02-04: Iterations and State Variables

1 - Purpose

- Get length of a vector
- Using a **for()** to perform iterative operations on a vector
- Embedding if() within a for()
- Using Boolean statements

2 - Questions about the material...

If you have any questions about the material in this lesson or your Class Project *feel free to email them to the instructor here*.

3 - Adding values in a vector

For loops can be used to find information about a vector. Some examples are:

- Find the high value in a vector
- Find the sum of a vector
- Find the mean of a vector
- Determine whether a certain value is an element of a vector

for() loops are powerful because they can scale to any amount of data easily. The challenge is that a ***for()*** cycles through the values in a vector iteratively -- in other words, *the values in the vector are presented one at a time instead of together as a group*. This means that intermediate results, what I call the *running total*, need to be saved. This is shown in the next example.

3.1 - Iterative addition

Let's say we have a vector with 4 temperature values: **52, 47, 60, 56** and we want to add the four values together. On paper, we would just add the four values in a single step: **52+47+60+56 = 215** -- but this is not how a ***for()*** operates. A ***for()*** views values in a vector *one at a time*. So, instead of adding four values at once, *you are adding one value to a running total four times*.

The process of iteratively adding four values in a vector using a ***for()*** is:

1. Create a value that stores the *running total* -- set it to **0**
2. Add the first value (**52**) to the running total (0 + 52 = **52**)
3. Add the second value (**47**) to the running total (52 + 47 = **99**)
4. Add the third value (**60**) to the running total (99 + 60 = **159**)
5. Add the fourth value (**56**) to the running total (159 + 56 = **215** -- this is the final value)

The code looks like this:

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

  tempValues = c(52,47,60,56); # vector that holds the four temperature values
```

```
5
6   runningTotal = 0; # state variable (initialized outside the loop)
7   lengthVector = length(tempValues);
8
9   # execute the codeblock using the sequence 1-2-3-4
10  for(i in 1:lengthVector)
11  {
12    # iteratively add the values from tempValues to the state variable, runningTotal
13    runningTotal = runningTotal + tempValues[i];
14  }
15 }
```

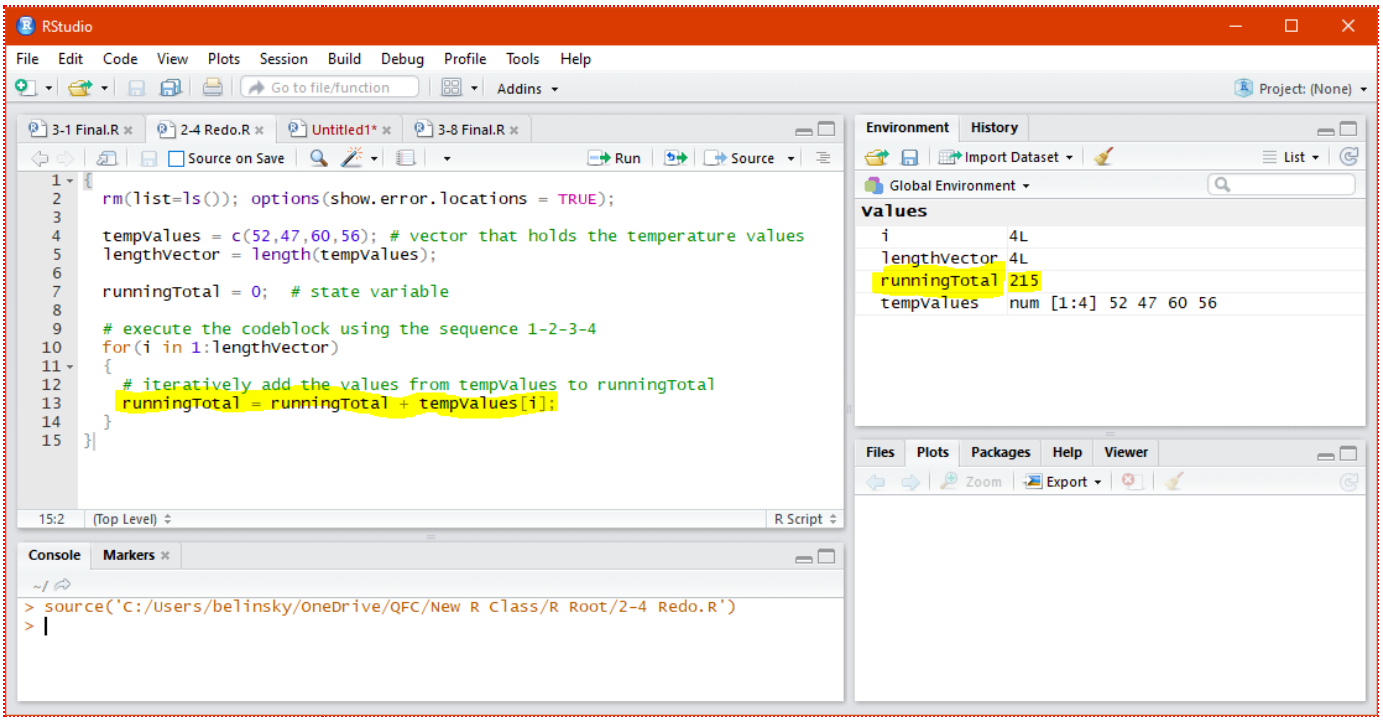

Fig 1: Adding the values in a vector iteratively.

We will break down the code in *Fig.1* into four parts:

1. Declare a *state variable* and set it to **0** (line 7)
2. Get the length of the vector so you can create a sequence (line 5)
3. Create a ***for()*** using the sequence (line 10)
4. Iterate through the codeblock attached to the ***for()*** and add the values in the vector to the state variable one at a time (line 13)

Note: We are using the vector created in line 4, ***tempValues*** with hardcoded values:

```
4 tempValues = c(52,47,60,56); # vector that holds the four temperature values
```

but the vector can come from anywhere, including a data frame column.

3.2 - The state variable

We need a way to save information through each iteration of the ***for()***. In this case the information is the running total of the addition of vector values. To do this we use a state variable, which we call ***runningTotal***.

```
runningTotal = 0; # state variable (initialized outside the loop)
```

The state variable:

- is declared before the ***for()*** to its initial state
- has intermediate states that are updated during the iterations of the ***for()***
- has a final state -- or the value after the ***for()*** has finished all its iterations

At this point we just need to know the initial state of ***runningTotal.*** The initial state is *the value before any values are added*. In other words, the initial state of ***runningTotal*** is **0**.
Trap: Not setting the initial state
Trap: Setting the initial state inside the ***for()***

3.3 - Getting the length of a vector

The length of the vector is the same as the number of values in the vector. In this case, we know we have a vector of length **4** -- in other words, the vector has **4** elements. However, we often do not know the length of a vector before-hand especially when we are dealing with data from an external source.

In R, it is easy to find how many values you have in a vector using the function ***length()***.

```
lengthVector = length(x=tempValues);   # the length of the vector
```

length() returns the number of values in the vector ***x*** -- in this case, ***x = tempValues.***

lengthVector is assigned the value 4. Note: the "L" (*Fig.2*) just says the value is a "long integer" -- it can be ignored.

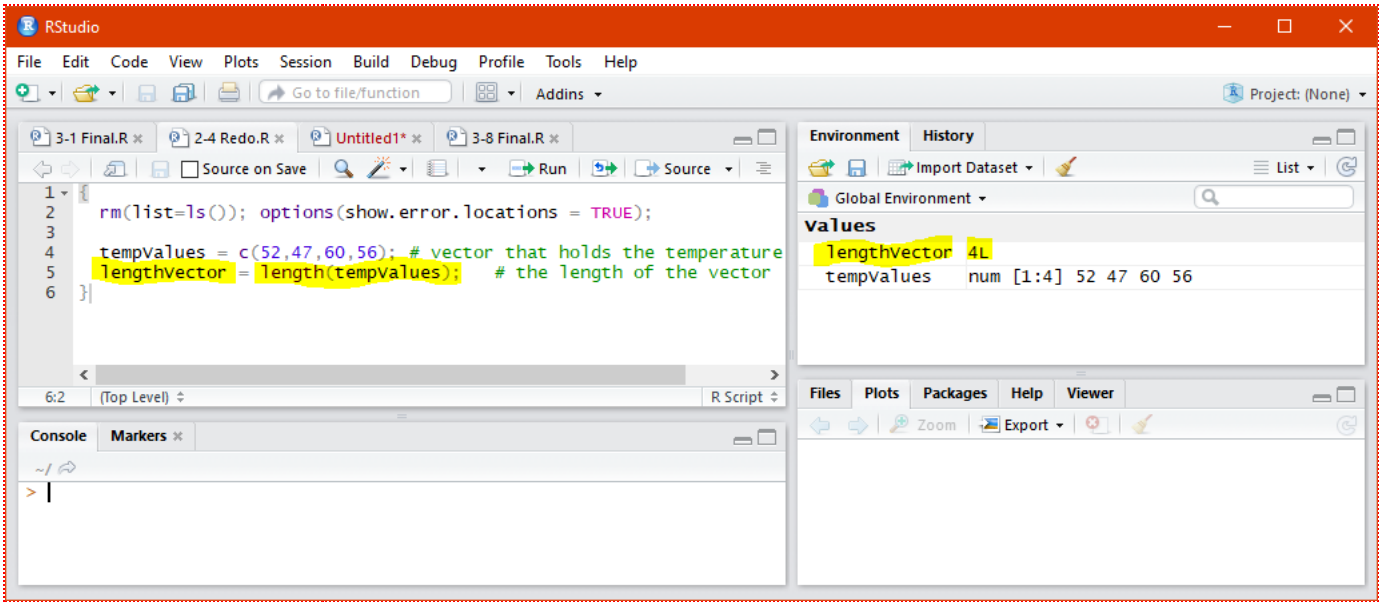

Fig 2: Getting the length of a vector.

3.4 - Creating a sequence to be used in the for()

The length of the vector can be used to create the sequence for the ***for()***.

The sequence tells the ***for()*** how to iterate and we want the ***for()*** to iterate through all of the values in the sequence vector (e.g., 1,2,...). In this case, there are four values so we want the sequence to be **1,2,3,4** or (**1:4)** or, more generically, so that it work with any sized vector: (**1:*lengthVector).***

```
for(i in 1:lengthVector)
{
  # add vector values here
}
```

So, the ***for()*** will iterate **4** times (***lengthVector***) and ***i*** will sequentially take the values **1** through **4**. Note: ***i*** is often used as the name of the iteration variable in a ***for()***.

3.5 - Iterate through the for()

The line of code:

```
runningTotal = runningTotal + tempValues[i];
```

changes the state of ***runningTotal*** each time the ***for()*** iterates.

tempValues has four values: **52, 47, 60, 56**:

- ***tempValues***[1] = **52**
- ***tempValues***[2] = **47**
- ***tempValues***[3] = **60**
- ***tempValues***[4] = **56**

and ***i*** iteratively goes **1, 2, 3, 4.**

This means ***tempValues[i]*** iteratively progresses: ***tempValues***[1]***, tempValues***[2]***, tempValues***[3]***,*** and ***tempValues***[4]

runningTotal starts at **0** and through the four iterations, the values from ***tempValues*** get added to it:

iteration 1... ***runningTotal*** = 0 + 52 = **52**
iteration 2... ***runningTotal*** = 52 + 47 = **99**
iteration 3... ***runningTotal*** = 99 + 60 = **159**
iteration 4... ***runningTotal*** = 159 + 56 = **215**

The final state of ***runningTotal*** is ***215.***

3.6 - Output all states of the state variable

The above example shows the initial state of ***runningTotal*** is **0** and the final state of ***runningTotal*** is **215**. Let's add some Console Window outputs to show the intermediate states. Note: "**\t**" is an instruction to tab, much like "**\n**" in an instruction to go to the next line.

```
{
```

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

  runningTotal = 0; # state variable

  # execute the codeblock using the sequence 1-2-3-4
  for(i in 1:lengthVector)
  {
    cat("Running Total before iteration", i, "=", runningTotal);
    runningTotal = runningTotal + tempValues[i];
    cat("\t\tAfter iteration", i, "=", runningTotal, "\n");
  }
}
```

Now we can see the intermediate states of 52, 99, and 159 for ***runningTotal***.

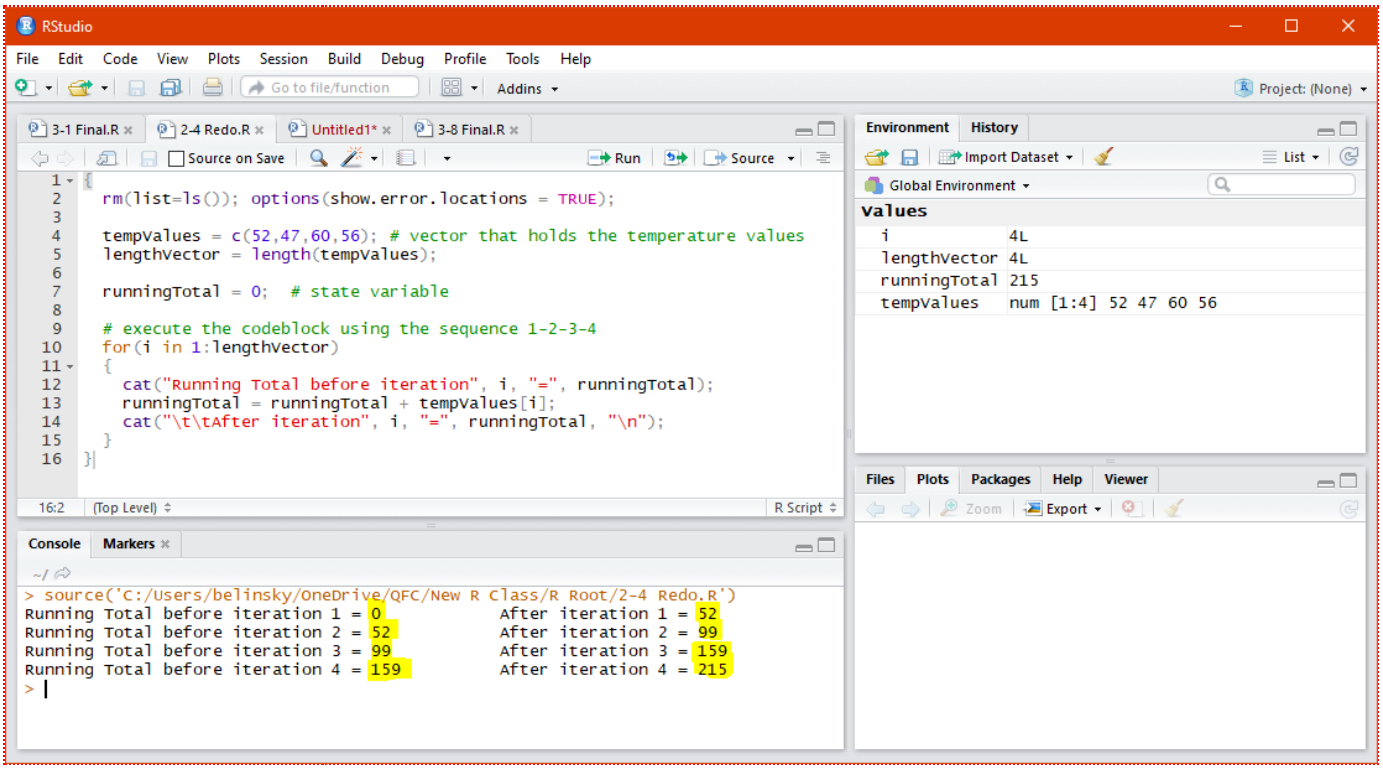

Fig 3: Output the intermediate states of the vector addition.

3.7 - Finding the mean value using length()

Right now we have the sum of the values in ***tempValues*** and we can use the sum to find the mean -- all we need in the number of values in ***tempValues***. But we already got this value in line 4 using ***length().*** ***lengthVector*** is the same as the number of values and we just need to divide ***runningTotal*** by ***lengthVector*** to get the mean. We just need to make sure we do this *after the* ***for()*** *finished all its iterations.*

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

  runningTotal = 0; # state variable

  # execute the codeblock using the sequence 1-2-3-4
  for(i in 1:lengthVector)
  {
    runningTotal = runningTotal + tempValues[i];
  }

  meanValue = runningTotal / lengthVector; # solve for the mean
}
```

The mean value is ***runningTotal*/*lengthVector*** = **215/4** = ***53.75***.

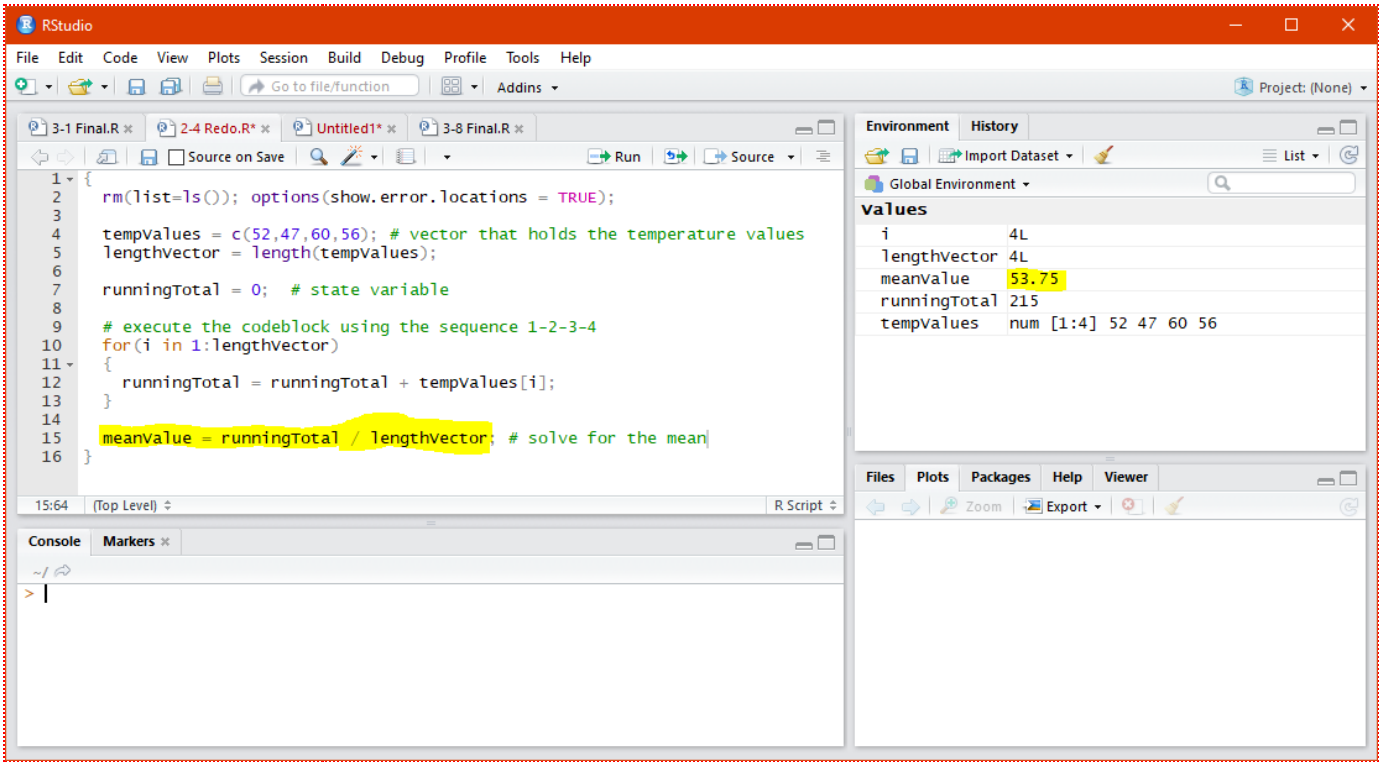

Fig 4: Getting the mean from the final value of the state variable and the vector length.

4 - Finding the maximum value

Another example where we can use a ***for()*** with a state variable is finding the maximum value in a vector. Again, we are only using a vector with four values, but the technique works with vectors that have any number

of values and whether or not the number of values in the vector is known before-hand.

The state variable in this example is named ***currentHighTemp***.

4.1 - Initial state

The initial state of ***currentHighTemp*** should be a value that is known to exist in the set. The best value to use is the first value in the vector we are checking, or ***tempValue***[1]. If we used a number that was not already in the vector and happened to be larger than any of the vector values we could mistakenly assign the initial state as the final high temperature.

4.2 - Intermediate states

The ***for()***, through each iteration, will check the other values in ***tempValues*** and change ***currentHighTemp*** whenever a new high value is found creating the intermediate values of ***currentHighTemp.***

4.3 - Final state

The final state of ***currentHighTemp*** is whatever value it has after all the iteration of the ***for()*** have executed.

4.4 - Code

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

  currentHighTemp = tempValues[1];       # state variable (initial)

  for(i in 1:lengthVector)
  {
    if(tempValues[i] > currentHighTemp)
    {
      currentHighTemp = tempValues[i];   # state variable (intermediate)
    }
  }
}
```

We can see that the *final state* of ***currentHighTemp*** is ***60***, which is the high temperature in the vector.

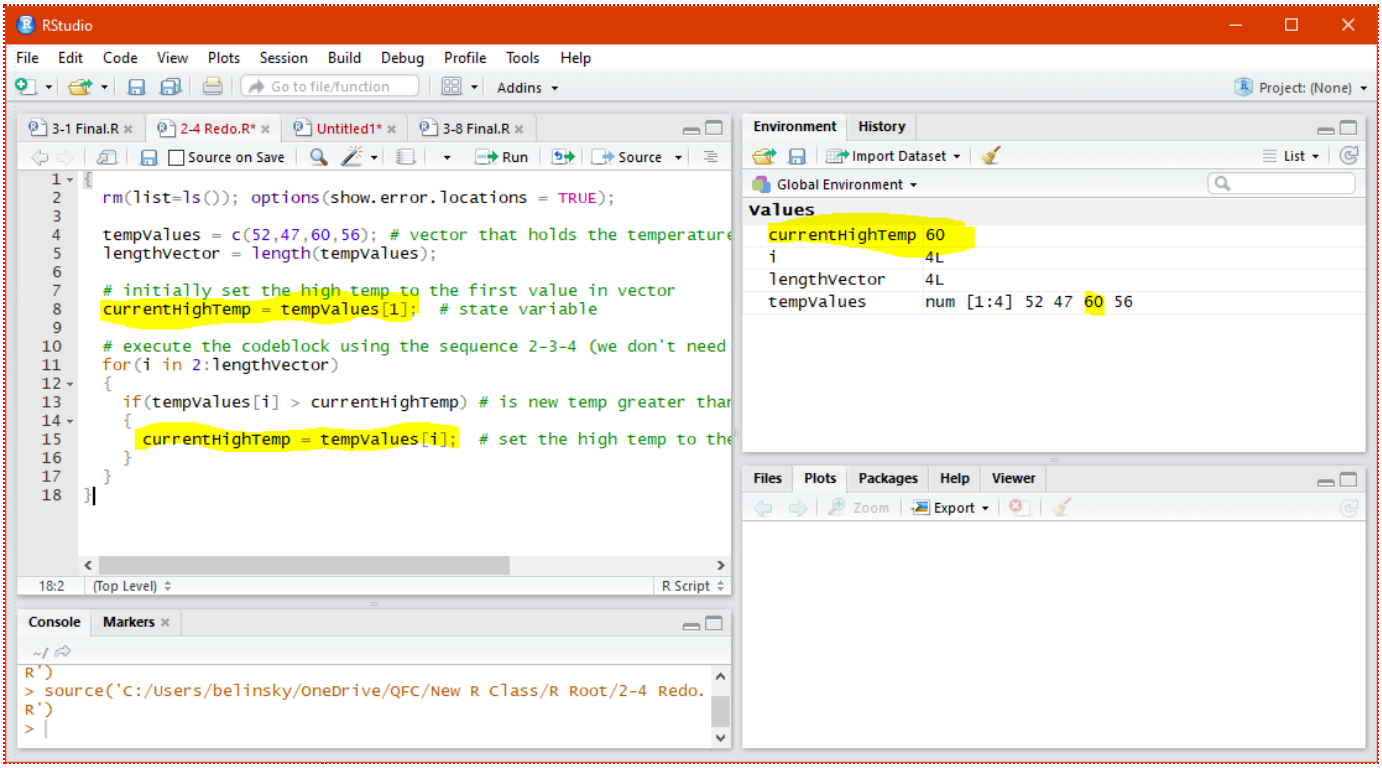

Fig 5: Finding the maximum value in a vector.

4.5 - Code with intermediate state output

Now let's add some ***cat()*** statements to better show what is happening each time the ***for()*** iterates.

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

  currentHighTemp = tempValues[1];       # state variable (initial)

  for(i in 1:lengthVector)
  {
    cat("\nChecking high temp,", currentHighTemp, ", against", tempValues[i], "\n");
    if(tempValues[i] > currentHighTemp)
    {
      cat(tempValues[i], "is greater, updating high temp\n");
      currentHighTemp = tempValues[i];    # state variable (intermediate)
    }
  }
}
```

currentHighTemp is initialized to the first value (***52***) in ***tempValues***. ***currentHighTemp*** is checked against **47** in the first iteration, **60** in the second iteration, and **52** in the third iteration. ***currentHighTemp*** only changes in the second iteration to 60 so the final state of ***currentHighTemp*** is 60 (shown in Environment Window).

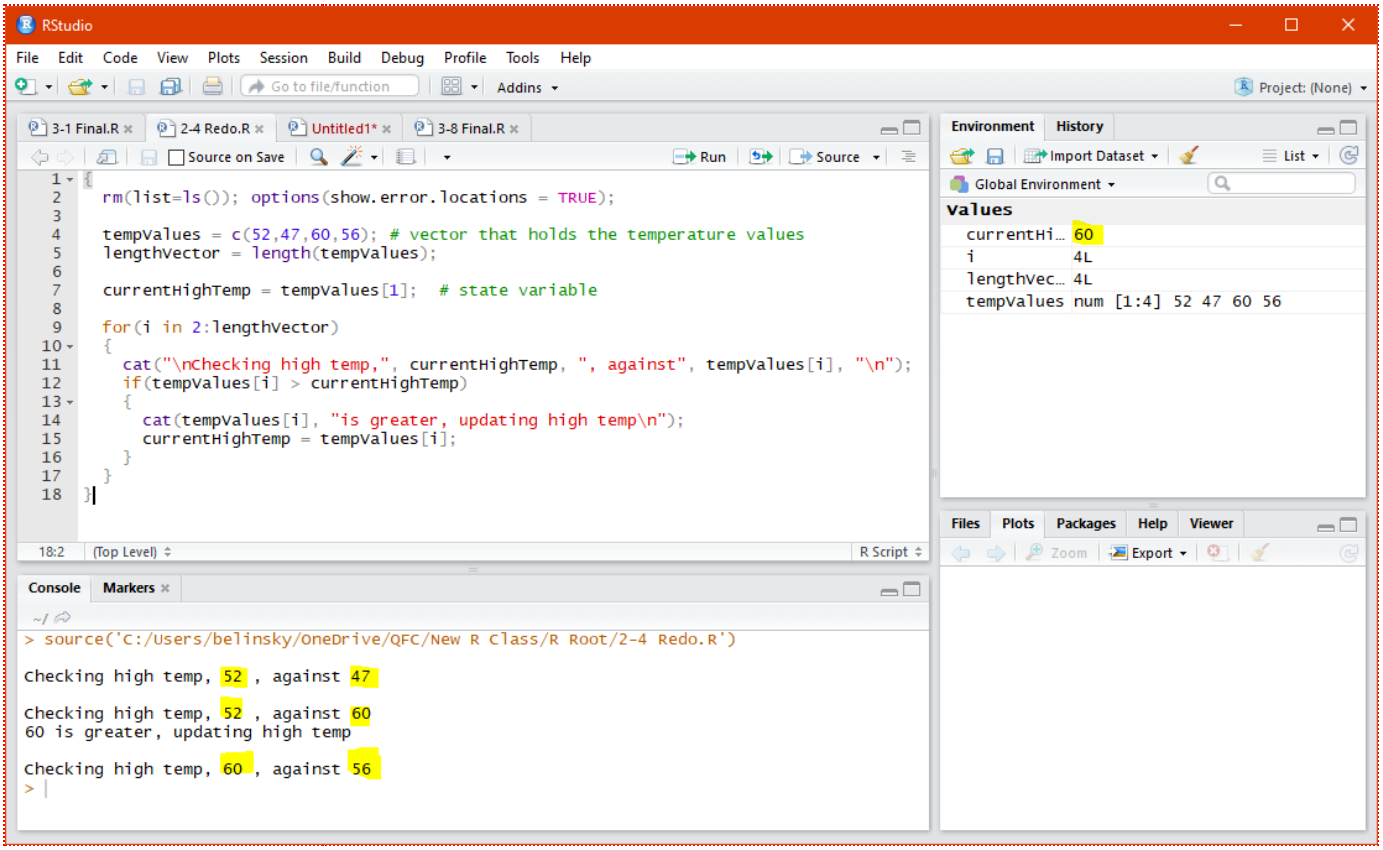

*Fig 6: The state variable (the vector maximum) only changes in the second iteration of the **for()**.*

Extension: could have started the sequence at 2

5 - Boolean variables as state variables

In our last example we will use a Boolean variable as a state variable. A Boolean variable can only take two values: ***TRUE*** and ***FALSE***. We can use a Boolean variable to ask a yes/no question about a vector like: is there a temperature less than 50?

In this example, the state variable is named ***tempLessThan50***

We assume there are no values less than 50 *until proven otherwise*, so the initial state of ***tempLessThan50*** is ***FALSE***.

Then we need to check each value in ***tempValues***. If any of the values are less than 50, we change ***tempLessThan50*** to ***TRUE***.

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

```

```
  tempLessThan50 = FALSE; # state/Boolean variable (outside loop)

  for(i in 1:lengthVector)
  {
    if(tempValues[i] < 50) # Is this temp less than 50
    {
      tempLessThan50 = TRUE; # set state to TRUE
    }
  }
}
```

The final state of ***tempLessThan50*** is ***TRUE*** because the second value in ***tempValues*** is less than **50**.

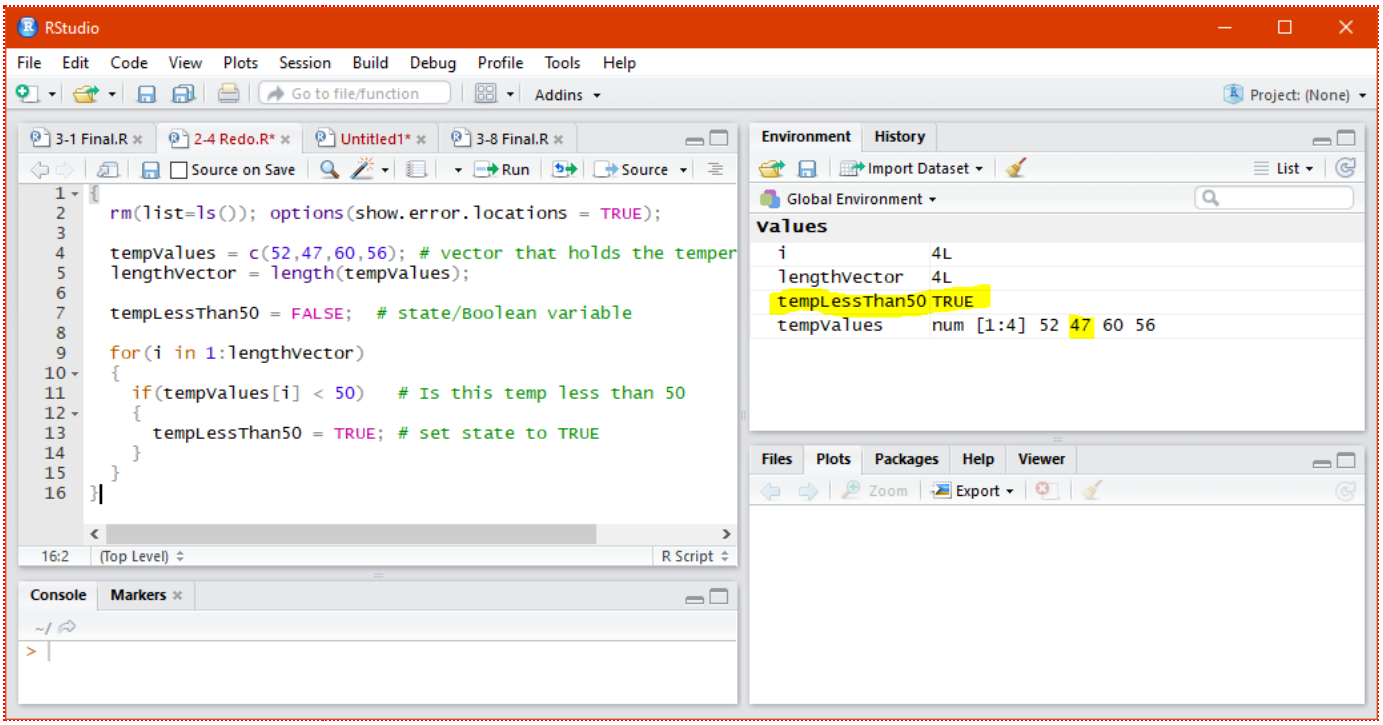

Fig 7: Using a Boolean variable as a state variable.

Extension: breaking out of for() loops

5.1 - Boolean state that does not change

If we run the same code but check for values under 40 instead of 50:

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values

  lengthVector = length(tempValues);

  tempLessThan40 = FALSE; # state/Boolean variable (outside loop)
```

```
  for(i in 1:lengthVector)
  {
    if(tempValues[i] < 40) # Is this temp less than 40
    {
      tempLessThan40 = TRUE; # set state to TRUE
    }
  }
}
```

We see the final state of ***tempLessThan40*** is ***FALSE***. In other words, it never changed through all the iterations of the ***for()*** because there were no values less than 40.

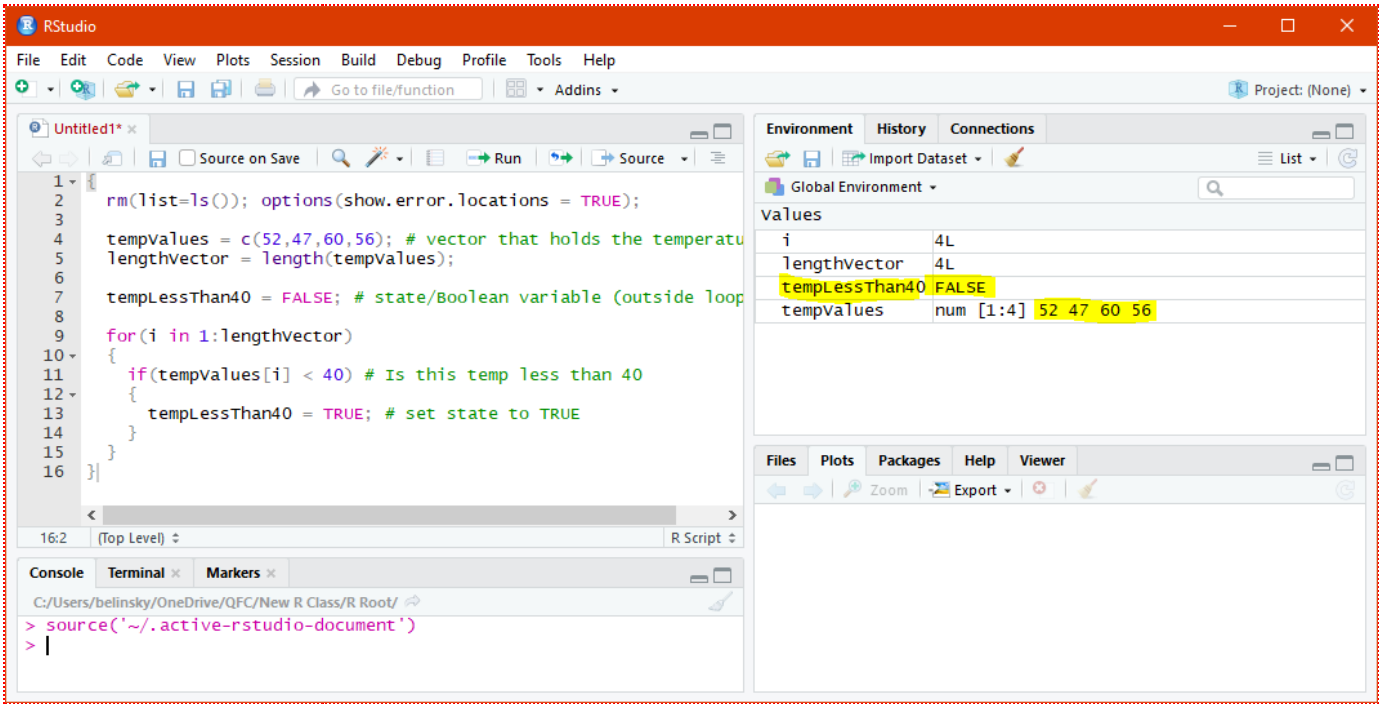

*Fig 8: The Boolean variable's initial and final state are both **FALSE**.*

6 - Application

If you have any questions regarding this application or your Class Project, feel free to email them to the instructor here. You can attach the whole Root Folder as a zipped file.

1) Use a ***for()*** to find the minimum, maximum , and mean value in the following vector: **c(94, 102, 89, 105, 78, 85)**. Find the length of the vector using ***length().*** Hint: you can find each of these within the same loop if you set three state variables before the loop begins.

Note: You can do the above using multiple ***for()*** loops or, as a challenge, with only 1 ***for()*** loop.

2) Use a ***for()*** and Boolean variables to check the vector **c(94, 102, 89, 105, 78, 85)** to see if there are values:

a) greater that **100**

b) less than **70**

c) equal to **78**

d) equal to **87**

*Save you script file as **app2-4.r** in the **scripts** folder of your RStudio Project for the class.*

7 - Extension: breaking out of loops

Notice that the codeblock attached to the ***for()*** will only set ***tempLessThan50*** to ***TRUE***. So once ***tempLessThan50*** is ***TRUE***, there is no way it can be turned back to ***FALSE*** (which makes sense given the context). This also means that once it is ***TRUE***, you are really just killing electrons (and wasting time!) if you continue to check values. This might not matter for 4 values but it probably matters for 10,000 values.

We can stop the execution of a ***for()*** loop using the ***break*** statement.

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

  tempLessThan50 = FALSE; # state/Boolean variable

  for(i in 1:lengthVector)
  {
    if(tempValues[i] < 50) # Is this temp less than 50
    {
      tempLessThan50 = TRUE; # set state to TRUE
      break;                 # break out of the for loop
    }
  }
}
```

Notice that the value of ***i*** is **2** (*Fig.9*) because that is when the ***break*** occurred meaning the ***for()*** iterated only twice. Without the ***break***, the ***for()*** would iterate ***4*** times and ***i*** would have a value of **4**.

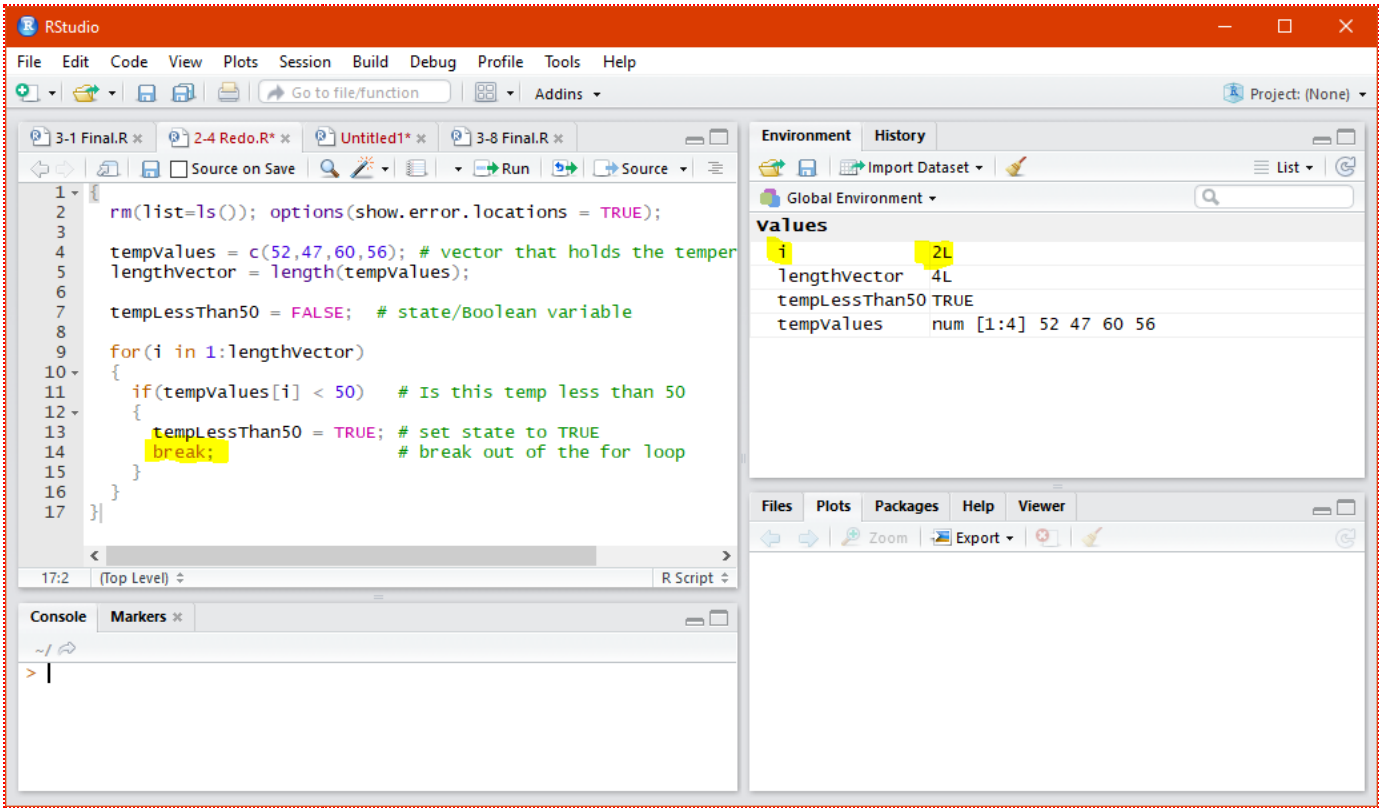

*Fig 9: Breaking out of a **for()** loop using **break**.*

8 - Trap: Not setting the initial state of a state variable

If we do not initialize the state of a state variable...

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

  tempValues = c(52,47,60,56); # vector that holds the four temperature values

  runningTotal = NULL; # declare state variable but don't initialize it
  lengthVector = length(tempValues);

  # execute the codeblock using the sequence 1-2-3-4
  for(i in 1:lengthVector)
  {
    # iteratively add the values from tempValues to the state variable, runningTotal
    runningTotal = runningTotal + tempValues[i];
  }
}
```

Then the final value of ***runningTotal*** is **numeric (empty)**

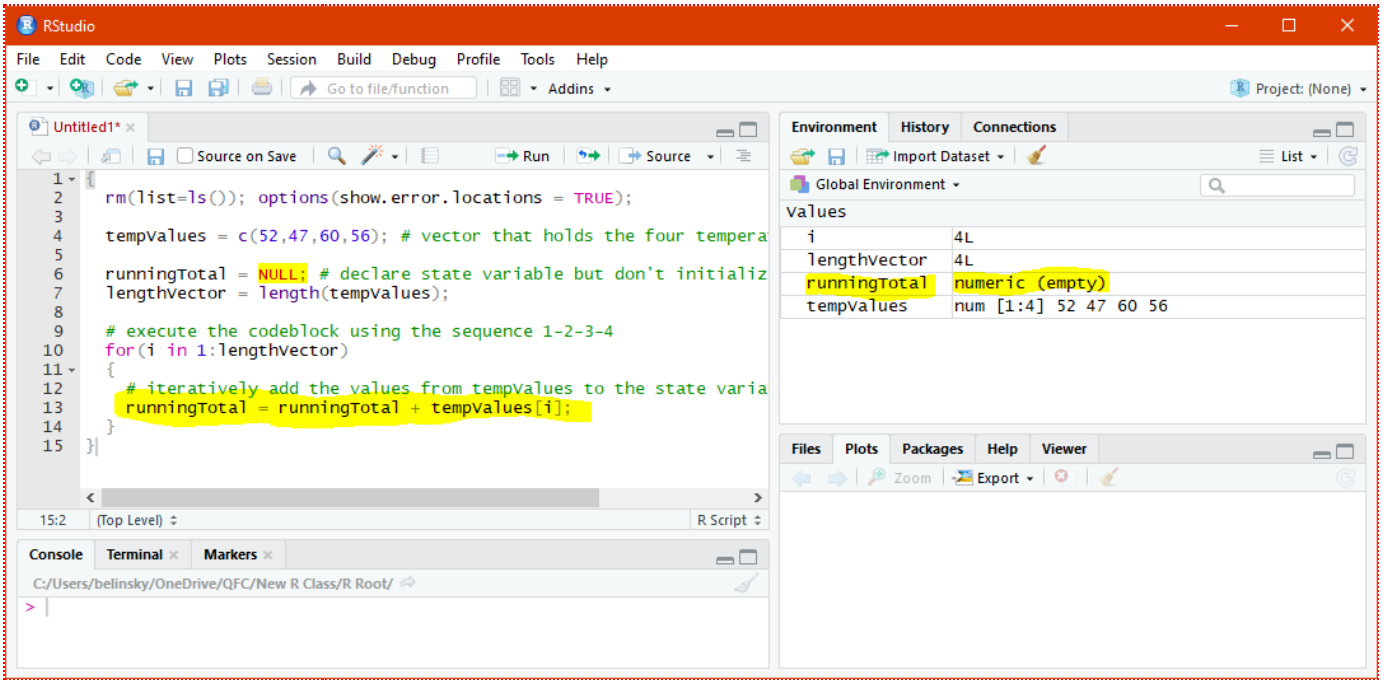

Fig 10: Forgetting to initialize the value of the state variable.

The reason runningTotal does not have a value after executing the script is because of line 13:

```
13   runningTotal = runningTotal + tempValues[i];
```

On the right side ***runningTotal*** is ***NULL*** and, in R, when you add ***NULL*** to any number the answer is still ***NULL***. **numeric(empty)** is sort of the same as saying "numeric NULL".

9 - Trap: Setting the initial state of a state variable inside a for()

If we accidentally initialize the state variable inside the ***for()***...

```
1  {
2    rm(list=ls()); options(show.error.locations = TRUE);
3
4    tempValues = c(52,47,60,56); # vector that holds the four temperature values
5
6    lengthVector = length(tempValues);
7
8    # execute the codeblock using the sequence 1-2-3-4
9    for(i in 1:lengthVector)
10   {
11     runningTotal = 0; # initialize state variable to 0
12     runningTotal = runningTotal + tempValues[i];
13   }
14 }
```

Then the final value of ***runningTotal*** is the last value in ***tempValues***, which is **56**.

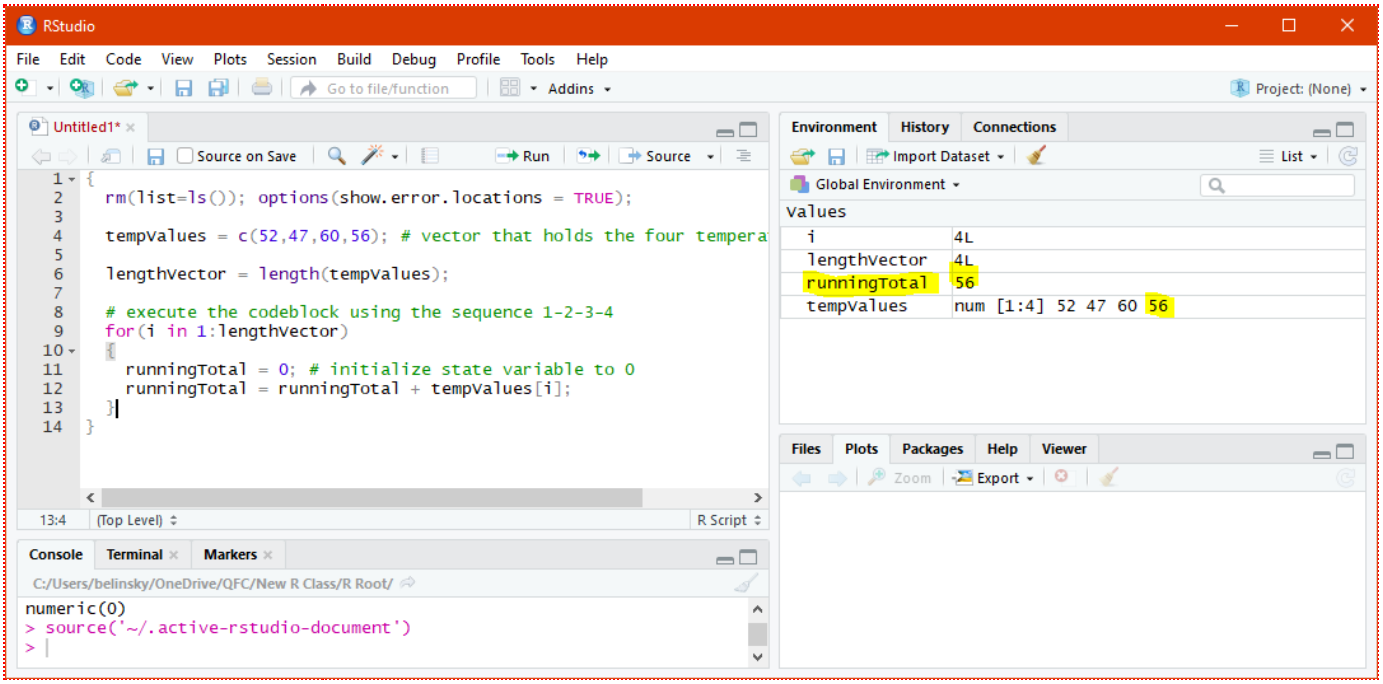

*Fig 11: Initializing the state variable inside the **for()** loop.*

This happens because you are resetting the value of ***runningTotal*** to **0** every time the ***for()*** iterates. In other words, the intermediate states are destroyed and the final value of ***runningTotal*** will just be **0** + the last value.

10 - Extension: Sequence could start with 2

Since we initialized the state variable to the first value in the vector, we could have started the sequence at **2** instead of **1**. In other words, *we don't need to check the first value against the first value*.

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

  tempValues = c(52,47,60,56); # vector that holds the temperature values
  lengthVector = length(tempValues);

  currentHighTemp = tempValues[1];       # state variable (initial)

  for(i in 2:lengthVector)  # start sequence at 2 instead of 1
  {
    cat("\nChecking high temp,", currentHighTemp, ", against", tempValues[i], "\n");
    if(tempValues[i] > currentHighTemp)
    {
      cat(tempValues[i], "is greater, updating high temp\n");
      currentHighTemp = tempValues[i];   # state variable (intermediate)
    }
  }
}
```

The script executes with the same result.

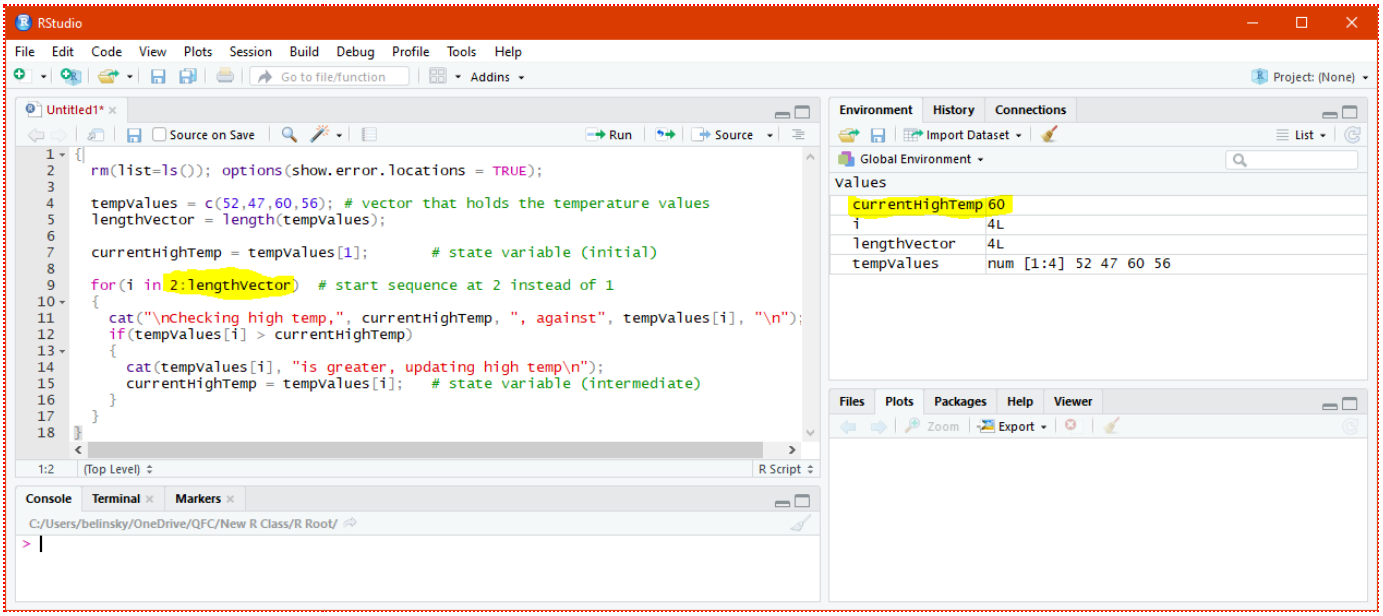

*Fig 12: Starting the sequence in the **for()** at **2** instead of **1**.*

The only real harm of starting the sequence at 1 is that an extra sequence is executed unnecessarily and a few electrons are inconvenienced.