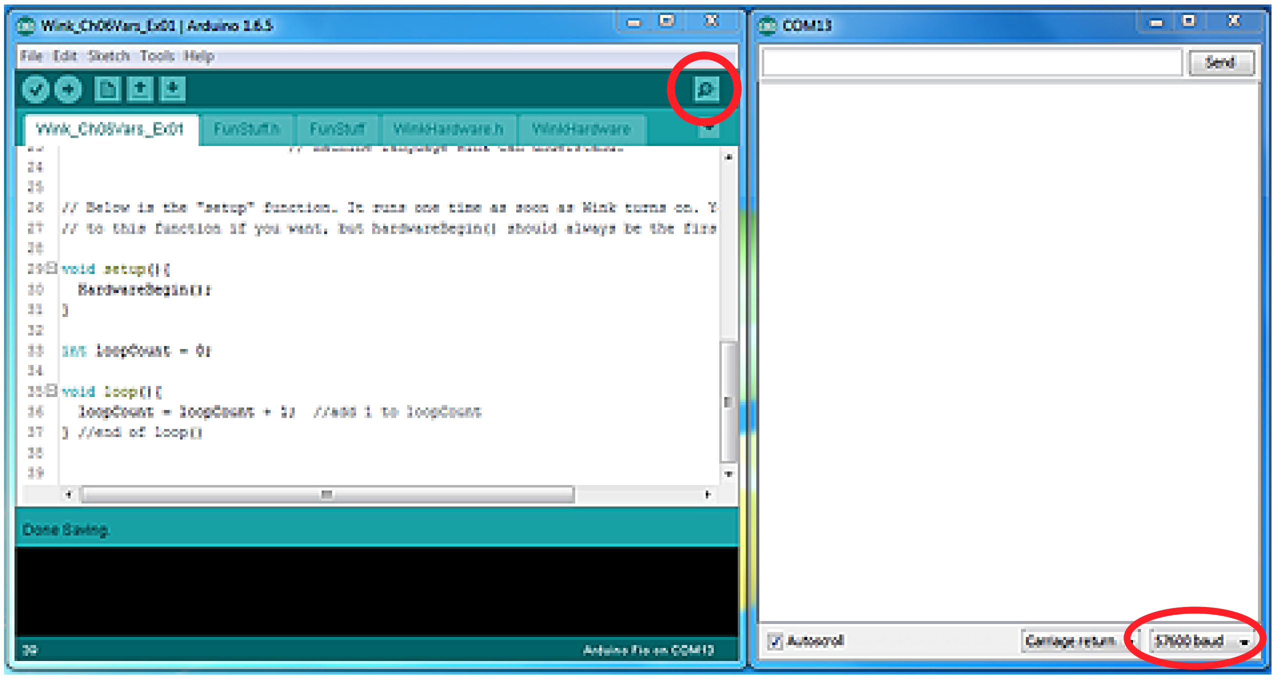
Making Wink tell you stuff...

Wouldn't it be cool if Wink could tell you what he is thinking? He can do exactly that, and we're going to learn how in this lesson. Often times it is useful for a computer program to give feedback to the user. This is especially useful when writing more complicated programs because the computer can tell you what things are happening as the program runs. Sometimes you'll have a block of code that is not behaving as expected, and having the program tell you what is happening as it runs can be very useful in finding the problem. This is often called "debugging" a program.

We are going to use the Serial.print functions in this lesson. The word "serial" in computer speak normally means to send some information over a connection one bit at a time. When you plug Wink into the programming adapter, and the programming adapter is connected to your computer via USB, Wink can use this connection to send information from his brain back to your computer. Serial can be used for several things, but in this case we are going to "print" information to your computer screen, thus, "Serial.print" describes what we are doing.



```
// Below is the "setup" function. It runs one time as soon as Wink turns on. Y
// to this function if you want, but hardwareBegin() should always be the firs

void setup(){
  hardwareBegin();
}

int loopCount = 0;

void loop(){
  loopCount = loopCount + 1;   //add 1 to loopCount
} //end of loop()
```

Your computer will show you the output of Serial.print in the Arduino Serial Monitor window. To open the Serial Monitor window, click the magnifying glass icon in the upper right corner of the Arduino IDE. This will make a white widow show up. You can move the window around and re-size however you want. Check the lower right corner of the Serial Monitor and make sure the speed is set to 57600 baud.

2 SKILL LEVEL

Using the Serial.print functions...

Now that you have your Serial Monitor window open, we'll see how to actually use the function. Serial.print can be used to send text (like letters and words) as well as the value of variables. Let's see an example...

```
# include WinkStuff.h

void setup(){
  hardwareBegin();
  playStartChirp();
}

void loop(){
  Serial.print("My name is Wink!");
  delay(1000);
} //end of loop()
```

} Normal setup

} Send text "My name is Wink!" over the serial port, then delay for 1 second before repeat.

Wink_Ch07Serial_Ex01

In this example, the function Serial.print will "print" anything inside the quotation marks. The quotation marks themselves are not printed. The quotation marks tell the program that the stuff inside the quotation marks should be treated like a string of letters and not a variable.

Try running the above example and see what happens. You should notice that My name is Wink! is printed in your Serial Monitor window every second. You should see the red Tx light flash on the programming adapter each time. This light flashes every time Wink transmits something to your computer over the serial line.

You will also notice, that each time the text is printed, it is run right up against the previous print in your Serial Monitor. There is no space, and no return or new line (it doesn't drop down to the next line). The reason there is no space or new line is because we didn't tell the program to print a new line or extra spaces. Serial.print sends exactly what you tell it to send and nothing more. If we want to create a new line each time we call Serial.print, we need to tell the function to send a new line, which is a special character.

To make a new line appear, you send a back-slash followed by a lower-case "n", which stands for "new line". When the program sees this back-slash, it knows not to actually send the back-slash, but rather, to interpret the letter that follows the back-slash and send the correct "special" character back to your computer.

Let's try this example...

```
void loop(){
  Serial.print("My name is Wink!\n");
  delay(1000);
} //end of loop()
```

} Send the special "new line" character at the end of the string. This causes the Serial Monitor to drop down to the next line.

Wink_Ch07Serial_Ex02

2
SKILL LEVEL

You can also make a new line appear by using a slightly different function, called Serial.print**ln()**. The print with an "ln" following it stands for "Serial Print Line", which works just like Serial.print, except it automatically adds a new line at the end.

This new version with Serial.println works just like the previous example where we added the special "\n" to the string of characters. You can write your code either way. Using Serial.println looks nicer and may be easier to read.

Try this...

```
void loop(){
  Serial.println("My name is Wink!");
  delay(1000);
} //end of loop()
```

} Serial.println automatically advances to a new line after it is sent.

Wink_Ch07Serial_Ex03

You can also send a string of characters using the normal Serial.print() without the special "\n", then in the next line of code, call Serial.println() by itself with nothing inside to make the new line happen. Like this example...

```
void loop(){
  Serial.print("My name is Wink!");
  Serial.println();
  delay(1000);
} //end of loop()
```

} Serial.println is used on a line by itself with nothing inside, which makes the Serial Monitor advance to a new line.

Wink_Ch07Serial_Ex04

Printing the values of variables...

You can print the value of a variable the same way, except you just use the name of the variable by itself with no quotation marks around it. Remember our previous lesson where we had Wink count the number of times he ran the loop() function? Let's have another look at that example, but this time, we'll have him tell us the value each time he runs the loop. Like this...

```
int loopCount = 0;

void loop(){
  Serial.print(loopCount);
  Serial.println();
  delay(1000);
  loopCount = loopCount + 1;  //add 1 to loopCount
} //end of loop()
```

} Declare the variable "loopCount", set it equal to 0.

} Print the present value of loopCount, then send a new line.

} Delay of 1 second.

} Add 1 to loopCount, then put the result back into loopCount

Wink_Ch07Serial_Ex05

2
SKILL LEVEL

Want to know how fast Wink can count? What if we remove the delay(1000) from the previous example?

Before we try it, let's consider a few things you should keep in mind with the Serial.print functions. The time required for Wink to increase the count by one is only about a millionth of a second. He will then send the result to your computer, then immediately count again, then send the result to your computer.

This is going to result in an extremely fast "flood" of serial data coming into your computer. If your computer can't read this information fast enough, it may begin to bog down. It could even crash the Arduino program (though this is unlikely, just something to keep in mind). You will also see that the Serial Monitor window can only update so fast, and the data will arrive far faster than it can update. You will instead see it update one chunk at a time.

If you try the below example and your computer bogs down, you can power off your Wink to make him stop sending the data. You can then re-start Arduino if it hangs (not likely, but possible), and close the Serial Monitor window so your computer won't try to display it. This will make it easier to re-program Wink with another program that doesn't flood the computer.

Give this a try if you want. (It is unlikely it will cause any real problems)...

```
int loopCount = 0;

void loop(){
  Serial.println(loopCount);
  loopCount = loopCount + 1;  //add 1 to loopCount
} //end of loop()
```

Wink_Ch07Serial_Ex06

} Declare the variable "loopCount", set it equal to 0.

} Print the present value of loopCount using Serial.println() which automatically adds a new line. Then add 1 to loopCount and do it all again, very quickly.

When you run this code, you may notice a few interesting things when looking at your Serial Monitor. The first is that the count is going up very quickly, which we would expect. But the time required for Wink to add 1 to loopCount is only about a millionth of a second. This means, Wink should be able to count to a million in about one second. But you see he is not counting anywhere near that fast.

The main reason for this, is that it takes a certain amount of time to send the loopCount value to your computer, one bit at a time, over the serial data line. This time is much longer than the time required to actually count from one number to the next.

The second thing you will notice, if you wait until the count reaches 32,767, is that it all of a sudden becomes a negative 32,768 (it displays as -32768). This is because, as we learned in the Variables lesson, that the "int" variable type will "roll over" once it reaches the largest amount that can be stored in the variable, and goes from being the largest possible value to the lowest possible variable, which for the "int" data type, is -32,768. Each time loop() runs, this negative number has 1 added to it, which makes it slowly less negative until it crosses 0 and becomes positive again.

Mixing Strings and Variables...

So far, we've talked about the stuff inside quotation marks as a "string" of letters. Now that you're comfortable using the function, let's quickly discuss what a "string" really is.

A string is made up of a set of characters (such as letters, numbers, spaces, and symbols like periods, exclamation marks, dollar signs, etc.). The word "rabbit" is a string. You can include numbers in a string if they don't change. For example, a string could say "I have 2 pet rabbits." This is fine if the number 2 is not expected to change.

But what if you want to print a number that does change as your program runs?

There are many ways to do this, and some of them are really complicated, but there's a really easy way to make this happen which you have probably already guessed in reading this.

Try this example...

```
int loopCount = 0;

void loop(){
  Serial.print("Loop has run ");    //first part of string
  Serial.print(loopCount);          //actual value of loopCount
  Serial.print(" times.");          //second part of string
  Serial.println();                 //print new line
  delay(1000);                      //delay 1 second
  loopCount = loopCount + 1;        //add 1 to loopCount
} //end of loop()
```

} Declare the variable "loopCount", set it equal to 0.

} Print the first part of the string, including an extra space at the end, then print the value of the variable, then print the last part of the string (notice the space before 'times.'), then print the new line character.

Wink_Ch07Serial_Ex07

Including the output of Functions in Serial.print...

You can also include the output of functions directly, but first, let's talk about "functions" and what they do. Don't get too hung up on the details at this point and don't let this section scare you. We'll go much deeper into the idea of functions in later lesson.

You have been using many functions during these lessons. motors() for example, is a function. A function can do something useful, like making the motors change speed. A function can do something useful like making the eyes turn red color, and so on. The functions you have used up to this point don't "return" any values. That is, they just run, do something, then come back and Wink's brain just runs the next line of code following the function. But sometimes it is useful for a function to return a useful value to you. For example, you could have a function that goes out and measures a temperature sensor, then comes back with a value, like, the degrees Fahrenheit that was read from the sensor.

There is an interesting function included in Arduino called millis(). The millis() function looks at a timer inside Wink's brain that begins counting as soon as he is turned on. The millis() function then "returns" the count of how many milliseconds it has been since Wink was turned on.

I will explain this concept in two steps so you can more easily understand what is happening.

Start with this code...

```
unsigned long onTime;     //declare the variable “onTime”.

void loop(){
  onTime = millis();
  Serial.print(“Wink has been running for “);
  Serial.print(onTime);      //print value of the onTime variable
  Serial.print(“ milliseconds.”);
  Serial.println();                //print new line
  delay(1000);                     //delay 1 second
} //end of loop()
```

Wink_Ch07Serial_Ex08

} Declare the variable onTime, instead of the “int” variable type, we’re making this one an “unsigned long” variable type.

} Put the value returned by millis() into the variable onTime

Now let’s discuss what is happening with the above code.

The first thing to note, is that when we declare the variable onTime, instead of using the “int” variable type we have been using, we are telling the computer to make this variable an “unsigned long” variable type. This is found in the chart at the end of the Variables lesson. We need this to be an “unsigned long” because that is the kind of variable returned by the millis() function. Because the millis() timer can count extremely high if Wink is left on for a long period of time, the value returned needs this special variable type, because the “int” type does not save enough space in Wink’s memory to hold this large value.

The more interesting point, however, is what we do on the first line inside loop(). This tells Wink to go off and run the special function called millis(). When millis() runs, it looks at the special timer inside Wink’s brain that counts how long he has been turned on, then “returns” back to your loop function. Not only does it just “return”, but it returns holding a number, in this case, the number of milliseconds since Wink was turned on. The equals sign tells Wink to take this returned number and store it in the variable called onTime.

In the next few lines of code, we print the string “Wink has been running for “, then, just in the previous example, we tell Serial.print to print the value stored in the variable onTime. The next line prints the string “ milliseconds.”

If you’re lost at this point, don’t worry, as I said, we’ll talk more about functions and returning values later.

The point of this example is to show you how the value returned by a function can be stored into a variable.

When you actually run this code and open your Serial Monitor window, you should see Wink reporting how long he has been running every second. You should notice that the value goes up by about 1000 each time.

Now let's consider the second step of this concept. As I said at the beginning of this section, the result of a function can be included directly in the Serial.print function directly. Like this...

```
void loop(){
  Serial.print("Wink has been running for ");
  Serial.print(millis());    //print result of millis() function
  Serial.print(" milliseconds.");
  Serial.println();                //print new line
  delay(1000);                     //delay 1 second
} //end of loop()
```

Print the first part of the string, then print the value returned by millis(), then print the last part of the string.

Wink_Ch07Serial_Ex09

The interesting part of this example is the second line, Serial.print(millis()); Let's look at this line of code and explain what is happening. In the previous example, we assigned the value returned by millis() to a variable, then we had the Serial.print function print the value of that variable.

In this new example, we've included the millis() function directly in the Serial.print() function, rather than a variable. It is a normal Serial.print() function, but inside the pair of parenthesis at the end of the Serial.print, where you normally include a string in quotes to print, or the name of a variable to print, we have actually included a whole extra function.

Because we're using the value returned by millis() directly, we don't have to declare a variable to hold the returned value. This saves us a step, and also saves some memory that would have been used by the variable. This isn't important now, but if you were writing a very complicated program, saving space here and there can really add up and save a lot of memory space in the end.

Summary...

So now you know the basics of how to make Wink tell you interesting things. This will be super useful in future lessons. Take some time to play with the Serial.print() functions on your own.

There have been entire books written on how to use various output functions in the C language, and there is a lot more you can do with the Serial functions. If you're interested, have a look at the official page at the Arduino website for Serial. We only covered 2 of the 21 ways to use Serial in this lesson. We'll get to more uses later.

Official Arduino Serial page:
https://www.arduino.cc/en/Reference/Serial