01-08: If-Else Statements

1 - Purpose

- Create condition statement that allow for both TRUE and FALSE conditions using an if-else structure
- do multiple conditional statements on one variable using an if-else-if structure
- create error conditions in an if-else-if structure with an else statement

2 - Concepts

3 - Dealing with FALSE conditions

In the last lesson we looked at the ***if()*** statement and how it changes the flow of a script by creating decision points. The decision point depends on a conditional statement inside the parenthesis of the ***if()***. The codeblock attached to the ***if()*** using curly brackets (***{ }***) gets executed only if the conditional statement is ***TRUE***.

Our next step is to execute a codeblock if the conditional statement is ***FALSE***. This is done using the ***if-else*** structure.

Trap: TRUE and FALSE are keywords

```
if (  )   # the condition in parenthesis is TRUE
{
   # execute the codeblock in these brackets
}
else # the condition above is FALSE
{
   # execute the codeblock in these brackets
}
```

If the condition is ***TRUE***, the codeblock in the curly brackets (***{ }***) attached to the ***if()*** gets executed.
If the condition is ***FALSE***, the codeblock in the curly brackets attached to the ***else*** gets executed.

Trap: Skipping brackets

3.1 - If-else statements and conditional operators

Using the same example from last lesson, we will ask the user their age -- a numeric value (remember, the input value is a string that needs to be converted to a number). Except, this time, the script will give a response for both a ***TRUE*** condition and a ***FALSE*** condition.

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

```

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

  if(yourAge < 20)  # yourAge is less than 20
  {
     cat("You are only ", yourAge, "? You have your whole life ahead of you!!");
  }
  else # yourAge is not less than 20 (it is greater than or equal to 20)
  {
      cat("You are quite the grown-up!");
  }
}
```

If you run this script then the codeblock attached to the ***if()*** will be executed if the age is less than *but not equal* to 20, and the codeblock attached to the ***else*** is executed if the age is greater than *or equal to* 20.

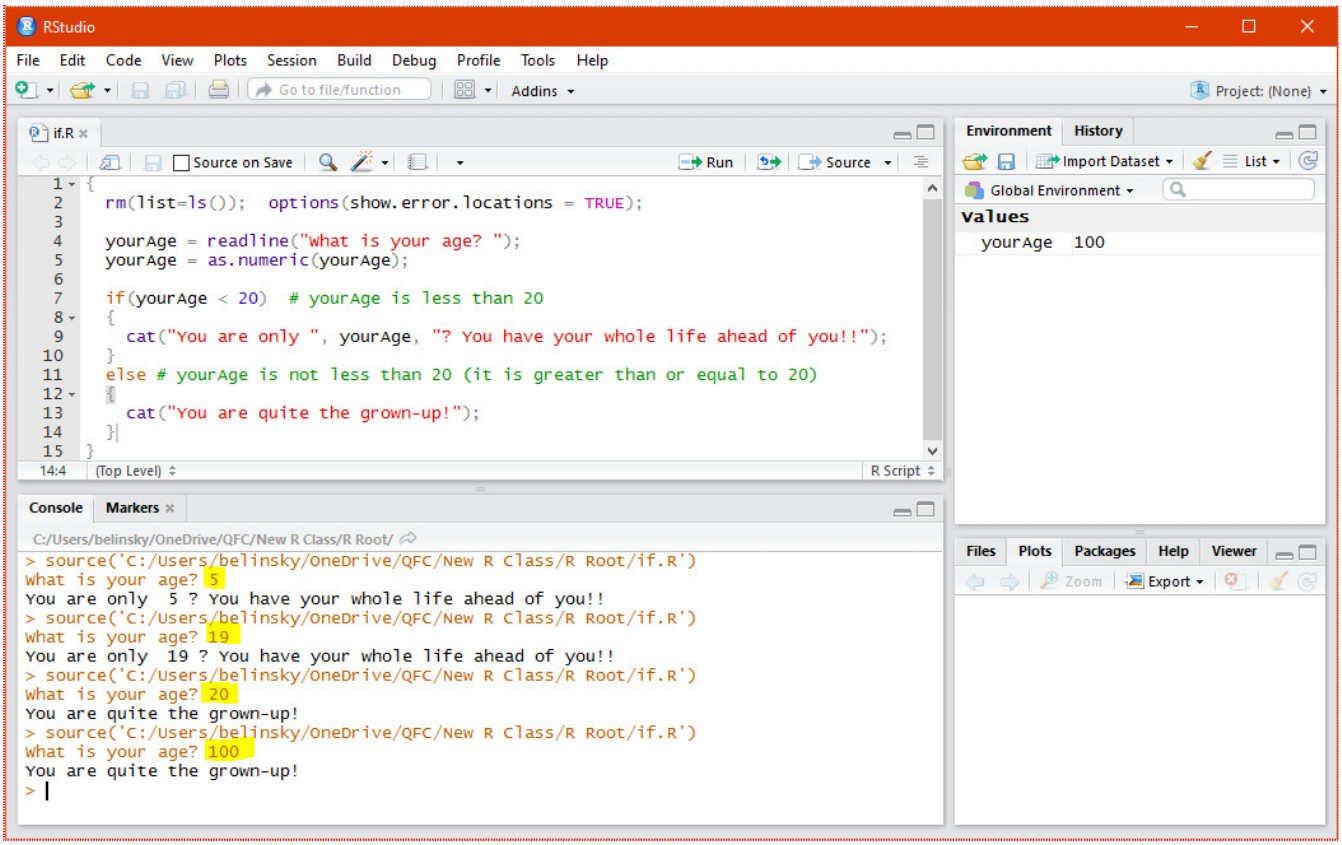

*Fig 1: **If-else** statement using a numerical variable*

3.2 - If-Else statements using strings

Conditional operators can also be used to compare two string values. Let's change the above script to one that asks the user for an input that is a string-- specifically their favorite type of cheese.

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

```
3
4   favCheese = readline("What is your favorite type of cheese? ");
5   if(favCheese == "Muenster")   # favCheese equal to "Muenster"
6   {
7       cat("You are a culinary genius!!");
8   }
9   else   # favCheese not equal to "Muenster"
10  {
11      cat(favCheese, "?!? You have much to learn!");
12  }
13 }
```

The codeblock attached to the ***if()*** will only be executed if the user entered "***Muenster"*** -- entering any other variation (e.g., "muenster", "Muunster", "Meunster") will result in a ***FALSE*** condition and the codeblock attached to the ***else*** statement is executed. In the next lesson we will look at how to check for misspellings.

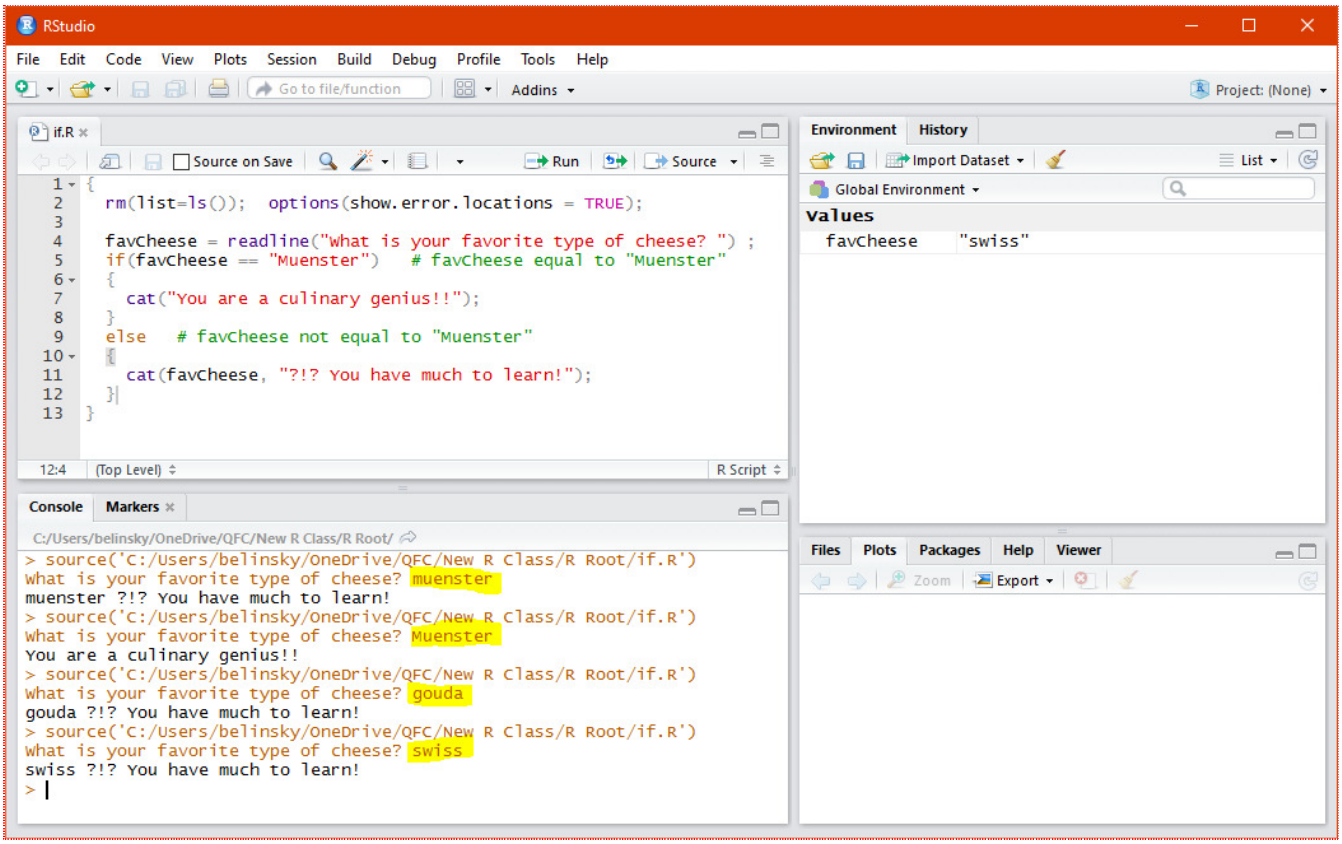

*Fig 2: **If-else** statement using a string variable*

3.3 - If-else using not-equal-to (!=)

The conditional statements in the above script, along with the attached code, can be reversed (**==** to **!=**) and the code will run exactly the same. We just have to change some of the logic around.

```
1 {
```

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

  favCheese = readline("What is your favorite type of cheese? ");
  if(favCheese != "Muenster")   # favCheese NOT equal to "Muenster"
  {
     cat(favCheese, "?!? You have much to learn!" );
  }
  else   # favCheese equal to "Muenster"
  {
     cat("You are a culinary genius!!");
  }
}
```

In this case, if the user's favorite cheese *is anything but "Muenster"*, the above script will execute the codeblock attached to the ***if()***. If the user's favorite cheese *is not anything but "Muenster"* (i.e, it ***is*** "Muenster"), the above script will execute the codeblock attached to the ***else***. This script is functionally the same as the previous script.

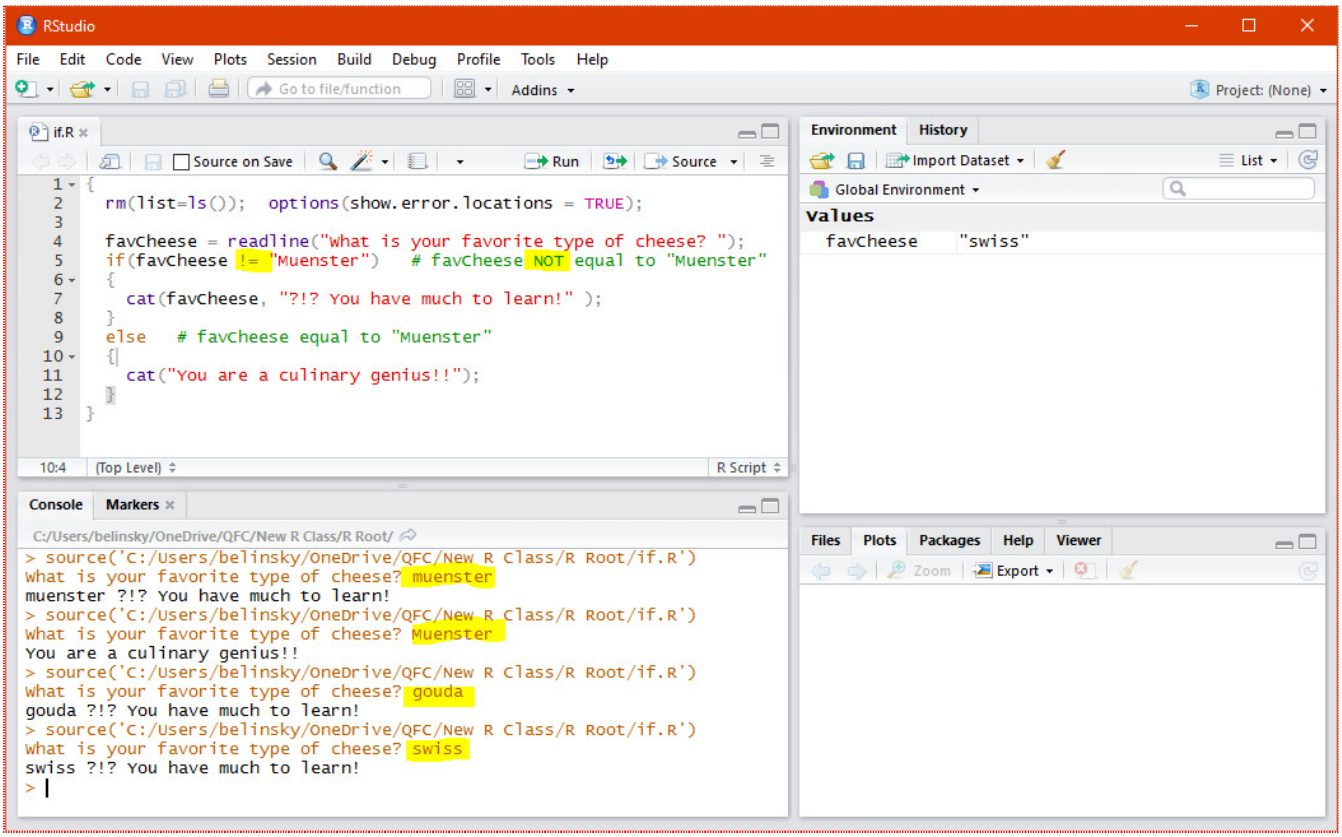

*Fig 3: **If-else** using not-equal-to (**!=**) condition*

4 - Adding in more conditions: If-Else-If Structures

We are limited to two outcomes when using ***if-else*** structures: (a) the ***if()*** condition is ***TRUE*** or (b) The ***if()*** condition is ***FALSE***. However, the examples above contain situations that could easily have more than two conditions. For instance, we might want to check more cheese types than just "Muenster". Perhaps we want to output a message if the user types in "Swiss" or "American".

In the code above, the codeblock attached to the ***else*** is executed in all situations where ***favCheese*** is not "Muenster". In other words, the ***else*** is a wastebasket for all situations in which the condition inside the ***if()*** is ***FALSE***.

We can break down the ***else*** statement to create more conditions by using the ***if-else-if*** structure.

Extension: curly bracket placement

The script below checks multiple cheese varieties (Muenster, American, Swiss, and Gouda) and executes a different codeblock for each. *If the user types in none of these choices, then the script outputs nothing.* Note that only the exact spelling of Muenster, Gouda, American, and Swiss will satisfy the conditional statement.

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

  favCheese = readline("What is your favorite type of cheese? ");

  if(favCheese == "Muenster")
  {
     cat("You are a culinary genius!!");
  }
  else if(favCheese == "Swiss")
  {
     cat("Holes don't make a cheese taste good!");
  }
  else if(favCheese == "American")
  {
     cat("Is there any real milk in there?");
  }
  else if(favCheese == "Gouda")
  {
     cat("So you like the smell of dirty socks?");
  }
}
```

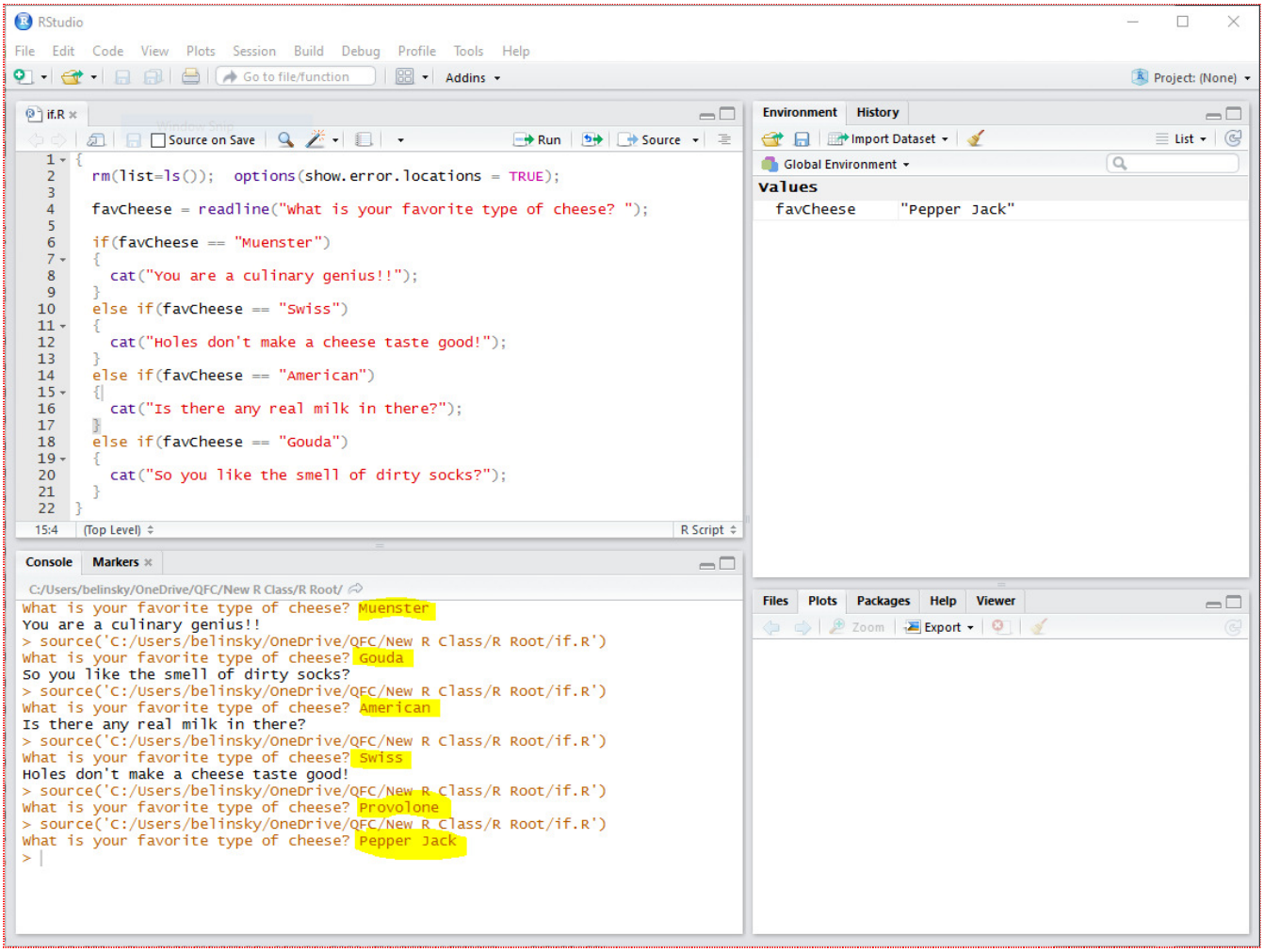

*Fig 4: **If-else-if** structure using string variable -- checking for multiple cheese varieties*

4.1 - The error/ everything else condition

We usually want to execute something for conditions we have not prepared for. For instance, we might have a menu options numbered 1 through 5 and the user chooses option 7. In this case we want a message saying something like "sorry, 7 is not an option".

The problem with the script above is that it does nothing if the user types in a cheese that is not listed. We cannot possibly list every cheese in the world, let alone every errant spelling. Instead, we create what is often referred to as an *error condition,* meaning we set up a codeblock that gets executed if none of the conditions in the ***if-else-if*** structure are ***TRUE***. This error codeblock gets attached to the ***else*** statement.

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

  favCheese = readline("What is your favorite type of cheese? ");

  if(favCheese == "Muenster")
  {
     cat("You are a culinary genius!!");
```

```
  }
  else if(favCheese == "Swiss")
  {
     cat("Holes don't make a cheese taste good!");
  }
  else if(favCheese == "American")
  {
     cat("Is there any real milk in there?");
  }
  else if(favCheese == "Gouda")
  {
     cat("So you like the smell of dirty socks?");
  }
  else  # error condition -- none of the conditional statements above were TRUE
  {
    cat("I'm sorry, I did not understand your response");
  }
}
```

Now there is guaranteed to be a response from the script no matter what the user types in...

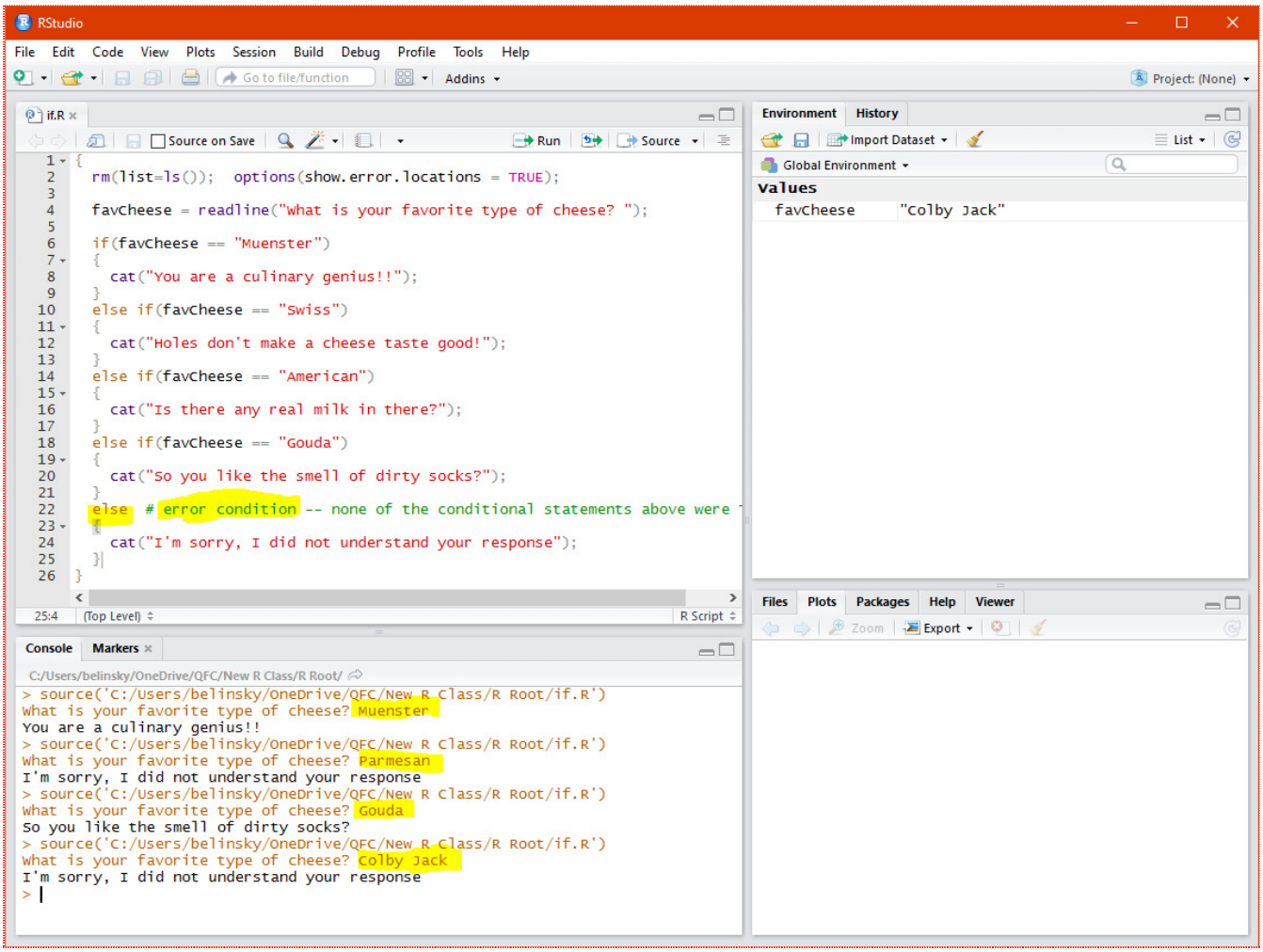

*Fig 5: Using the **else** statement as an error condition*

Trap: Else never has an attached condition

5 - Creating number ranges using if-else structures

The script above checks a string variable (***favCheese***) for specific values (e.g., Muenster, Swiss, etc..) and outputs a message for each specific value. However, you often want to look at ranges of values. For instance, class grades are based on a range of numeric values (e.g, a "C" is 70%-80%).

The following code asks the user for an input score and assigns the score a grade.

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

  score = readline("Enter your score: ");
  score = as.numeric(score);

  if(score > 90)
  {
     cat("A");
  }
  else if(score > 80) # score between 80-90
  {
     cat("B");
  }
  else if(score > 70) # score between 70-80
  {
     cat("C");
  }
  else if(score > 60) # score between 60-70
  {
     cat("D");
  }
  else  # all above conditions were FALSE
  {
     cat("E");
  }
}
```

"A" will be output anytime the user type in a number above 90, "E" will be output anytime the user types in a number below 60.

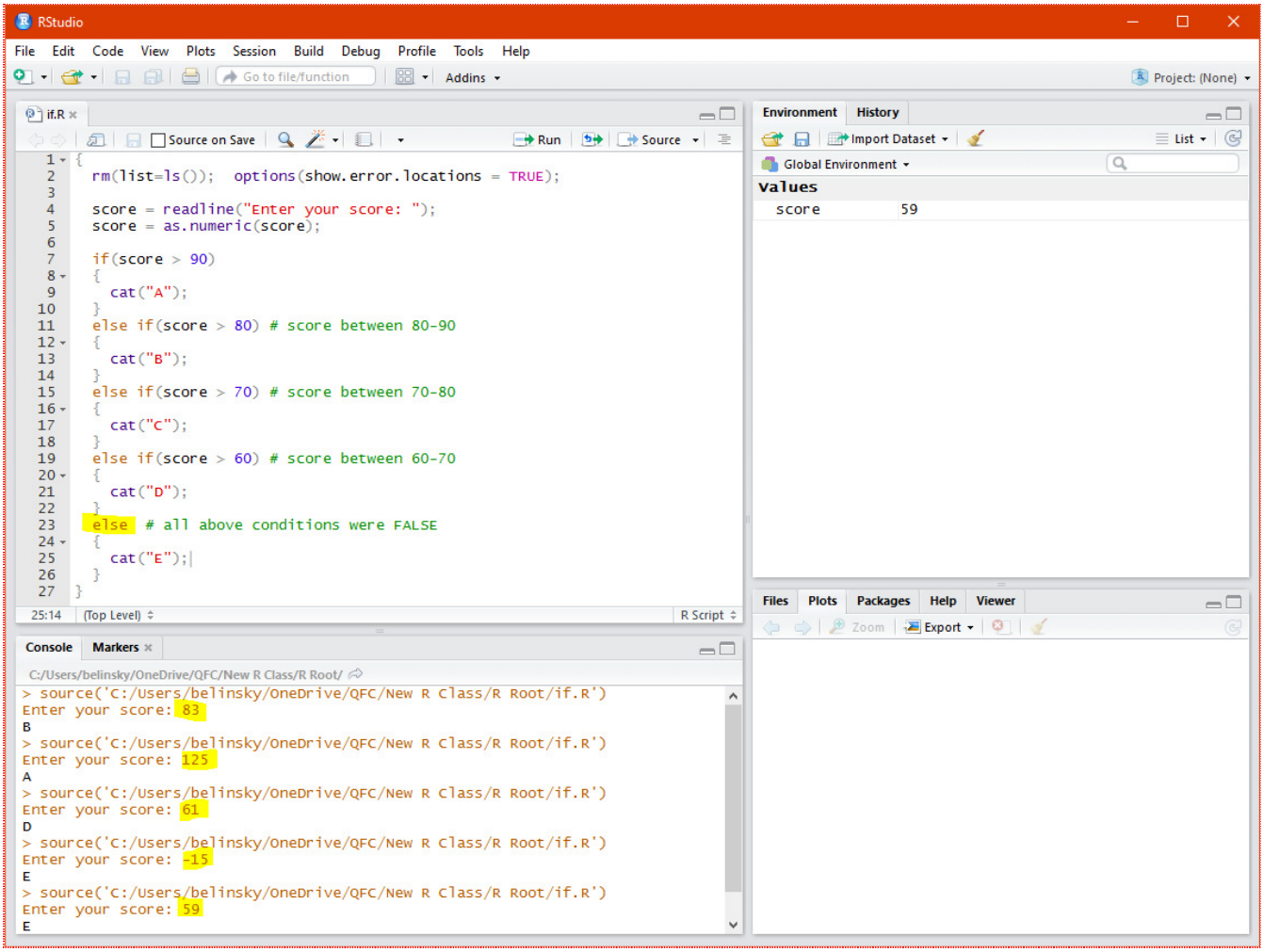

Fig 6: ***If-else-if*** *structure for grades*

5.1 - If-Else-If statements are mutually exclusive

An ***if-else-if*** *structure ends as soon as a condition is determined to be* ***TRUE*** so at most one codeblock in an ***if-else-if*** structure gets executed. In the above example (*Fig 6*), a grade of 97 means the first condition (grade > 90) is ***TRUE*** and "A" will be output. The ***if-else-if*** ends here and none of the other conditions are checked.

So, if the score of ***83*** was entered in (*Fig 6*) the first condition (***grade > 90)*** is ***FALSE*** but the second condition (***grade > 80***) is ***TRUE*** -- so ***B*** is output to the console. The third and fourth conditions (***grade > 70***) and (***grade > 60***) are both ***TRUE*** if ***grade*** is **83** *but the third and fourth conditions do not get checked by the script* because an above condition (***grade > 80***) has already evaluated to ***TRUE***.

In an ***if-else-if*** structure, the code-block executed is the one attached to the first conditional statement that is ***TRUE***. If none of the conditional statements are ***TRUE*** then the codeblock attached to the ***else*** statement is executed.

6 - Application

A) Create a script where the user enter in a temperature value
1) Have the script output:

- "cold" if the temperature is between -20 and 30 (exclusive of -20 and 30)
- "warm" if the temperature is between 30 and 60 (inclusive of 30, exclusive of 60)
- "hot" if the temperature is between 60 and 100 (inclusive of 60 and inclusive of 100)
- "invalid value" if the temperature is below -20 or above 100
- hint: do the if-else-if structure strictly in ascending order or strictly in descending order -- like the grades example above. This mean splitting the "invalid" category into two conditions (less than -20 and greater then 100)

2) Challenge: In the same script -- change the output if the user enters a "border" value (in this case: 30 or 60)

- for 30 output: "cold-ish"
- for 60 output: "hot-ish"

B) Create a second script that reverses the grading script from this lesson. In this script:

1) Have the user enter a grade (A, B, C, D, or E)
2) Have the script output the score range for the grade entered (so C would be "70-80")
3) Have the script output an error message if the user enters anything except A,B,C,D,E
4) Challenge: Have the script give the correct scoring range for the lowercase letters (a,b,c,d,e)

7 - Trap: TRUE and FALSE are keywords, T and F are not

The terms ***TRUE*** and ***FALSE*** are reserved *keywords* in R -- this means that ***TRUE*** *and* ***FALSE*** are predefined and cannot be used as variable names in R.

If you try to assign a value to a "variable" named TRUE or FALSE, you will get an "*invalid (do_set) left-hand side to assignment*" error (*Fig 7*). This is the same error you get if you try to assign a value to a number (i.e., 10 = "a").

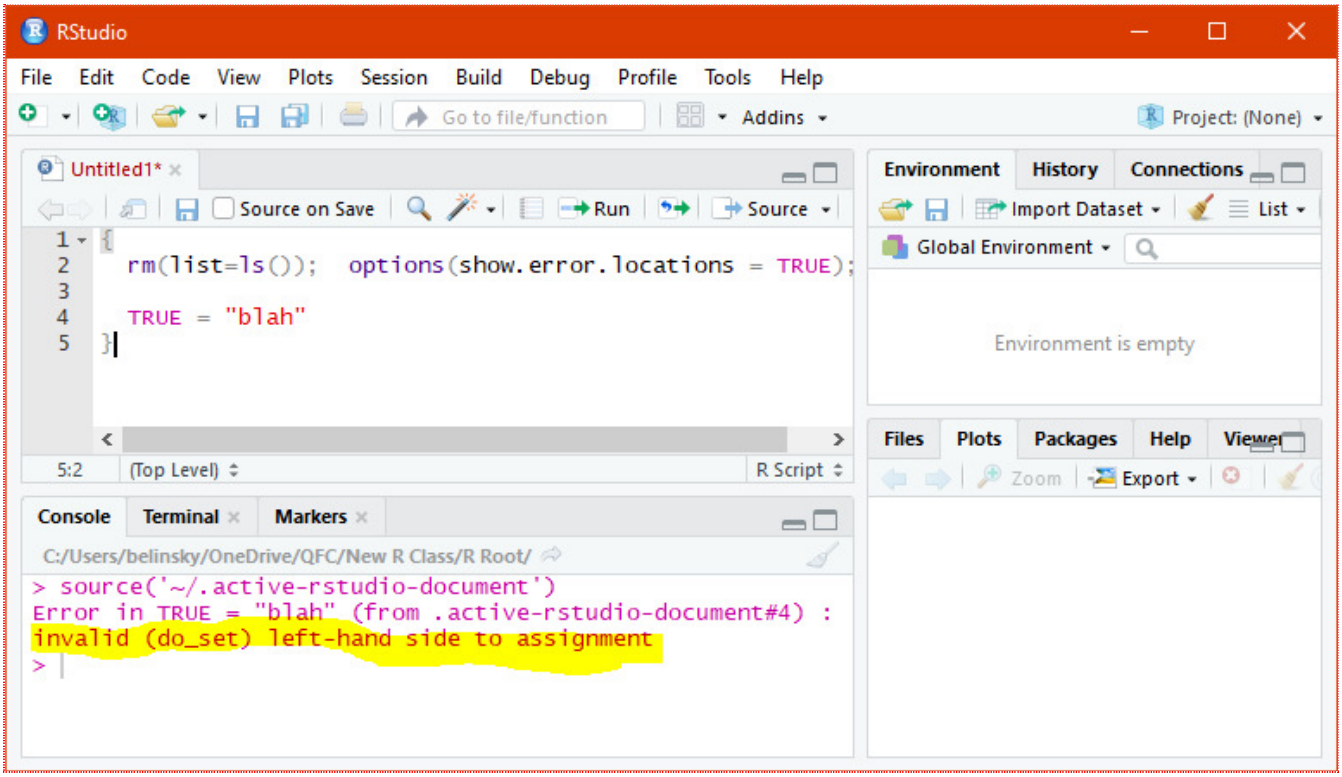

Fig 7: Assigning a value to a keyword causes an error.

You will often see R scripts that use ***T*** and ***F*** as shortcuts for ***TRUE*** and ***FALSE***. R accepts ***T*** and ***F*** as

shortcuts for ***TRUE*** and ***FALSE*** ***but T and F are not keywords***. This means that someone can assign a value to ***T*** or ***F*** that overrides the default ***TRUE/FALSE*** values. If this happens, using ***T*** or ***F*** as shortcuts for ***TRUE*** or ***FALSE*** would produce unexpected results. It is best to stick with the reserved (and protected) keywords ***TRUE*** and ***FALSE***.

8 - Extension: brackets and code blocks

A codeblock enclosed by curly brackets ***{ }*** usually follows an ***if()*** statement. The brackets designate the start and end of the code block and R does not care how the brackets are placed in code -- as long as the order is correct.

The following four code-blocks all execute exactly the same in R:

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

  yourAge = readline("How old are you? ")

  if(yourAge < 20)
  {
      cat("You have your whole life ahead of you!!");
  }
}
```

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

  yourAge = readline("How old are you? ")

  if(yourAge < 20){
      cat("You have your whole life ahead of you!!");
  }
}
```

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

  yourAge = readline("How old are you? ")

  if(yourAge < 20){ cat("You have your whole life ahead of you!!"); }
}
```

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

  yourAge = readline("How old are you? ")
```

```
5
6   if(yourAge < 20)
7                                                 {
8   cat("You have your whole life ahead of you!!");
     }
9 }
```

All the above code blocks are valid. R just looks for a start and end bracket and executes everything in between -- it completely ignores the spaces in between. The decision on where to place the brackets has to do with coding standards and user viewability. The first two code blocks above are the most commonly accepted standards and your code should use one of these two methods.

9 - Extension: Skipping brackets in if-else structures

You will often see ***if-else*** statements with no curly brackets. This can only be done if *there is only code statement* attached to the ***if()*** or the ***else***.

```
1 {
2   rm(list=ls());   options(show.error.locations = TRUE);
3
4   favAnimal = readline("What is your favorite animal? ");
5
6   if(favAnimal == "Llama")
7     cat("You are wise beyond your years");
8   else
9     cat("You still have a lot of room for growth");
10 }
```

The code above is functionally equivalent to the code below

```
1 {
2   rm(list=ls());   options(show.error.locations = TRUE);
3
4   favAnimal = readline("What is your favorite animal? ");
5
6   if(favAnimal == "Llama")
7   {
8      cat("You are wise beyond your years");
9   }
10  else
11  {
12     cat("You still have a lot of room for growth");
13  }
14 }
```

The only advantage to not using curly brackets is that it takes up less space. I generally recommend that you put in curly brackets even if there is only one line of code in the codeblock -- it is more explicit and you avoid accidentally putting more than one line of code without using brackets:

The code below will give an *"unexpected 'else'"* error. This is because R believes that the ***if()*** structure ends right after the first ***cat()*** on line 7 because there are no curly brackets attached to the ***if().*** This means the ***else*** statement is completely detached from the ***if()***, which is an error.

```
1  {
2    rm(list=ls());   options(show.error.locations = TRUE);
3
4    favAnimal = readline("What is your favorite animal? ")
5
6    if(favAnimal == "Llama")
7       cat("You are wise beyond your years\n");
8       cat("No doubt about that.");       # this line is NOT attached to the if()
9    else
10      cat("You still have a lot of room for growth");
11 }
```

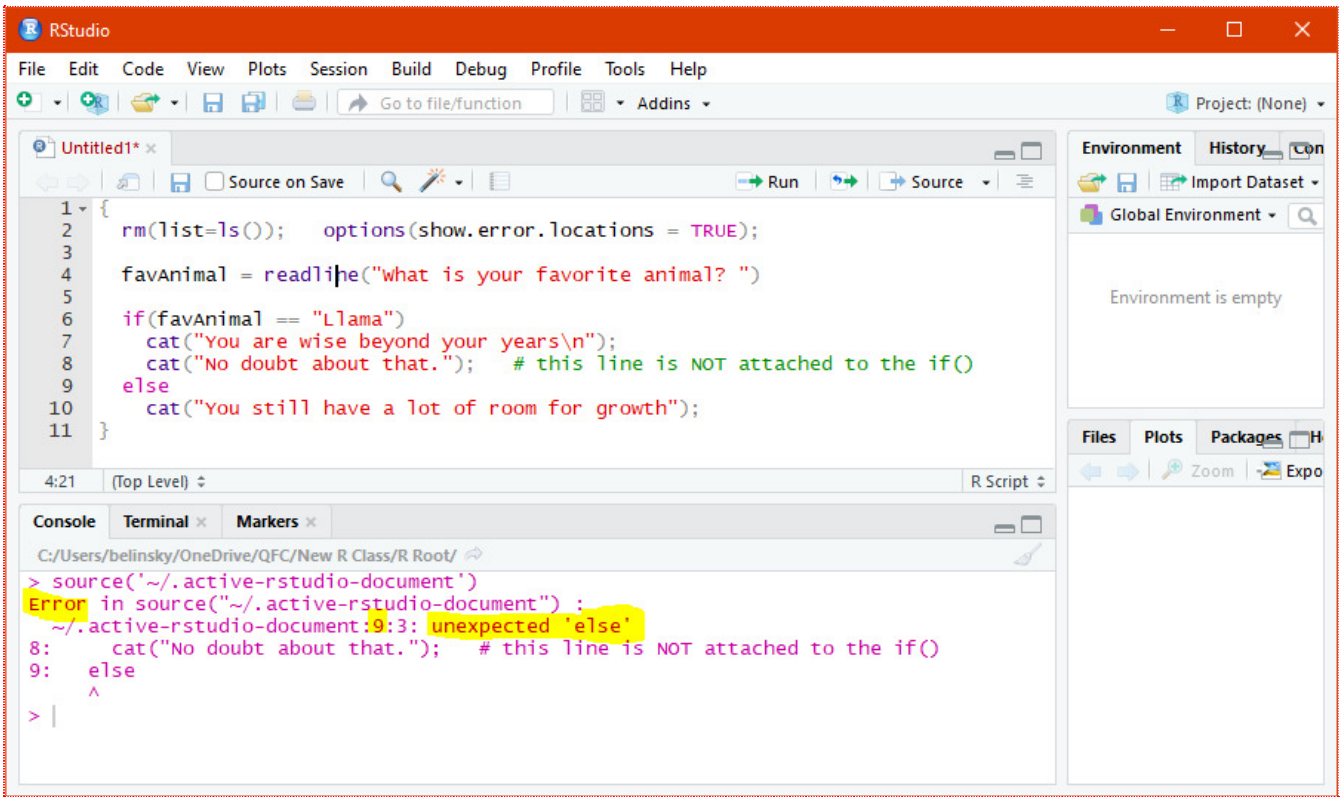

*Fig 8: Unexpected **else** error caused by not using curly brackets in **if-else** structures.*

10 - Trap: Else never has an attached condition

A common mistake for people new to scripting is to treat ***else*** like an ***if()*** or an ***else if()*** and attach a condition to it. In the script below, the code attached to the ***else*** statement is supposed to be executed if all other conditional statements in the ***if-else-if*** structure fail, which is when the score is less than 60.

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

  score = readline("Enter your score: ")
  score = as.numeric(score);

  if(score > 90)
  {
     cat("A");
  }
  else if(score > 80) # score between 80-90
  {
     cat("B");
  }
  else if(score > 70) # score between 70-80
  {
     cat("C");
  }
  else if(score > 60) # score between 60-70
  {
     cat("D");
  }
  else (score <= 60)  # will cause an error
  {
     cat("E");
  }
}
```

The script seems to work at first. If you type in a value below 60 then "E" will be the output. However, if you type in 75 the output is "CE". If you type in 95 the output is "AE". In fact, "E" will be part of the output no matter what you type in.

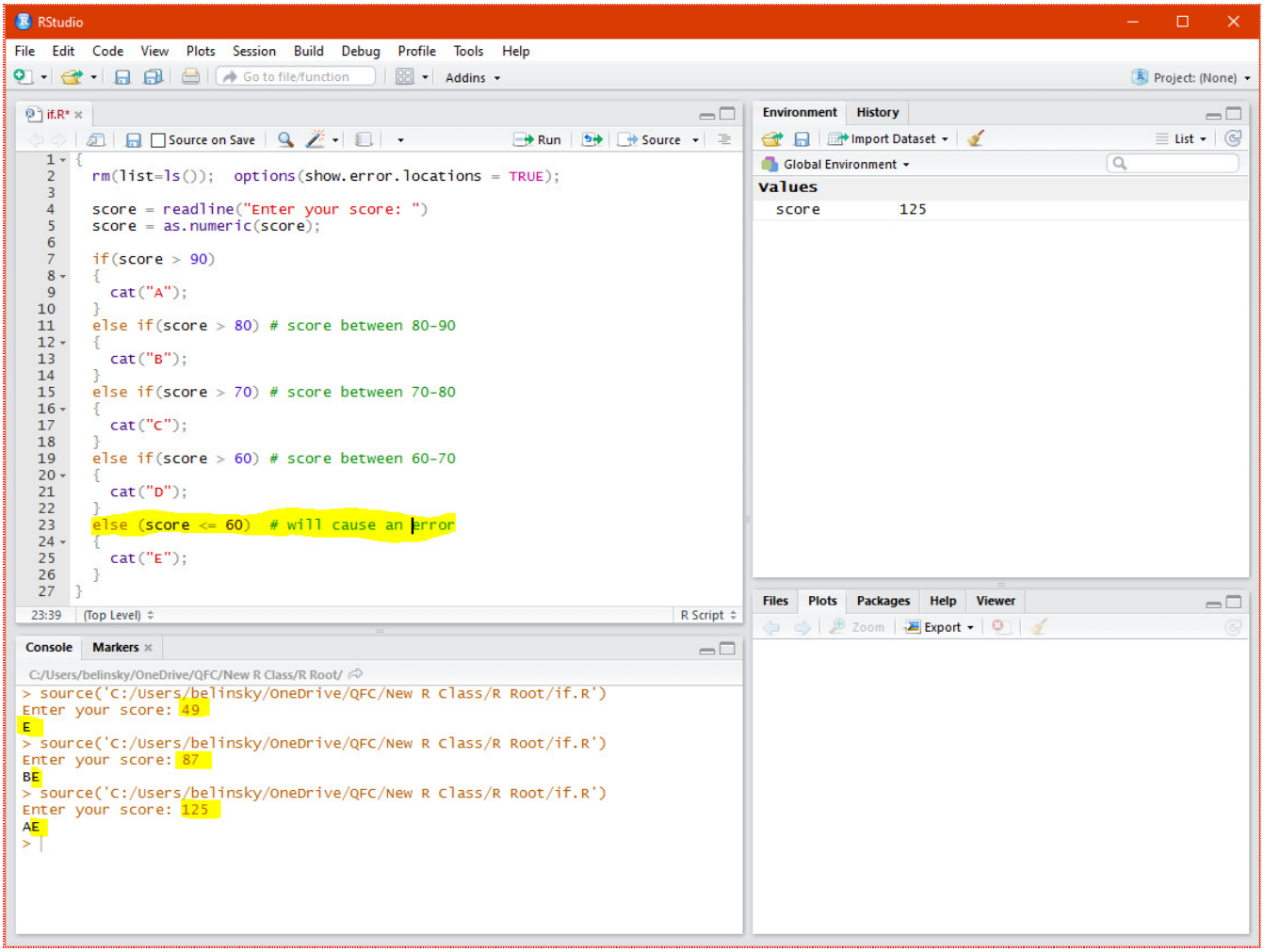

*Fig 9: Putting a condition on an **else** -- causes an error*

Essentially, you have detached the codeblock

```
{
   cat("E");
}
```

from the ***else*** statement and, because of this, ***cat("E")*** will execute every time the script is executed.

The proper way to code the else is as follows:

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

  score = readline("Enter your score: ");
  score = as.numeric(score);

  if(score > 90)
  {
     cat("A");
  }
  else if(score > 80) # score between 80-90
```

```
12    {
13       cat("B");
14    }
15    else if(score > 70) # score between 70-80
16    {
17       cat("C");
18    }
19    else if(score > 60) # score between 60-70
20    {
21       cat("D");
22    }
23    else  # all other conditions failed
24    {
25       cat("E")
26    }
27 }
```

Now the ***else*** will execute only when all other conditions fail. In other words, it will execute when ***score*** is less than or equal to ***60***