



## The Big Idea:

This lesson extends what we know about working with text in an Arduino™ sketch by adding the ability to change it as the sketch is running.

### Background:

In Lesson 2, an Arduino™ sketch used the serial port to send text to a computer screen, where it appeared in the Serial Monitor. The programming statement that sent the text was:

```
Serial.println("Hello, world!");
```

The text was contained inside double quotation marks. Such information is pre-set. It cannot be changed as the sketch runs. It is used literally. Such information that is programmed exactly as it is to be used is called a *literal*.

Further, a collection of characters, such as the `Hello, world!` message, is called a *String*.

A String (note that this word always begins with a capital letter) is a kind of data. Kinds of data are referred to as *types*. Putting these together, then, the String in the programming statement is a *String literal*. Another way of saying this is that the message `Hello, world!` is a literal of type *String*.

Most Arduino™ sketches, including nearly all the lessons in this book, need a way to store values so the values can change over time and so that multiple parts of the sketch can access the values. This is accomplished by employing a *variable*. You may be familiar with variables from algebra. Here the variable `X` is set equal to the number 42.

```
X = 42
```

The variable name is `X`. The value is the integer 42.

Computer programming languages, including C, provide ways to create and name variables. Along with each variable name `C` also sets aside spaces in computer memory to store the values being represented. Once created, a variable may be assigned a value. That value may be retrieved or replaced with another whenever the sketch requires.

What use would a sketch have for a variable? Making cool sketches possible. Table 3-1 provides some examples.

Table 3-1. Uses of variables in sketches

| Kind of sketch           | Possibly use for a variable  |
|--------------------------|--|
| Laser Tag                | A variable to store energy level is set when the game is started. The sketch refers to the variable when tagging or receiving a tag.   |
| Quad Copter              | A variable to store the desired throttle setting to determine if the copter is climbing, hovering or descending. Its value is set by the user's manipulation of a control and is compared to the copter's actual throttle setting. |
| Digital Musical Keyboard | Lots of variables are used to hold the frequencies of different notes and to provide the correct note output when the corresponding key is pressed.  |

Just as in algebra, variables have names. Unlike with algebra, however, programmers can give variables meaningful names, which aid in making the programming instructions in an Arduino™ sketch understandable. Suppose, for example, a sketch that programs the Arduino™ to play a game. A variable to keep track of a player's name might be `playerName`.

Notice this name is really two words: `player` and `name`. How they are combined into one is by means of a naming convention called *camel notation*. Under this convention the first letter of the first word of the variable is always lowercase, and there are no spaces between words. The first letters of all subsequent words in the variable are capitalized.

The process of setting aside memory space for a variable and assigning that variable's name to that space is called *declaration*. Before it can be used, a variable must be declared. The programmer has the option of assigning an initial value to the variable at that time.

Table 3-2. Vocabulary

| Term                       | Definition   |
|----------------------------|--|
| <b>assignment operator</b> | The symbol used in a programming statement to store a value to a variable. The symbol is the equals sign, <code>=</code> .   |
| <b>camel notation</b>      | A convention for naming variables where words are joined together to form a meaningful phrase to describe what is being assigned. Example of a possible variable in camel notation: <code>playerHighScore</code> |
| <b>concatenation</b>       | The process of appending the value of one <code>String</code> variable to the value of another <code>String</code> variable.   |
| <b>declaration</b>         | A programming statement that sets aside memory for a particular type of data and assigns the variable name that will refer to that type.   |
| <b>delimiter</b>           | The character used to identify the beginning and end of the values for some types of data. For data of the type <code>String</code> the delimiter is the quotation mark: <code>"</code>                          |

| Term                  | Definition   |
|-----------------------|--|
| <b>initialization</b> | The initial value assigned to a newly declared variable.   |
| <b>literal</b>        | A notation for representing a fixed value in source code. Its value cannot be changed as a sketch runs. Literals are often used to initialize variables.   |
| <b>scope</b>          | The portions of an Arduino™ sketch where a variable can be accessed. Scope comes in two kinds: <ul style="list-style-type: none"> <li>• global: the variable is declared at the beginning of the sketch and may be accessed anywhere.</li> <li>• local: the variable is declared within a set of curly braces and may be accessed only within those curly braces. This will be discussed in a later lesson.</li> </ul> |
| <b>String</b>         | A sequence of characters treated as one object. Example: “Hello, World!”.  |
| <b>type</b>           | The kind of data to be assigned to a variable. The type used in this lesson is <code>String</code> . Other types, which will be introduced in future lessons, are: <code>boolean</code> , <code>int</code> , <code>double</code> , and <code>char</code> .   |
| <b>variable</b>       | A name given to a location in memory where a value can be stored. A variable is for a specific type. The name must follow some naming rules. Putting a value into memory is referred to as assigning that value to the variable.   |

## Description:

The rules for using variables in C are:

1. Declare a variable before assigning a value to it.
2. Assign a value to a variable before using it for some other purpose, such as printing or having its value assigned to another variable.
3. Give variables valid names, meaning the names follow some simple rules.
4. Give variables meaningful names in accordance with good practices. Do not access a variable outside of its scope. Local variables may be accessed only from within their set of curly braces, while global variables may be accessed from anywhere within a sketch. The limitation on access is referred to as scope.

## Declaring variables

In order for an Arduino™ sketch to use a variable, the sketch must first know two things about the variable: its name and its type.

## Naming variables

A variable can be given any name, subject to the following rules:

1. A variable name may not begin with a number but can begin with an underbar (`_`) or a dollar sign (`$`).

2. A variable name may not contain spaces.
3. A variable name may not contain mathematical operators: + - / \* % =
4. A variable name may not contain the symbols for logical operators: > < !
5. A variable name may not contain a comma.

*Table 3-3. Examples of names of variables*

| Example            | Comment   |
|--------------------|---|
| volumeOfCube       | valid and descriptive.  |
| correct_answer     | valid.  |
| 3ForAChange        | invalid; cannot begin with a number.  |
| answerForQuestion5 | valid; number is allowed, just not the first character.   |
| Location of wumpus | invalid; contains a space.  |
| tax%rate           | invalid; contains mathematical operator.  |
| age1, age2         | invalid as one name. C will interpret this as two variables, one named age1 and the other named age2.   |
| FrodoLives         | valid but not good practice since the name is not likely to be meaningful in the context of the sketch. |

## Declaration

A **declaration** is the C-language programming statement that makes a variable available to a sketch.

For these first few lessons, all variables will be given global scope, meaning they are declared near the top of the sketch, before the `setup()` method.

The declaration statement consists of two required parts and one optional part. The type and the name are required. As part of declaring a variable, the programmer has the option of giving the variable an initial value. The format of the variable declaration is simple, consisting of three parts: the type, followed by the name and, optionally, an initial value for the variable.

### *Example 3-1. String variable declarations in the C language*

```
String nameOfAccountHolder;
String playerName = "Deputy Dog";
String capital;
```

Notice the following about each of the declarations in Table 3-4:

1. Each declaration begins with the type of variable. In this case each variable is of type `String`.
2. The type of variable is followed by the variable name.
3. These names follow the naming rules, convey meaning, and comply with the camel notation naming convention.
4. The variable `playerName` is assigned an initial value.

### Assigning and using values

Once declared, a variable can be assigned a value by using the equals sign. In C, the equals sign is referred to as the *assignment operator*. That value may be replaced by the assignment of a new value. For example, the statement:

```
nameOfAccountHolder = "Flintstone";
```

stores the `String` literal `Flintstone` to the variable `nameOfAccountHolder`.

This statement:

```
Serial.println(nameOfAccountHolder);
```

will cause the `String` `Flintstone` to appear on the Arduino™ IDE's Serial Monitor.

This statement changes the value stored to the variable `nameOfAccountHolder`:

```
nameOfAccountHolder = "Rubble";
```

Now the statement:

```
Serial.println(nameOfAccountHolder);
```

will cause the `String` `Rubble` to appear on the Serial Monitor.

### Concatenation

Finally, the plus sign (+) may be used to append one `String` to another. This is called *concatenation*. For example, consider the following two declarations:

```
String actorFirstName = "Yogi";  
String actorFamilyName = "Bear";
```

Suppose the programmer needs to have the full name stored to another variable, called `actorFullName`. Further, a space is required between the two names. One way to do this is with concatenation, where the first name, a space `String` literal, and the last name are combined. See Example 3-2.

### Example 3-2.

```
String fullName;  
fullName = actorFirstName + " " + actorLastName;
```

The statement

```
Serial.println(fullName);
```

results in the following to appear on the Serial Monitor:



```
Yogi Bear
```

### Goals:

By the end of this lesson readers will:

1. Know that a variable is a name that can be assigned a value.
2. Be able to follow naming rules and conventions.
3. Know that before a variable can be used it must be declared.
4. Be able to declare variables.
5. Be able to declare variables and assign initial values as part of the declaration.
6. Know how to work with the `String` data type, including use of the concatenation operator `+`.

### Materials:

| Quantity | Part         | Image   | Notes  | Catalog Number |
|----------|--------------|---|--|----------------|
| 1        | Arduino™ Uno |  | Single-board computer. This board is delicate and should be handled with care. When you are not using it, keep it in a box or plastic bag. | 3102           |
| 1        | USB Cable    |  | This is the standard USB adapter cable with the flat connector on one end and the square connector on the other.                           | 2301           |

| Quantity | Part  | Image | Notes  | Catalog Number |
|----------|---|-------|--|----------------|
| 1        | Computer with at least one USB port and access to the Arduino™ website, <a href="http://www.arduino.cc">http://www.arduino.cc</a> . | ---   | The operating system of this computer must be Windows, Macintosh OS/X, or Linux. | ---            |

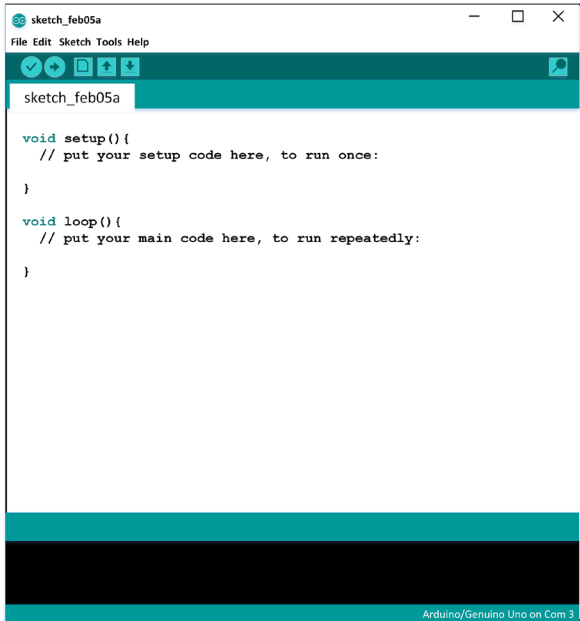
**Procedure:**

*Part I: Set up, upload, and run the first sketch.*

1. Connect the Arduino™ to the computer then start the Arduino™ Integrated Development Environment (IDE).

Arduino™ IDE as it appears when first opened. Notice the type of Arduino™ and the COM port being used appear in the lower-right corner.

COM refers to the communications port. This is assigned by the computer's operating system and may change from time to time.



2. Enter the header comments as shown in Snippet 3-1.

*Snippet 3-1.*

```

/* Lesson3LearnStringVariables
  <author>
  <date>
*/

```

3. Declare three **String** variables just below the header comments as shown in Snippet 3-2. By declaring them here, outside of any methods, the variables are global and can be accessed anywhere in the sketch.

### Snippet 3-2.

```
...  
String str1 = "Hello,";  
String str2 = "world!";  
String str3;
```

4. Add the `setup()` method to your sketch as shown in Snippet 3-3. Use it to send the initial values to the Serial Monitor:

### Snippet 3-3.

```
...  
void setup(){  
    serial.begin(9600);  
  
    serial.print("str1 is: ");  
    Serial.println(str1);  
    serial.print("str2 is: ");  
    serial.println(str2);  
}
```

5. Add the `loop()` method, as shown in Snippet 3-4, but place no programming statements within it.

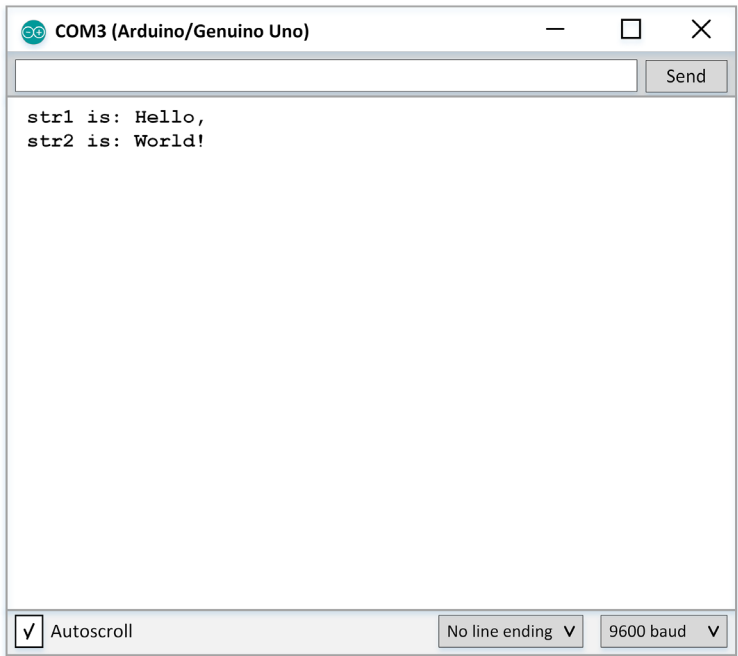
### Snippet 3-4.

```
...  
void loop(){  
  
}
```

6. Save the sketch as `Lesson3LearnStringVariables`.



7. Upload the sketch, then open the Serial Monitor. The following should appear:



Notice the text does not repeat. This is because the print statements are inside the `setup()` method. Since the `setup()` method is run only once, these statements are run only once.

*Part II: Experiment with concatenation.*

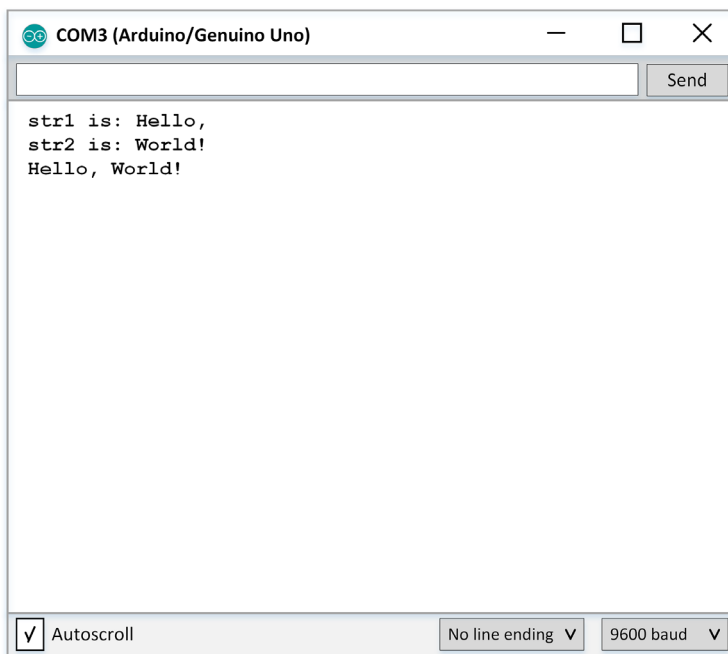
As in Part I, these steps will place programming statements in the `setup()` method. Keep in mind that they could easily be put in the `loop()` method instead. But statements in the `loop()` method are executed over and over. This means the text will be sent to the Serial Monitor over and over.

- 8. Add the programming statements to the bottom of the `setup()` method (existing statements are in gray, new statements in black), as shown in Snippet 3-5.

### Snippet 3-5.

```
...  
void setup(){  
  Serial.begin(9600);  
  
  Serial.print("str1 is: ");  
  Serial.println(str1);  
  Serial.print("str2 is: ");  
  Serial.println(str2);  
  
  // Concatenate str1 with a space  
  // and str2 to produce the message  
  // hello world!  
  // Assign result to str3 then  
  // print it.  
  str3 = str1 + " " + str2;  
  Serial.println(str3);  
}  
...
```

9. `str3` now contains the concatenation of `str1` with a space, followed by `str2`. The next line sends the contents of `str3` to the Serial Monitor.
10. Save the sketch, then upload to the Arduino.™ Open the Serial Monitor. The Serial Monitor should look like this:



## Exercise:

### Exercise 3-1. "There was an old lady"

Create a new sketch called `SpiderLady.ino`. Pattern this sketch after `Lesson3LearnStringVariables.ino`. Initialize variables `str1` and `str2` as follows:

```
String str1 = "There was an old lady who swallowed a ";
String str2 = "I don't know why she swallowed a ";
String sentence;
```

Then, after initializing the Serial port in `setup()` add the programming statements necessary to print a truncated version of the children's poem. The first few statements will look like this:

```
sentence = str1 + "fly";
Serial.println(sentence);
sentence = str2 + "fly";
Serial.println(sentence);
sentence = str1 + "spider";
Serial.println(sentence);
```

... and so on through the critter of bird.



Note

The Arduino™ does not have sufficient data storage for this sketch to print the entire poem.

