01-05: Strings and Inputs

1 - Purpose

- Introduce string variable
- Discuss differences between numeric and string variables
- Assign input from the user to a variable.
- Convert input values to a numeric value.

2 - Concepts

3 - String variable types

Up to this point, we have only used numeric variables (e.g., fish's height or weight, a person's speed or kinetic energy). The main attribute of numeric variables is that mathematical operations can be performed on them. However, there are many variables that contain values that are not numeric.

In statistics, these variables are often called *categorical variables* and could include the location where a fish was caught, the gender of a fish, or the species of fish. In scripts, these are called *string variables* and the values assigned to them are put in quotes:

```
fishLocation = "SouthPort"
fishSpecies = "Guppy"
fishGender = "Male"
```

Extension: Strings vs. characters

3.1 - Common errors when dealing with string variables:

Probably the two most common errors when using strings in R is:
1) Using mathematics operators on the string
2) Forgetting the quotes

3.1.1 - Using mathematics operators

```
{
  # moved these two lines to one line -- still runs the same
  rm(list=ls()); options(show.error.locations = TRUE);

  fishSpecies = "Guppy";
  totalValue = fishSpecies + 10; # error
}
```

The above script tells R to store the string value "Guppy" in the variable named ***fishSpecies***. Since R knows ***fishSpecies*** is a string, R does not permit the use of mathematics operations and will give an error. The error given is: "*non-numeric argument to binary operator*" (*Fig 1*), an awkward sounding error that

means you need to use numbers if you are going to perform a numeric (binary) operation.

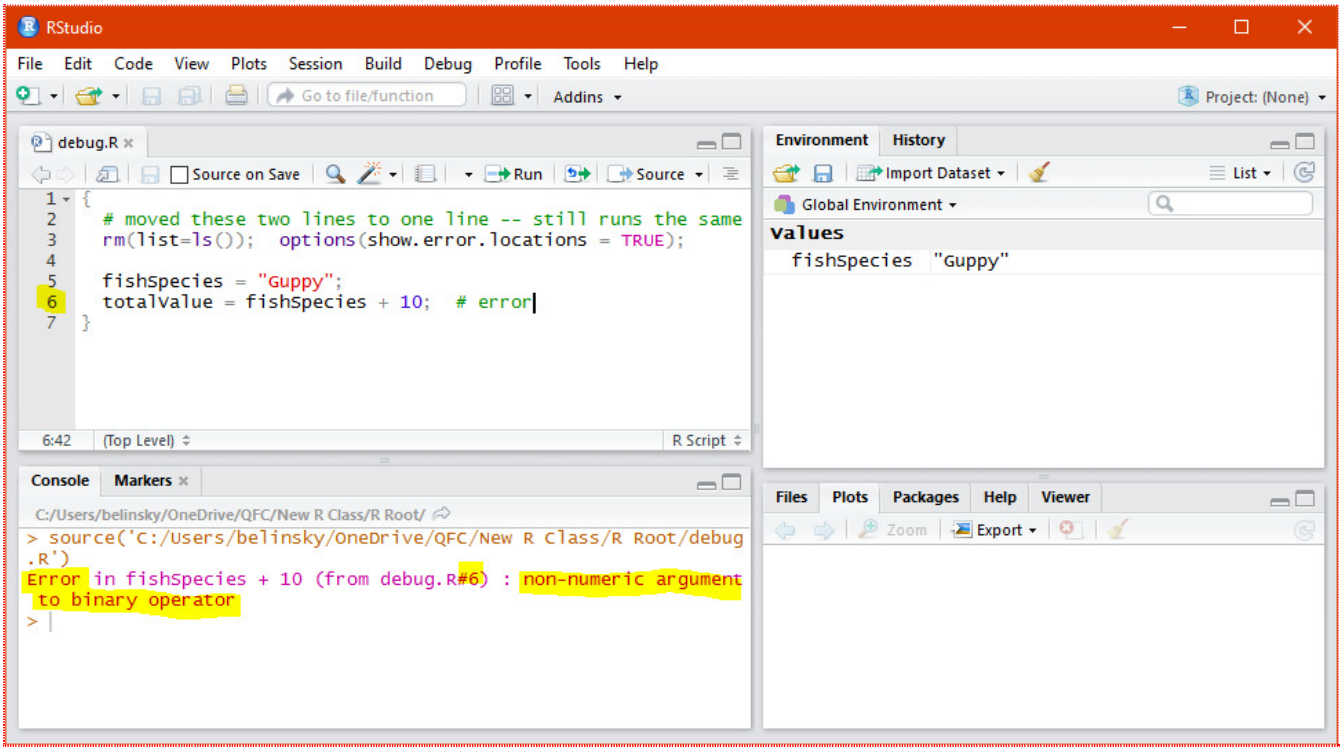

Fig 1: Using a mathematics operations on a string -- error

3.1.2 - Forgetting the quotes

If you forget to put quotes around the value assigned to the string variable -- for example:

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls()); options(show.error.locations = TRUE);
4
5   fishSpecies = Guppy;    # error -- need quotes
6 }
```

You will get the error: "object 'Guppy' not found" (*Fig 2*).

This is because R thinks that ***Guppy*** is a variable name and the variable named ***Guppy*** does not exist in this script.

Another way to look at line 5 is that line 5 is attempting to set the value of the variable named ***fishSpecies*** to the value of the variable named ***Guppy*** but the variable named ***Guppy*** does not exist.

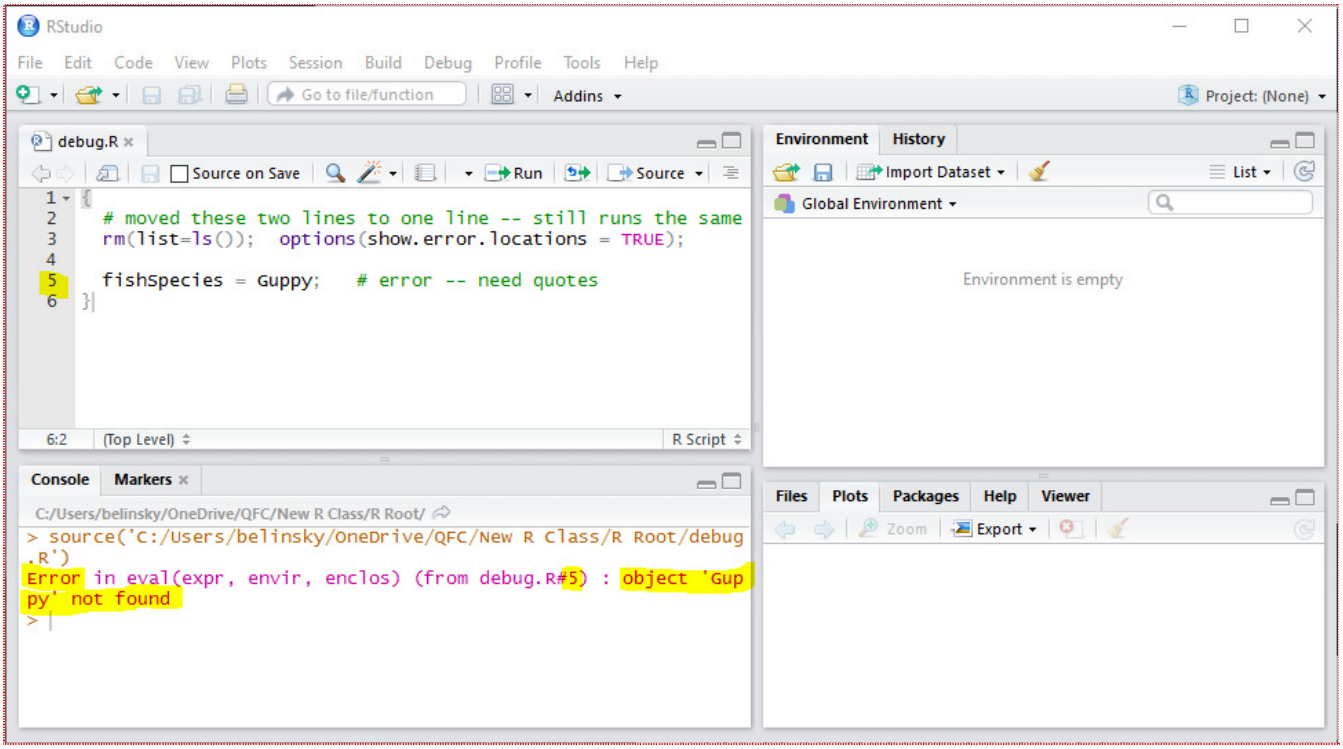

Fig 2: Forgetting quotes around the value of a string variable

4 - Inputs

Once again, most variables used in a script are not hardcoded in to the script. Many variables are input from an external source like a csv file or a database. We will talk about file inputs in a later lesson. For now we are going to get input values from a simpler source -- the person running your script.

The code to get input from the user is: ***readline()*** -- and inside the parenthesis is the message you want to display in the Console Window for the user. *Extension: functions (and parameters)*

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls()); options(show.error.locations = TRUE);
4
5   readline("What is your name? ");
6 }
```

When you execute the script, the text "What's your name? " appears in the Console Window along with a prompt cursor that is waiting for input from the user (*Fig 3*).

Click inside the Console Window and type your name at the prompt and hit enter.

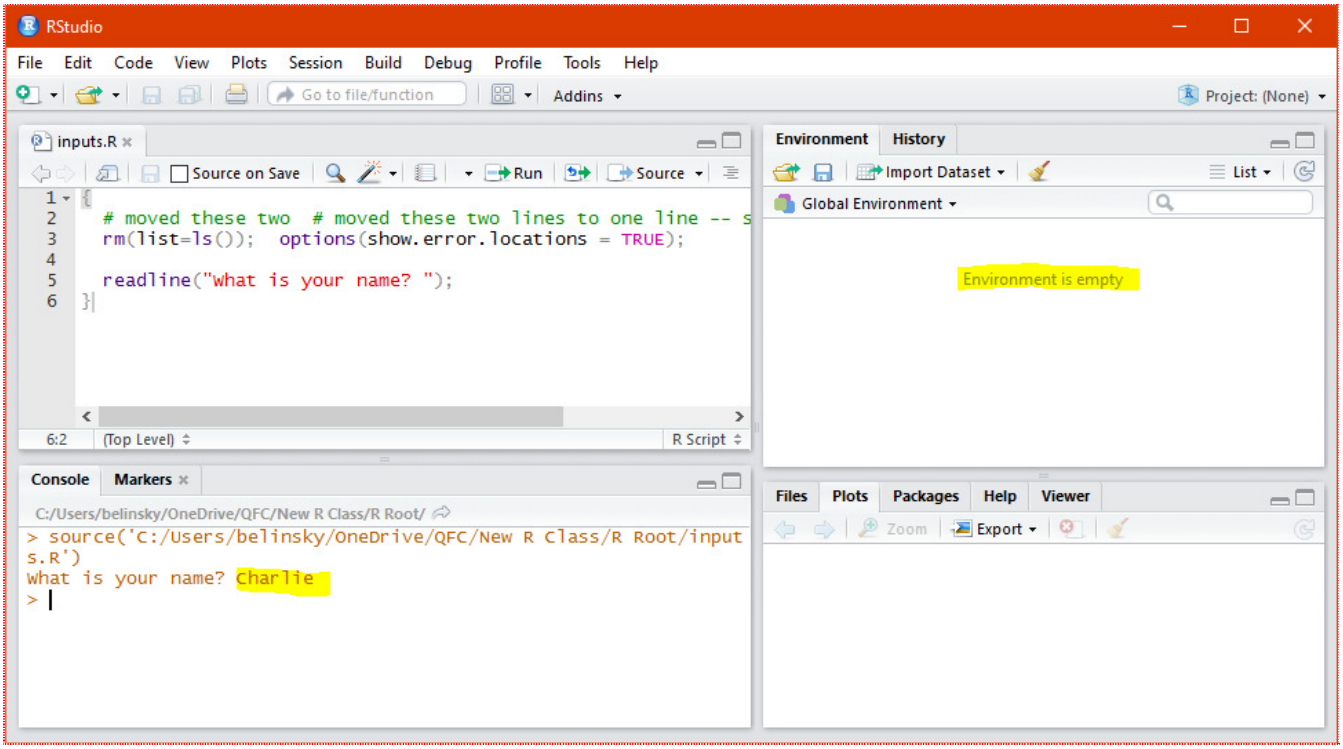

Fig 3: Prompting input from the user.

After you your name and hit enter, the script finishes executing but the Environment Window is empty (*Fig 3*)and the Environment Window holds all the information stored by the script. In other words, the script did not remember the value typed in by the user.

4.1 - Remembering (storing) input values

In programming, values are only remembered if you specifically instruct the script to remember the value -- and this is done by assigning the value to a variable. The following script assigns the value typed in by the user to the variable named ***yourName***.

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls());  options(show.error.locations = TRUE);
4
5   yourName = readline("What's your name? ");
6 }
```

When your execute the script, the variable named ***yourName*** with the value you entered is placed in the Environment Window (*Fig 4*).

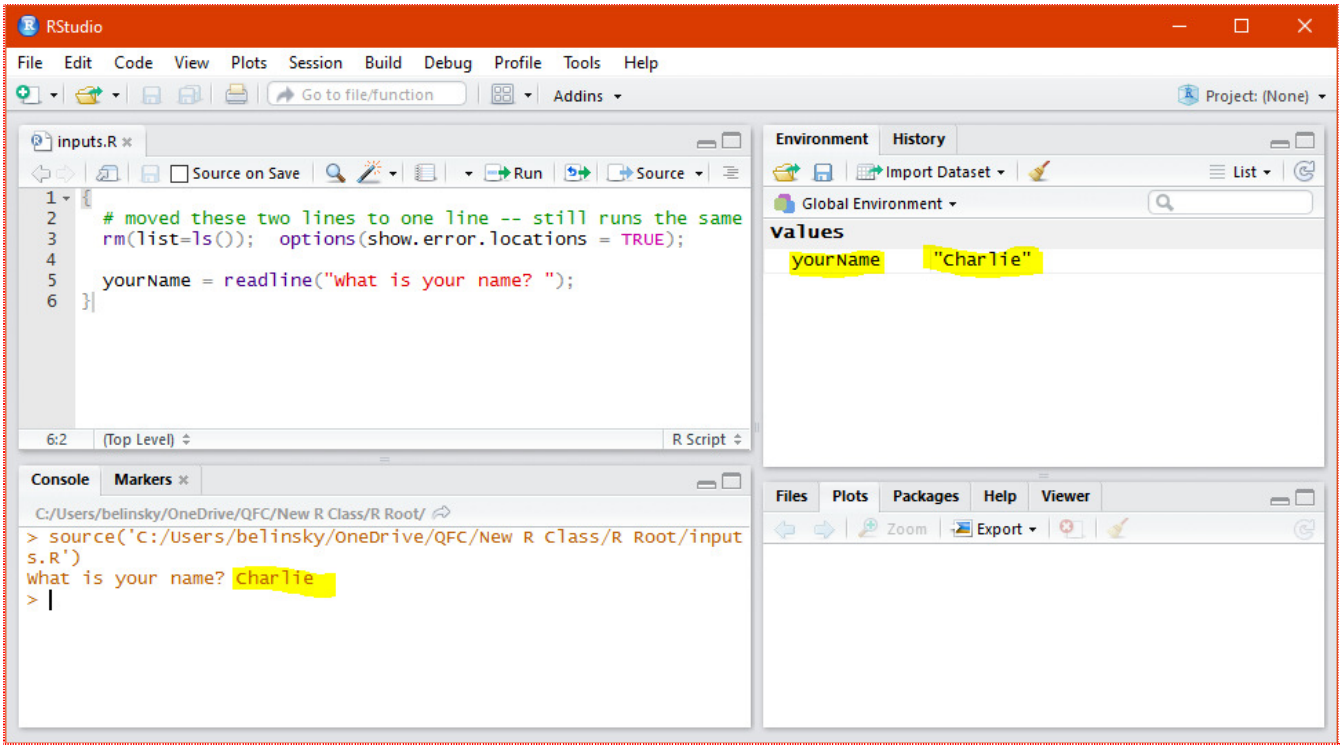

Fig 4: Saving input value to a variable

Extension: Inputs act like breakpoints

Notice that the value for ***yourName*** is put in quotes. This is because ***yourName*** is a variable with a string value.

4.2 - Storing a string input value

Let's add a second prompt for more information from the user -- this time asking for height, a numeric value.

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls());  options(show.error.locations = TRUE);
4
5   yourName = readline("What's your name? ");
6   yourHeight = readline("What is your height in inches? ");
7 }
```

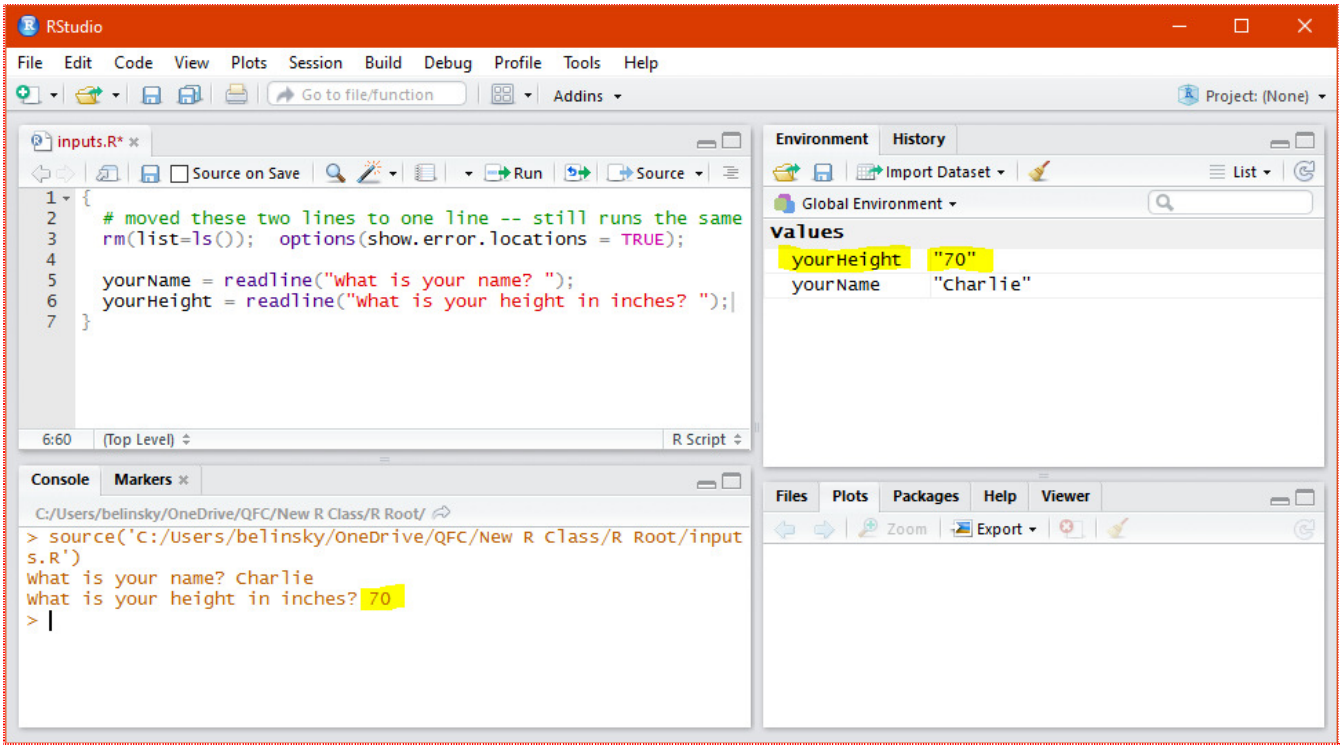

Fig 5: Numeric input still treated by R as a string

The script prompts you twice and stores the values the user types in ***yourName*** and ***yourHeight*** respectively but the height also has quotes around it (*Fig 5*) -- ***yourHeight*** has the value ***"70"***.

R always assumes that the input is coming in from the user is a string -- R sees ***"70"*** as the character ***"7"*** followed by the character ***"0"***. In other words a variable with a value ***"70"*** is a string whose characters all happen to be numbers. Whereas a variable with a value ***70*** is a number. This mean you cannot perform mathematical operations on the variable ***yourHeight*** even though it looks like a number.

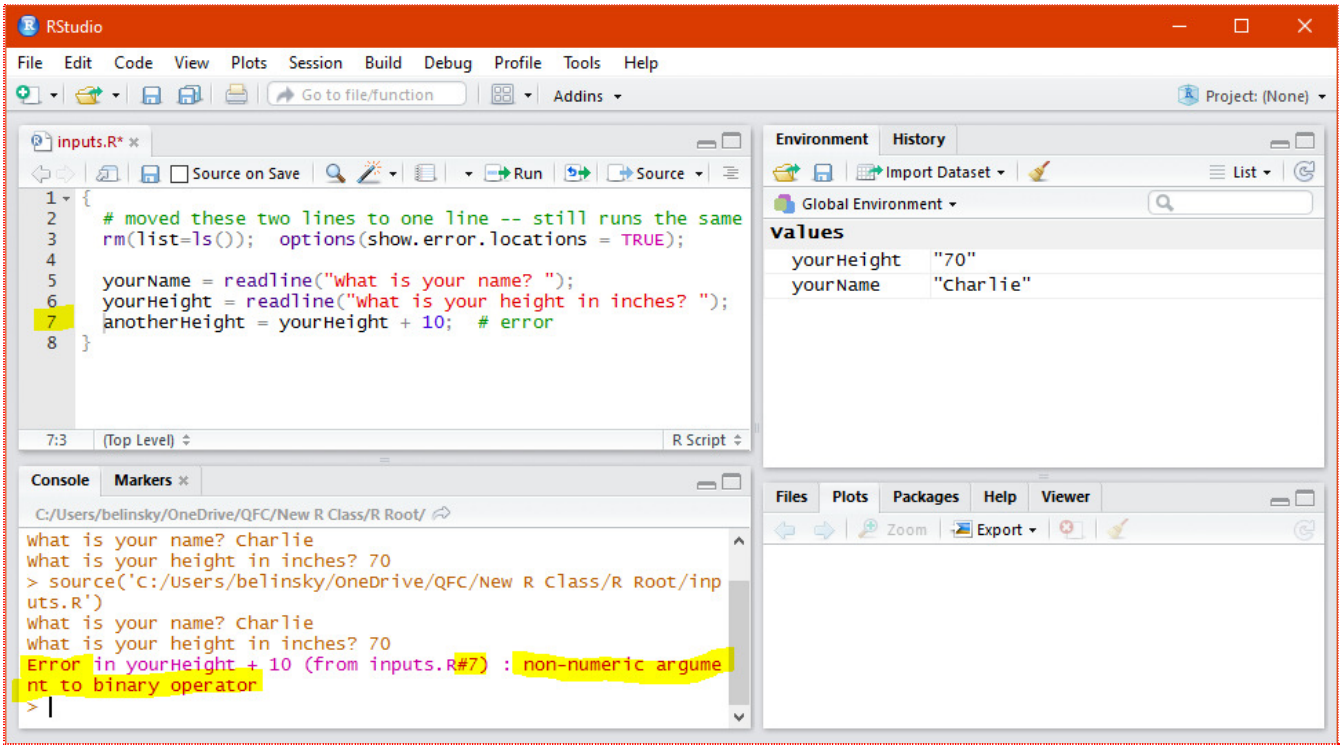

Fig 6: Error performing a mathematics operation on a string -- even if it looks like a number

Trap: Doing a mathematical operation on the string value

4.3 - Storing a numeric input value

If you want R to treat a variable as a number instead of a string, you need to instruct R to treat the variable as a number. This is done by using the ***as.numeric()*** function on line 6 below.

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls());  options(show.error.locations = TRUE);
4
5   yourName = readline("What's your name? ");
6   yourHeight = readline("What is your height in inches? ");
7   yourHeight = as.numeric(yourHeight);
8 }
```

Line 7 first finds the value of the statement ***as.numeric(yourHeight)***, which is the number ***70***, and then the assignment operator on line 6 tells R to save the value ***70*** to the variable ***yourHeight***.

So, line 7 changes the value of ***yourHeight*** from ***"70"*** to ***70***.

yourHeight now has the numeric value ***70*** (*Fig 7*) and you can now perform mathematical operations on it.

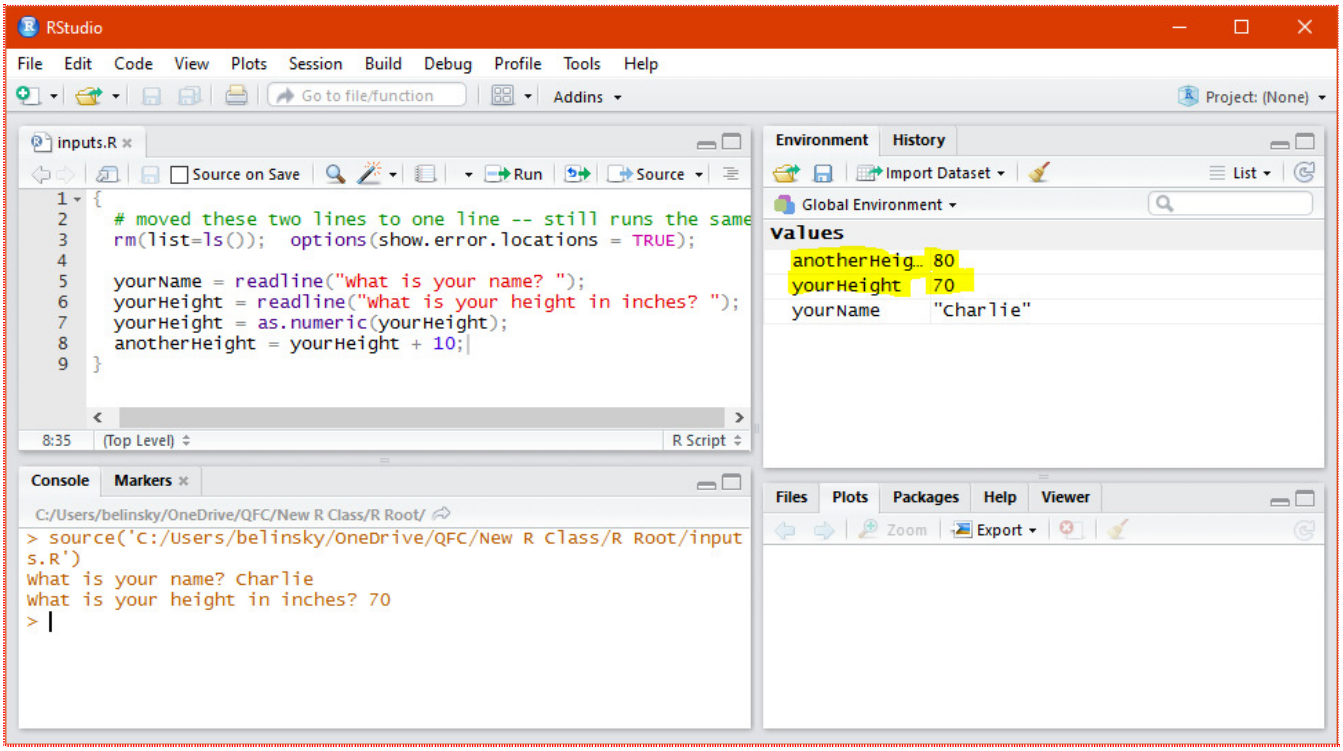

Fig 7: String converted to number and used in a mathematical operation

5 - Application

Create a script that finds the mean of five temperature measurements from one location -- the five

measurements and the location are given by the user (so the user inputs 6 values).

1) The script will ask the user for the location of the temperature measurements (e.g., Lansing, New York...).
2) The script will ask the user for the five temperature measurements.
3) The script will calculate the mean value of the five temperature measurements and save it as a variable.

6 - Extension: Inputs act like breakpoints

When a script asks for an input from the user, the script is stopped at that line of code until it gets the input. So the following script will be stopped at line 4 until the user enters in their name and again at line 5 until the user enters in their height. The input acts like a breakpoint in the script.

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls());  options(show.error.locations = TRUE);
4
5   yourName = readline("What's your name? ");
6   yourHeight = readline("What is your height in inches? ");
7 }
```

This can cause some interesting issues if a person tries to (re)*Source* the script while the script is waiting for an input as highlight in the following figure (*Fig 8*).

7 - Trap: Sourcing script while waiting for input

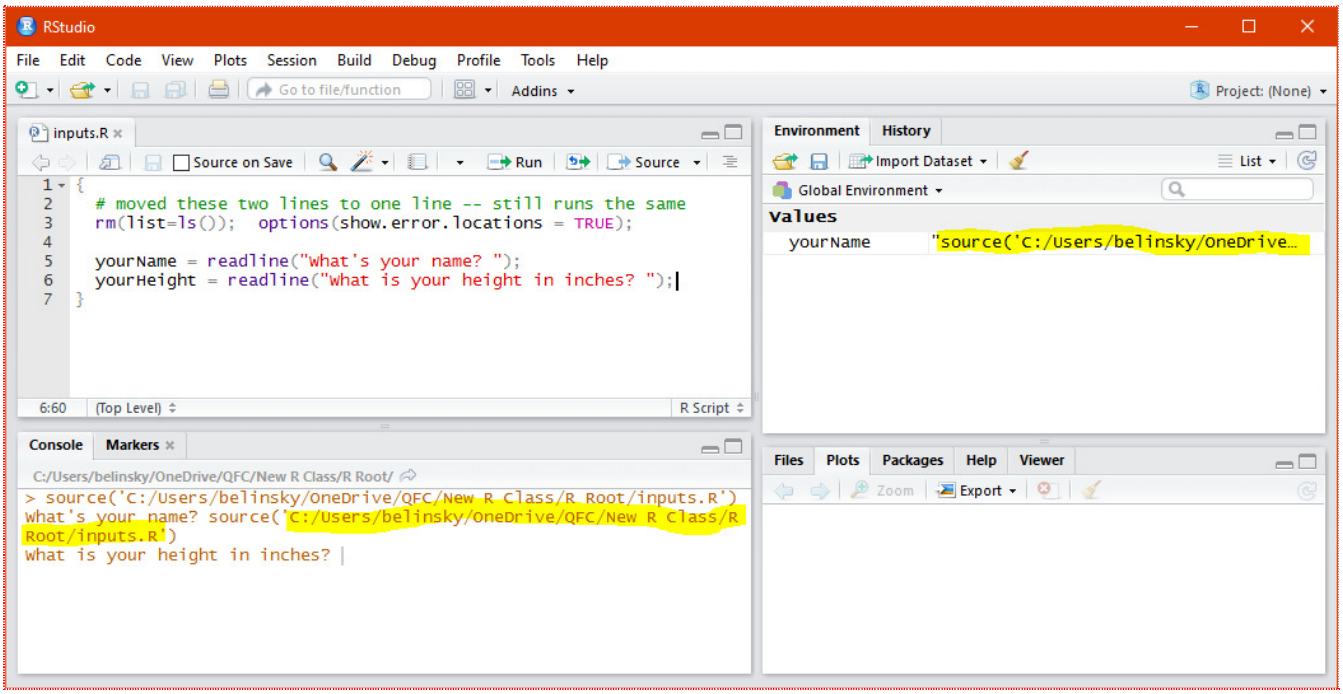

Fig 8: What happens when you hit source when R is expecting a user input

What happens here was:

1) The user clicked *Source*
2) The user answered "*Charlie*" to the first question
3) The user clicked *Source* again while the script was waiting for the second question to be answered.

The script was still running when the user asked to execute the script again. So, the original script was still waiting for an input. The input the original script received was the *Console Windows* description of the action the user just implemented -- the second source command: *source('~/.active-rstudio-document')*, which the script mistook for an input to the second question and assigned it to the variable *yourHeight*.

Basically, the message here is be careful and try not to re-source a script that is already running.

8 - Trap: Mathematical Operations on a string

Let's try to do a mathematical operation on ***yourHeight*** (line 7), a string value that only has numbers as characters:

```
1 {
2   # moved these two lines to one line -- still runs the same
3   rm(list=ls()); options(show.error.locations = TRUE);
4
5   yourName = readline("What is your name? ");
6   yourHeight = readline("What is your height in inches? ");
7   myHeight = yourHeight + 10;
8 }
```

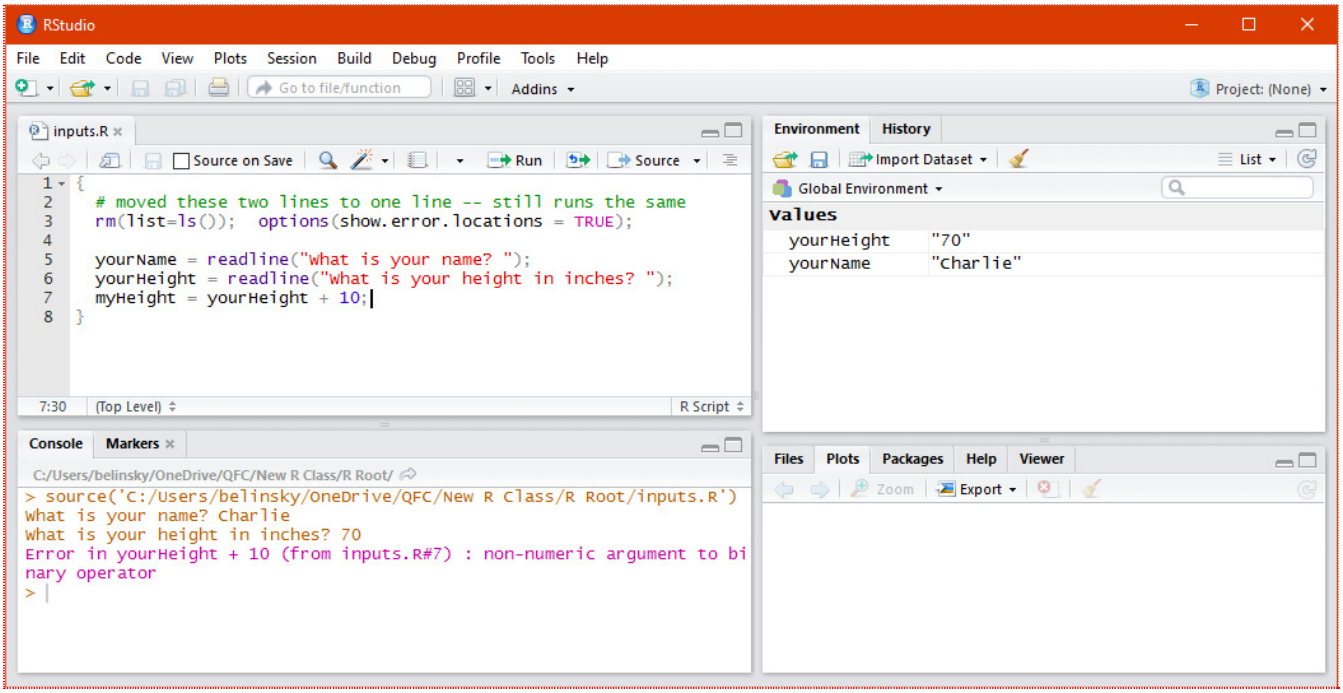

Fig 9: Mathematical operation on a string -- error

The error you get for using a mathematical operation on the string ***yourHeight*** is "*non-numeric argument to binary operator*" (*Fig 9*). As a default, R treats any input as a string even if it only contains numbers.

One way to fix this is to force R to treat ***yourHeight*** as a number by changing line 7 to:

```
7  myHeight = as.numeric(yourHeight) + 10;
```

9 - Extension: Strings vs. characters

These two terms are often conflated in R, and programming in general. String variables are variables that consist of any number of characters. The characters can be anything from letters, to symbols, or numbers -- in any combination. Many programming languages, including C, have both character variables (only one character) and string variables (one or more characters).

R does not functionally distinguish between single character and multiple character variables. But, you will often see these variables referred to as both strings and characters -- just know that, in R's case, they are the same thing.

10 - Extension: functions (and parameters)

A lot of the functionality (pun intended) in a programming language lies in functions. Functions are repeatable procedure that encapsulate a certain task -- and there are numerous functions in R. What is special about functions is that they only get executed when called upon from inside a script. ***readline()*** is a function, which means that somewhere in the deep, dark recesses of R there is a codeblock that is called ***readline()*** and that codeblock gets executed when you invoke ***readline()*** in your script.

The majority of functions take some sort of input. The input for ***readline()*** is the prompt you give to the user and it is put in the parenthesis. These inputs are called parameters. In this case, ***readline()*** takes as input one parameter called ***prompt*** as shown on the help page for ***readline()***. It is good practice to put the parameter name in the function so that instead of:

```
1  readline("Enter your name");
```

you have

```
1  readline(prompt="Enter your name");
```

Both the above line of code do the same thing. We will be talking much more about function in future lessons.

Other functions used in this lesson -- the parentheses are the giveaway:

- ***rm()***
- ***options()***
- ***as.numeric()***