Programming Arduino

BASICS

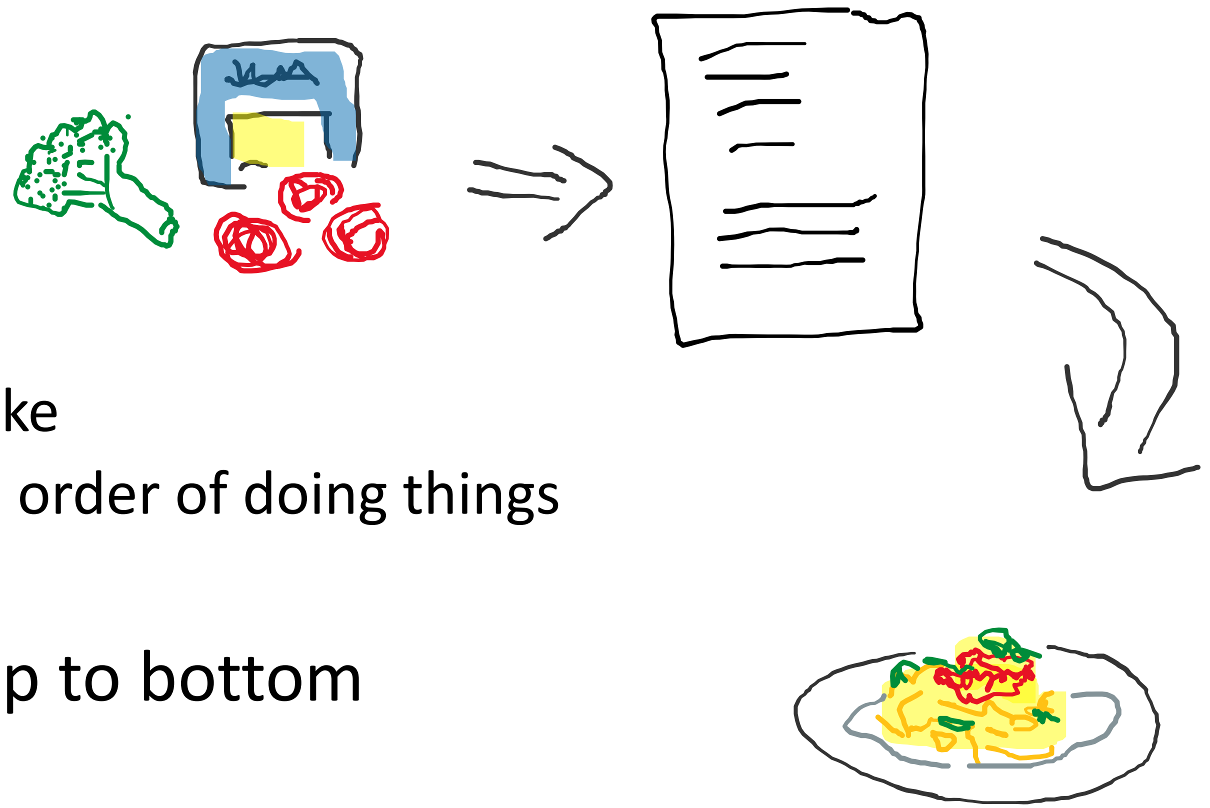
Basics for Arduino Programming

- This basic tutorial will include explanation of:
  - Intro
    - Instruction to Machines
  - Part I
    - Variables
    - Functions
    - Logical statements
  - Part II
    - variable scope
    - custom functions

Intructions to Machines

(Syntax and structure)

How to write code?



- Code is like a recipe:
  - variables are incredients
  - functions are some actions to take
  - syntax and structure defines the order of doing things

- As default code is read from top to bottom

How to write code?

keywords are colored

- Code consist of keywords (reserved for specific use), ”free words” (something you can come up yourself) and data (numbers)
- Eventually everything is boiled down to binary numbers (zeros and ones) but luckily we don’t need to learn binary code but we can write more abstract language. Huh!
- In case of Arduino the language we use is C++

$$\begin{array}{r} 0101100 \\ +\,1101100 \\ \hline 10011000 \end{array}$$

How to write code?

;

- Each language has its own syntax. Here is some notes on C++ syntax:
    - Every line (statement) ends with semicolon ;
    - Code inside a function is placed between curly brackets { }
        - Never use semicolon after curly bracket*
    - Function names or variable names can’t have spaces in them or numbers in the beginning. To mark space use underscore _
    - Variables need data type when introduced for the first time.
    - Functions need return type when they are defined.

}

* unless you are defining “a class”, “a struct” or “an enum” which is not happening in the scope of this tutorial

How to write code?

- Functions calls and conditional statements can be used to change the order.
- The following examples are identical:

```
1.  void setup() {
2.      pinMode(13, OUTPUT);
3.  }
```

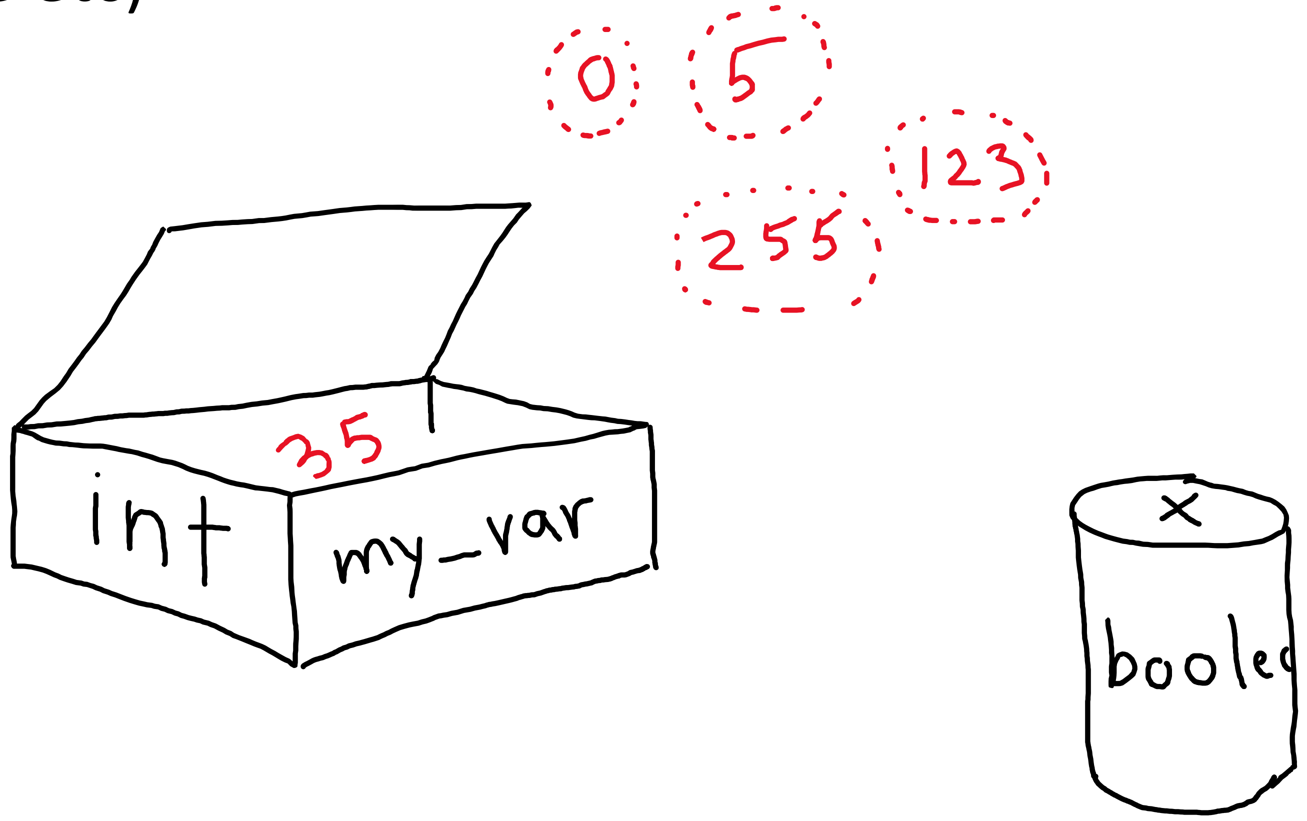
```
1.
    void setup() {          2.
      setPin();
    }  4.

    void setPin() {
  3.  pinMode(13, OUTPUT);
    }
```

Part I

Variables

Variables

- Variables are containers for data (such as number, character, text, table, image etc)



Variables

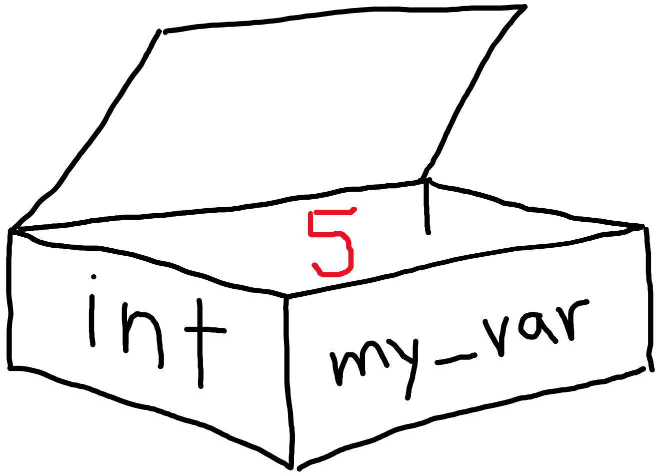
- Variable consist of 3 parts (data type, name, value)

Type of the box

Name of the box

What's inside

```
boolean example_variable = true;
int my_variable = 5;
```



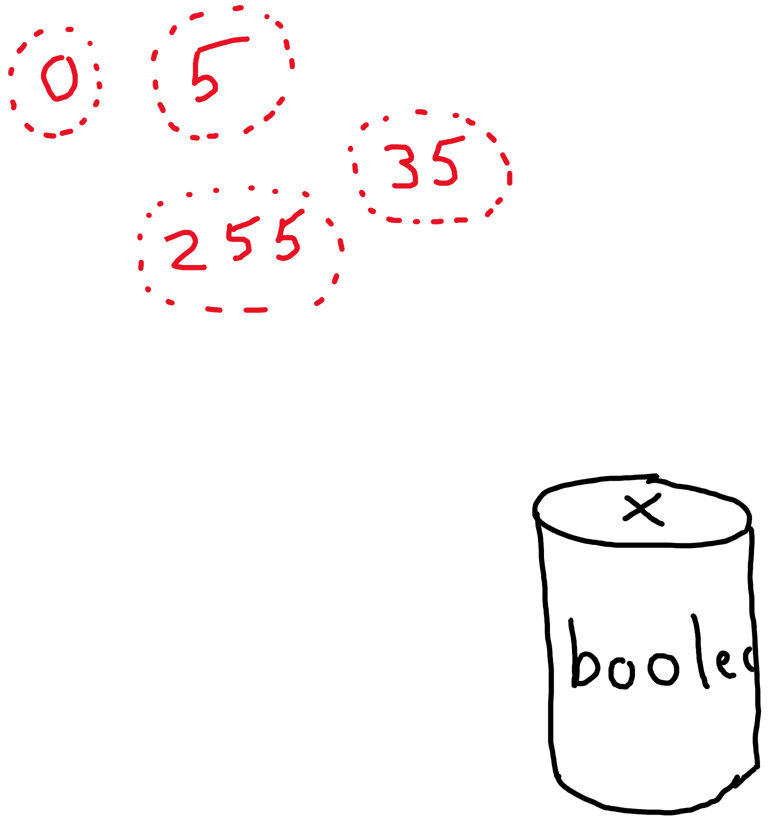
Variable data type should match with the data you are about to store inside it!

Variable name should be unique and readable!

Variable value is assigned to variable with equal sign =

Variables

- How to create new variable?



Variables

- To declare and initialize a new variable, you need to write data type, variable name and assign some value. For example:

```
int my_var;     ← Declaring
my_var = 0;     ← Initializing
```

You can also do the same thing in more simple form like this:

```
int my_var = 0;     ← Declare + Initialize
```

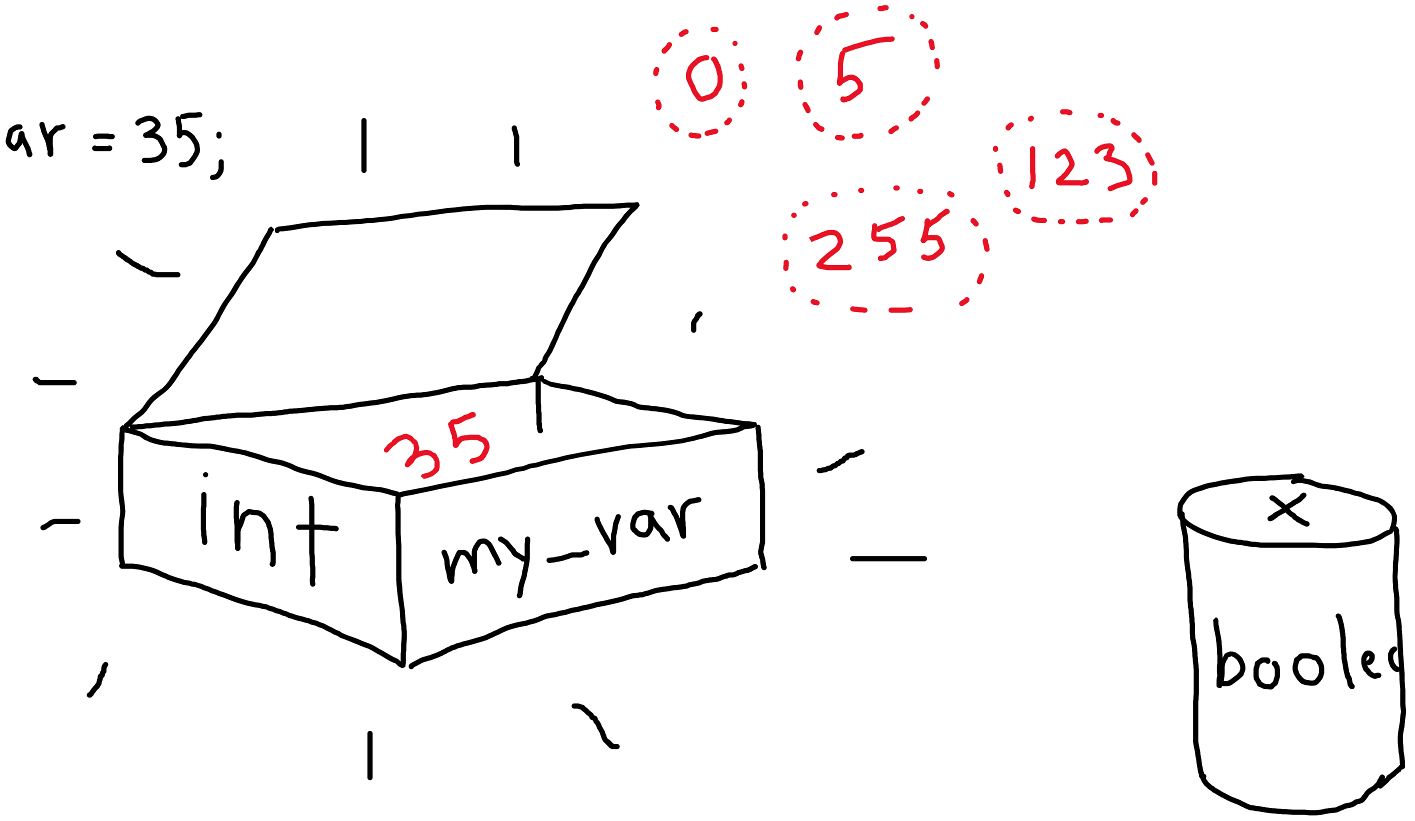
- To change data stored into variable, use equal sign to assign new value. For example:

```
my_var = 0;     ← Assigning
```

Variables

- New variable is created by declaring a variable

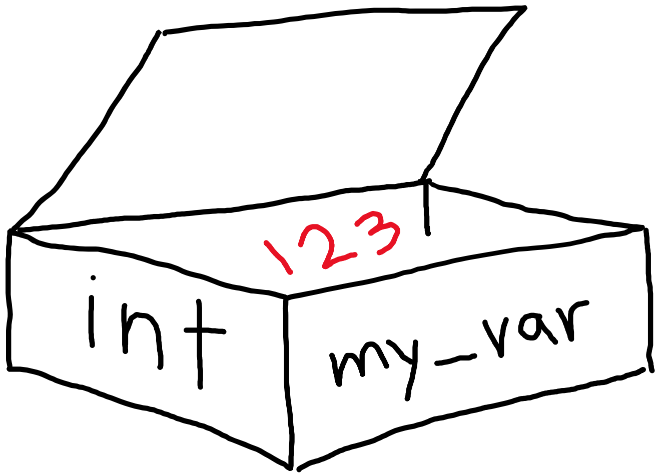
```
int my_var = 35;
```

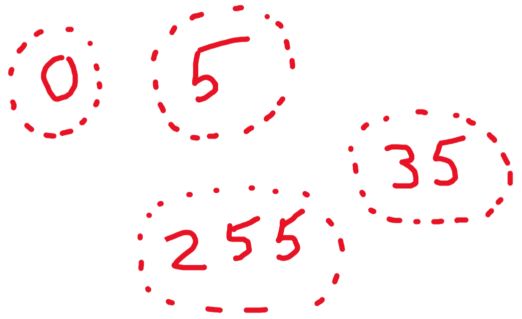


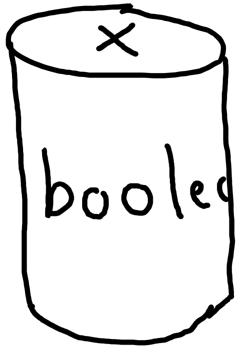
Variables

- The data inside a variable can be changed by assigning a new value with equal sign =

```
my_var = 123;
```







Variables (data type)

- Variables have data type for different type of data
- To declare new variable write data type keyword infront of variable name.
- Data types we already know:

  byte (any integer number between 0-255)

  int (any integer number between -32768 to 32767 / 0 -65536 = 2^{16})

  float (any floating point number, also large numbers)

  boolean (true or false value)

NOTE! Always small caps ! String (any text) ... except String

Functions

Functions

- Function is a building block of code

Functions

- Function has 4 parts (return data type, name, parameters, code) for example:

```
void exampleFunction(){
   // some code here…
}
```

Kāāk!

- Function name is usually written with small caps
- Function parameters are variables written inside parenthesis ( )
- Function starts and ends with curly brakets { }

Functions

- There are two types of functions:

  1. void functions
     - These functions don’t return any value
     - These functions only execute some code .

  2. other funtions
     - These functions always return some value
     - These functions execute some code and return some value

Functions

- Functions that we already know :

  1. void functions

     ```
     setup()
     loop()
     pinMode()
     digitalWrite()
     analogWrite()
     ```

  2. other funtions

     ```
     digitalRead()
     analogRead()
     ```

Functions (example of void functions)

- Arduino has two special functions that every program should include. Those are highlighted with green color.

```
void setup() {
}

void loop(){
}
```

Things that run only once!

Things that run continuosly!

Functions

- Arduino has many built-in functions that you can use. Those are highlighted with orange color.

```
void setup() {
        pinMode()
        digitalWrite()
        analogWrite()
}

void loop(){
}
```

Full list of built-in functions is here: https://www.arduino.cc/reference/en/#functions

Functions (parameters)

- Some of Arduino built-in functions need a specific parameters. Write parameters inside parenthesis

```
void setup() {
        pinMode(13,OUTPUT);
        digitalWrite(13,HIGH);
        analogWrite(9,255);
}

void loop(){
}
```

Full list of built-in functions is here: https://www.arduino.cc/reference/en/#functions

Logics

Logic (logical statements)

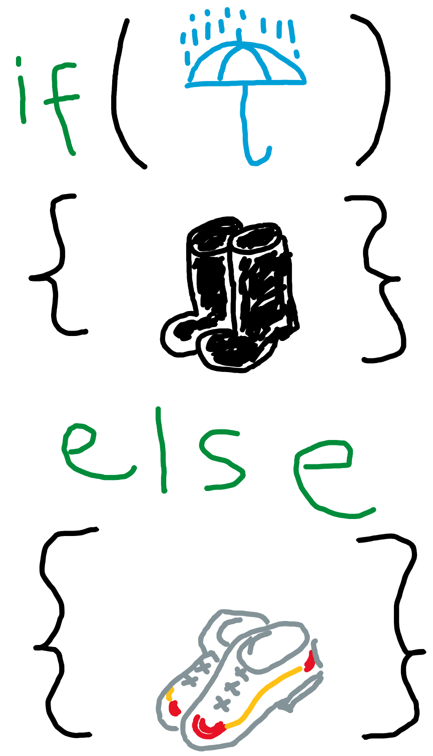
- Your code will always execute from top to bottom, line-by-line.
- To change this default structure, you can create conditional statements
- Some of most well-known conditions are:
  - if
  - if else
  - while
  - do … while
  - switch … case

  - You can find all conditions for Arduino here: https://www.arduino.cc/reference/en/#structure

Logic (logical statements)

- Conditions consist typically from 3 parts (statement, condition, block of code)

What shoes to wear?



Logic (logical statements)

- Conditions you already know:

```
if (🌧️ == true) {
☂️+🥾
} else if (🌥️ == true) {
🧣+👞
} else {
🕶️+👒+👟
}
```

Logic (logical statements)

- Conditions you already know:

```
if (temperature<=❄️){
🏒
} else {
🏊‍♀️
}
```

Logic (logical statements)

- Conditions you already know:

```
if(button_pressed == true) {
    // do something here
} else {
    // do something else here
}
```

Conclusions

Conclusions

- You know basics of variables, functions and conditions! That’s great since that’s most of the knowledge you need to understand programming!

Variable = container for data

Functions = code block that do stuff (and sometimes return data)

Conditions = Organize functionality of your code according a condition

II

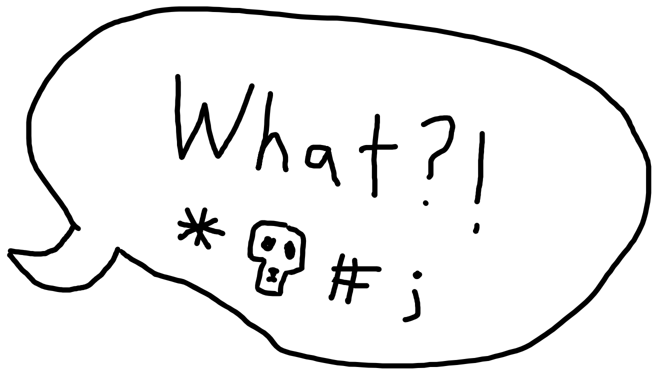
Variable scope

Extra (variable scope)

- There is one thing that you should know about variables that combines all of your knowledge. Variables are only ‘visible’ int their scope!
- Variables can be either global (visible everywhere) or local (visible only inside function or condition statement)
- Declaring variable defines where it is visible!

Extra (variable scope)

- All variables declared outside of functions (setup, loop) are global.
- Write global variables in the beginning of the code.
- All variables declared inside function or condition statement are local and only visible inside that function/condition



Extra (variable scope)

= introduced for the first time

- Scope of variable is defined when variable is declared!
  - For example:

```
int my_global_variable = 0;

void setup() {
int my_local_variable_1 = 0;
}
void loop() {
int my_local_variable_2 = 0;
}
```

global variable = visible anywhere in the code

local variables

Extra (variable scope)

- Scope of variable is define when variable is declared!
  - For example:

```
int my_global_variable = 0;

void setup() {
        int my_local_variable_1 = 0;
        my_local_variable_1 = 4;
        my_global_variable = 0;
}
void loop() {
        int my_local_variable_2 = 0;
        my_local_variable_2 = 5;
        my_local_variable_1 = 100;
        my_global_variable = 0;
}
```

Where is the error ?

Extra (variable scope in functions)

- Do you still remember our function example?

```
void exampleFunction(int input){
   // some code here…
}
```

- Function inputs are variables written inside parenthesis ( ). Input gets the value outside of function but variables are local to the function!
- Function starts and ends with curly brakets { }. Those symbols define the limits of the scope.

Extra (variable scope in functions)

```
void exampleFunction(int input){
   int example_variable = 0;
   example_variable = input;
}
```

Custom functions

Function calls

- You can make your own functions and call the function from the setup or the loop function

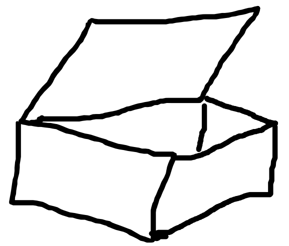
```
void loop(){
   myFunction();
}

void myFunction(){
   // some code
}
```

Functions call with parameters

- You can make your own functions with your own parameters as well:

```
void loop(){
   myFunction(100);
}

void myFunction(int input){
   // some code
}
```

Parameter is a variable!



Functions (example of sum function)

- You can make your own functions with your own parameters as well:

```
void loop(){
   sum(1432,42354);
}

void sum(int x, int y){
   int result = x + y;
}
```

Two inputs!

OK, nice but how to get the result back?

Functions (example of sum function)

- You can make your own functions with your own parameters as well:

```
void loop(){
   int example_variable = sum(1432,42354);
}

int sum(int x, int y){
   int result = x + y;
   return result;
}
```

return type

Return value from function