogRead(analogPin);
```

The use of the constant is not strictly required, but it makes for much more readable code

Component 3:

Data Collection and Analysis

- Take at least 5 samples each for: 5, 6, 8, 10, 14, 20, 30, 40, 60, 80 cm.
- Two plots for each sensor:
 - Sensor value as a function of distance (cm)
 - Sensor value as a function of $1/\text{distance}$ (1/cm)

Component 4: Sensor Model

Fit a *simple* function to your data

- 5-6cm should be captured well
- Adjust the other parameters of your function to capture the rest of your data as best as possible

Component 5: Implement the Model

- Define a new variable type in "project.h":

```
typedef enum {  
    DISTANCE_LEFT = 0,  
    DISTANCE_RIGHT = 1  
} DistanceSensor;
```

- Implement the function:

```
float read_distance(DistanceSensor side)
```

- Return value in cm

Component 6: Test

- Take at least 5 samples each for: 5, 6, 8, 10, 14, 20, 30, 40, 60, 80 cm.
- Plot sensed distance value as a function of true distance (one curve for each sensor)

Hints

- The sensors can interfere with one-another
- Start this project early
- Keep things simple