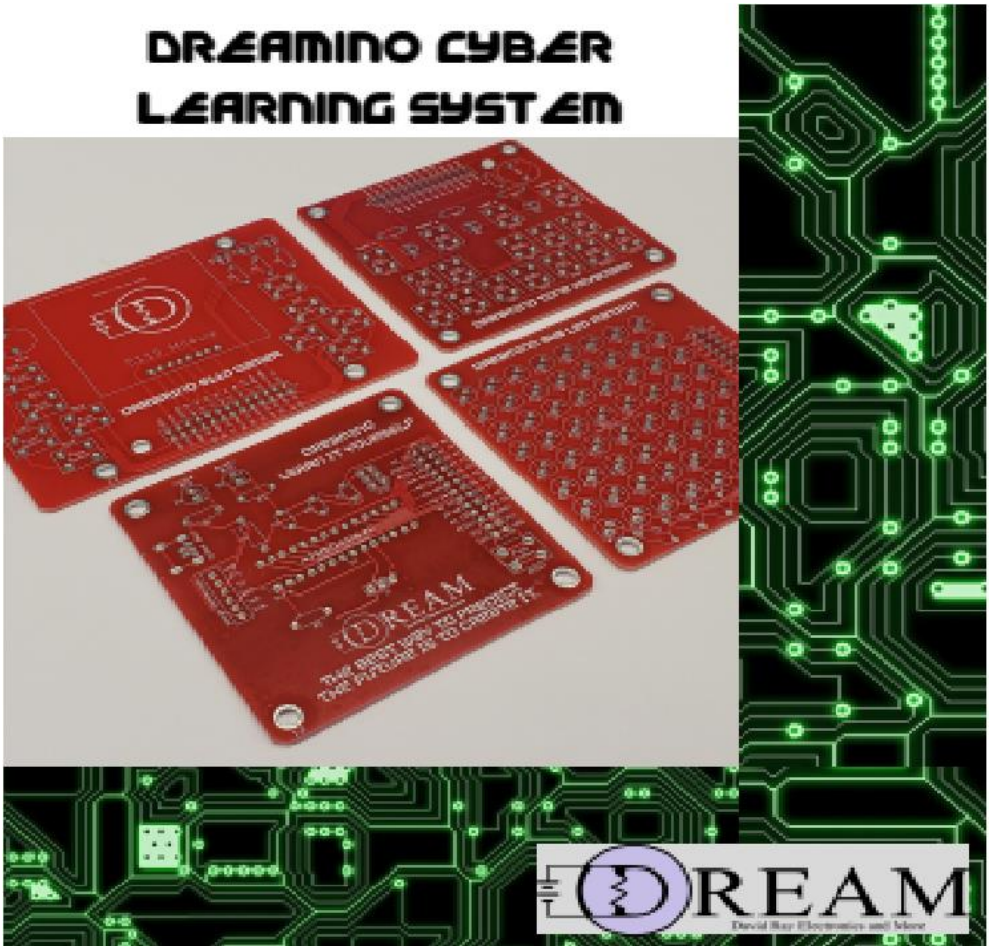

DREAMINO CYBER LEARNING SYSTEM

David Ray Electronics and More LLC

Booklet v0.02.01 – 20190922

David Ray and Angel Andino

Book Two

Section 1: Introduction to Programming

Now that we have the starter kit built, we can write code for it and make it do things! We are going to be using the Arduino IDE software to program the starter kit.

1.01: Programming Concepts:

There are several basic concepts that we need to go over first. We won't be going over all of them, but you will be getting a great start. The Arduino uses a type of the C++ programming language. The programs you write for Arduino is called a 'sketch'.

1.02: Comments - Comments are just that, comments. Comments are ignored by the compiler and do not affect your code at all when used correctly. They are used to make notes and reminders in the code for reference in the future. All good code has a lot of comments in it. There are two types of comments, single line and multiline.

- **Single Line Comments -** Single line comments always start with '//'. For example,
  '//The LED is connected to digital pin 13.'
  This line tells the reader that there is a LED and it is connected to pin 13 and that pin is a digital pin.
- **Multi Line Comments -** Multi line comments start with '/*' and end with '*/'. They are used when you have a lot of information you want to block off. Example:

```
/*
  Blink
  Turns an LED on, then off, repeatedly.
*/
```

1.03: Variables – Variables are used to store values. Here is an example: 'int a = 13;'. Here you will find three parts of this example.

- **Variable Type** - 'int' is used to declare the variable type. 'Int' is used to tell your sketch that the variable is an integer. There are a lot of different variable types, but we will primarily use this type in the beginning. An integer can store any number from -32,768 to 32,767. You can't store any number outside this range as an integer though.

- **Variable Name** – 'a' is the name of the variable. A variable name can be any letter from A to Z, a to z, any number from 0 to 9 and the underscore.

- **Variable Value** – In the example, the value we are setting 'a' to is '13'.

1.04: Functions – Functions are a set of instructions that can be anywhere in a sketch. They are used to perform very specific and repetitive tasks. A function has four major components. Here is an example of a small function.

```
int multi(int a, int b){
  return a * b;
}
```

- **Return Type** – In this example, 'int' is called the return type. You might find other things in this spot, like void or float (…and we'll go over what those mean later…). If you have a function that is used to send a value from that function to another, then you need to tell your 'sketch' what type of value that will be. In this example, the value being returned in an integer, so we use 'int'.

- **Function Name** – The function name in this example is ‘multi’. A function name can be made up of characters from A to Z (lowercase or upper case), numbers from 0-9 and we can use underscores (_). You cannot use the same function name more than once in your sketch. There are also some internal names (…like tone…) that you cannot use.

- **Arguments** – Inside the parenthesis you can place arguments. These are used to give the functions additional information. Even if your function doesn’t need an argument, you must put parenthesis, just leave them empty. In the example there are two arguments, ‘ int a’ and ‘int b’. These are two variables that the function uses to do math with. If you don’t tell the function what numbers to use it won’t work correctly.

- **Statement** – The statement is the actions that the function does when it is called. In the example, the statement is ‘{ return a * b; }’. A statement will always begin and end with curly brackets. This statement will take the number for ‘a’ and the number for ‘b’ and multiply them together. Then, it will ‘return’ the product of the two to the sketch. Almost everything you do, outside of the basic setups, will take place in a function.

- **Essential Functions** - Every sketch will have two functions to start with. These are ‘void setup()’ and ‘void loop()’.

    - The **‘void setup()’** function is an essential part of an Arduino sketch, even if you don’t actually use it. It is the first function read by your program and setups your starter kit to run your sketch. Remember, the setup function only runs once and at the beginning.

- The **'void loop()**' function is also very essential. It is read after the setup function and will usually contain a very large portion of the operations for the sketch. The function is called loop and it does exactly that – loops. The loop function will run over and over until something tells it to stop. Remember, that the loop function runs over and over again.

1.05: Statements – Statements come in two flavors. Simple and conditional.

- **Simple Statements** - A simple statement is usually a single line and will **always** end with a semi-colon (;). That last part is important. If you have issues with your code, it is most likely a missing semi-colon. An example would be 'digitalWrite(13, HIGH);'.

- **Conditional Statements** – Conditional statements comprise of a condition followed by a series of other statements that only run when the condition is met. Let's walk through another example:

```
1  if (a = = 5){
2     digitalWrite(13, HIGH);
3  }
4  else{
5     digitalWrite(13, LOW);
6  }
```

Okay, I know it looks like a lot, but it's pretty simple. Basically when 'a' is equal to 5, then a light will turn on. We will go over it line by line.

Line 1 – This is the conditional statement portion. It reads out as 'if the variable a is equal to 5, then do the following'.

You may have noticed that the conditional statement has a double-equals sign (= =) in it. This is different than a single equals sign. The double-equals sign means ‘is equal to’. So, the statement after the curly bracket will only execute when ‘a’ is equal to 5.

Line 2 – This line is a simple statement. digitalWrite is a function name that takes one of the pins of the starter kit and either turns it on or turns it off. When you make a pin ‘HIGH’ it turns it on, alternatively when you make a pin ‘LOW’ it turns it off. In this case, it is pin 13 on your starter kit.

Line 3 – This is the closing bracket for the conditional statement we started on line 1.

Line 4 – This is another kind conditional statement. Else is a used with ‘if’ statements. Basically, when an ‘if’ condition is not met, then the ‘else’ statement is executed. So if ‘a’ is not equal to 5, then the statements after the curly brackets will be executed.

Line 5 – This is the statement for the ‘else’ condition. When ‘a’ is not equal to 5, then we will make the led pin LOW. Remember, this turns the pin off.

Line 6 – This is the closing bracket for the else conditional statement we started on line 4.

1.06: Math and Logic – The Arduino is capable of basic math.

- **Basic Operators** – These are used to do super basic math.

This table shows the difference math operators:	
Operator	**Action**
+	Addition
-	Subtraction
*	Multiplication
/	Division
=	Set Value

These should all pretty straight forward. The one thing to remember is that the microcontroller can only do basic math, so it doesn't always know how to handle division. When dividing two integers, the controller will only return an integer. For example, 7 / 4 would normally give your 1.75, but because it will only return an integer it will give you 1 instead, but if one of the numbers is a float type (decimal number) is will give you a more accurate number. So, if you tried 7.0 / 4 then it will give you 1.75, which is a float instead of an integer.

- **Compound Operators –** These are basically shorthand for math. You can use them to make statements shorter and more concise.

This is the full list of compound operators	
Operator	**Action**
+ +	Increment by 1
- -	Decrement by 1
+ =	Compound addition
- =	Compound subtraction
* =	Compound multiplication
/ =	Compound division

Here is an example: ‘X = X + 1’ - In this example we take whatever value is stored in the ‘X’ variable and we’re just going to add 1 to it. So, if ‘X’ was 7, after this it will be 8. Instead of writing that, we could just write ‘X + +’.

Now we understand how ‘+ +’ and ‘- -’ work, let’s talk about how to increment by more than 1. If we use ‘X = X + 4’, just like before it will add whatever value is stored in X and add 4 to it. We could also use ‘X + = 4’ and it will do the same thing.

- **Comparison Operators** – Comparison operators are used to compare two different values. We used one of these earlier when we were talking about if statements. The ‘= =’ is a comparison operator.

This is a list of comparison operators:	
Operator	**Action**
= =	Is equal to
! =	Is not equal to
<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to

These work just like they sound like. You will mostly uses these in conditional statements, like:

```
if ( a > = 20 ){
        digitalWrite(13, HIGH);
}
```

In this example, when a is greater than or equal to 20 the pin 13 will turn on.

- **Boolean Operators** – Boolean operators, sometimes called logic operators, are used to test different conditions in if statements. These will return either TRUE or FALSE.

There are three different Boolean operators.		
Operator	**Symbols**	**Action**
AND	&&	True only if both conditions are true
OR	||	True if either condition is true
NOT	!	True if neither condition is true

The words AND, OR, and NOT aren't written into the code, but instead you will use symbols. Here is an example:

```
If (x = = 15 && y > = 50){
        digitalWrite(13, HIGH);
}
```

Here the statement, 'digitalWrite(13, HIGH);' is only executed when both 'x' is equal to 14 AND y is greater than or equal to 50.

Section 2: Using the Arduino IDE

2.01: Downloading the Arduino IDE - Go to the following website: http://arduino.DREAM-Enterprise.com to download the Arduino IDE and the FTDI device driver. The IDE stands for Integrated Development Environment. We use the IDE to program the starter kit. The FTDI driver is used to tell your computer how to communicate the starter kit through the USB adapter.

2.02: Blinking a LED – Let's go ahead and connect the USB adapter to the computer and to the starter kit. We're going to make the light on the starter board blink on and off.

Open the Arduino IDE and open the File menu. Open the following: File -> Examples -> 01. Basics -> Blink. This should open a new window with the Blink sketch in it. Now we're going to send it to the starter board.

We need to configure the Arduino IDE to communicate to the starter board. Open the Tools menu and in the third section you will find two options, Board and Port. Change the Board option to 'Arduino/Genuino Uno' and then change the Port option to the highest 'COM' number listed.

We're ready to send the code to the starter board. Go to the Sketch menu and click upload. You could also push 'CTRL + U' to upload to the board.

YAY! It should have worked. If you get any error, check the connections between the board and the USB adapter. If it still doesn't work try changing the 'COM' number under the Port option.

2.03: Understanding the Blink Code – Now that it is blinking, let's figure out how it works!

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// 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
}
```

The first section is a multi-line comment. It is used to describe the sketch, list authors, and give further references.

The next section is 'void setup()'. There is one 'simple statement' inside the setup function. 'pinMode(LED_BUILTIN, OUTPUT);' pinMode is used to configure a specific pin to act as an input or output pin. In this case, we are using the pin 'LED_BUILTIN' and making it an OUTPUT. The pin named 'LED_BUILTIN' is pin 13 and is hard coded into the microcontroller in the starter board.

The next section is the 'void loop()' section.

```
1 void loop() {
2     digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
3     delay(1000);    // wait for a second
4     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
5     delay(1000);    // wait for a second
6 }
```

This one is made up of four 'simple statements' and four single-line comments. We are going to walk through it line-by-line.

Line 1 – This is the return type and function name for the loop function.

Line 2 – This line has a 'simple statement' and a single-line comment. The digitalWrite function turns the LED on and then the comment following the function describes what the line is doing.

Line 3 – This statement has the delay function followed by another comment. Delay is a built-in function that is used to make the sketch wait until continuing further. It counts in milliseconds, so 'delay(1000);' will make the sketch wait 1000 milli-seconds or 1 second.

Line 4 – This line turns off the LED.

Line 5 – This line makes the sketch wait another one second.

Line 6 – This line closes the curly brackets for the loop function.

To review, the sketch makes pin 13 an output pin, then it turns the pin HIGH, waits one second, then turns the pin LOW, waits another second, and then starts the loop over again.

Off Week Exercises:

1. Using your breadboard and two LEDs, write a program that will alternate the two LEDs being on for one second at a time.

2. Using your breadboard, button, and LED, make a program where the LED is on until you push the button. When you press the button, the LED turns off.

3. Using your breadboard, two button inputs, and an LED, write a program where the LED will only turn on when both buttons are pressed at the same time.

4. Using your breadboard, a button, and a speaker, write a program that will make the speaker sound at a frequency of 500 when the button is pressed.