

PROGRAMMING YOUR ARDUINO

Here is a brief introductory guide to writing code for your Arduino. All information is taken from www.arduino.cc if more detailed information is required you can access tutorials at: <https://www.arduino.cc/en/Tutorial/HomePage>

and reference material at:

<https://www.arduino.cc/en/Reference/HomePage>

INTRODUCTION

Arduino programs can be divided in three main parts: structure, values (variables and constants), and functions.

STRUCTURE

The two function every program will need are

```
setup()
```

and

```
loop()
```

The `setup()` function is called when the program starts and is used to initialize variables, pin modes, libraries, etc. The `setup()` function will only run once, after each powerup or reset of the Arduino board.

The `loop()` function will run consecutively over and over until a reset or board power-down. This function is used to actively control the Arduino board.

Control Structures

The main control structures we will use are `if...else`, `while`, `for` and `break`.

`if` is used in conjunction with a comparison operator (`==`, `!=`, `<`, `>`), tests whether a certain condition has been reached, such as an input being above a certain number. The format is as follows:

```
if(variable > value){  
    //do something  
}
```

The `if` statement can be used in conjunction with the `else` statement. The `else` statement allows multiple tests to be grouped together. The format is as follows:

```
if(variable > value){
    //do thing A
}else if(variable == value){
    //do thing B
}else {
    //do thing C
}
```

`while` loops will loop continuously, and infinitely, until the expression inside the parenthesis, `()` becomes false. Something must change the tested variable, or the `while` loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor. The syntax is as follows:

```
while(condition == TRUE){
    //statement(s)
}
```

The `for` statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The `for` statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

```
for(initialize; condition; increment){
    //statement(s)
}

for(int i=0; i<100; i++){
    println(i);
}
```

`break` is used to exit from a `do`, `for`, or `while` loop, bypassing the normal loop condition.

```
while(TRUE){
    if(condition){
        break;
    }
}
```

Operators

Arithmetic Operators

These are used for operations on variables:

- `=` assignment

- + addition
- - subtraction
- * multiplication
- / division
- % modulo

Comparison Operators

These are used inside parenthesis () (`if`, `for`, `while`, etc.) to compare variables/constants:

- == equal to
- != not equal to
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to

Boolean Operators

These are used in the same way as logic statements:

- && and
- || or
- ! not

VALUES

Variables

Variables can be set using the assignment operator `=` and must be initialized in C.

```
//assign integer variable
int variable = 56;

//assign character variable
char variable = 'A';

//assign decimal variable
float variable = 3.141593;
```

Here is a short list of the data types you can use in the Arduino IDE and their sizes:

- `boolean` true or false
- `char` -128 to 127 (character)
- `unsigned char` 0 to 255 (character)
- `int` -32,768 to 32,767 (integer)
- `unsigned int` 0 to 65,535 (integer)
- `float` -3.4028235E+38 to 3.4028235E+38 (floating point/decimal)
- `unsigned long` 0 to 4,294,967,295 (integer)

The variable `void` is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

```
void setup() {
  //
}
```

Constants

Constants are predefined expressions in the Arduino language. They are used to make the programs easier to read.

- `false` defined as zero (0)
- `true` defined as anything but zero (!0)
- `HIGH` reading or writing to digital pin high
- `LOW` reading or writing to digital pin low
- `INPUT` configures pin to input
- `OUTPUT` configures pin to output
- `INPUT_PULLUP` configures pin to input with internal pullup resistor
- `LED_BUILTIN` number of the pin which on-board LED is connected

Constants can also be set using `#define` at the top of the file:

```
#define PI 3.141593
#define K 89

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

}

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

}
```

FUNCTIONS

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program. Below is the syntax that should be followed for writing a function:

```
//datatype_of_data_returned function_name(parameters_passed_to_function){  
int my_function(int x, int y){  
  
    //function body  
    int result;  
    result = x + y;  
  
    //return statement  
    return result;  
}
```

When calling this function we write:

```
int k;  
int i = 1, j = 2;  
k = my_function(i,j); //k now equals 3
```

The Arduino IDE contains a number of pre-defined, inbuilt function, some useful ones are detailed below and again more are available on the reference page of their website.

Digital I/O

`pinMode()` configures the specified pin to behave either as an input or an output. This function must be used within the `setup()` function to initialize the pins. The syntax is `pinMode(pin, mode)` where `mode` is one of: `INPUT`, `OUTPUT` or `INPUT_PULLUP`.

`digitalWrite()` is used to write a `HIGH` or `LOW` value to a digital pin. The syntax is `digitalWrite(pin, value)`

`digitalRead()` is used to read a `HIGH` or `LOW` value from a digital pin. The syntax is `digitalRead(pin)`

```

#define LED 13
#define SWITCH 15

void setup() {
    // initialize digital pins
    pinMode(LED, OUTPUT);
    pinMode(SWITCH, INPUT);
}

void loop() {
    int on = false;
    //read input from pin SWITCH
    on = digitalRead(SWITCH);

    if(on == true){
        //if SWITCH is on, turn on LED
        digitalWrite(LED, HIGH);
    }
}

```

Analog I/O

`analogRead()` Reads values from the specified analog pin, with a 10-bit ADC. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5V/1024 units or 0.0049V per unit. The input range and resolution can be changed using `analogReference()`. The maximum reading rate is roughly 10,000 times per second. The syntax is `analogRead(pin)` and returns `int`(0 to 1023).

`analogWrite()` writes a pulse width modulated (PWM) wave to a pin. The frequency of the PWM signal on most pins is 490Hz. The syntax is `AnalogWrite(pin, value)` where value is the duty cycle between 0 (always off) and 255 (always on).

```

#define ANALOG_IN 6
#define LED 8

void setup() {
}

void loop() {
    int value = 0;
    int duty = 0;

    //read from analog input
    value = analogRead(ANALOG_IN);

    //adjust duty cycle
    duty = value/4; //1024/256 = 4

    //output PWM signal to LED
    analogWrite(LED, duty);
}

```

Time

`millis()` returns the number of milliseconds since the Arduino board began running the current program. This number will return to zero after approximately 50 days. This function returns an `unsigned long` data type.

`micros()` returns the number of microseconds since the program began and will return to 0 after approximately 70 seconds. This function returns an `unsigned long` data type.

`delay()` pauses the program for the amount of time specified in milliseconds and takes an `unsigned long` as input data type.

`delayMicroseconds()` pauses the program for the amount of time specified in microseconds and takes an `unsigned int` as input data type.

```
unsigned long Time;

void setup() {
  //initialize serial
  Serial.begin(9600);
}

void loop() {
  Serial.println("Time(ms): ");
  //calculate ms since program started
  Time = millis();
  //print to serial
  Serial.println(Time);
  //delay by 1 second
  delay(1000);

  Serial.println("Time(us): ");
  //calculate us since program started
  Time = micros();
  //print to serial
  Serial.println(Time);
  //delay by 1 second
  delayMicroseconds(1000000);
}
```

Serial

The `Serial` function is used to communicate between the Arduino and computer.

This should be initialized in the `setup()` function using `Serial.begin(data_rate)` where the `data_rate` is in bits per second and can be one of 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.

The main functions we will need are `Serial.print()`, `Serial.println()`, `Serial.available()` and `Serial.read()`.

`Serial.print(value)` or `Serial.print(value, format)` is used to print value to the serial monitor. Value can be of any data type and format specifies the number base (for integral data types) or number of decimal places (for floating point types). `Serial.println()` can also be used and will automatically upend the statement with a newline '\n' character.

Some examples:

- | | | |
|---|-------|----------------|
| • <code>Serial.print(78)</code> | gives | "78" |
| • <code>Serial.print(1.23456)</code> | gives | "1.23" |
| • <code>Serial.print('N')</code> | gives | "N" |
| • <code>Serial.print("Hello world.")</code> | gives | "Hello world." |
| • <code>Serial.print(78, BIN)</code> | gives | "1001110" |
| • <code>Serial.print(78, OCT)</code> | gives | "116" |
| • <code>Serial.print(78, DEC)</code> | gives | "78" |
| • <code>Serial.print(78, HEX)</code> | gives | "4E" |
| • <code>Serial.println(1.23456, 0)</code> | gives | "1" |
| • <code>Serial.println(1.23456, 2)</code> | gives | "1.23" |
| • <code>Serial.println(1.23456, 4)</code> | gives | "1.2346" |

`Serial.available()` gets the number of bytes (characters) available for reading from the serial port.

`Serial.read()` is used to read data from the serial input and returns the first byte of the incoming signal

These can all be used in conjunction as follows:

```
int incomingByte = 0;  // for incoming serial data

void setup() {
  Serial.begin(9600);  // opens serial port, sets data rate to 9600 bps
}

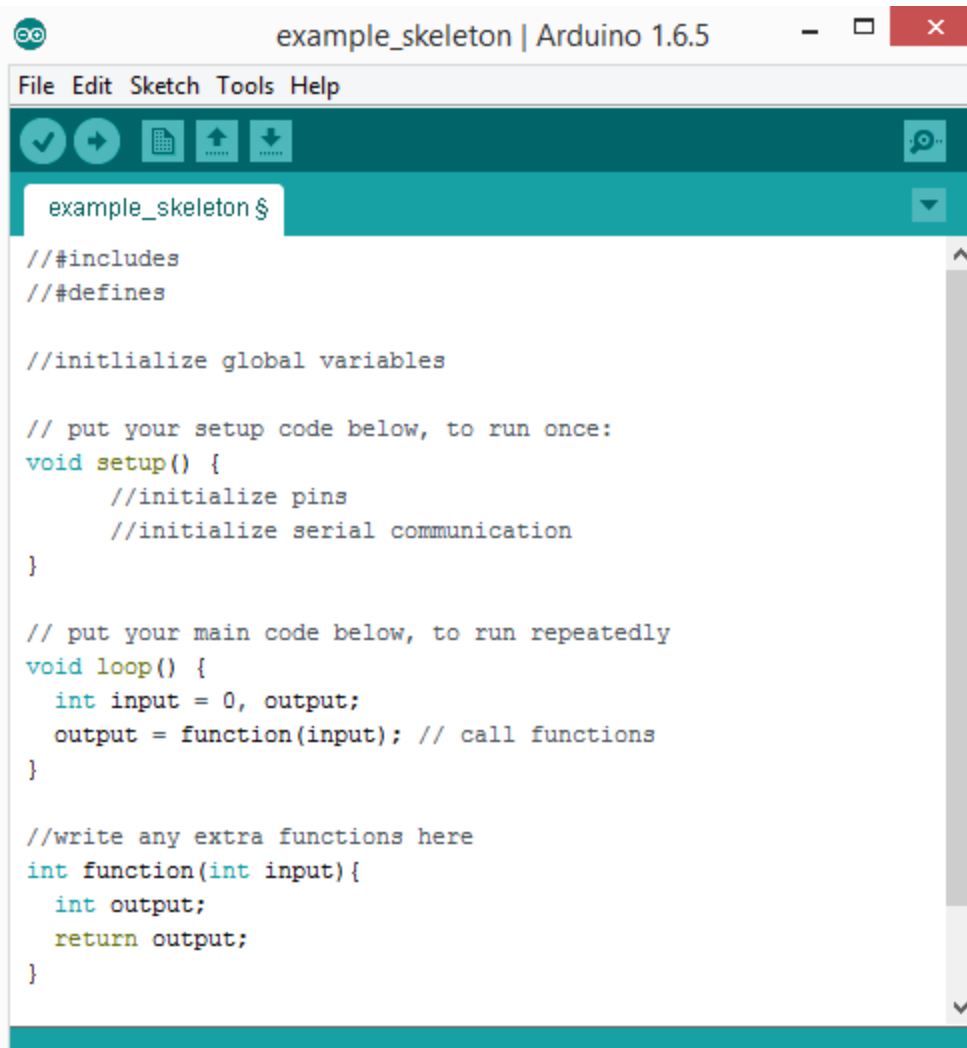
void loop() {

  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```

EXAMPLE PROGRAM

The following is an example of how your program should be laid out:



The screenshot shows the Arduino IDE interface with a window titled "example_skeleton | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for checking, running, saving, and uploading. The main text area contains a skeleton program template with the following code:

```
example_skeleton $
//#includes
//#defines

//initlialize global variables

// put your setup code below, to run once:
void setup() {
    //initialize pins
    //initialize serial communication
}

// put your main code below, to run repeatedly
void loop() {
    int input = 0, output;
    output = function(input); // call functions
}

//write any extra functions here
int function(int input){
    int output;
    return output;
}
```