02-08: Functions

1 - Purpose

- Create a function to do mathematical operations
- Using parameters in functions
- Using Boolean values as parameters
- Using return values in functions

2 - Concepts

3 - The road to repeatable code (functions)

mean(), ***max()***, ***plot()*** are all examples of functions. Functions are reusable code that get executed from within a script -- it is sort of a script within a script. We use functions to avoid writing the same code over and over again. In the last lesson we used a function called ***plot()*** each time we wanted to produce a plot. ***plot()*** is a function built into R to handle the multitude of plotting situations that R programmers implement. We are going to create our own function that is a lot simpler but, like ***plot()***, can be used over and over again.

The function we will create is called ***pythagoras()*** and it will take two input values representing the smaller sides of a right triangle. ***pythagoras()*** solves for the long side of the triangle, or the hypotenuse and returns the answer to the caller using an R function called ***return()***.

So ***pythagoras()*** executes the formula: $c^2 = a^2 + b^2$ or $c = \sqrt{a^2 + b^2}$.

```
1  {
2    rm(list=ls()); options(show.error.locations = TRUE);
3
4    pythagoras=function(a,b)
5    {
6      c = (a^2 + b^2)^(1/2);
7      return (c);
8    }
9
10   hypoteneuse = pythagoras(a=5, b=7);
11 }
```

Functions operate a lot like a black box. There are input parameters (***a*** and ***b***), a bunch of processing in a codeblock, and a return value (***c***). On line 10, ***pythagoras()*** is called and the two parameters in parentheses (***a*** and ***b***) are assigned the values ***5*** and ***7*** by the caller. On line 12, ***c*** is calculated using ***a*** and ***b***. On line 13, ***pythagoras()*** uses the ***return()*** function to return the answer (***c*** = ***8.602***) and this value gets saved in the variable named ***hypotenuse***.

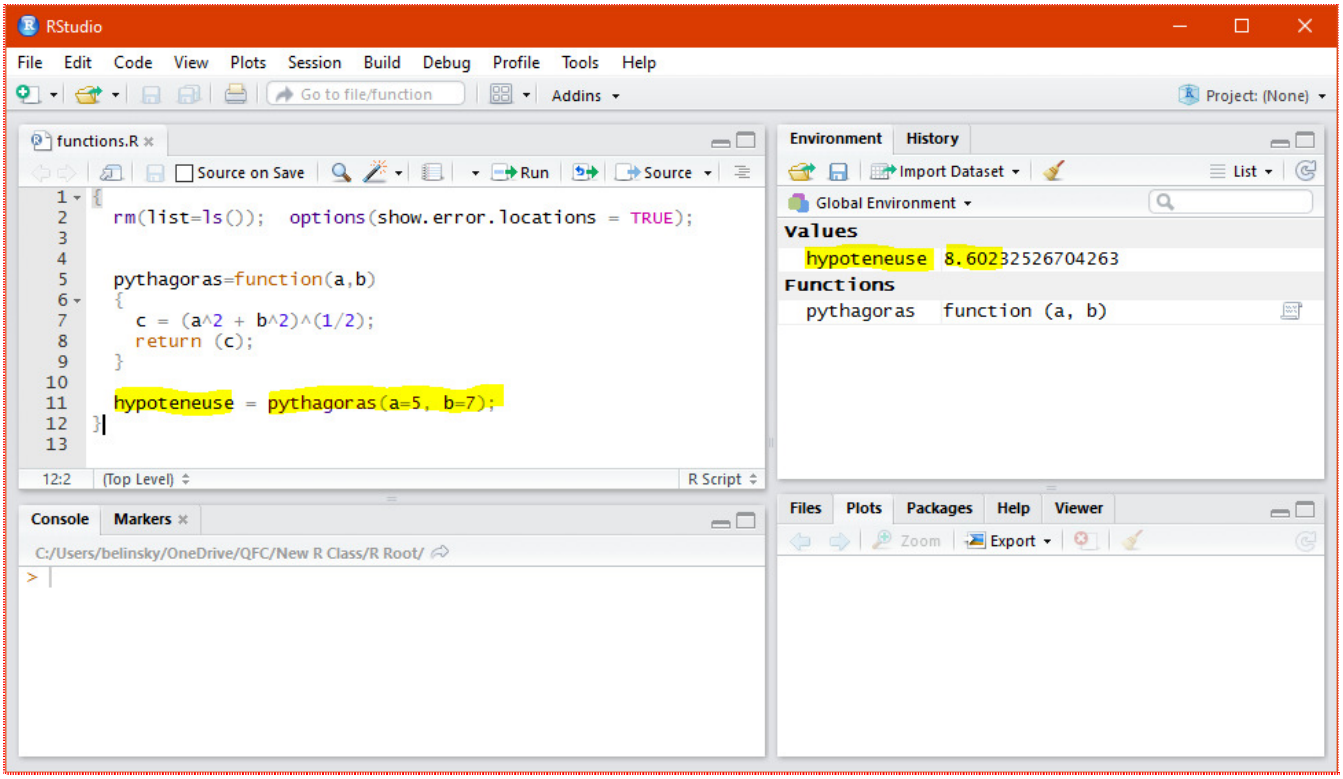

*Fig 1: Calling the **pythagoras()** function with two values (sides of a triangle) to solve the third value (**hypotenuse**)*

4 - Function Details

There is a whole lot going on here and we will take it step by step

```
1 pythagoras=function(a,b)
2 {
3   c = (a^2 + b^2)^(1/2);
4   return (c);
5 }
6
7 hypoteneuse = pythagoras(a=5, b=7);
```

pythagoras() is declared like a variable except there is the term ***function().*** *Variable are assigned values, functions are assigned codeblocks.* The codeblock assigned to ***pythagoras()*** is in between the curly brackets on lines 3 and 6 (***{ }***). This means that whenever ***pythagoras()*** is called in the script, the codeblock attached to ***pythagoras()*** will be executed.

```
1 pythagoras=function(a,b)
2 {
3   c = (a^2 + b^2)^(1/2);
4   return (c);
5 }
6
```

```
7  hypoteneuse = pythagoras(a=5, b=7);
```

Inside the function parentheses are the parameters used by the function (***a*** and ***b***). ***a*** and ***b*** are variables used in ***pythagoras()*** but the values of ***a*** and ***b*** are assigned values by the caller -- so ***a*** and ***b*** are similar to input values.

```
1  pythagoras=function(a,b)
2  {
3    c = (a^2 + b^2)^(1/2);
4    return (c);
5  }
6
7  hypoteneuse = pythagoras(a=5, b=7);
```

The variables (***a*** and ***b***) are assigned values by the user, in this case: ***a=7, b=5.*** ***pythagoras()*** uses the variables ***a*** and ***b*** assigned by the user to calculate the third side of the right triangle. The result of the calculation is assigned to the variable named ***c*** (line 4).

```
1  pythagoras=function(a,b)
2  {
3    c = (a^2 + b^2)^(1/2);
4    return (c);
5  }
6
7  hypoteneuse = pythagoras(a=5, b=7)
```

The function ***pythagoras()*** returns the value of the variable ***c*** back to the user. In other words, ***pythagoras()*** is returning the answer, which is the value of ***c***, **8.507**, to the user.

```
1  pythagoras=function(a,b)
2  {
3    c = (a^2 + b^2)^(1/2);
4    return (c);
5  }
6
7  hypoteneuse = pythagoras(a=5, b=7);
```

To summarize: the function ***pythagoras()*** executes with parameters ***a***=5, and ***b***=7, does its calculation, and assigned the answer to ***c***. The value of ***c***, which is ***8.602***, is returned to the caller and assigned to the variable ***hypotenuse***.

5 - Reusing a function

The reason to create functions is to easily re-use a block of code. We don't have to create new code every time we want to solve for the third side in a right triangle, we just call ***pythagoras()*** and pass in the value of the two smaller sides of the right-triangle.

```
{
  rm(list=ls()); options(show.error.locations = TRUE);

  pythagoras=function(a,b)
  {
    c = (a^2 + b^2)^(1/2);
    return (c);
  }

  hypoteneuse1 = pythagoras(a=5, b=7);
  hypoteneuse2 = pythagoras(a=10, b=23);
  hypoteneuse3 = pythagoras(a=18, b=12);
  hypoteneuse4 = pythagoras(b=12, a=18);
  hypoteneuse5 = pythagoras(18, 12);
}
```

Note that you will often see code that skips the variable names as in:

```
  hypoteneuse3 = pythagoras(18, 12);
```

This line of code works as long as the values are put in the correct order -- it is the equivalent of:

```
  hypoteneuse3 = pythagoras(a=18, b=12);
```

You can also move the variables around and it will execute exactly the same:

```
  hypoteneuse3 = pythagoras(b=12, a=18);
```

All three of these lines will call ***pythagoras()*** and assign ***a*** the value ***18*** and assign ***b*** the value ***12***

5.1 - Variable names do not matter... to R

Variable names are generally chosen to make it easier for the reader to understand the script. But the script could care less what variable names you use -- as long as you keep the variable names consistent. The following script executes the exact same calculation and return the exact same value as the script above -- it just uses variable and function names that are not intuitive to the user.

```
{
  rm(list=ls()); options(show.error.locations = TRUE);

  doStuff=function(aValue, anotherValue)
  {
    answerToStuff = (aValue^2 + anotherValue^2)^(1/2);
    return (answerToStuff);
  }

  whatIGot = doStuff(aValue=5, anotherValue=7);
}
```

6 - Function to do conversions of vector values

We are going to do one more example-- a function that converts all temperature values in a vector from Fahrenheit to Celsius and from Celsius to Fahrenheit.

6.1 - Single value conversion

We will start with the conversion of a single value from Fahrenheit to Celsius

```
{
  rm(list=ls()); options(show.error.locations = TRUE);

  convertTemp=function(tempVal)
  {
    convertedTemp = (5/9)*(tempVal -32); # Fahrenheit to Celsius conversion
    return (convertedTemp);
  }

  temp1 = convertTemp(32);
  temp2 = convertTemp(-20);
  temp3 = convertTemp(80);
  temp4 = convertTemp(tempVal = 80);
}
```

Notice that ***temp4*** explicitly sets the function parameter ***tempVal = 80,*** whereas ***temp3*** does not. But ***temp3*** and ***temp4*** evaluate to the same value.

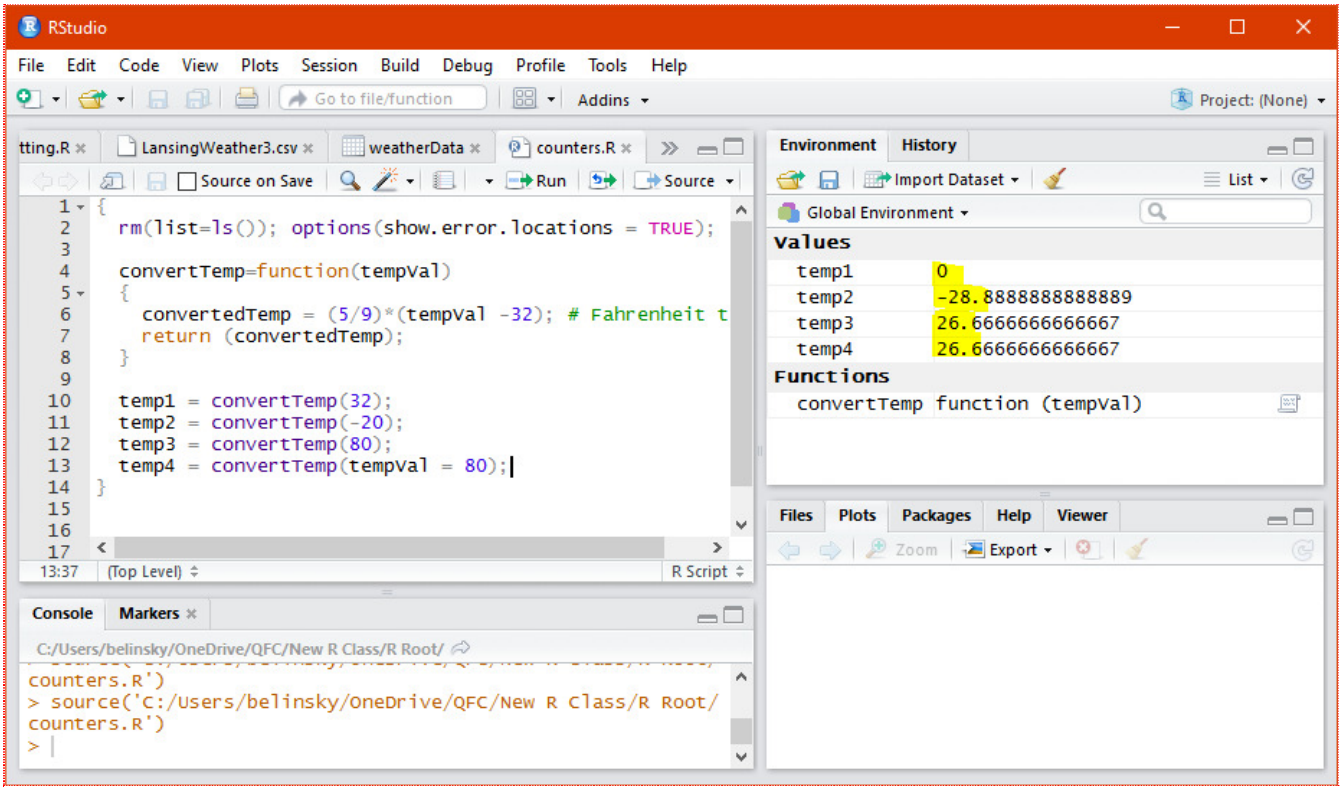

Fig 2: Converting single temperature values from Fahrenheit to Celsius

6.2 - Converting in both directions: Boolean Variable

In the previous unit we generated ***TRUE*** and ***FALSE*** statements by performing conditional operations on a variable like:

```
1  if( yourAge > 20 && yourAge < 50) # yourAge betwen 20 and 50
```

or

```
1  # checking two spellings of Muenster
2  if(favCheese == "Muenster" || favCheese == "Meunster")
```

TRUE and ***FALSE*** are predefined words in R and can be used as values for a variable. A variable that has ***TRUE*** or ***FALSE*** as a value is called a *Boolean Variable*. In the next example we will use a Boolean variables as a function parameter (***toCelsius***) to check how the user wants to convert the temperature values:

if the parameter ***toCelsius*** is set to ***TRUE***: convert from Fahrenheit to Celsius
if the parameter ***toCelsius*** is set to ***FALSE***: convert from Celsius to Fahrenheit

Boolean variables can only be set to ***TRUE*** and ***FALSE*** so they are convenient when you have a situation that exclusively has two options.

6.3 - Function that convert in two ways

Now we want a function that can do conversions in both direction: **Fahrenheit (F) -> Celsius (C)** and **Celsius (C) -> Fahrenheit (F)**. So the function needs to differentiate between an **F -> C** conversion and a **C -> F** conversion. Since there are only two possibilities we can use a ***TRUE***/***FALSE*** scenario. In the script below, the function has two parameters that the user needs to set: ***tempVal*** and ***toCelsius***.

toCelsius *can only have two values:* ***TRUE*** *and* ***FALSE****.*
If ***toCelsius*** is ***TRUE***, we will use the **F -> C** conversion, if ***toCelsius*** is ***FALSE***, we will use the **C -> F** conversion.

```
1  {
2    rm(list=ls()); options(show.error.locations = TRUE);
3
4    convertTemp=function(tempVal, toCelsius)
5    {
6      if(toCelsius == TRUE)
7      {
8        convertedTemp = (5/9)*(tempVal -32); # Fahrenheit to Celsius conversion
9      }
10     else  # toCelsius is FALSE
11     {
12       convertedTemp = (9/5)* tempVal + 32; # Celsius to Fahrenheit conversion
```

```
    }
    return (convertedTemp);
  }

  temp1 = convertTemp(32, TRUE);    # without parameter names
  temp2 = convertTemp(32, FALSE);   # without parameter names
  temp3 = convertTemp(tempVal = 0, toCelsius = TRUE);  # with parameter names
  temp4 = convertTemp(tempVal = 0, toCelsius = FALSE); # with parameter names
}
```

Notice that ***TRUE*** and ***FALSE*** are not in quotes. ***TRUE*** and ***FALSE*** are reserved keywords in R.

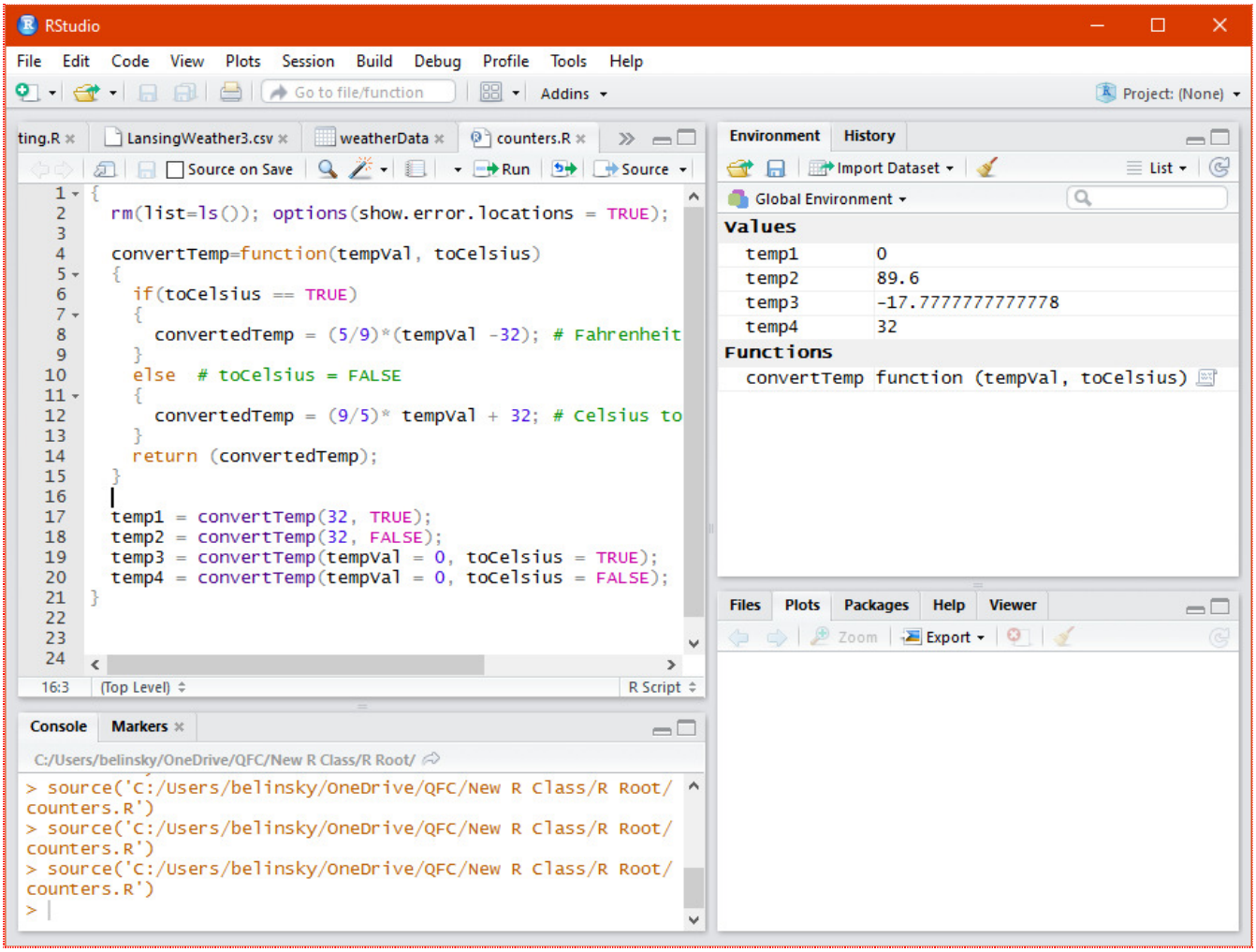

*Fig 3: Using a **TRUE**/**FALSE** value to determine the direction of the temperature conversion.*

6.4 - Using a vector instead of a single value

The function ***convertTemp()*** works with a vector of temperature values without any modification.

```
{
  rm(list=ls()); options(show.error.locations = TRUE);
```

```
3
4   convertTemp=function(tempVal, toCelsius)
5   {
6     if(toCelsius == TRUE)
7     {
8       convertedTemp = (5/9)*(tempVal -32); # Fahrenheit to Celsius conversion
9     }
10    else  # toCelsius = FALSE
11    {
12      convertedTemp = (9/5)* tempVal + 32; # Celsius to Fahrenheit conversion
13    }
14    return (convertedTemp);
15  }
16
17  temp1 = convertTemp(c(-10,0,10,20), TRUE);
18  temp2 = convertTemp(c(-10,0,10,20), FALSE);
19  temp3 = convertTemp(tempVal = c(40,50,60,70), toCelsius = TRUE);
20  temp4 = convertTemp(tempVal = c(40,50,60,70), toCelsius = FALSE);
21 }
```

When ***convertTemp()*** is called in lines 17-20:

- a vector with four values is assigned to ***tempVal***.
- ***tempVal*** gets used to calculate ***convertedTemp*** (lines 8 and 12)
- ***convertedTemp*** will be a vector with the same number of values as ***tempVal***

convertedTemp, a vector, is returned to the caller (line 14). The value of ***convertedTemp*** is saved into the variables named ***temp1***, ***temp2***, ***temp3***, and ***temp4***, which will also be vectors.

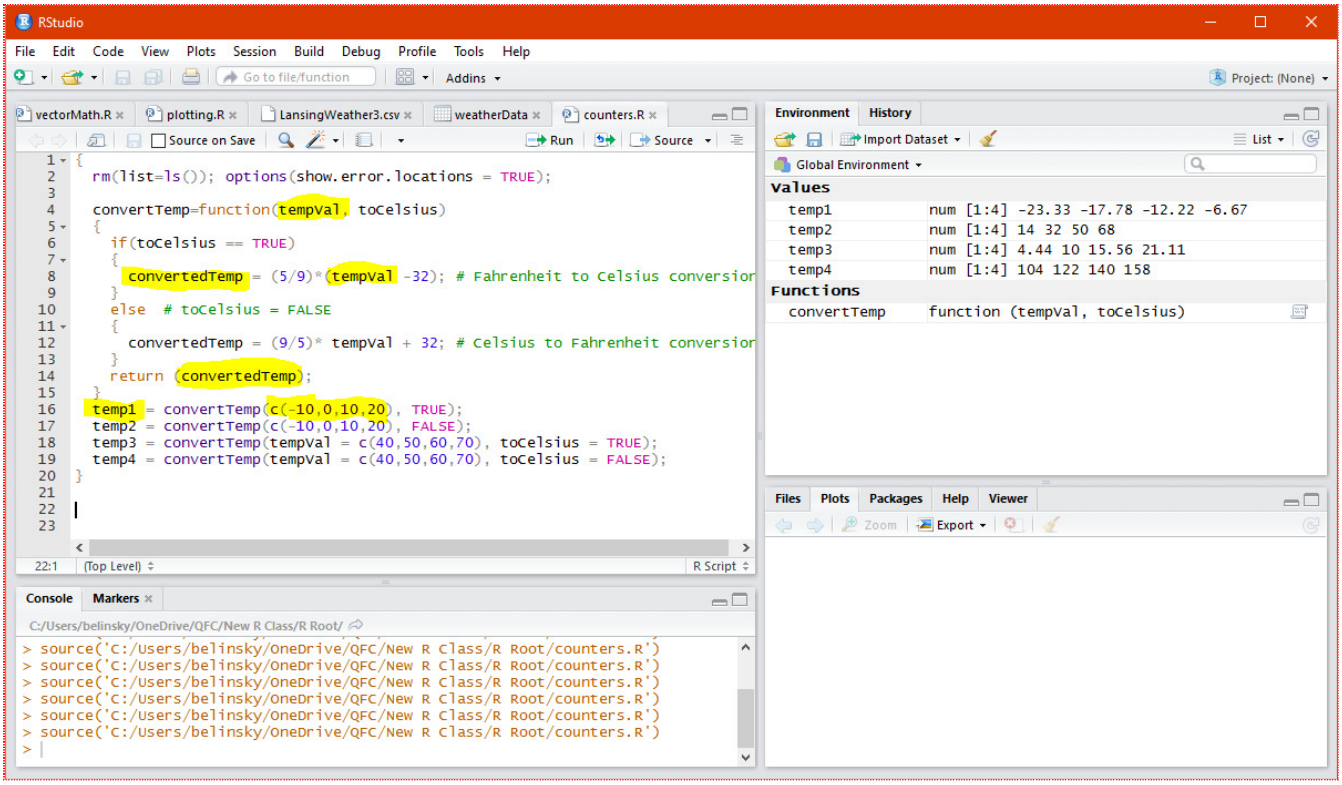

Fig 4: Converting multiple values in a vector.

Extension: Using a for() to iterate through the vector values

7 - Application

1) Create ***one function*** that does all of the following weight conversions:

a) kg -> g
b) g -> kg
c) lb -> g
d) g --> lb
e) lb -> kg
f) kg -> lb

2) Create a function that finds the *difference in temperatures* between consecutive days and returns the temperature differences as a vector.

So if you have four high temperatures: **40, 45, 35, 42**:
the temperature difference returned by the function would be: **5, -10, 7**

Note: *the return vector has one less value* than the vector given to the function.

8 - Extension: Using a for() to iterate through vector values

The following example functionally does the same thing as *Fig 4*except with a ***for().*** While it takes more code to do it this way, it also opens up the possibility of doing more processing on the values (e.g., checking for invalid values).

```
{
  rm(list=ls()); options(show.error.locations = TRUE);

  convertTemp=function(tempVal, toCelsius)
  {
    convertedTemp = c();  # declare convertedTemp as a vector

    for(i in 1:length(tempVal))  # go through each value in the vector
    {
      if(toCelsius == TRUE)
      {
        convertedTemp[i] = (5/9)*(tempVal[i] -32); # F to C conversion
      }
      else  # toCelsius = FALSE
      {
        convertedTemp[i] = (9/5)* tempVal[i] + 32; # C to F conversion
      }
    }
    return (convertedTemp);
  }
  temp1 = convertTemp(c(-10,0,10,20), TRUE);
  temp2 = convertTemp(c(-10,0,10,20), FALSE);
  temp3 = convertTemp(tempVal = c(40,50,60,70), toCelsius = TRUE);
  temp4 = convertTemp(tempVal = c(40,50,60,70), toCelsius = FALSE);
}
```