01-06: Outputs

1 - Purpose

- Output variables to the Console Window
- Output messages to the Console Window
- Output special characters to the Console Window, including a line feed
- Output mixed message and variable statements to the Console Window

2 - Questions about the material...

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

3 - Output to console

In the last lesson, we had examples of R output when we asked the user for information using ***readline()***. The ***readline()*** function instructs R to output the statement inside the parentheses to the console and then wait for an input from the user. In this lesson we will more formally go through the different ways to output information to the user. The output, in these cases, will always go to the Console Window. In a later lesson we will talk about outputs to the Plot Window and to a file.

3.1 - Output a message

If you just want to output a message to the user then you use the function ***cat()***. Note that ***cat()*** stands, rather unintuitively, for concatenate. *Extension: The concatenate, cat(), function.* But, for our sake, ***cat()*** is simply an instruction to output to the Console Window what is inside the parentheses.

Put the message you want to output to the Console Window in quotes inside the parentheses of ***cat()***.

```
1 cat("Hello, World");
```

If you do not use quotes (***" "***), R will think you are referring to a variable. For instance the line ***cat(Hello)*** gives you the *"Object not found"* error because R thinks this is a reference to a variable named ***Hello*** -- and, in this case, there is no variable named ***Hello.***

You can add as many ***cat()*** statements to your script as you like and they will be sequentially sent to the Console Window:

```
1 {
2   rm(list=ls()); options(show.error.locations = TRUE);
3
4   cat("Hello, World.");
5   cat("How are you?");
6   cat("I am fine?");
```

```
7 }
```

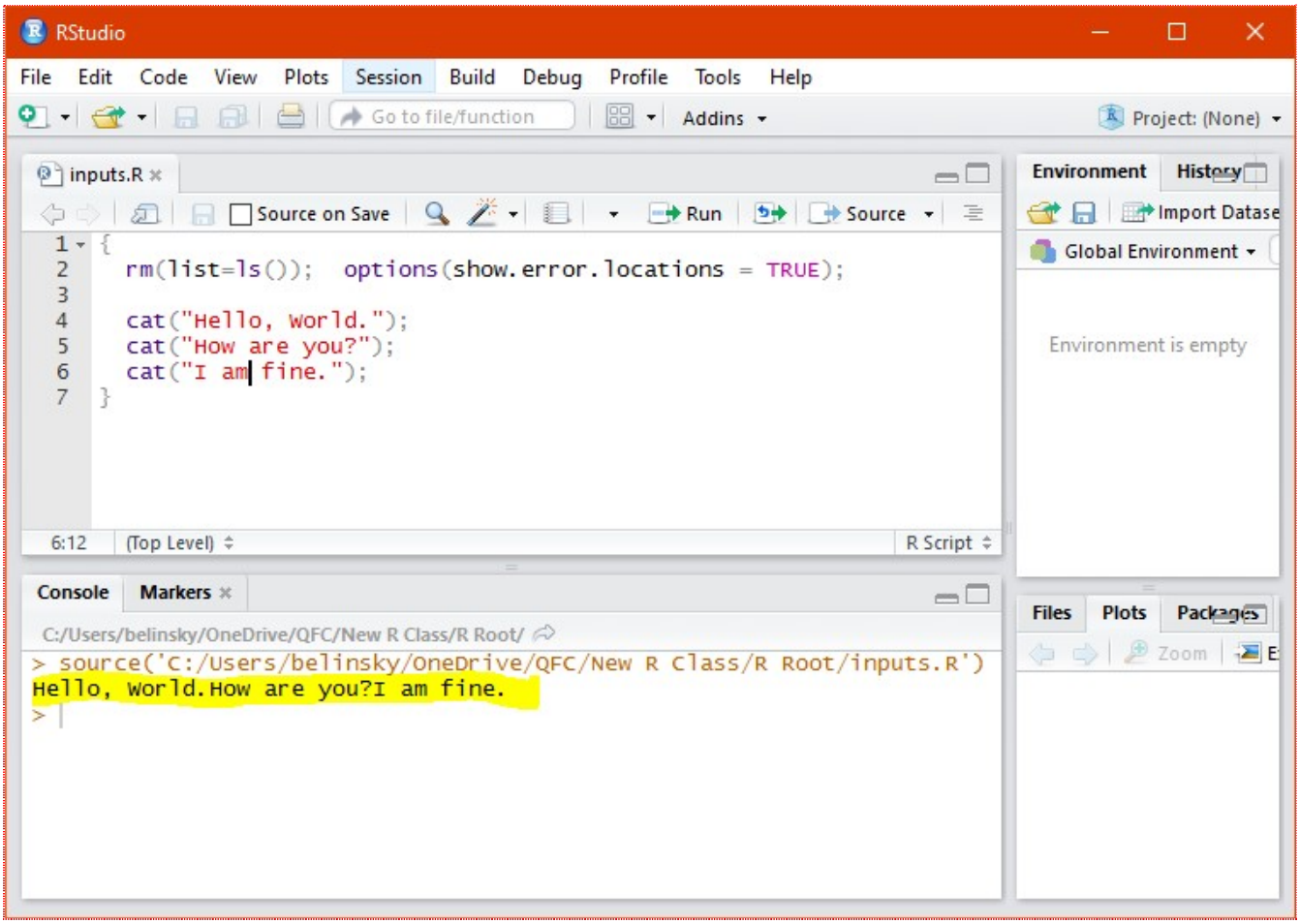

Fig 1: Output message without spacing

3.2 - The line feed: \n

But there is a problem here -- R puts no spacing in between the lines (*Fig.1*). This is actually typical behavior of most programming languages -- they do not add line feeds (e.g., enter or return) unless the script specifically requests that a line feed is added.

The instruction to add a line feed is ***\n*** and ***\n*** is put inside the quotes (***" "***):

```
1 {
2   rm(list=ls()); options(show.error.locations = TRUE);
3
4   cat("Hello, World.\n");
5   cat("How are you?\n");
6   cat("I am\n fine?\n");
7 }
```

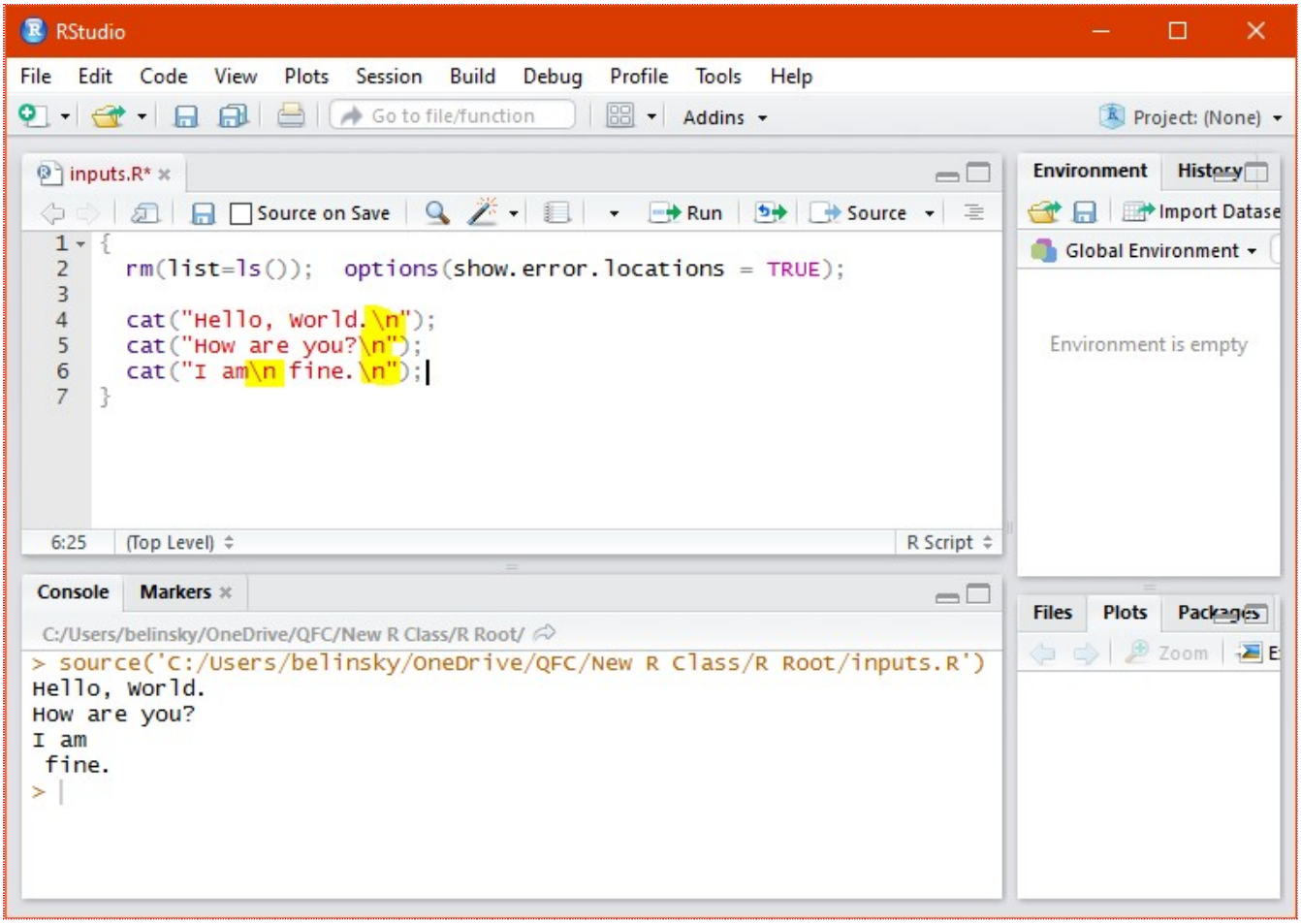

Fig 2: Message with line feeds

Now we have our text on multiple lines (*Fig.2*). Notice that the ***\n*** does not get printed to the screen and \n is not a variable. ***\n*** *is an instruction inside the message to add a line feed, sometimes called a newline character,* and needs to be put inside the quotes.

EXTENSION: the backslash (\ )

\n can be put anywhere in a quote. In the above code there is a ***\n*** put in between "am" and "fine", hence the line feed between the two word. The exact same message as *Fig 2* can be output to the Console Window using only one ***cat()*** statement with multiple ***\n***:

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

  cat("Hello, World.\nHow are you?\nI am\n fine?\n");
}
```

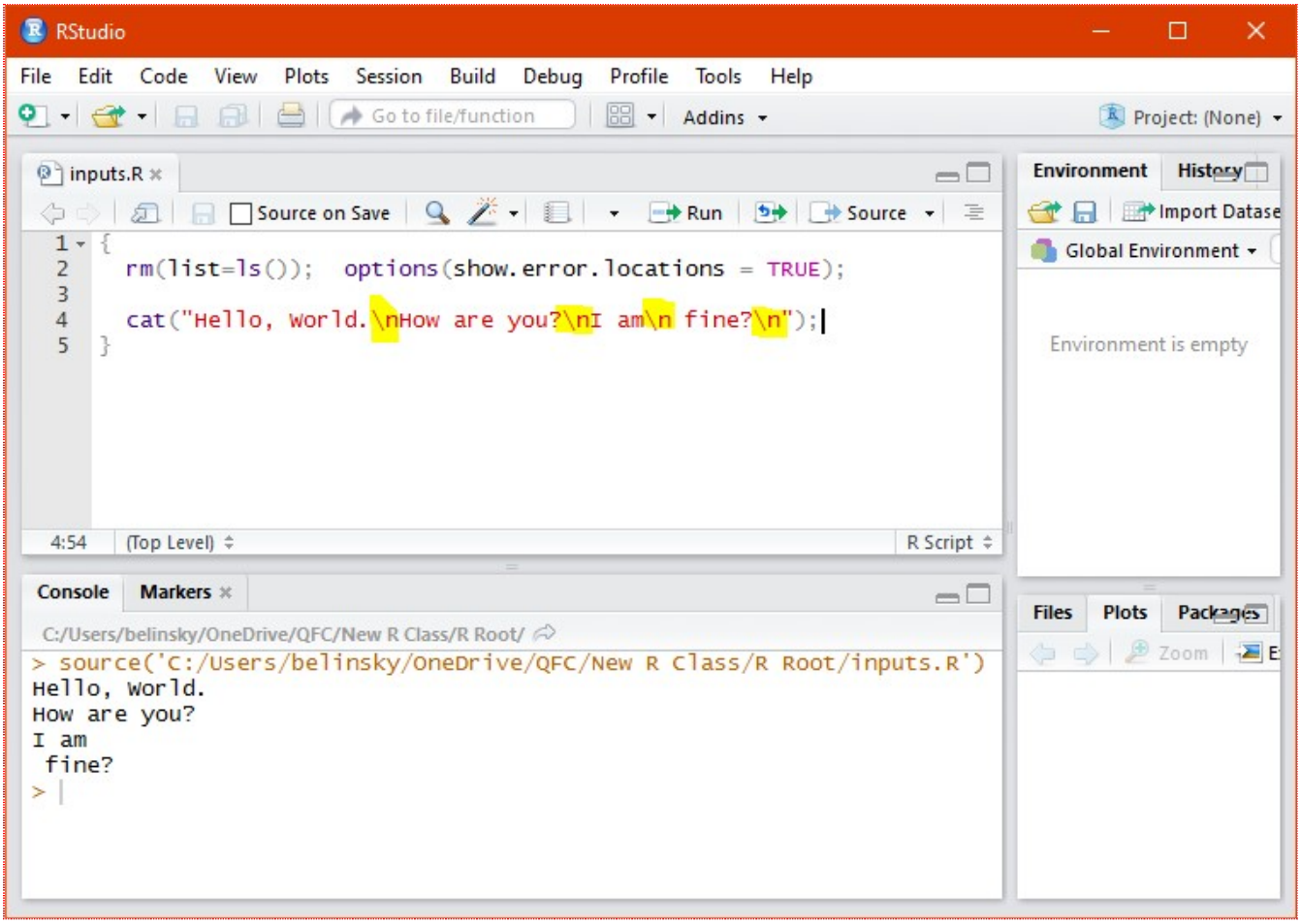

*Fig 3: Using one **cat()** statement with multiple **\n** to output multiple lines of text*

4 - Output messages with variables

If you are outputting a message to the user, you probably also want to output the value of variables from the script. The function ***cat()*** can be used to output the value of a variable by putting the variable's name in between the parentheses *without quotes*. The following example calculates ***velocity*** and outputs it to the console.

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

  distance = 100;
  time = 50;
  velocity = distance/time;
  cat(velocity);  # print(velocity); also works here
}
```

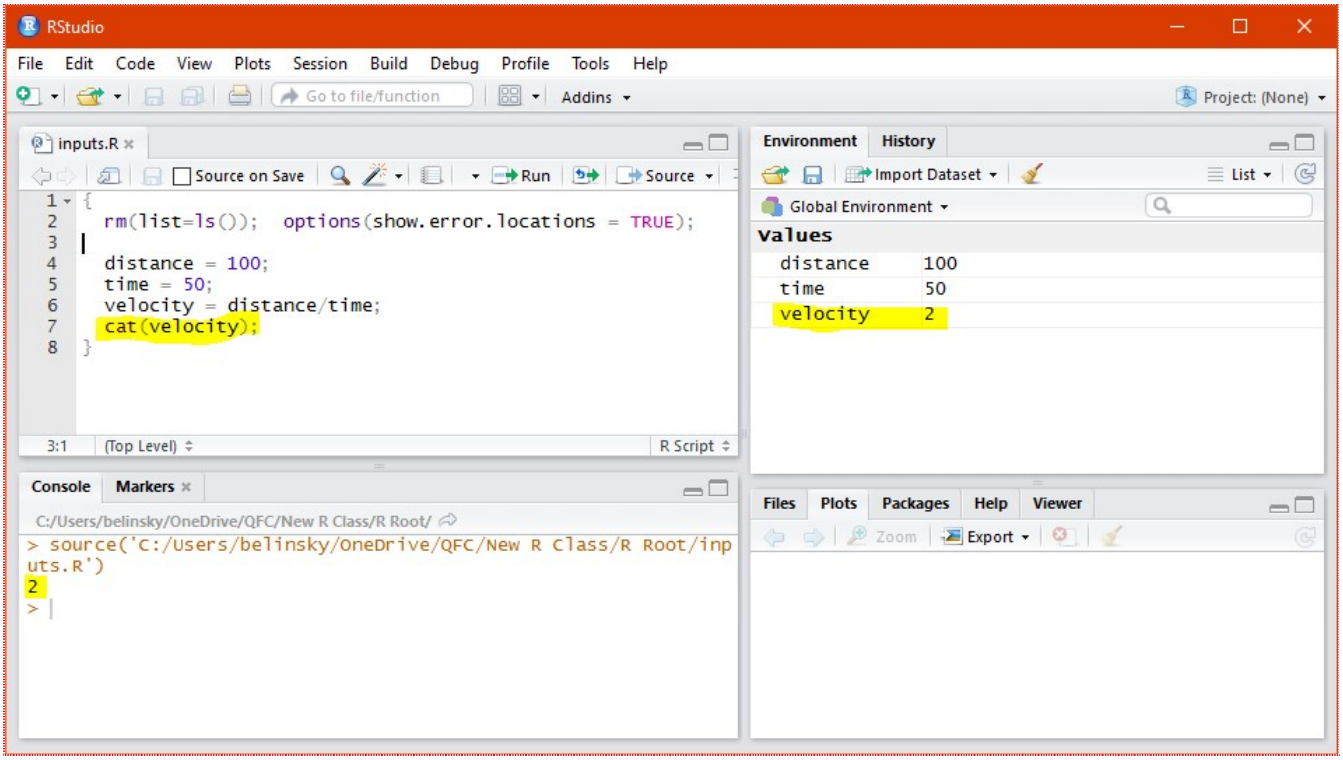

Fig 4: Outputting the value of a variable to the Console Window

The output from the script above is far from robust (*Fig.4*) as only a number (the velocity) is output -- with no other information. A more robust output would include a message that explains to the user that this value is a velocity.

5 - Outputting mixed messages and variables

In most cases you want to output both a message and a variable's value.

The following script outputs a message and a variable using multiple ***cat()*** statements:

```
1  {
2    rm(list=ls());  options(show.error.locations = TRUE);
3
4    distance = 100;
5    time = 50;
6    velocity = distance/time;
7    cat("Your velocity is: ");
8    cat(velocity);
9    cat("miles/hour");
10 }
```

Lines **7** and **9** both output a message, the content inside the double quotes, to the Console Window
Line **8** outputs the value of the variable ***velocity*** (***2)*** to the Console Window

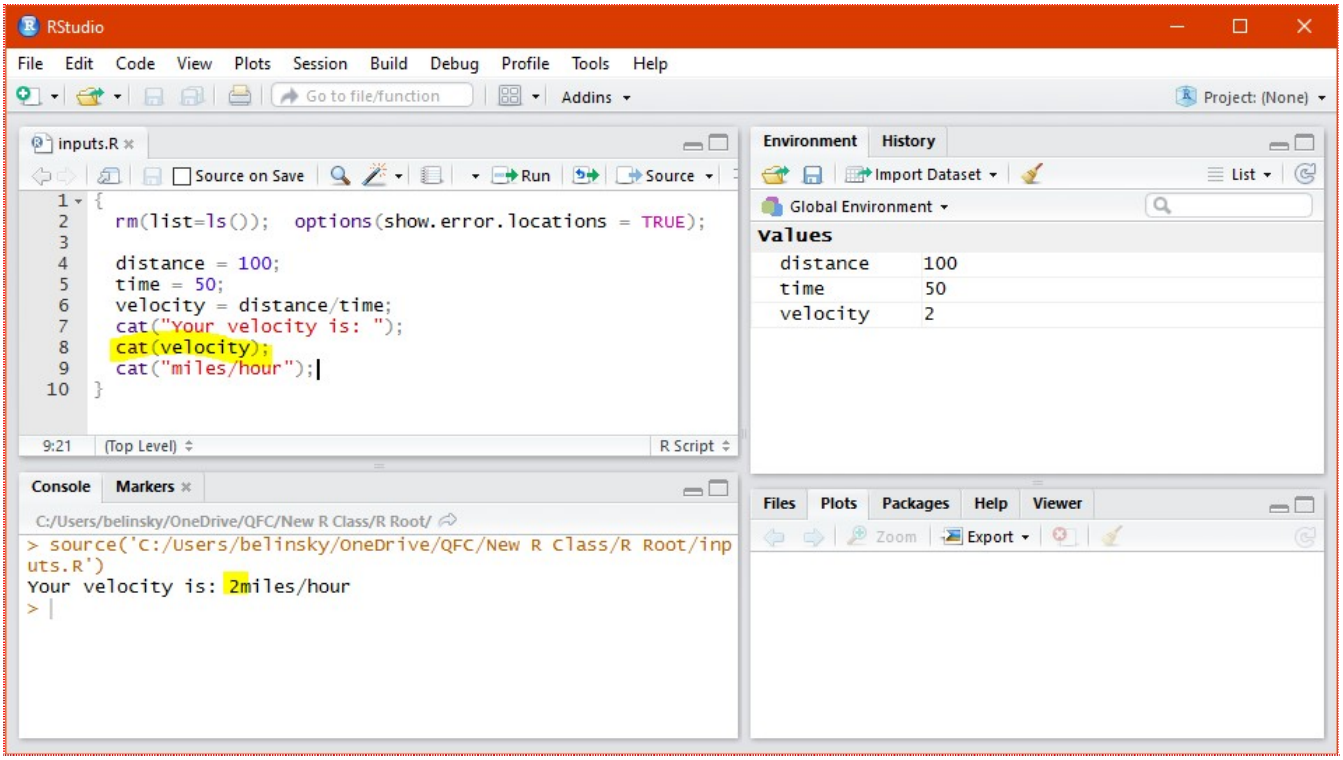

Fig 5: Output strings and variables to the Console

5.1 - Output strings and variables in one statement

In the previous example (*Fig.5*) we used ***cat()*** three times to:

1. output a message ("Your velocity is:")
2. output the value of a variable (***velocity*** -- which is ***2***)
3. output a message ("miles/hour")

We could also output all three parts using just one ***cat()*** statement.

To do this, we use a comma to separate the different parts of the output, so the following three ***cat()*** statements:

```
cat("Your velocity is: ");
cat(velocity);
cat("miles/hour");
```

can be written as one statement:

```
cat("Your velocity is: ", velocity, "miles/hour");
```

The commas effectively "stitch" together, or concatenate, the different parts of the output:

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

  distance = 100;
  time = 50;
  velocity = distance/time;
```

```
7
8   cat("Your velocity is: ", velocity, "miles/hour");
9 }
```

Note: By default, ***cat()*** put a space in the output wherever a comma is used.

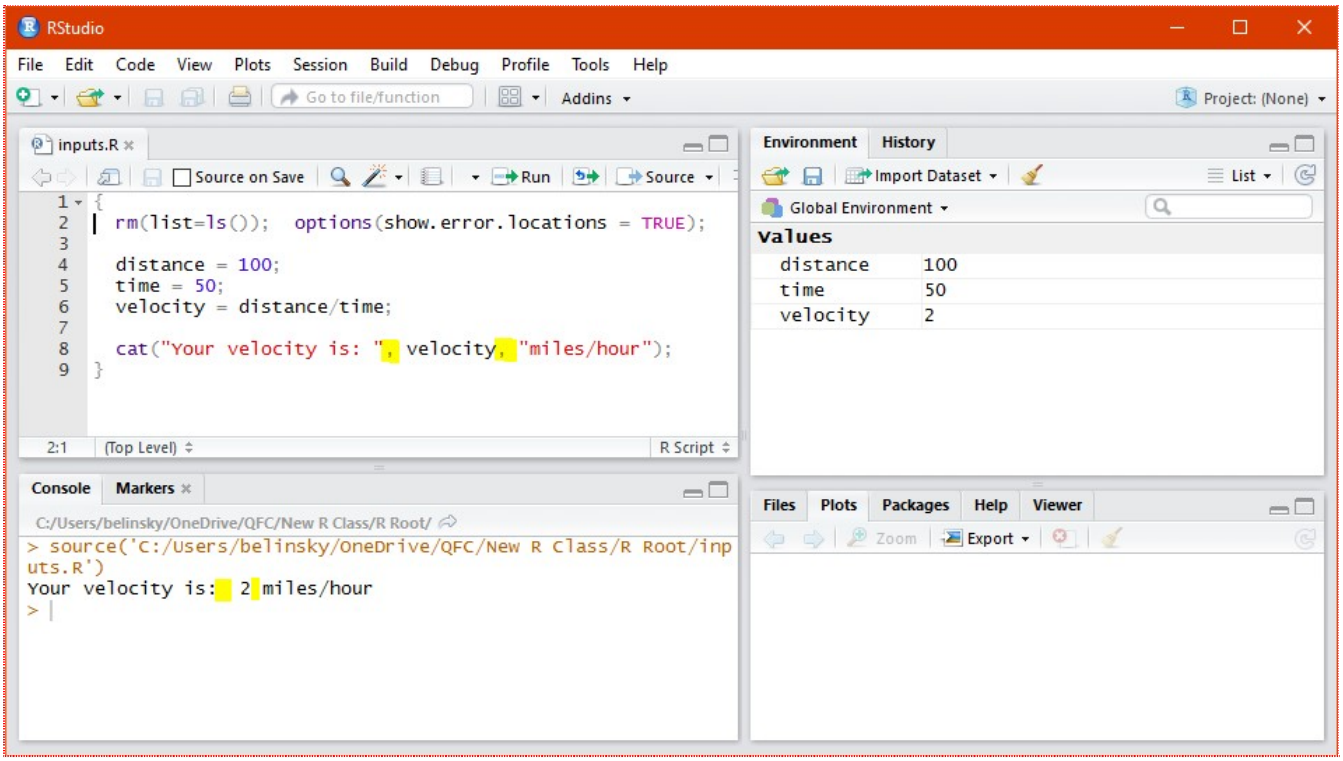

*Fig 6: Using **cat()** to output a message with mixed variables and strings*

TRAP: Misplacing Quotes.

5.2 - Output multiple strings and multiple variables using one cat() statement

You can stitch together an output with multiple variables and multiple strings using commas in ***cat()*** .

The following example outputs the value of three variables (velocity, distance, and time) in the script and gives an explanation of the values (*Fig.7*):

```
1 {
2   rm(list=ls());  options(show.error.locations = TRUE);
3
4   distance = 100;
5   time = 50;
6   velocity = distance/time;
7   cat("The values are...", "\nDistance: ", distance, "\ntime: ", time, "\nvelocity: "
      velocity);
8 }
```

Notice the use of ***\n*** to add line feeds to the output.

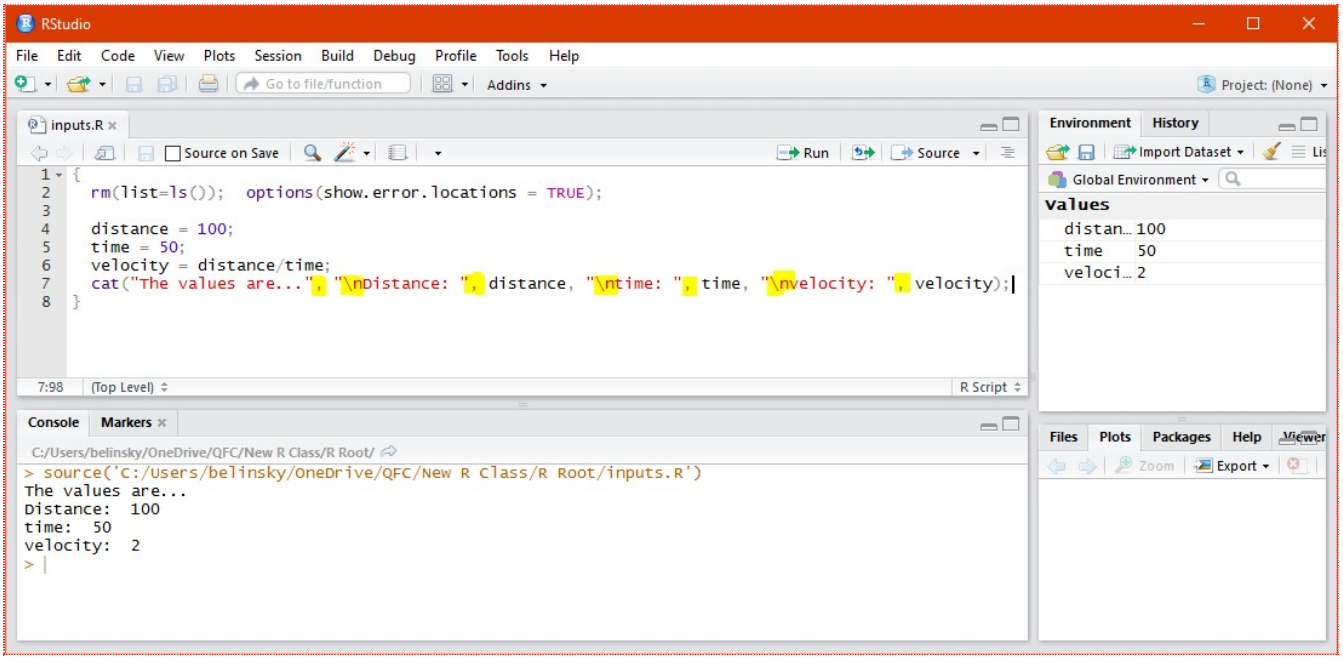

Fig 7: Multiple variables and messages strung together using commas

6 - Breaking up long lines of code

Line 6 in the code above (*Fig. 7*) gets pretty long. Sometime we want to break up a line of code into multiple lines to make it more readable.

In R you can break most lines of code into multiple line (one exception is long file-path names). You just need to be judicious about how you break up the line -- *the best places to break up a line of code are after a comma or where a space occurs*.

You can break up the ***cat()*** statement from the script above (*Fig.7*) into two lines *without changing its functionality:*

```
7 cat("The values are...", "\nDistance: ", distance,
8       "\ntime: ", time, "\nvelocity: ", velocity);
```

Notice that I only put a semicolon at the end of the second line. Just like a period ends a sentence, *a semicolon designates the end of a statement*. In this case, the ***cat()*** statement is two lines long so the semicolon goes at the end of the second line.

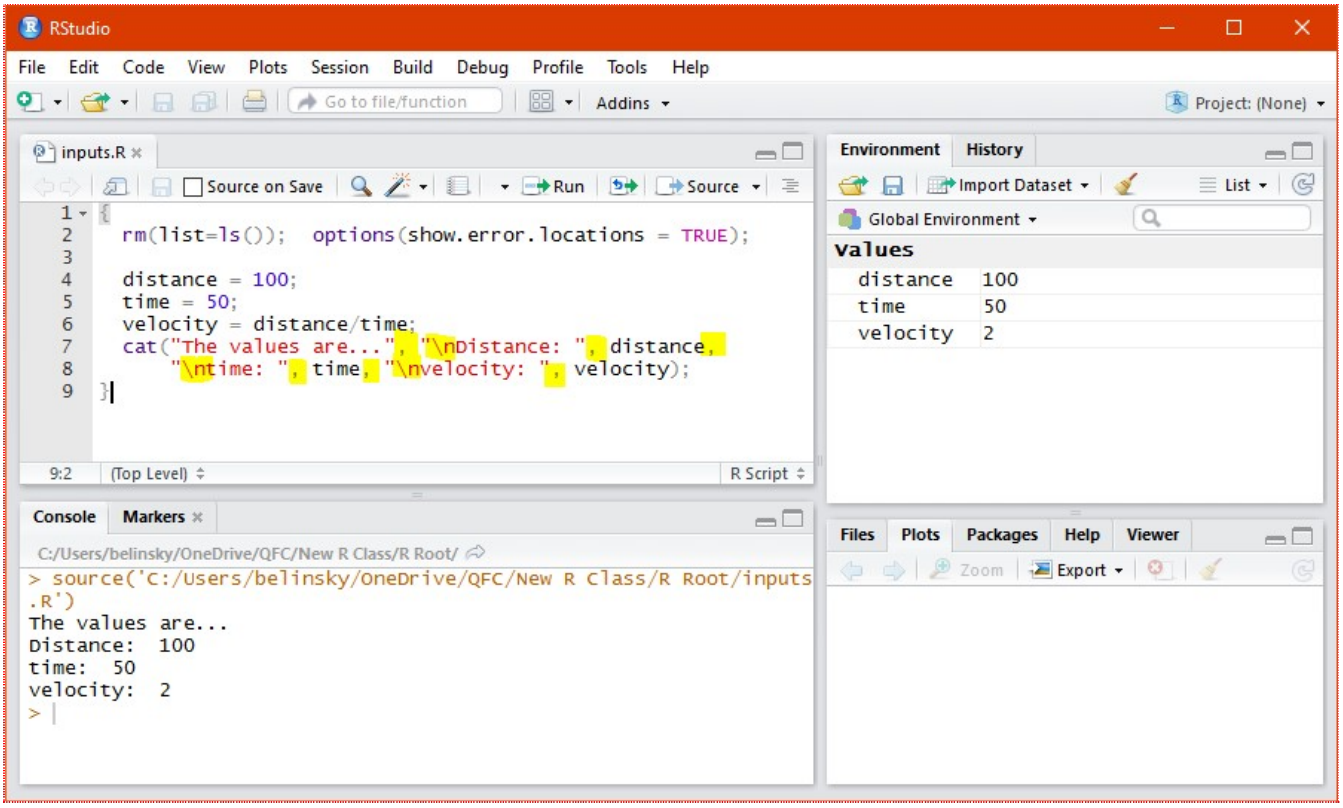

*Fig 8: **cat()** statement divided over two lines*

7 - Putting margin lines in your script

The generally accepted standard for the maximum number of characters in a line of code is 80. This is not a hard-and-fast rule but keeping your lines 80 characters or less makes it a lot easier to read your code especially on smaller montiors. You can put a margin line at 80 characters in RStudio by clicking ***Tools -> Global Options -> Code.*** On the ***Display*** tab check ***Show margin*** and set ***Margin column*** to **80.** *One of the requirements of the class project is that you keep your lines 80 characters or less.*

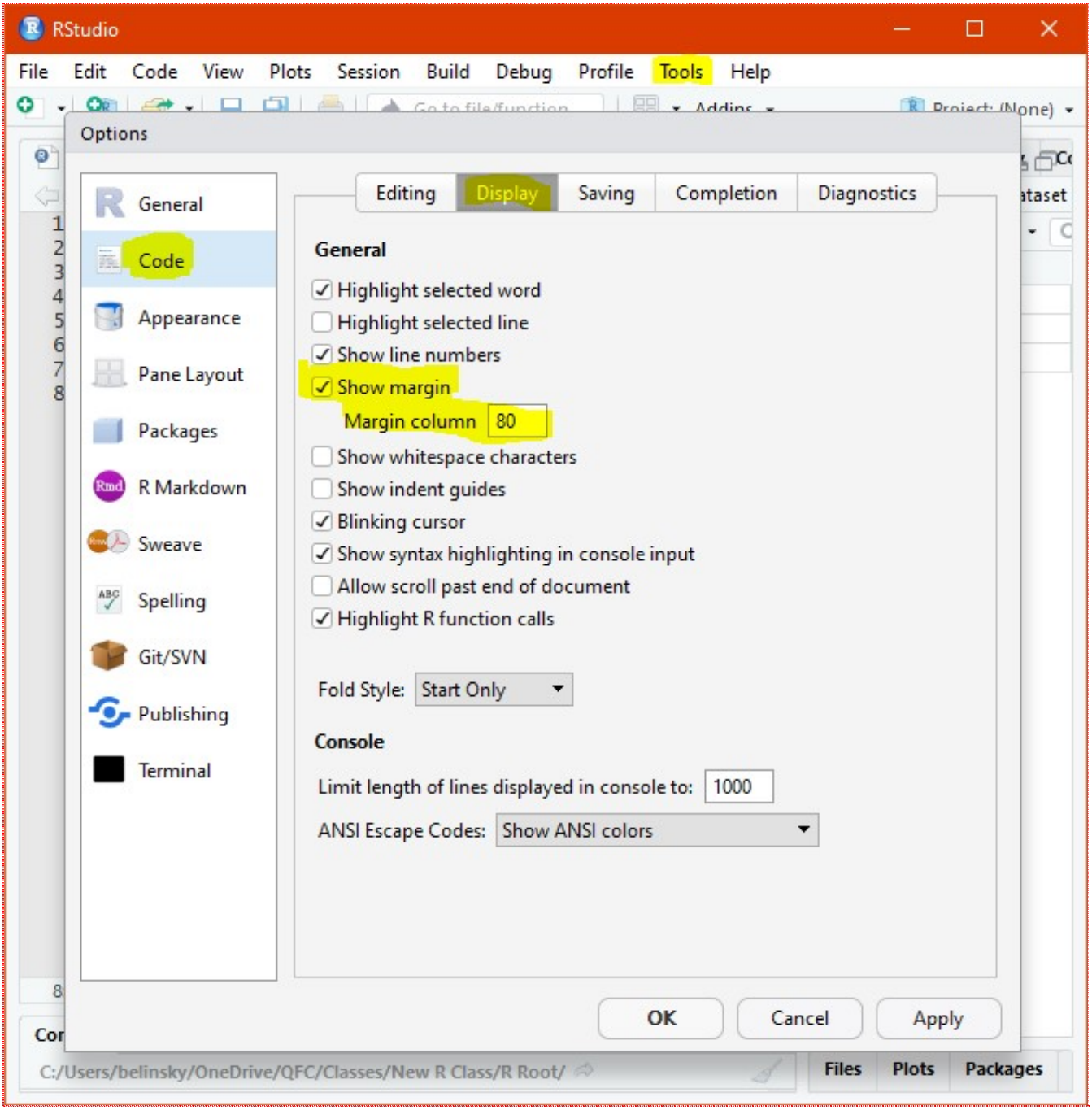

Fig 9: Setting the line margin in RStudio.

Now you will see a grey vertical line at 80 characters (*Fig.10*). The line is there as a guide -- you can still type beyond this line. There are times when it is not possible to keep a line to 80 characters -- the most common reason is a long file-path name and *file-path names cannot be broken up*.

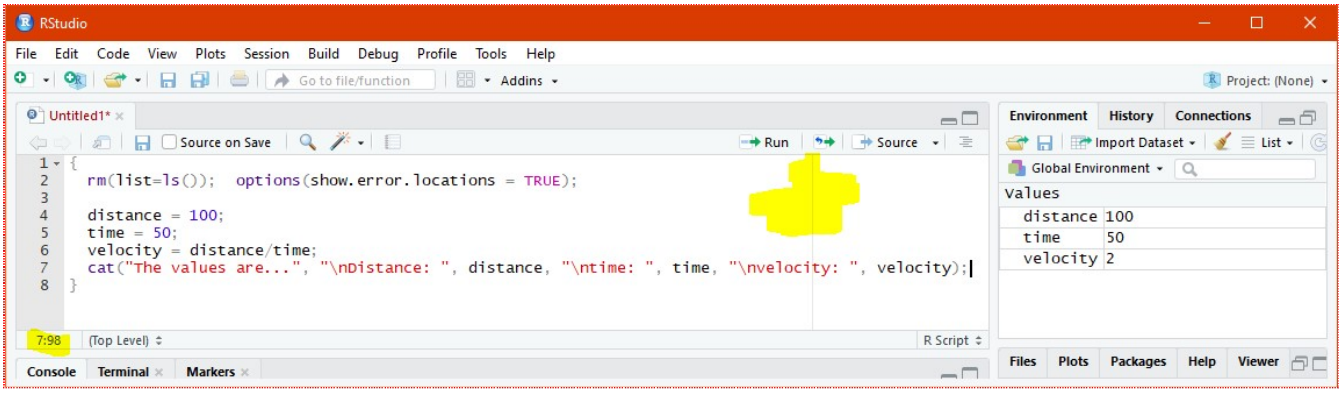

Fig 10: The grey line that acts as a margin guide.

Note: You can also see the character number the cursor is on in the lower-left corner (*Fig.10*) -- in this case, character 98.

8 - Application

If you have any questions regarding this application, feel free to email them to the instructor here.
You can attach files to the email above or send the whole Root Folder as a zipped file. Instructions for zipping the Root Folder are here.

A) Just like the last lesson, get 6 inputs from the user, which are:
1) Five temperature measurements for a given location
2) The name of the location

B) Have the script calculate the mean of the 5 temperature measurements

C) Using ***cat()***, output to the screen a message that gives:
1) On line 1: The location the temperature measurements are from (e.g., Lansing)
2) On line 2: The five temperature measurements (and a message that says these are the measurements)
3) On line 3: The mean temperature (and a message that says this is the mean)

So the full message in the Console Window should look something like this:
The temperatures are from: [location].
The temperature measurements are [temp1], [temp2], [temp3], [temp4], and [temp5].
The mean temperature is [meanTemp].

Don't forget to output the commas in between the temperature values and the other punctuation.
Where the values in brackets ***[ ]*** represent the values of the variables in your script

*Save you script file as **app1-6.r** in the **scripts** folder of your RStudio Project for the class.*

9 - Extension: The backslash (\ )

The backslash in programming is used in a string as an *escape character.* This means that the character after the backslash is treated in some special manner -- similar to the way the **Shift**, **Control** or **Alt** keys on your keyboard change the meaning of the character that is pressed with them.

Backslash can be used to

- put in a line feed (**\n**)
- print to the Console a double quote -- in other words, treat the double quote as a character, not as the start or end of a message
- print our special characters not on the keyboard. Here is a link to a partial list of special characters.

```
1 cat("And the boy said, \"Hello\" \n");
2 cat("To print to console a backslash, use \\ \n");
3 cat("Some hex characters: \xB1 \xC6 \20AC");
```

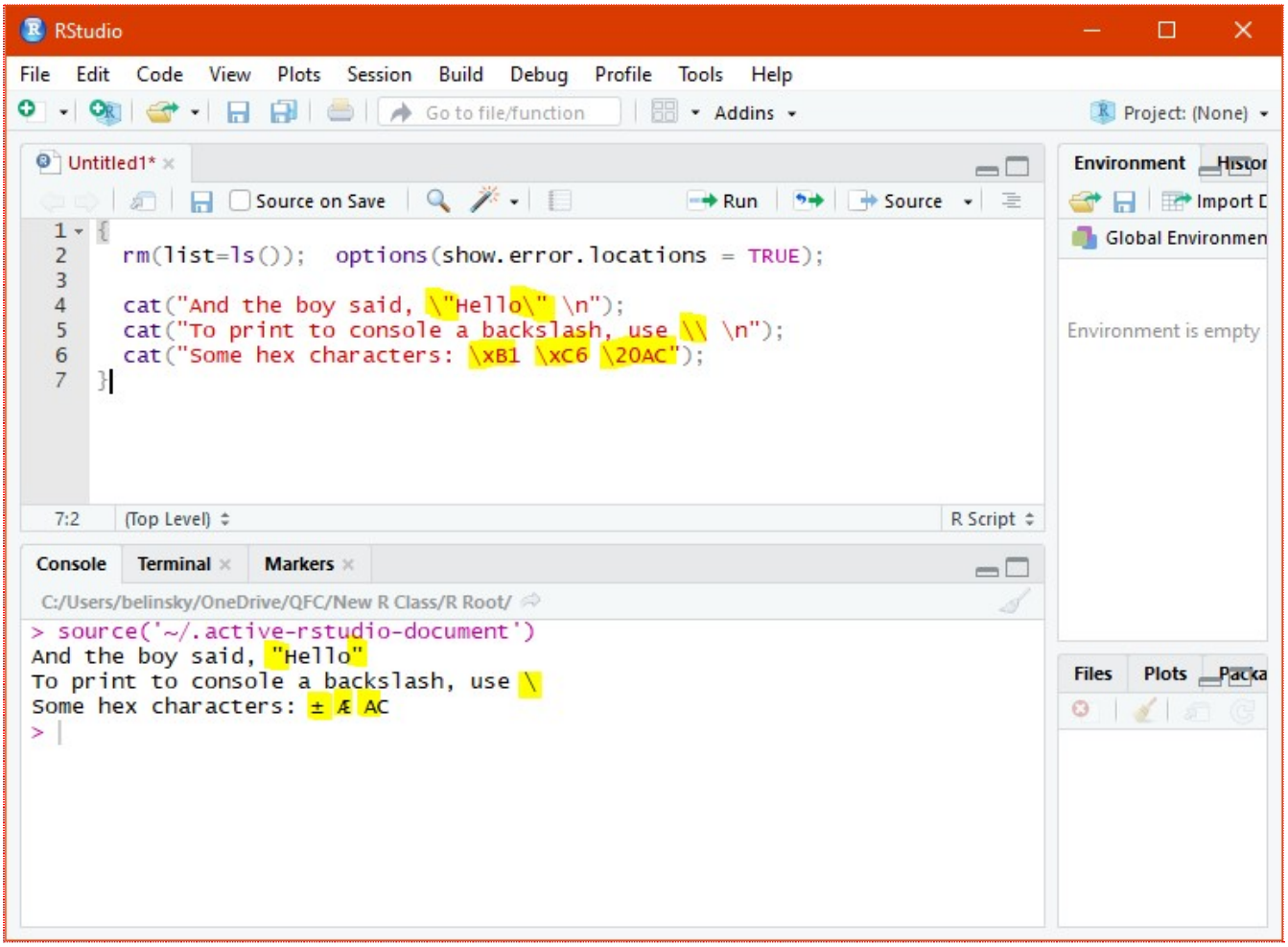

Fig 11: Using backslash (\ ) to output special characters.

10 - Trap: Misplacing (or forgetting) quotes

It is easy to accidentally forget a quote, add a quote, or simply move a quote to a place it does not belong. The following code has a missing quote on line 7 right after "*The values are...*

```
1 {
2   rm(list=ls());  options(show.error.locations = TRUE);
3
4   distance = 100;
```

```
  time = 50;
  velocity = distance/time;

  cat("The values are..., "\nDistance: ", distance, "\ntime: ",
      time, "\nvelocity: ", velocity);
}
```

The execution of the code causes an "*Unexpected Symbol*" error that is not too helpful. However, RStudio has a built in feature to help you find misplaced quotes -- the color scheme. Quoted text is in a different color than other parts of the code. For example, in the XCode color scheme -- text in quotes is colored red.

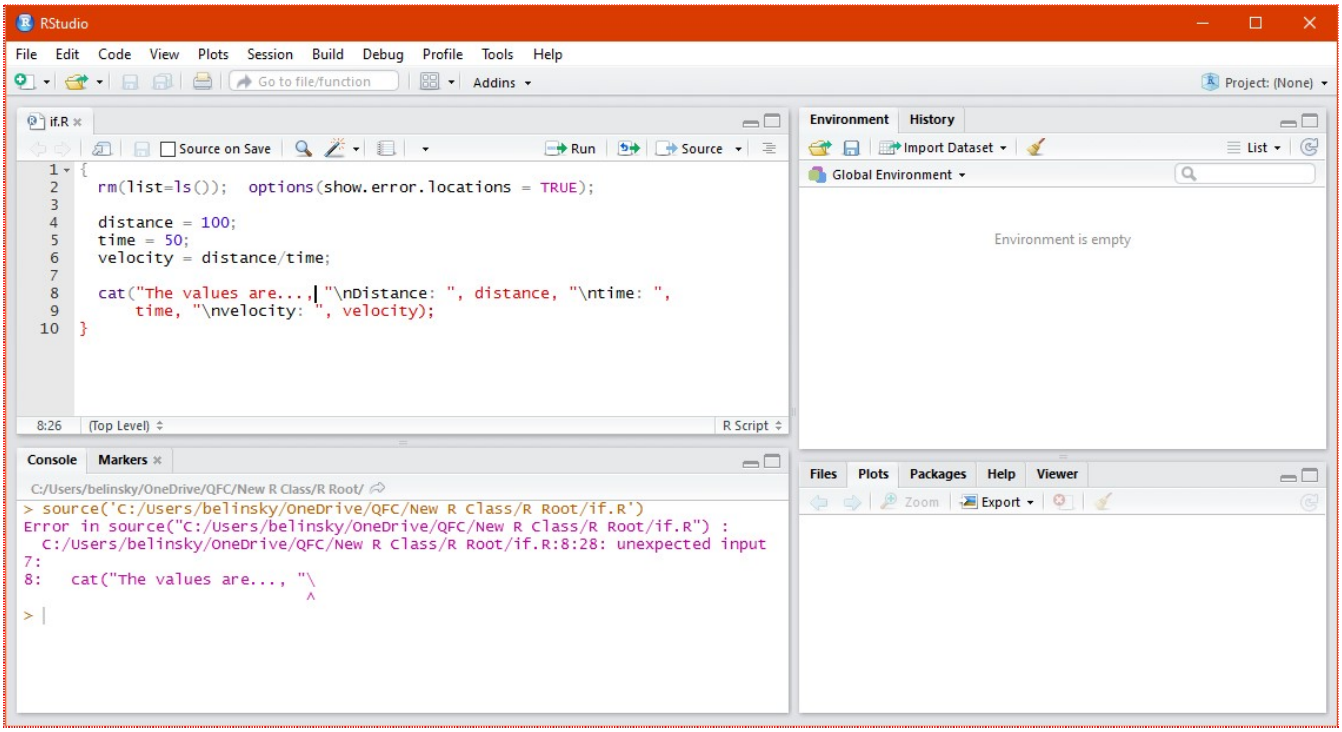

Fig 12: Missing quote

(*Fig. 12*) shows that, right after "The value are... , the variables are all treated as if they are in quotes and the messages are outside the quotes. Since one quote was missing, the error propagated and assumed characters to be variables/numbers and variables/numbers to be characters.

We will add the missing quote to line 7:

```
cat("The values are..._", "\nDistance: ", distance, "\ntime: ",
    time, "\nvelocity: ", velocity);
```

Now the correct components of the output are colored red and the script executes without an error:

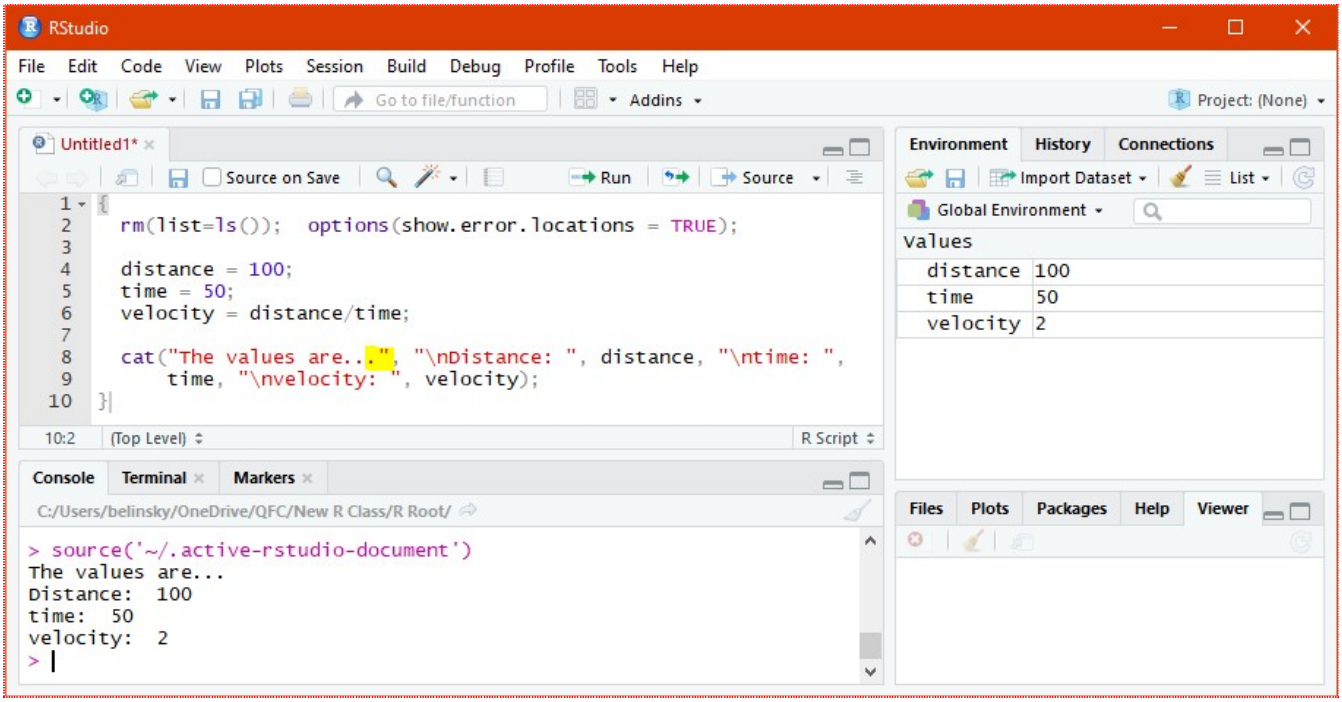

Fig 13: Missing quote added

11 - Extension: The concatenate, cat(), function

Let's switch to the "Help" tab in the Plot Window and type "cat" in the search bar:

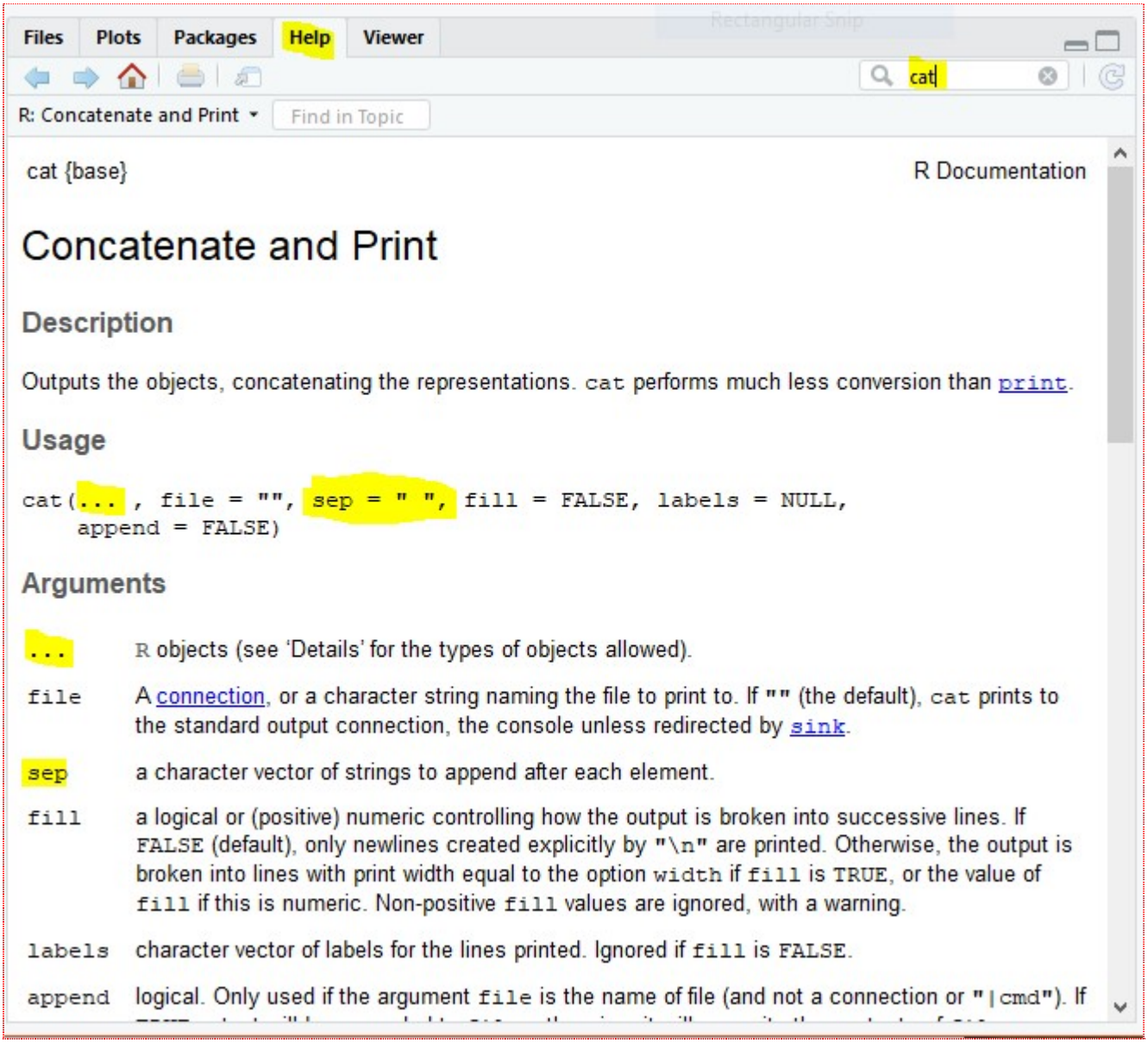

*Fig 14: Information about the **cat()** function in the Help Window.*

The first part of ***cat()*** is the three dots "**...**". The three dots say that ***cat()*** *will accept any number of objects* as inputs that it will tun into outputs. The objects can be messages in quotes, string variables, numeric variable, or other objects we have not discussed yet like vectors.

After the three dots, ***cat()*** has **5** parameters (or arguments): ***file***, ***sep***, ***fill***, ***labels***, and ***append.***

Parameters are used to modify the behavior of a function. We are going to change the ***sep*** parameter. ***sep*** *changes how the different components of the output are separated*. The default is to put a single space between the components of the output.

Let's take the code from above and add the ***sep*** parameter to ***cat()*** and give ***sep*** the value " ** ":

```
{
  rm(list=ls());  options(show.error.locations = TRUE);
  distance = 100;
  time = 50;
```

```
  velocity = distance/time;
  cat("The values are...", "\nDistance: ", distance, "\ntime: ",
      time, "\nvelocity: ", velocity, sep=" ** ");
}
```

And you will notice that the different component are separated by " ** ".

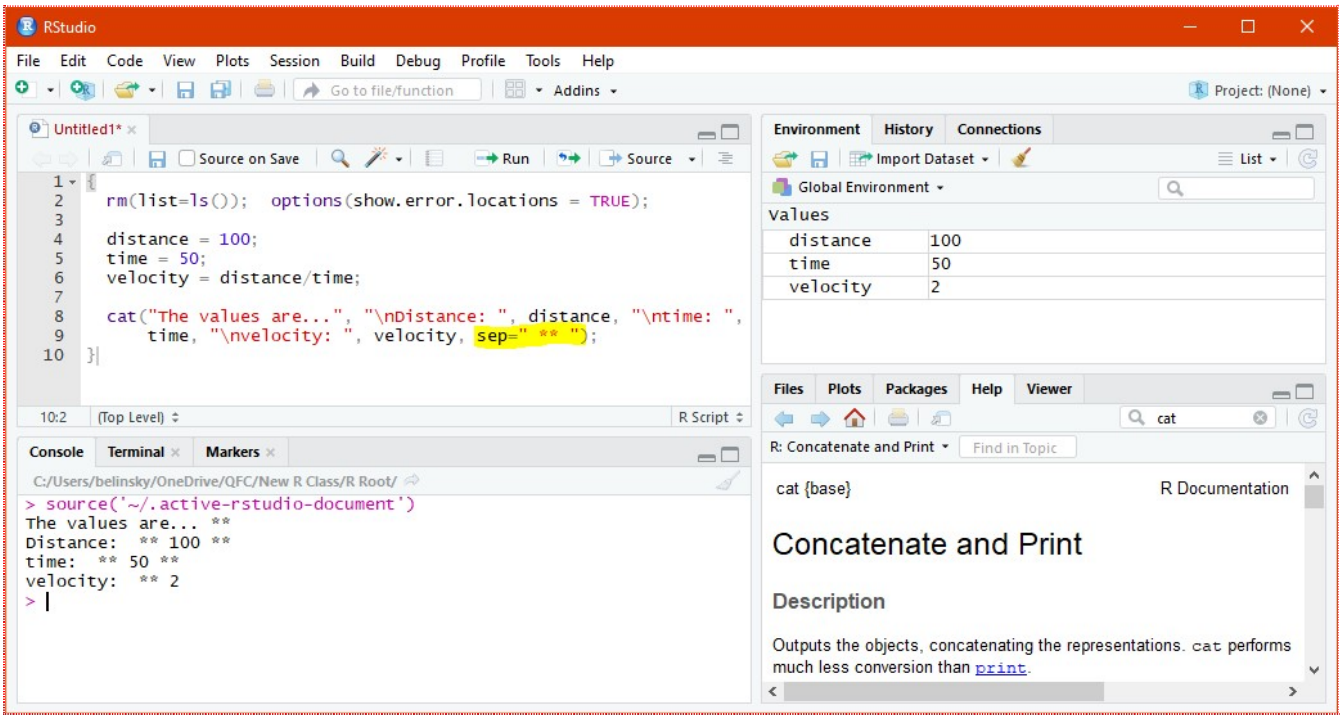

*Fig 15: Using the **sep** parameter in **cat()***