Advanced Activities - Information and Ideas

Congratulations! You successfully created and controlled the robotic chameleon using the program developed for the chameleon project. Here you'll learn how you can change the original program using Arduino IDE, upload it to the microcontroller, and test it on the robot!

To give a robot life, it needs programmable components. These components can read sensor values or control motors. They serve as the 'brain' of the robot. The robots in the Bionics Kit use a microcontroller with a programmable processor and various inputs and outputs.

A microcontroller is a programmable device. In most cases, it consists of a processor and associated components such as memory and digital or analog inputs and outputs. In order to program a microcontroller, a computer with the appropriate software is needed. The software allows you to develop the logic using a programming language and then load the program onto the microcontroller.

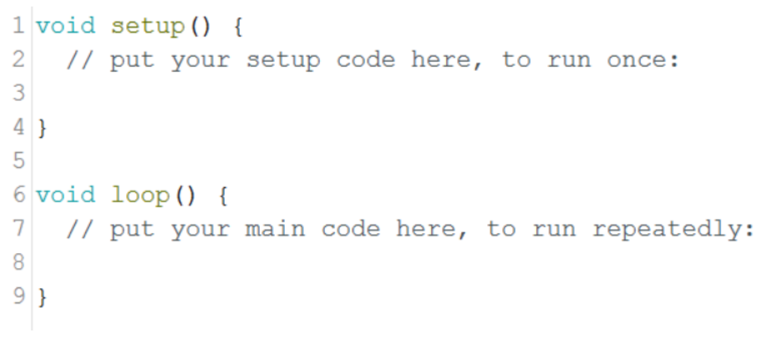
You will use the Arduino IDE (Integrated Development Environment). An IDE is a tool for software development and provides various libraries and tools in one interface. The IDE converts (compiles) the written software into machine code that carries out your real-world commands. The Arduino IDE is used for Arduino and Arduino-compatible microcontroller boards. It allows software to be written in C/C++ and loaded onto the microcontroller.

Basic Programming

Programs written using Arduino Software (IDE) are called sketches and are saved in the text editor. When you write a program in Arduino, you are using a combination of C/C++ languages.

An Arduino program has the same basic structure. It includes the "setup" and "loop" functions. These functions are required by the Arduino system and you must have one of each. The "setup" function contains everything that is to be executed once, at the beginning of the program. The actions to be performed are written between the braces "{ ... }". Each command you write is executed once before moving to the next command, in order. When the program reaches the end of the "setup"function, it moves to the "loop" function.

The "loop" function is where your main code is entered, to run in an endless loop. Each command you write between the braces is executed in order. When the program reaches the end of the "loop" function, it then moves to the top of the "loop" function and executes the code again.



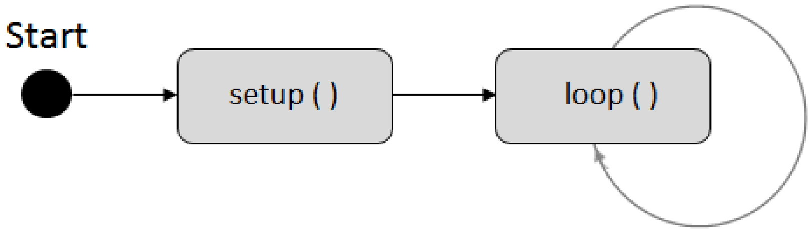
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

In Arduino, the "setup" function is called on once, and the "loop" is called on repeatedly.



An Arduino program consists of letters, words, and symbols, known as tokens. In this example, the program turns an on-board LED on for one second, then off for one second, repeatedly.

The "setup" code sets the LED pin as the output using the function "pinMode". The "pinMode" function tells the code to run a section of code written elsewhere in Arduino. The parentheses are tokens that enclose the inputs, known as arguments, to the "pinMode" function. The "pinMode" function uses these arguments to determine which pin it should control and if that pin should be an input or an output. The comma token is how C/C++ separates parameters. The semicolon is a special C token, known as a statement terminator. It is used to denote the end of a command.

The "loop" function contains the code necessary to turn the LED on, wait the one second delay (1000 ms), turn the LED off, and wait another one second delay (1000 ms). The program then returns to the top and executes again.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Data Types

A data type tells the compiler how to use a piece of data. In C/C++ there are three fundamental types of data.

Fundamental Data Type	Abbreviation	Example
Integer	int	42
Floating point	float	42.12
Character	char	h and 2

Void is a type, but it means 'nothing' or 'no type'. In Arduino, it's used to define the 'setup' and 'loop' functions, and returns no value.

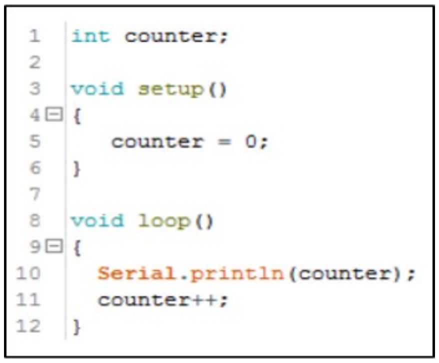
There are only three fundamental types of data in C/C++, however, keywords are used to change how the types work.

Variables

In order to write programs, you must understand the concept of variables. A variable is a portion of memory (container) used to store a value and reuse it later in the program. In C/C++, each variable needs a data type and a name, or identifier, that identifies it and distinguishes it from the others.

Identifiers can be a combination of upper or lowercase letters and numbers. The only special character you can use is the underscore. C/C++ uses many reserved keywords to identify operations and data descriptions, so you must not create identifiers that match those keywords. They have special meaning to the compiler, therefore, if you use a reserved keyword in a program you will not be able to compile it. You can review Arduino's reserved keywords on the Arduino website.

If you want to implement a variable that is visible in all functions, it must be defined outside a function. This makes it globally visible. In the example program below, Line 1 defines a global variable "counter" of the data type "int". This is available throughout the program and can be changed by any function. In the example, the variable is initialized with the value 0 in the "setup" function. If you display the value of each iteration in the serial monitor of the Arduino IDE, you can see that it always increases by 1.



```
 1  int counter;
 2
 3  void setup()
 4  {
 5      counter = 0;
 6  }
 7
 8  void loop()
 9  {
10    Serial.println(counter);
11    counter++;
12  }
```

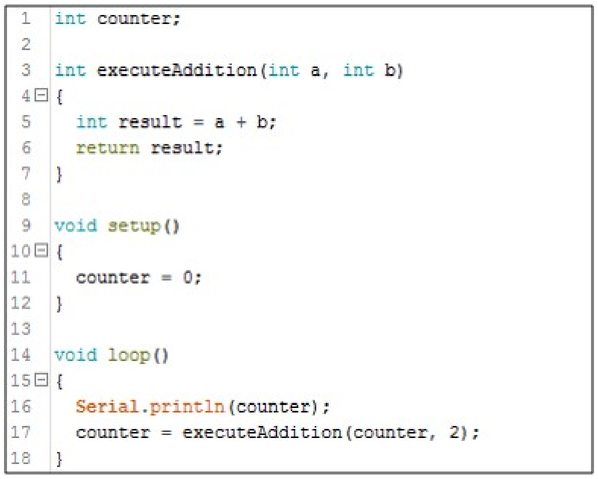
If a variable is only to be visible within the function, it must be embedded in the instruction block ("{... }") of the function.

Functions

You know that the two standard functions "setup" and "loop" must appear in an Arduino program. Functions can also be structured differently. They can have a return value and/or a transfer parameter.

In the example below, the program stores the addition of two values in a separate function that returns the result. This is used to avoid writing a certain sequence more than once. Function "executeAddition" accepts two transfer parameters of the data type int, adds them together, and returns the result of the data type int.

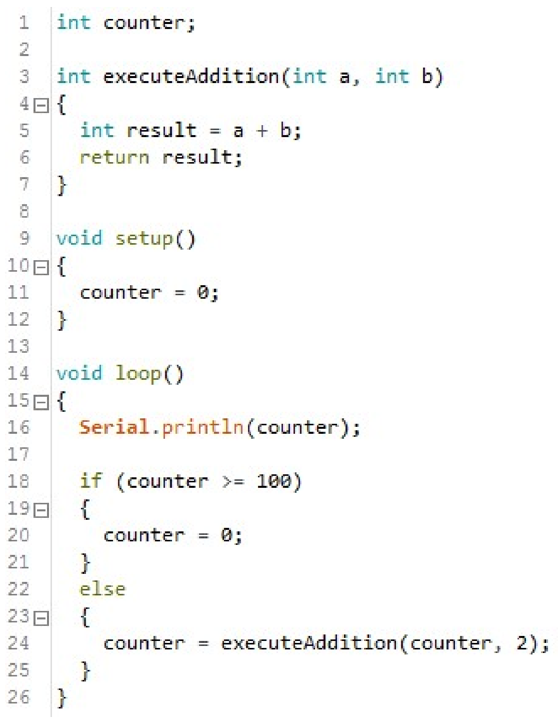
In line 17, the function is called with the variable "counter" and the value 2. The function implemented in line 3 adds these two values together and returns the result, which is stored in the variable "counter".



```
 1  int counter;
 2
 3  int executeAddition(int a, int b)
 4  {
 5    int result = a + b;
 6    return result;
 7  }
 8
 9  void setup()
10  {
11    counter = 0;
12  }
13
14  void loop()
15  {
16    Serial.println(counter);
17    counter = executeAddition(counter, 2);
18  }
```

If queries

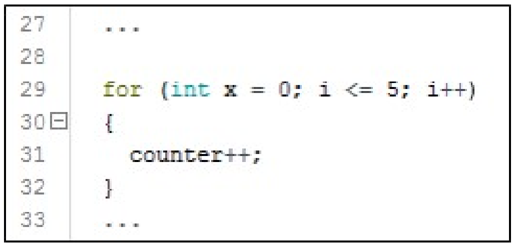
When writing your own program, variables must always be checked for certain values. This is implemented with "If queries". Such a query could check whether the value of the variable "counter" is greater than or equal to 100 and then execute an action. In the example program below, line 18 checks the value of the variable "counter", if it is greater than or equal to 100, everything within the statement block of the If query is executed. In this case, the variable "counter" is set to 0. If the condition is not fulfilled, an alternative statement block (else) is executed. In this case, the function "calculateAddition()" is called.



```
 1  int counter;
 2
 3  int executeAddition(int a, int b)
 4  {
 5    int result = a + b;
 6    return result;
 7  }
 8
 9  void setup()
10  {
11    counter = 0;
12  }
13
14  void loop()
15  {
16    Serial.println(counter);
17
18    if (counter >= 100)
19    {
20      counter = 0;
21    }
22    else
23    {
24      counter = executeAddition(counter, 2);
25    }
26  }
```

Loops

A loop is used to execute an action more than once. You can use the "for loop" to define repetitions up to a certain termination condition. In the example below, the initialization (int x = 0), the termination condition (i <= 5) and the continuation (i++) are passed in parentheses. During the first run, the variable **x** has the value 0 and the counter is increased by 1. If the end of the statement block of the loop is reached, the continuation statement is executed and **i** is increased by 1. This action is repeated until **x** reaches the value 6.



```
...

for (int x = 0; i <= 5; i++)
{
  counter++;
}
...
```

Comments

Comments are built into code for better readability. In C/C++ there are two ways to comment: single-line and multi-line. Single-line comments are introduced by two slashes "//". Multi-line comments start with a slash and a star "/*" and end with a star and slash "*/". Everything within these characters is not executed by the program.

Installing Arduino IDE

To install the Arduinio IDE, you need the following:

- Computers with Internet access
- Micro-USB cable
- ESP32 microcontroller (included in the Bionics Kit)

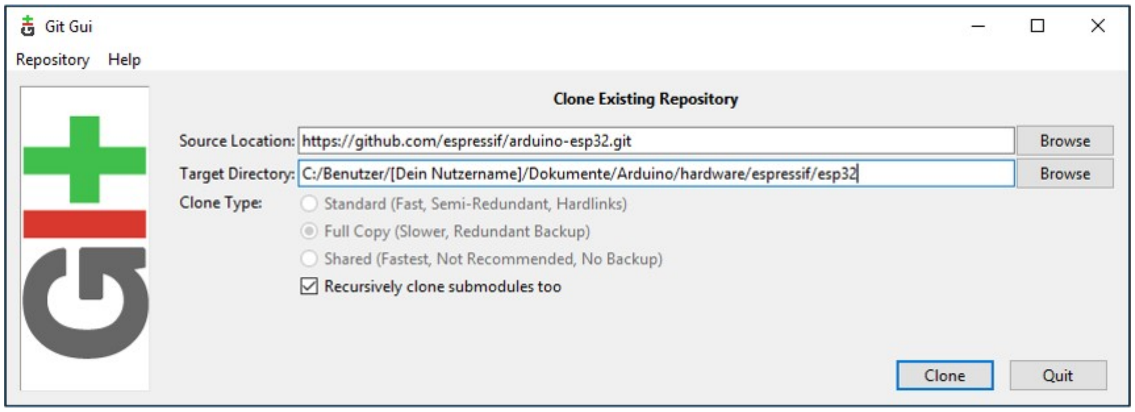
1. Download and install the Arduino IDE from the official Arduino website (https://www.arduino.cc/en/Main/Software). You are now able to program various microcontrollers.

To program the microcontroller used with the bionic robots (ESP32), information relevant to the board must be added to the IDE. To program the ESP32 with the Arduino IDE, the following steps must be performed. (For more information, go to https://github.com/espressif/arduino-esp32).

Installing Git

Git is a software program for version control of files. In this case, it will be used to get an up-to-date version of the software for using the ESP32 with the Arduino environment.

1. Go to the Git website (https://git-scm.com/download) to download the program for your operating system.
2. Perform the installation with the default settings. After the installation there are two new entries in the Windows context menu (right click in Windows Explorer).The "Git GUI here" entry starts the interface for downloading Git folders. With the entry "Git Bash here" a Git command line opens, in which further functions, which are not necessary for our purposes, are available.
3. Start "Git Gui" and click on "Clone Existing Repository". Enter the following input mask (Note: this is for a Windows download) and then click "Clone".
   -**Source Location:** "https://github.com/espressif/arduino-esp32.git"
   -**Target Directory:** "C:/Benutzer/[Dein Nutzername]/Dokumente/Arduino/hardware/espressif/esp32"
4. After all files have been successfully downloaded, a new window appears, which can be closed.
5. In Windows Explorer, change the path of "Target Directory". Open the "tools" folder and start "git.exe". This downloads additional software necessary for development with the ESP32. Now all necessary preparations are finished and the Arduino IDE can be started.



Clone Existing Repository

Source Location: https://github.com/espressif/arduino-esp32.git

Target Directory: C:/Benutzer/[Dein Nutzername]/Dokumente/Arduino/hardware/espressif/esp32

Clone Type:
- Standard (Fast, Semi-Redundant, Hardlinks)
- [✓ Full Copy (Slower, Redundant Backup)]
- Shared (Fastest, Not Recommended, No Backup)

[✓] Recursively clone submodules too

Testing the Environment

For a complete function test of the ESP32, load the example program onto the microcontroller.

- First, select the appropriate board. Tools -> Board: -> ESP32 Dev Module
- Then open the example program. File -> Examples -> 01.Basics -> BareMinimum
- Load the program to the microcontroller. Sketch -> Upload. If no error messages appear, the IDE has been completely set-up.

The example program "BareMinimum", has no executable code, it is only used for testing the upload to the microcontroller. At startup, the "setup()" function is called (line 1). This contains all initializations for the program. Line 6 contains the function "loop()", which is called and executed infinitely often.