01-07: Conditional Operations

1 - Purpose

- Creating "decision points" in a script using if-statements
- Introduce the six different conditional operators
- Code conditional statements on strings and numbers

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 - Decision Points

So far all of our scripts have been linear -- in other words, the execution of the script goes line-by-line until the end of the script. However, most scripts do not work like this -- they have multiple points within them where the execution of code depends on some condition. These conditions can be thought of as questions: *Is the fish's weight above 100 grams? Is one runner's speed greater than another runner's speed? Did the player hit the fire button? Was the temperature less than 30?*

In script, the questions are presented as conditional statements using ***if()***. The basic structure of an ***if()*** statement is:

```
if (  )   # some condition to check goes in these parenthesis
{
   # execute the code between these curly brackets
   # if the condition in parenthesis is TRUE
}
# next lesson we will cover what happens if the condition is FALSE.
```

Inside the curly brackets (***{ }***) attached to the ***if()*** is a *codeblock* that gets executed if the conditional statement is ***TRUE***. If the conditional statement is ***FALSE***, the codeblock is ignored.

Extension: embedded curly bracket

3.1 - Operators in R

To perform a conditional statement using ***if()*** we need to introduce a new type of operator called the ***conditional operator.*** Along with conditional operators, let's also look at the different types of operators we have already used.

Operator Type	Purpose	R Symbols
Assignment	assign a value to a variable	= or <-

Mathematical	Perform a mathematical operation on a numeric value	+, -, *, /, ^
Conditional	Compare two values	==, !=, >, <, >=, <=

A *conditional operator* does two things:
1) It compares two values using one of six operators (***==, !=, >, <, >=, <=***).
2) It outputs either a ***TRUE*** or ***FALSE*** statement based on results of the comparison.

3.2 - Conditional Operators and Conditional Statements

Conditional Statements are statements that have conditional operators:

Conditional Statement in English	Conditional Statement in R
fish's age greater than 10	fishAge > 10
fish species is "Walleye"	fishSpecies == "Walleye"
fish weight less than or equal to 500 grams	fishWeight <= 500

Note: Conditional Operators exclusively compare two values (e.g., ***fishSpecies*** == "Walleye"). In lesson nine (multiple conditions), we will learn how to deal with more complex conditions (e.g., how to check if ***fishSpecies*** is "Walleye" or "Carp").

3.3 - If statements and conditional operators

The conditional statement goes inside the parenthesis of the ***if():***

If statement in English	If statement in R
if the fish's age is greater than 10	if (fishAge > 10)
if the fish species is "Walleye"	if(fishSpecies == "Walleye")
if the fish weight less than or equal to 500 grams	if(fishWeight <= 500)

The conditional statement in an ***if()*** returns as output either ***TRUE*** or ***FALSE.*** ***TRUE*** means *the codeblock attached to the **if()** gets executed*, ***FALSE*** means *the codeblock attached to the if() does not get executed* (i.e., it is ignored in the code).

An ***if()*** statement contains:
1. A conditional statement
2. Curly brackets (***{ }***) that contain a codeblock that executes if the conditional statement is ***TRUE***

Our first example will ask the user their age and output a statement depending on the value of the age. Don't forget that line 6 is needed to tell R to explicitly designate the input, ***yourAge***, from the user as a number!

Trap: string/number comparison

```
1 {
2   rm(list=ls());  options(show.error.locations = TRUE);
3
4   # R will treat the input, yourAge, as a string,
5   # so we need to force it to become numeric
```

```
  yourAge = readline("What is your age? ");
  yourAge = as.numeric(yourAge);

  if(yourAge < 20)  # check the conditional statement: yourAge less than 20
  {
    # this codeblock is executed if the conditional statement is TRUE
    cat("You are only", yourAge, "?\n");
    cat("You have your whole life ahead of you!!");
  }
}
```

If you run this script then the codeblock attached to the ***if()*** is executed if the age entered by the user is less than *but not equal* to **20**, and the codeblock is ignored if the age is greater than *or equal to* **20**. This script also works with negative numbers and decimals. The values -2, 10, 0.1, 19, and 19.9 all cause the codeblock attached to the ***if()*** to be executed. The values 20, 20.1, and 100 cause the codeblock to be ignored (*Fig.1*).

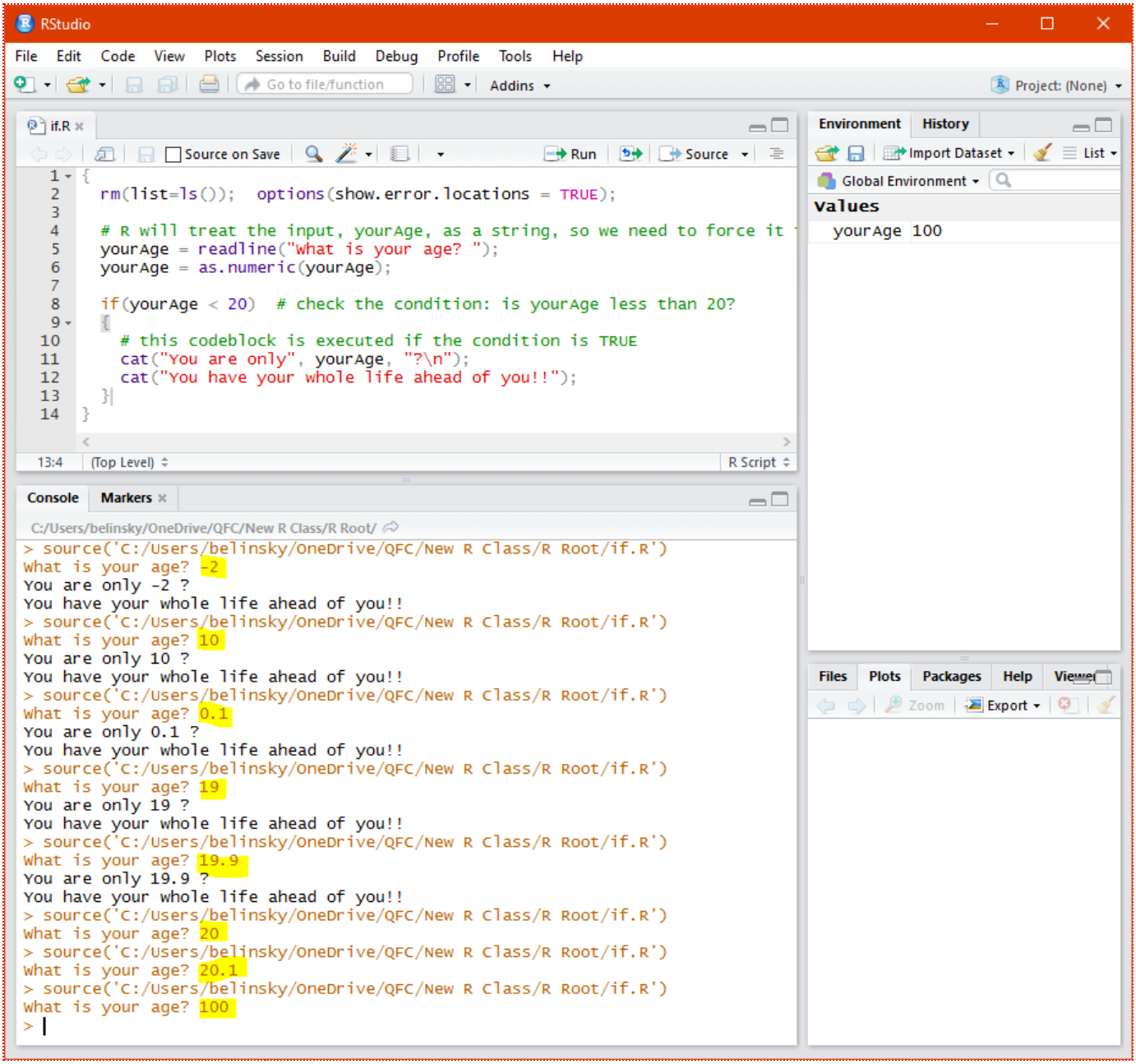

Fig 1: Testing the less-than (<) Conditional Operator by asking the age of the user

Note: the script in (*Fig.1*) ran multiple times but the Environment Window only shows the last value for ***yourAge*** that was input by the user.

4 - Conditional Operators with strings and numbers

The above example used (**<**) to compare the variable ***yourAge*** to the value ***20. Less-than*** (**<**) represents one of six conditional operators -- the table below present all six conditional operators and their effect when used to compare ***yourAge*** to the number ***20.***

Operator	Meaning	results when comparing *yourAge* to *20*
==	equal to	if(***yourAge*** == ***20***) is ***TRUE*** if ***yourAge*** is ***20*** ***FALSE*** if ***yourAge*** is anything but ***20***
!=	not equal to	if(***yourAge*** != ***20***) is

		TRUE if ***yourAge*** is anything but ***20*** ***FALSE*** if ***yourAge*** is ***20***
>=	greater than or equal to	if(***yourAge*** >= ***20***) is ***TRUE*** if ***yourAge*** is greater than or equal to ***20*** ***FALSE*** if ***yourAge*** is less than (but not equal to) ***20***
<=	less than or equal to	if(***yourAge*** <= ***20***) is ***TRUE*** if ***yourAge*** is less than or equal to ***20*** ***FALSE*** if ***yourAge*** is greater than (but not equal to) ***20***
>	only greater than	if(***yourAge*** > ***20***) is ***TRUE*** if ***yourAge*** is greater than (but not equal to) ***20*** ***FALSE*** if ***yourAge*** is less than or equal to ***20***
<	only less than	if(***yourAge*** < ***20***) is ***TRUE*** if ***yourAge*** is less than (but not equal to) ***20*** ***FALSE*** if ***yourAge*** is greater than or equal to ***20***

Fig 2: The six conditional operators and the output of their operation.

The following code will check the *equal* (**==**) condition and the *greater than or equal to* (**>=**) condition

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

  yourAge = readline("What is your age? ");
  yourAge = as.numeric(yourAge);

  if(yourAge == 20)  # check the condition: yourAge is equal to 20
  {
     cat("Welcome to the third decade of your life!\n");
  }
  if(yourAge >= 20) # check the condition: yourAge is greater than or equal to 20
  {
     cat("You still have plenty of life left in you!\n");
  }
}
```

The output (*Fig.3*) shows that the script executes (for example):

1. neither codeblock if the user enters ***10***
2. the second codeblock if the user enters ***30***
3. both codeblocks if the user enters ***20***

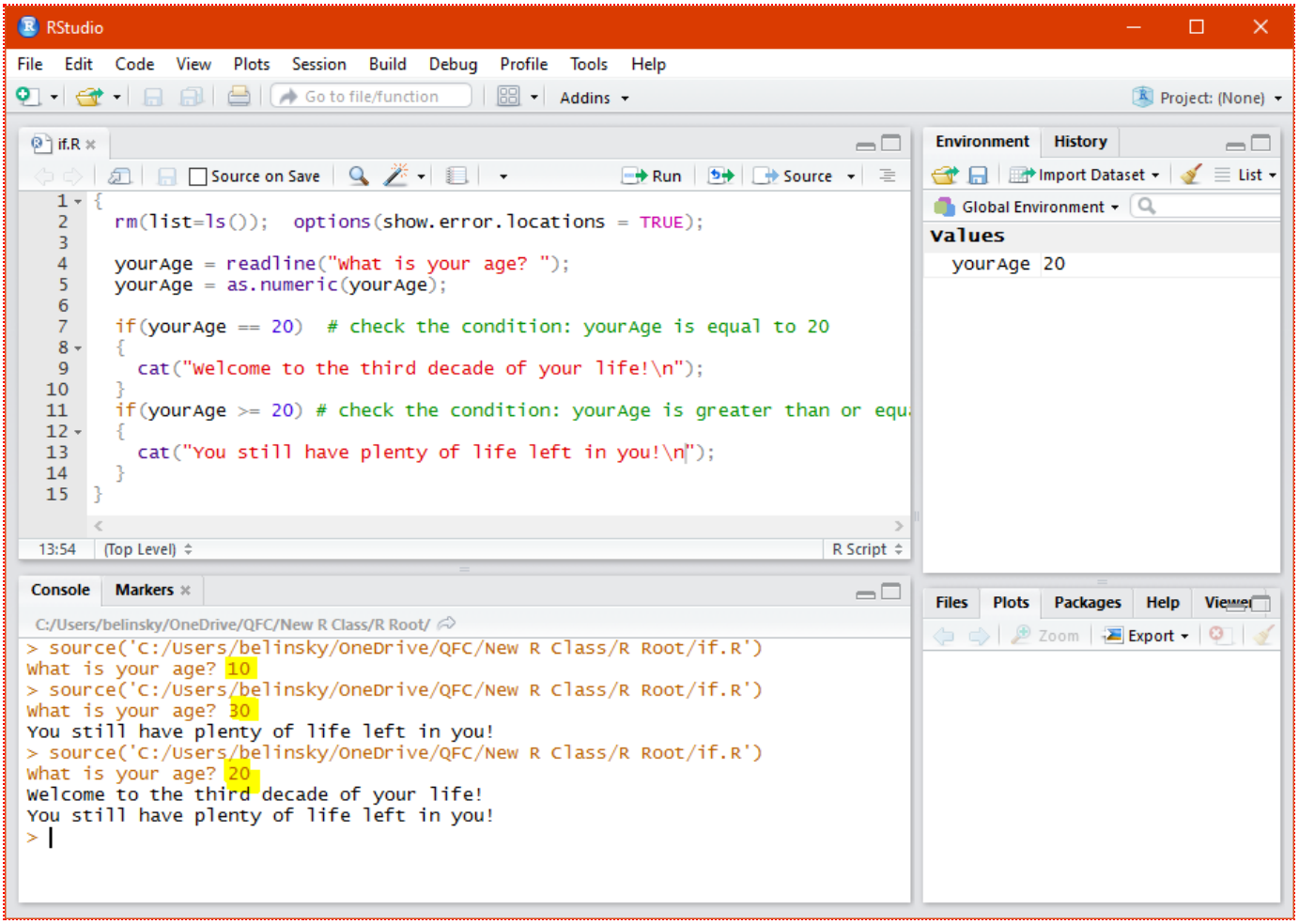

*Fig 3: Multiple if statements checking the same variable, **yourAge***

5 - Conditional operators on strings

The previous examples used conditional operators to compare two numerical values. Conditional operators can also be used to compare two string values. Let's change the above script to one that asks the user for a string input -- their favorite animal.

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

  favAnimal = readline("What is your favorite animal? ");

  # check the conditional statement: favAnimal is equal to "Llama"
  if(favAnimal == "Llama")
  {

    # if the conditional statement is TRUE, execute this codeblock
    cat("You have the wisdom of the elders!!");
  }
}
```

The codeblock attached to the ***if()*** will only be executed if the user entered ***"Llama"***. Even ***"llama"*** (lowercase "***l***" to begin the word) is not seen by R as equal to ***"Llama"*** (*Fig. 4*).

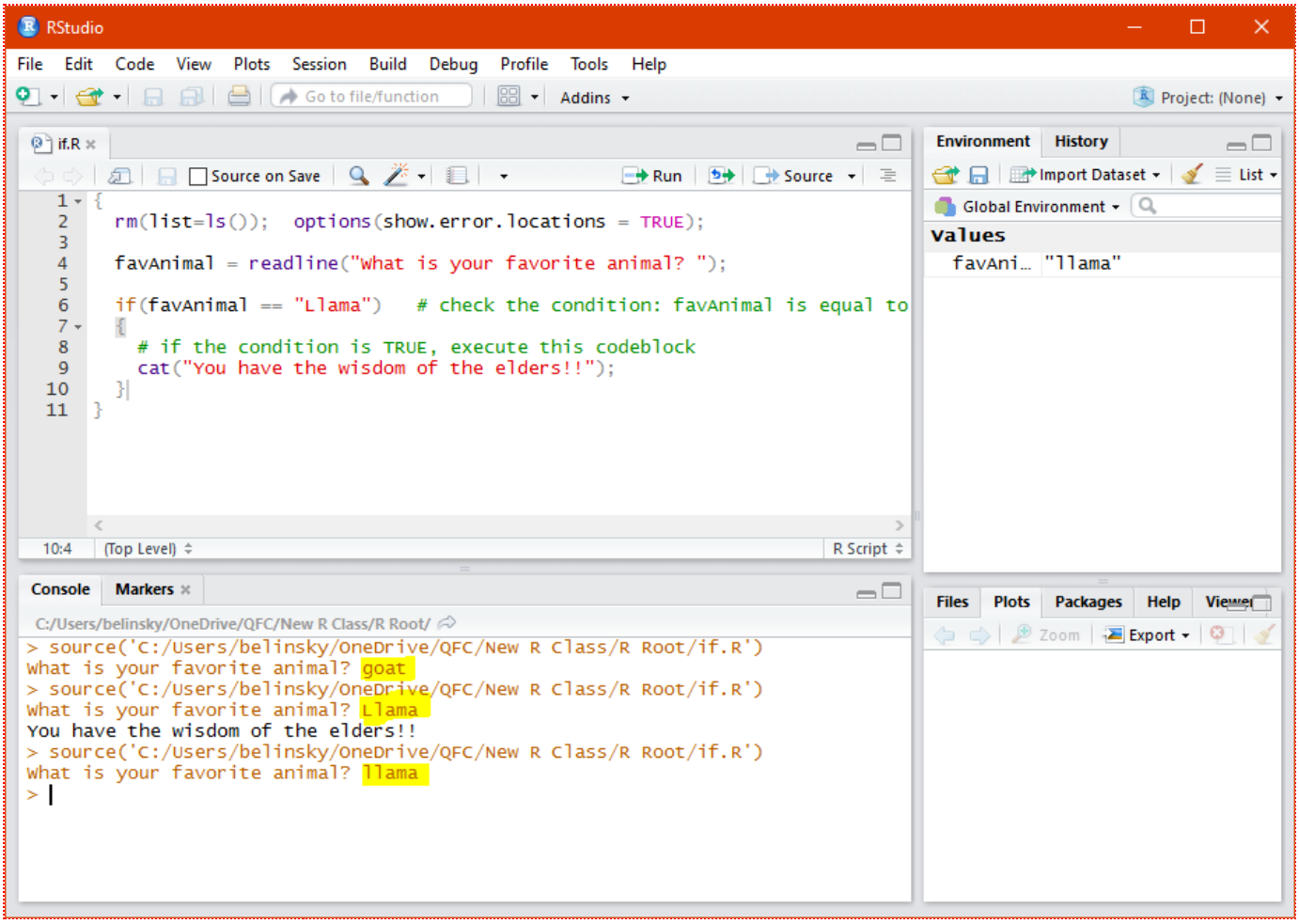

Fig 4: Using a Conditional Operator to compare string values

Only two of the six conditional operators (**==**) and (***!=***) make sense to use when comparing string values.

- **==** returns ***TRUE*** when the string values being compared are *exactly the same.*
- **!=** returns ***TRUE*** when the string values being compared *are not exactly the same*.

EXTENSION: Greater than and less than on string values

5.1 - Using the not equal to operator (!=)

Let's change the script above to use a (***!=***). In this case, the conditional statement is ***TRUE*** if ***favAnimal*** *is not equal* to ***"Llama"***

```
1 {
2   rm(list=ls());  options(show.error.locations = TRUE);
3
4   favAnimal = readline("What is your favorite animal? ");
5
6   # check the conditional statement: favAnimal is NOT equal to "Llama"
7   if(favAnimal != "Llama")
```

```
8   {
9     # if the conditional statement is TRUE, execute this codeblock
10    cat("Well, you still have some growing up to do!!");
11  }
12 }
```

The above code will execute the codeblock attached to the ***if()*** when the user inputs anything but "Llama" as shown below (*Fig. 5*)

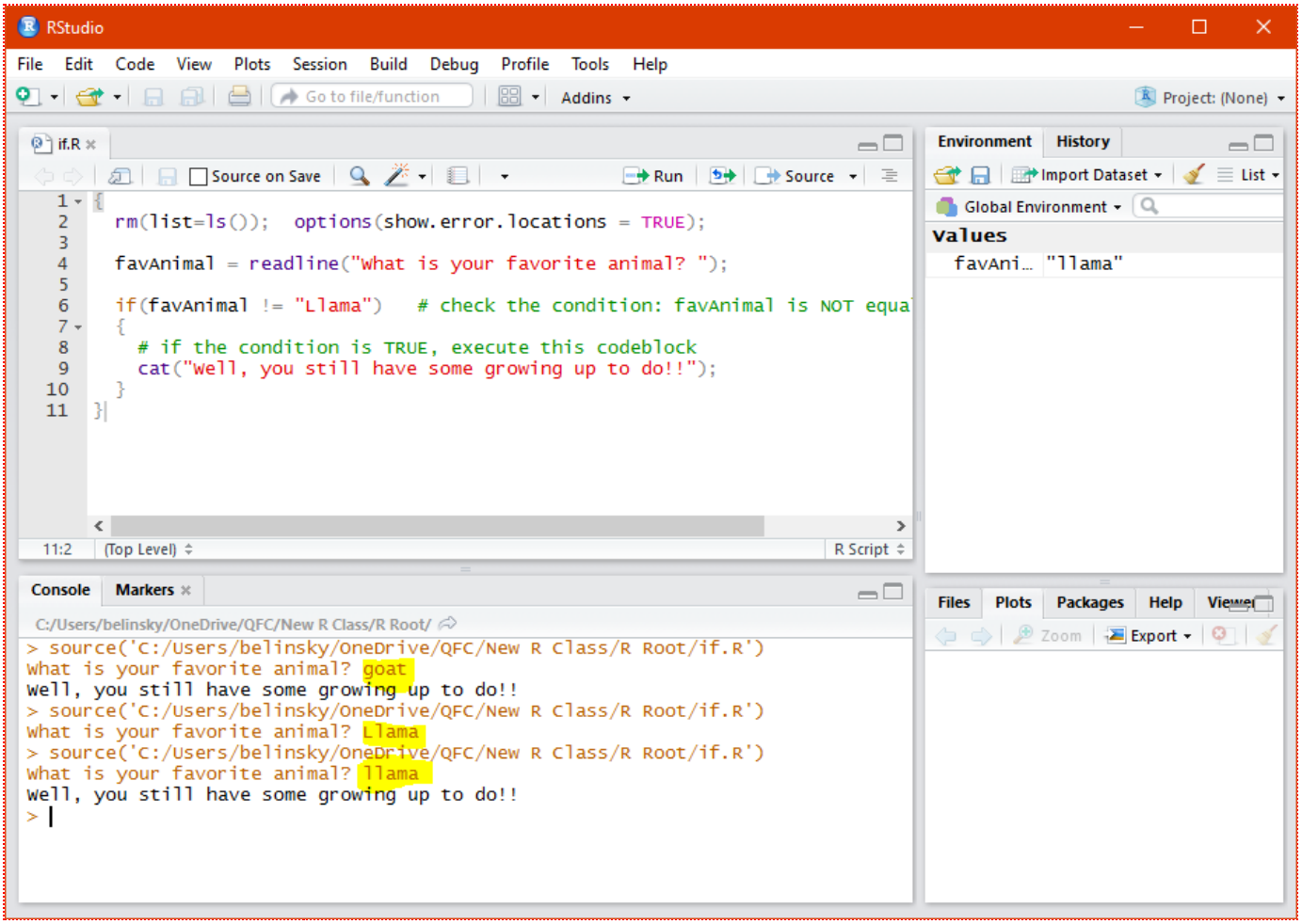

Fig 5: Using the not-equal conditional operator to compare values

6 - 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.

All of the following code should be in *one script file*.
A) Have a user enter two values and save these values to variables:
1) The high temperature for the day
2) The type of weather (either "sunny", "cloudy", or "rainy")

B) Using ***if()*** , give a message if the high temperature is less than 30.

C) Using ***if()***, give a different message if the high temperature is greater than or equal to 80.

D) Using ***if(),*** give a message if the weather is cloudy.

E) Using ***if(),*** give a message if the weather is not rainy.

*Save you script file as **app1-7.r** in the **scripts** folder of your RStudio Project for the class.*

7 - Extension: Embedded curly brackets

Curly brackets are somewhat similar to paragraph in that they bring together multiple statements that are connected. However, curly brackets can also be embedded, meaning you have a codeblock within a codeblock. In all the examples in this lesson, the codeblock attached to the ***if()*** are embedded within the codeblock which is the entire script. You can also embed codeblocks attached to ***if()*** within codeblocks attached to a different ***if()***. For example:

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

  yourAge = as.numeric(readline("What is your age? "));

  if(yourAge < 30)  # check the condition: yourAge is less than 30
  {
    cat("You are less than 30!!");

    if(yourAge < 20)# check the condition: yourAge is less than 20
    {
      cat("And you are less than 20!!!");
    }
  }
}
```

In this case, the codeblock attached to the second ***if()*** is embedded within the codeblock attached to the first ***if().***

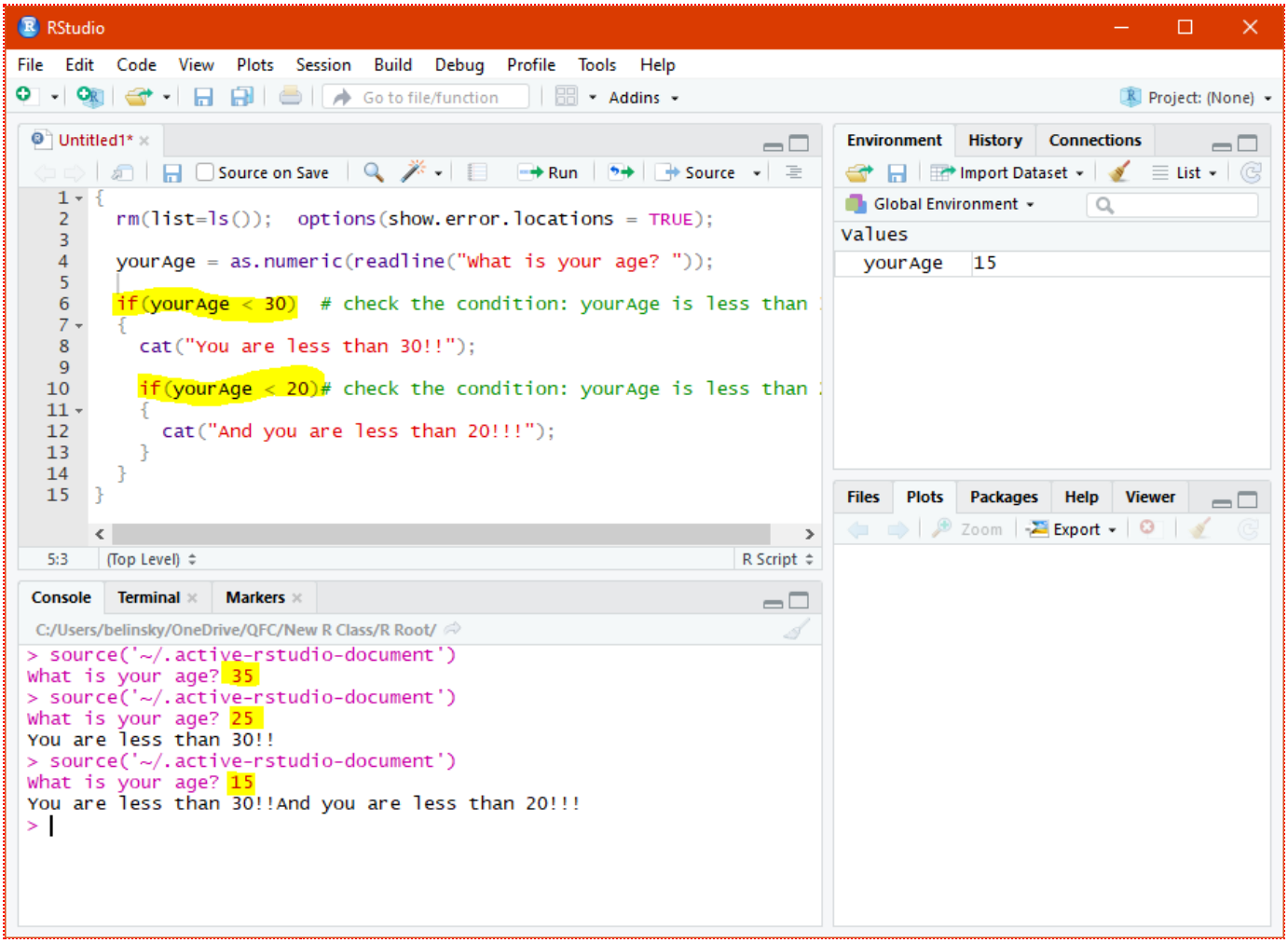

*Fig 6: **if()** embedded within an **if()***

8 - TRAP: String/Number comparisons

If you forget to explicitly say that ***yourAge*** is a number, meaning R thinks of it as a string, R will still attempt to make a comparison between ***yourAge*** and ***20*** in line 5:

```
1  {
2    rm(list=ls());  options(show.error.locations = TRUE);
3
4    yourAge = readline("What is your age? "); # but, R will treat this as a string
5
6    if(yourAge < 20)  # check the condition: yourAge is less than 20
7    {
8        cat("You have your whole life ahead of you!!");
9    }
10 }
```

The result seems to work at first if you use values between 1 and 99 but if you look a little deeper, many problems emerge. For instance the values ***100***, ***!!!***, or ***"bob"*** (with the quotes) are all considered to be less than 20. This is because R converts the number ***20*** to a string in order to compare it to the string that was entered by the user (even if the string has only numbers).

String comparisons are really checking to see how the values alphabetically compare
So "10" is less than "20" because "1" is alphabetically less than "2". The same logic applies for "10000" is less than "20"

The rules for determining whether one character is greater than another character are complicated and have to do with ASCII character codes.

Needless to say, this is not the kind of functionality you want. Most other scripting language will give you a warning or an error if you attempt to make a comparison between a number and a string. In R, you just need to be careful about explicitly ensuring a number is treated as a number.

9 - Extension: Greater than and less than on strings

Greater than and less than conditional operator do work on strings. If the strings only have letters from the English alphabet, then > and < will do an alphabetical comparison between the values.

However, if there are characters other than letters from the English alphabet, R will look at the ASCII character code. ASCII character codes are unique numbers assigned to every character. The main use for ASCII character codes is that they allow someone to use and output characters that are not on your keyboard like the ζ, which is character number 950.