Programming Assignment

Lesson 9

CISC 071

By

Jeremy Reuwer

Date: 04/21/1998

Purpose

* To learn about ***functions***

Rubric

* Correctness: 2 Points. Program should work as specified
* Input/Output: 2 Points. Show the inputs and outputs to the program. If Applicable multiple examples needed
* Coding style/Comments: 1 Points.

Project Assignment

* Use the setup function to set the serial monitor and print “Serial monitor setup done”
* Define a function AreaOfSquare that takes an integer as input (length of the side of square) and returns an integer which is the area of a square.
* Call the function AreaOfSquare in the **setup** function three times with value of 8, 10 and 24. Print the “Side of the square =” and then the “Area of the square =” in the serial monitor.
* Cut and paste your program and serial monitor in this document.
* Leave the **loop** function empty.

Learning Notes:

Functions were briefly encountered in [part 1 of this programming course](https://startingelectronics.org/software/arduino/learn-to-program-course/01-program-structure-flow/) where

**The Structure of a Function**

Some basic facts about functions are

1) each function must have a unique name,

2) the function name is followed by parentheses ()

3) functions have a return type, e.g. void,

4) the body of a function is enclosed in opening and closing braces {}.

We will start by examining the structure of a function and then see how to write functions.

Before a function can be used in a sketch, it must be created. The following code is an example of a function that was created to print a dashed line in the Arduino IDE.

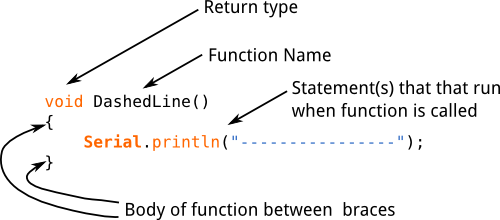
*void DashedLine()*

*{*

*Serial.println("----------------");*

*}*

The code above that creates the function is called the function definition. The image below shows the components of a function.



**Function Name**

When we create a function, it must be given a name. The naming convention for functions is the same as for variables:

* The function name can be made up of alphanumeric characters (A to Z; a to z; 0 to 9) and the underscore (\_).
* The function name may not start with a number i.e. the numbers 0 to 9.
* A function name must not be used that is the same as a language keyword or existing function.

The function name ends with parentheses (). Nothing is passed to the example function above, so the parentheses are empty. Passing values or parameters to functions will be explained later in this tutorial.

**Return Type**

A function must have a return type. The example function does not return anything, so has a return type of ***void***. Returning a value from a function will be explained later.

**Function Body**

The function body is made up of statements placed between curly braces {}. The statements make up the functionality of the function (what the function will do when it is called).

When a function is used, it is said to be "called". We will look at how to call a function next.

**Calling a Function**

To use the function that was created above, it must be called in a sketch as shown in the sketch below.

*void setup() {*

*Serial.begin(9600);*

*DashedLine();*

*Serial.println("| Program Menu |");*

*DashedLine();*

*}*

*void loop() {*

*}*

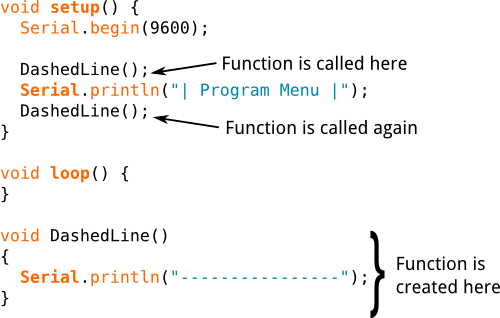
*void DashedLine()*

*{*

*Serial.println("----------------");*

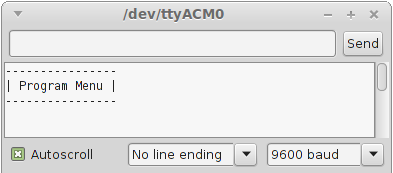
*}*

In the sketch above, the DashedLine() function is created at the bottom of the file and then called twice at the top of the file as shown in the image below.



To call a function, use the function name followed by opening and closing parentheses. Finally terminate the statement that calls the function with a semicolon.

Load the sketch to an Arduino and then open the terminal window. The sketch prints some text in a box as shown below.



The first time that the function is called, it prints the dashed line shown in the top of the image. Text is then written to the serial monitor window by the statement below the function call. The function is then called again to print the same dashed line that completes the box.

**Why Use Functions**

The function used in the example above is very simple, so all the benefits of using functions will not be seen immediately.

One advantage of using functions is that they avoid having to write the same code over and over again in a sketch which saves time and memory. Every time that a function is called, we are just reusing code that has been written once.

If a function needs to be modified, it only has to be done once and the modifications will take effect every place in a sketch that the function is called. If a function was not used, each place that the statements are found in a sketch to do a particular task would need to be located and modified.

Functions can be used to break a sketch up into pieces which make it more modular and easier to understand. Functions can be reused in other sketches.

**Passing a Value to a Function**

In the sketch above, the length of the line that the function prints out is fixed in the function. If we change the text that is in the box, it may not fit in the box properly. The function needs to be modified so that we can tell it what size line it must draw.

The above function can be modified to pass a value to it that will tell it how many characters long to make the line that it draws.

The modified sketch is shown below.

*void setup() {*

*Serial.begin(9600);*

*// draw the menu box*

*DashedLine(24);*

*Serial.println("| Program Options Menu |");*

*DashedLine(24);*

*}*

*void loop() {*

*}*

*void DashedLine(int len)*

*{*

*int i;*

*// draw the line*

*for (i = 0; i < len; i++) {*

*Serial.print("-");*

*}*

*// move the cursor to the next line*

*Serial.println("");*

*}*

Can't see the video? [View on YouTube →](https://youtu.be/HBmrEkqQ4Ds)

The DashedLine() function in the above sketch is modified so that an integer value can be passed to it. The line needs to be 24 characters long to fit the new menu text into it, so we pass it a value of 24.

DashedLine(24);  // passing a value of 24 to the function

Of course the function has to be modified to handle the value that is being passed to it:

*void DashedLine(int len)*

*{*

*int i;*

*// draw the line*

*for (i = 0; i < len; i++) {*

*Serial.print("-");*

*}*

*// move the cursor to the next line*

*Serial.println("");*

*}*

The function needs to be able to accept an integer value that is passed to it. The variable type and the name of the variable are inserted between the opening an closing parentheses after the function name.

*void DashedLine(int len)*

We can now use the len integer variable in the body of the DashedLine() function. The variable will contain whatever value was passed to it when the function was called.

The body of the sketch uses the len variable in a for loop to print out the correct number of dashes that make up the dashed line of the menu box.

The cursor is moved to the next line in the serial monitor by calling Serial.println(""); with an empty string.

After the function has run the last statement in its body, it is said to "return". When a function returns, program execution continues below the statement that called the function – i.e. the statement below the function call is run next.

**Returning a Value from a Function**

Getting a value back from a function is called "returning" the value from the function. The return keyword is used at the end of the function to get the value back. We must also say what type of value the function is returning, e.g. int, float, etc.

The example sketch below uses a function to do a mathematical calculation and then return the result of the calculation which can then be used in the main Arduino sketch.

**Function that Returns a Value**

*void setup() {*

*float area;*

*Serial.begin(9600);*

*// calculate the area of a circle with radius of 9.2*

*area = CircleArea(9.2);*

*Serial.print("Area of circle is: ");*

*// print area to 4 decimal places*

*Serial.println(area, 4);*

*}*

*void loop() {*

*}*

*// calculate the area of a circle*

*float CircleArea(float radius)*

*{*

*float result;*

*result = 3.141592654 \* radius \* radius;*

*return result;*

*}*

**What the Sketch Does**

The sketch calculates the the area of a circle from a radius value of the circle that is hard-coded into the sketch – in the example sketch the value is set to 9.2, but you can set it to any value that you want. The result of the calculation is then sent out of the serial port so that it can be seen in the Arduino IDE Serial Monitor window.

The formula for calculating the area of a circle is:

*A = π × r²*

*OR*

*A = π × r × r*

*Where:*

*A = area of the circle*

*π = PI = 3.141592654*

*r = radius of the circle*

In other words, if we know the radius of the circle (radius is the distance from the centre of the circle to the edge) we can calculate the area of the circle.

The unit that the radius is in can be any unit that is used to measure distance and the area will be squares of the unit used, e.g. if the radius is in centimetres, the area will be in square centimetres, if the radius is in feet, the result will be in square feet.

Can't see the video? [View on YouTube →](https://youtu.be/2NY0XVUG0Ac)

**Amazon.com**

**How the Sketch Works**

The CircleArea() function must return a value, so is preceded by the type of value that it must return – in this case float. A float value called radius is also passed to the function as explained in the previous part of this course.

*float CircleArea(float radius)*

Inside the function body, the radius calculation is done and the result of the calculation is put into the variable result which is a variable created in the function.

The function then returns the result using the return keyword at the bottom of the function.

*return result;*

The formula is translated into code for the Arduino as follows:

*A = π × r × r*

*Becomes:*

*result = 3.141592654 \* radius \* radius;*

In the part of the sketch that calls the CircleArea() function, the function basically becomes the value that it returns and can be assigned to a variable.

The variable area is assigned the value that the CircleArea() function returns:

*area = CircleArea(9.2);*

After this, the result of the calculation, which is the area of the circle, is sent out the serial port to be displayed in the Arduino IDE Serial Monitor window.

**A Shorter Version of the Sketch**

The sketch above can be written in a shorter way without using some of the intermediate variables as shown below.

*void setup() {*

*Serial.begin(9600);*

*Serial.print("Area of circle is: ");*

*// print area to 4 decimal places*

*Serial.println(CircleArea(9.2), 4);*

*}*

*void loop() {*

*}*

*// calculate the area of a circle*

*float CircleArea(float radius)*

*{*

*return (3.141592654 \* radius \* radius);*

*}*

In this sketch, the CircleArea() function returns the result of the calculation on one line without first assigning it to a variable.

*return (3.141592654 \* radius \* radius);*

This method of doing the calculation and returning the value is fine, although it may not be as easy to read the code as the first example.

When the CircleArea() function is called in the sketch, it is passed to Serial.println() as if it were a variable. This is possible because when a function returns a variable, it takes on the value of the variable. The sketch therefore works the same way as the first sketch, although again, it is more difficult to read the code.

*Serial.println(CircleArea(9.2), 4);*

**For further details refer to the Arduino programming reference guide**

<https://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf>

Program

void setup() {

Serial.begin(9600);

Serial.println("Serial monitor setup done");

Serial.print("the side of the square = ");

Serial.println(8);

Serial.print("the side of the square = ");

Serial.println(AreaOfSquare(8));

Serial.print("the side of the square = ");

Serial.println(10);

Serial.print("the side of the square = ");

Serial.println(AreaOfSquare(10));

Serial.print("the side of the square = ");

Serial.println(24);

Serial.print("the side of the square = ");

Serial.println(AreaOfSquare(24));

}

void loop() {

}

int AreaOfSquare(int numIn)

{

int numOut;

numOut= numIn \* numIn;

return numOut;

}

Inputs/Outputs

Serial monitor setup done

the side of the square = 8

the side of the square = 64

the side of the square = 10

the side of the square = 100

the side of the square = 24

the side of the square = 576