Sprintf() With Arduino | Print Multiple Variables To The Serial Monitor

Bottom of Form

**sprintf() –** Ugly to say, but incredibly useful to master. Sprintf() stands for “string print format(ted). It’s your go-to function if you’re looking for a more efficient way to combine text and variables into a string for output to the Serial Monitor. I’ll get into the details shortly, but here’s the main reason you should master sprintf():

**Leaner, clearer, less-cluttered code.**

How much leaner, you ask?

What would take you **nine** lines of code with standard serial.print commands you could do in just **three** using sprintf(). Add that up over a whole project, and it’s a pretty big deal. If that sounds like a handy chunk of knowledge, you’re in the right place. In this lesson, you’ll learn all the above and a little bit more.

Ready? Let’s go.

**First, Let’s Revisit The Clunky Way To Print A String**

So here’s an Arduino 101 question for you. How would you print a string using text and variables using good old Serial.print()? Let’s say you want to print this line of text to the [Serial Monitor](https://www.programmingelectronics.com/serial-communication/):

**“The 3 burritos are 147.7 degrees F”**

In this example, consider both the number of burritos and the temperature value as variables. Using Serial.print() you’d typically write five lines of code to print out that single line of text.

**Serial**.print("The ");

**Serial**.print(numBurritos);

**Serial**.print(" burritos are ");

**Serial**.print(tempStr);

**Serial**.println(" degrees F");

Now to be clear, there’s nothing inherently wrong with using Serial.print() to build a string. It’ll do the trick. But Great Auntie Gertrude’s Carbunkle, is it clunky. In fact, using Serial.print() to build a string gets even more clunky the more variables you add.

What if you wanted to print a line with 4 [variables](https://www.programmingelectronics.com/understanding-using-variables/) inserted into a string like this:

**“The 3 burritos are 147.7 degrees F, weigh 14oz, and were finished 3 minutes ago.”**

For every variable you add to the output, you add *two more serial prints in the code.* So for this helpfully informative burrito update, you’d be staring down the barrel of **nine** lines of code.

**Sprintf() To The Rescue**

This is where sprintf() comes in handy. We can print out as many variables into our string as we want, and the amount of code required always stays at a manageable three lines.

Here are the three lines of code you’ll need:

 char buffer[40];

 sprintf(buffer, "The %d burritos are %s degrees F", numBurritos, tempStr);

**Serial**.println(buffer);

Let’s break each line down.

1. First, you’ll create in a character array to save the output string.
2. Then you use the sprintf() function to combine our text and variables into a string.
3. Finally, you’ll tell [Serial.print()](https://www.programmingelectronics.com/using-the-print-function-with-arduino-part-1/) to display the formatted string.

Let’s take a closer look at each line of code.

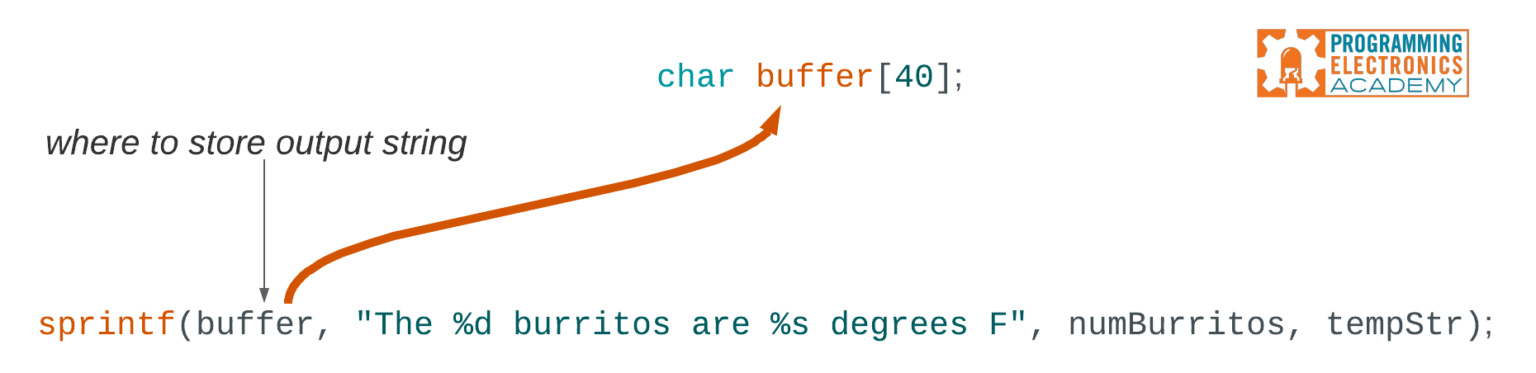
 char buffer[40];

The character array needs to be as large, or larger than the final output string. Just count the characters you plan to store in that string and make sure the buffer is at least that large.

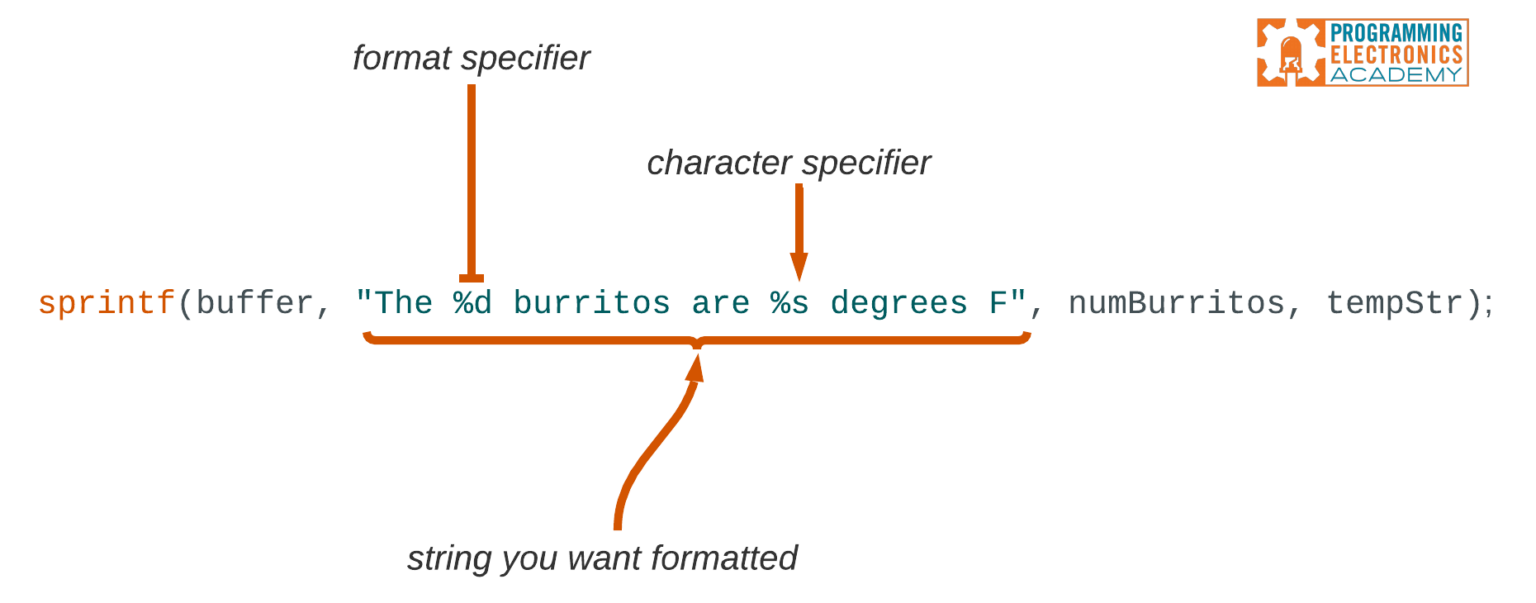
The next line of code is the actual sprintf() function.

 sprintf(buffer, "The %d burritos are %s degrees F", numBurritos, tempStr);

Note that sprintf() requires a minimum of 2 arguments. The first argument is where you plan to store the string that sprintf() will be making for you. This is where you use the character buffer that you created on the previous line.



The next argument is the string you want to create, filled in with **format specifiers** where you want to insert your variables. The format specifier is the % sign. The letter following the format specifier is called the **format character**, and it tells sprintf() what datatype will be used for that variable.



 "The 3 burritos are 147.7 degrees F"

In this example, we have two format specifiers (%). This means we want two variables inserted into the output string. The character specifiers are a little weird at first. They are simply letters that stand for the kind of data type that will be inserted.

It starts to make a lot more sense once you learn what each letter means, so let’s look at that now.

**Character Specifiers**

Here are some of the common character specifiers:

**d** or **i** – signed decimal integer  
**u** – unsigned decimal integer  
**s** – a string of characters

For example, if you use **%d**, you’re telling sprintf() to format the inserted variable as a signed decimal integer.

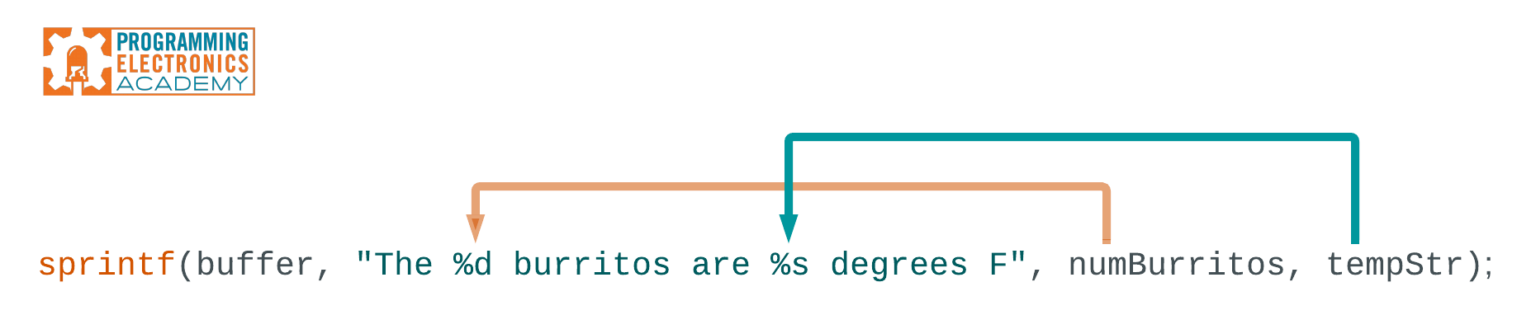
You may be wondering what it means when a character is defined as a **signed decimal integer**. Here’s the scoop:

* **Signed** means it can be positive or negative.
* **Decimal** means we want it to show up in decimal form, instead of formatted into octal or hexadecimal
* [**Integer**](https://www.arduino.cc/reference/en/language/variables/data-types/int/) means it is a whole number, that is, there aren’t any decimal points.

Now, where does sprintf() actually find the variables to insert? Well, we don’t have to look too far, because those are the arguments added right after the string.

 sprintf(buffer, "The %d burritos are %s degrees F", numBurritos, tempStr);

For every format specifier, you must pass a matching value. These values are added as additional arguments to sprintf(), each one separated by a comma.



In this example, we have two format specifiers, and therefore we have two arguments at the end. The first one, numBurritos, is inserted at the first format specifier. The second one, tempStr, is inserted at the second format specifier.

If we had more format specifiers in our string, we’d need to add more arguments to sprint(). However, it all goes onto that one line of code. That’s what makes sprintf() such a useful function to add to your coding toolkit.

**What About Floating Point Numbers?**

Now you might be like… *“Wait a second now – I thought you said the “s” character formatter was for a string of characters, but the temperature in Fahrenheit is a floating point value – what gives?!”*

Well, here’s the deal.

Sprintf() with Arduino cannot handle floating point values. So if you have to print something that has a decimal point, like 3.14 or 156.7, then you need to convert that float value to a character string first, and then print the string. A handy way to do that is with [dtostrf()](https://www.programmingelectronics.com/dtostrf/), which converts a floating point value to a string. I won’t get into that now, but be sure to check out our other video on using [dtostrf()](https://www.programmingelectronics.com/dtostrf/) with Arduino.

The final line of code in our trifecta is the good ol’ Serial.print(). What we pass as an argument is the character buffer where sprintf() stored our formatted string. You’ll notice that sprintf() isn’t returning the string itself. It saves that string into the character buffer we specified. This is why all we have to do is print the buffer’s contents to display our string.

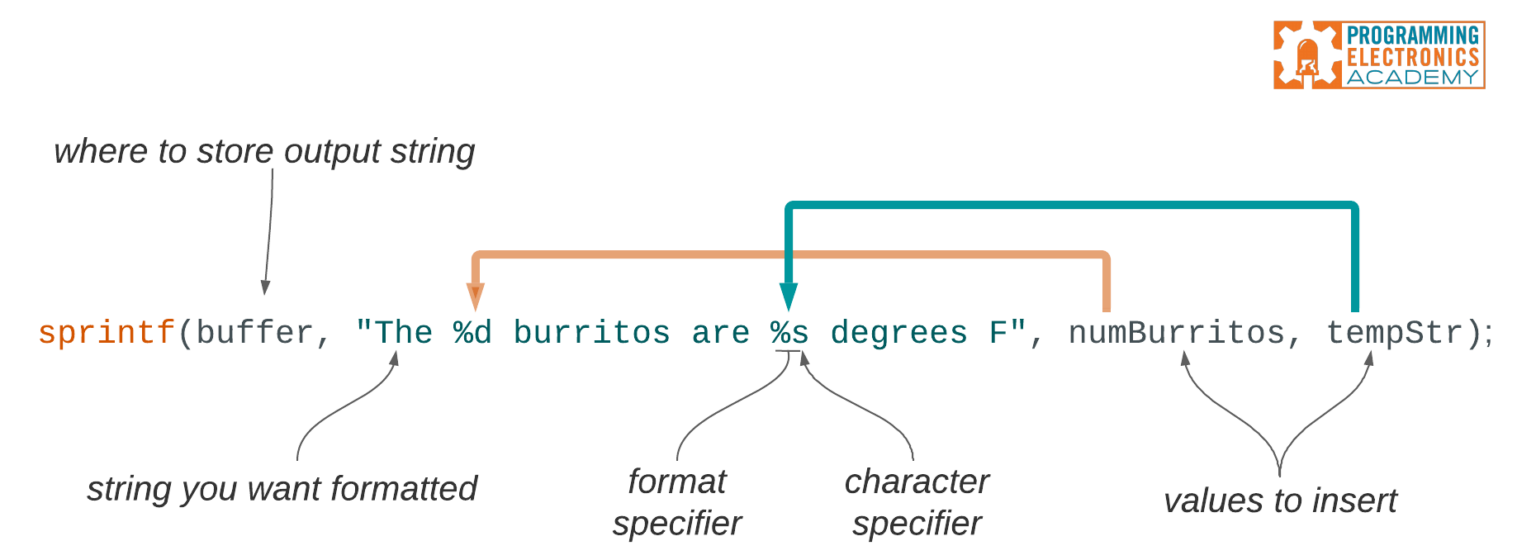
Whew. OK, so yes, let’s name the elephant in the room; floating numbers are a lil’ bit more fiddly than working with other values. However, the incentive to use the sprintf() approach still applies — once you get used to using it, you’ll be writing leaner, better code.

Oh, and before I dive into a quick review of everything you’ve learned, here’s an interesting detail to store away at the back of your mind. It’s worth noting that sprintf() does return a value if you choose to use it, which is the total number of characters that have been stored in the buffer, excluding the null terminating character added by sprintf().

**A Quick Sprintf() Review**

OK, let’s do a 30-second review of sprintf().

**Here’s a line of code using the sprint(f) function:**

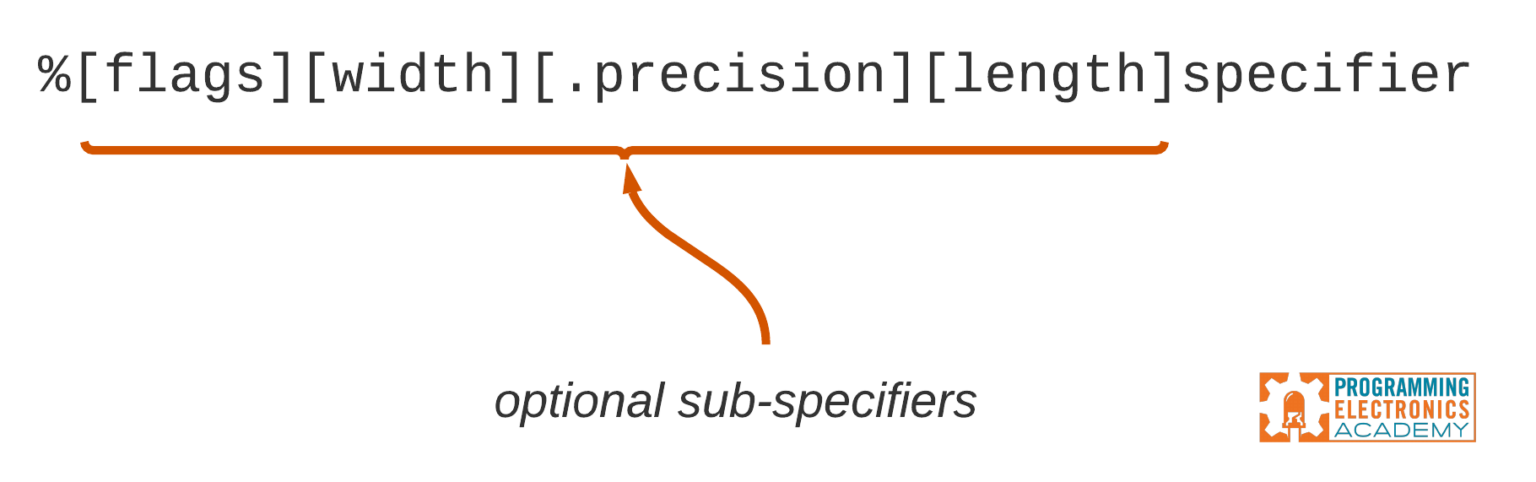


**Here’s what it does:**

1. The first value sprintf() expects is a character buffer. This is where the formatted string will be stored.
2. The second value in sprintf() is the string you want to format, including any format specifiers.
3. The final arguments are the values that’ll replace the format specifiers.

**And if you’re feeling ambitious…**

Hungry for even more efficiency? You’re a productivity animal! (I mean that in a good way.) Believe it or not, there’s a ton of additional stuff you can do with sprintf(). Try playing with **sub-specifiers.** These optional elements live between the % sign and the character specifier. You can use sub-specifiers to perform useful formatting functions, such as left-justifying inserted values and adding leading zeros.



So there you have it; proof positive that sometimes life *isn’t* a marathon but a good old-fashioned sprintf. Let us know in the comments if you’re interested in a follow-on lesson including more detail on those optional sprintf() sub-specifiers.