01-03: Mathematical Operations

1 - Purpose

- Perform mathematical operations, including powers, on numerical variables
- Explicit use of multiplication symbols in formulas
- Convert algebraic formulas to programming formulas
- Use parentheses to establish order of operations for formulas

2 - Concepts

3 - Putting a formula in code

Once again we will calculate ***velocity*** using ***distance*** and ***time*** except we will now use the full version of the ***velocity*** formula which looks at the *changes in distance and time* as opposed to absolute distance and time.

The full velocity formula is (subscript ***i*** means "initial", subscript ***f*** means "final"):

$v = \frac{d_f - d_i}{t_f - t_i}$ (If t_i and d_i are zero then you get the original formula: $v = \frac{d}{t}$)

We are going to code this formula but there are a couple of issues:

1) fractions are "stacked" -- but in script, equations can only be read left to right (similar to formulas in Excel)
2) variable names have subscript characters (e.g., t_i) and subscript and superscript characters are not allowed in script

3.1 - Unstacking formulas

The velocity formula above is read in two dimensions. In other words, you read it both left-to-right and, because of the fraction, you are also reading it up-and-down. In script, everything goes left-to-right so you only have one dimension to work with. Before we code the formula, *we need to rewrite the formula to something we can put on one line* that maintains the same order-of-operations.

In order to do this, we need to first group the items in the numerator and denominator by putting them in parenthesis:

$v = \frac{(d_f - d_i)}{(t_f - t_i)}$

And then pull out the fraction between the numerator and denominator and replace it with a division sign:

$v = (d_f - d_i) / (t_f - t_i)$

Now the formula is all on one line but the symbols need to be replaced with valid variable names that do not

have subscripts:

$$velocity = (finalDist\ -\ initDist)\ /(finalTime\ -\ initTime)$$

This is now a valid line of code in R -- assuming all variables on the right have assigned values.

```
1  velocity = (finalDist - initDist) / (finalTime - initTime);
```

The line of code above says that velocity *will be assigned* the value equal to the calculations on the right side of the equation.

3.2 - Variable naming error -- or cases count

Here is the full script calculating velocity with a small error on line 7:

```
1  {
2    rm(list=ls());                          # clean out the environment
3    options(show.error.locations = TRUE);  # give line numbers on errors
4
5    finalDistance = 100;
6    initDistance = 50;
7    finaltime = 20;       # error on this line
8    initTime = 15;
9    velocity = (finalDistance - initDistance) / (finalTime - initTime);
10 }
```

Every variable on the right side of the equation on line 9 must be given a value before-hand -- otherwise you will get the pesky "*Object not found*" error as shown in the image below (*Fig 1*)

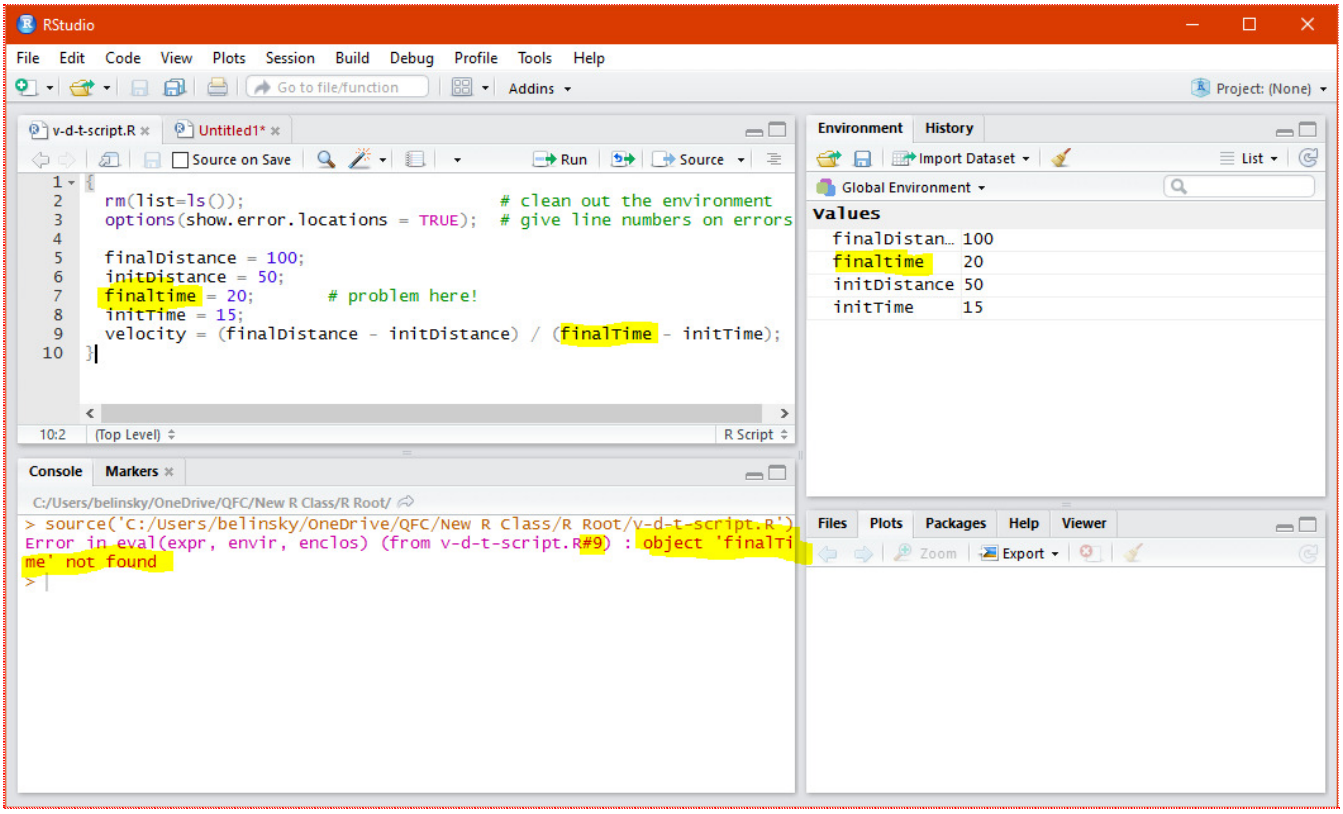

Fig 1: Object not found because the variable name does not exist (it is misspelled).

Notice this error occurred because I "spelled" the variable name wrong by changing the case of the ***T***. ***finaltime*** is not the same as ***finalTime*** -- case counts in programming!

Here is the working code:

```
{
  rm(list=ls());                          # clean out the environment
  options(show.error.locations = TRUE);   # give line numbers on errors

  finalDistance = 100;
  initDistance = 50;
  finalTime = 20;
  initTime = 15;
  velocity = (finalDistance - initDistance) / (finalTime - initTime);
}
```

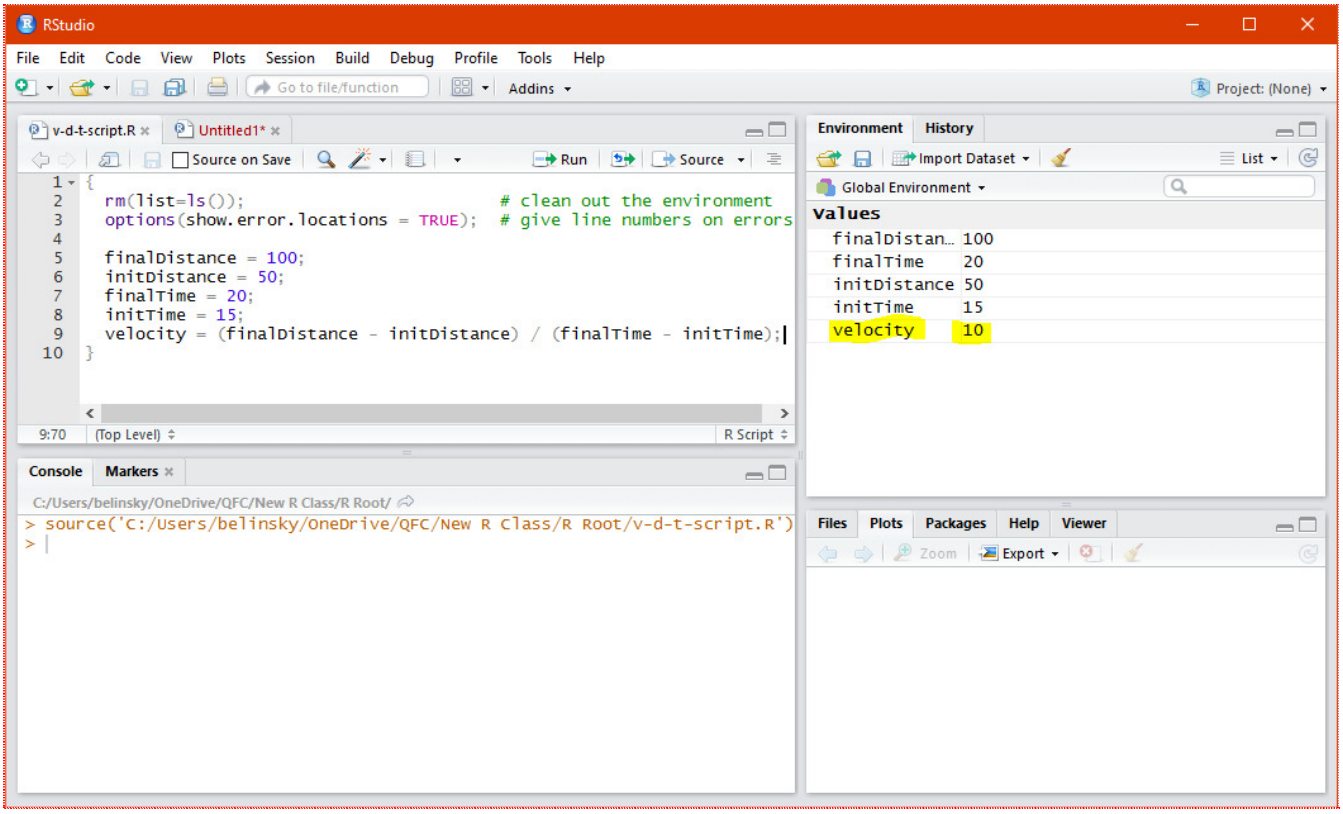

*Fig 2: Misspelled variable name corrected -- script now calculates **velocity**.*

4 - Powers and multiplication

We will look at one more formula that relates ***kinetic energy*** to ***mass*** and ***velocity***:

$$E_k = \frac{1}{2}mv^2$$

There are two new issues with coding this formula:

1) the square function is a superscript -- you cannot use superscript characters in R
2) the implicit multiplication -- we know that values on the right-side are being multiplied but there is no multiplication sign

4.1 - Dealing with parenthesis and multiplication

So let's first pull the ***one-half*** out of fraction form and into division form:

$$E_k = 1/2mv^2$$

We need to be more explicit because this formula could be misinterpreted as $E_k = 1/\left(2mv^2\right)$ so we need to put the one-half in parenthesis:

$$E_k = \left(1/2\right)mv^2$$

Next we will explicitly put in the multiplication symbols -- a necessity in programming:

$$E_k = \left(1/2\right) * m * v^2$$

Trap: Forgetting multiplication symbol

And then change the symbols to script-friendly variable names:

$$kineticEnergy = (1/2) * mass * velocity^2$$

4.2 - Dealing with powers

We still have to deal with the square power. The easiest way in script is to just multiply a second velocity:

```
# this does work but it is not pretty
kineticEnergy = (1/2)*mass*velocity*velocity;
```

The above solution is limited and useless for powers in root form (e.g., square roots, cube roots). We want a more robust solution that works for all powers and roots. In R we use the (**^**) symbol to represent a power:

```
# much more robust solution
kineticEnergy = (1/2)*mass*velocity^2;
```

While the above works correctly, it is often helpful to be explicit and add parenthesis around the value or values that are getting raised to a power:

```
# more explicit solution
kineticEnergy = (1/2)*mass*(velocity)^2;
```

5 - The power operator (^)

The (**^**) operator works for all powers including square roots, cubed roots, or mixed power (e.g., raising to the 3/2 or 5/3).
Let's rearrange the ***kinetic energy*** formula to solve for ***velocity***, *which requires a square root*

$$v = \sqrt{\frac{2E_k}{m}}$$

To put the above formula into a script form, we need to:

1) Put the numerator and denominator on one line by taking out the fraction and replacing it with a division sign.

$$v = \sqrt{(2E_k \ / \ m)}$$

2) Be explicit and put in multiplication symbols.

$$v = \sqrt{(2 * E_k \ / \ m)}$$

3) Spell the formula out using friendly variable names:

$$velocity = \sqrt{(2 * kineticEnergy \ / \ mass)}$$

4) Use the power operator (^) to square root the whole formula. Square rooting something is the same as saying "raise it to the 1/2 power". Since we are square rooting multiple variables, we need to put the whole formula in parenthesis.

$$velocity = (2 * kineticEnergy/mass)^{1/2}$$

So we have this in R:

```
{
  rm(list=ls()); # clean out the environment
  options(show.error.locations = TRUE); # give line numbers on errors

  kineticEnergy = 50;
  mass = 5;
  velocity = (2*kineticEnergy / mass)^1/2; # still a problem here!
}
```

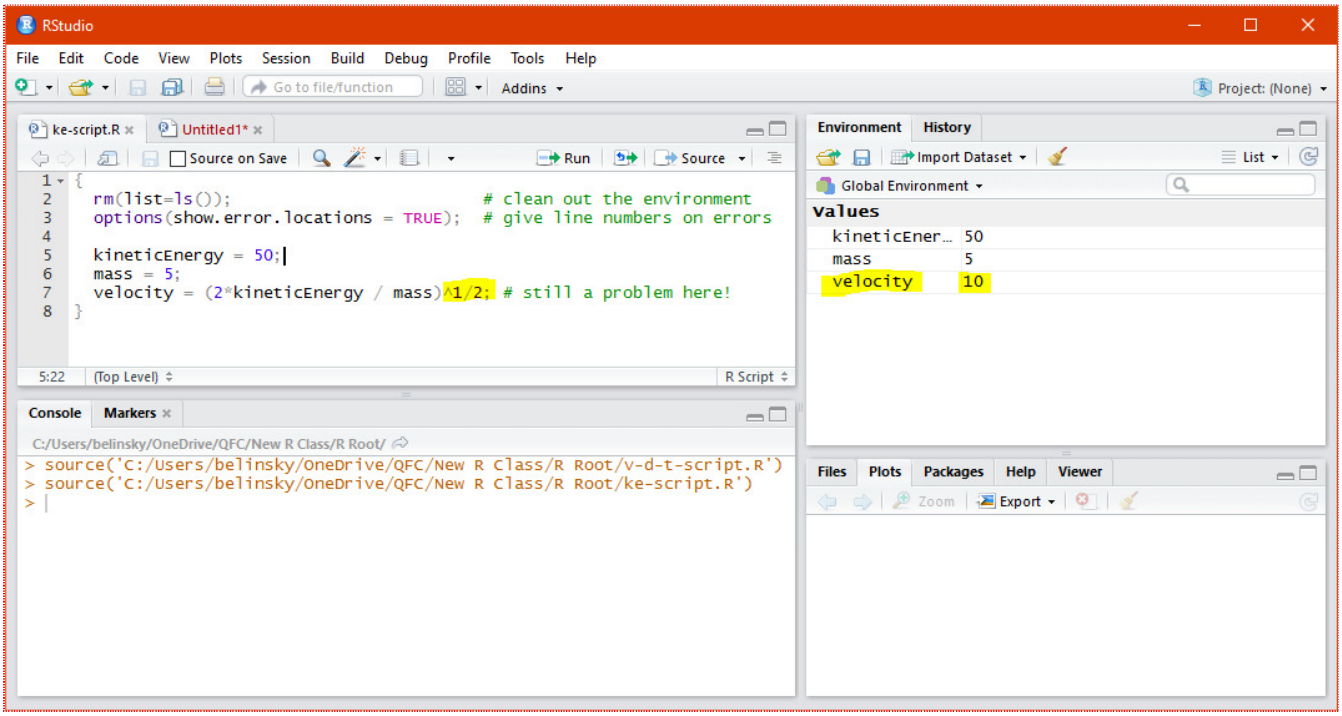

*Fig 3: Incorrect answer for **velocity** because the power is missing a parenthesis*

The Environment Window (*Fig 3*) shows that ***v*** is, unexpectedly, ***10***. This is because of the order-of-operations. Instead of raising the ***(2*kineticEnergy/mass)*** to the ***1/2*** power, the above code raised ***(2*kineticEnergy/mass)*** to the ***first (1)*** power and then divided everything by ***2***. We need to be more explicit and put the ***1/2*** in parenthesis.

Extension: Alternate way to compute square roots

```
{
  rm(list=ls()); # clean out the environment
  options(show.error.locations = TRUE); # give line numbers on errors

```

```
  kineticEnergy = 50;
  mass = 5;
  velocity = (2*kineticEnergy / mass)^(1/2); # now we are good!
}
```

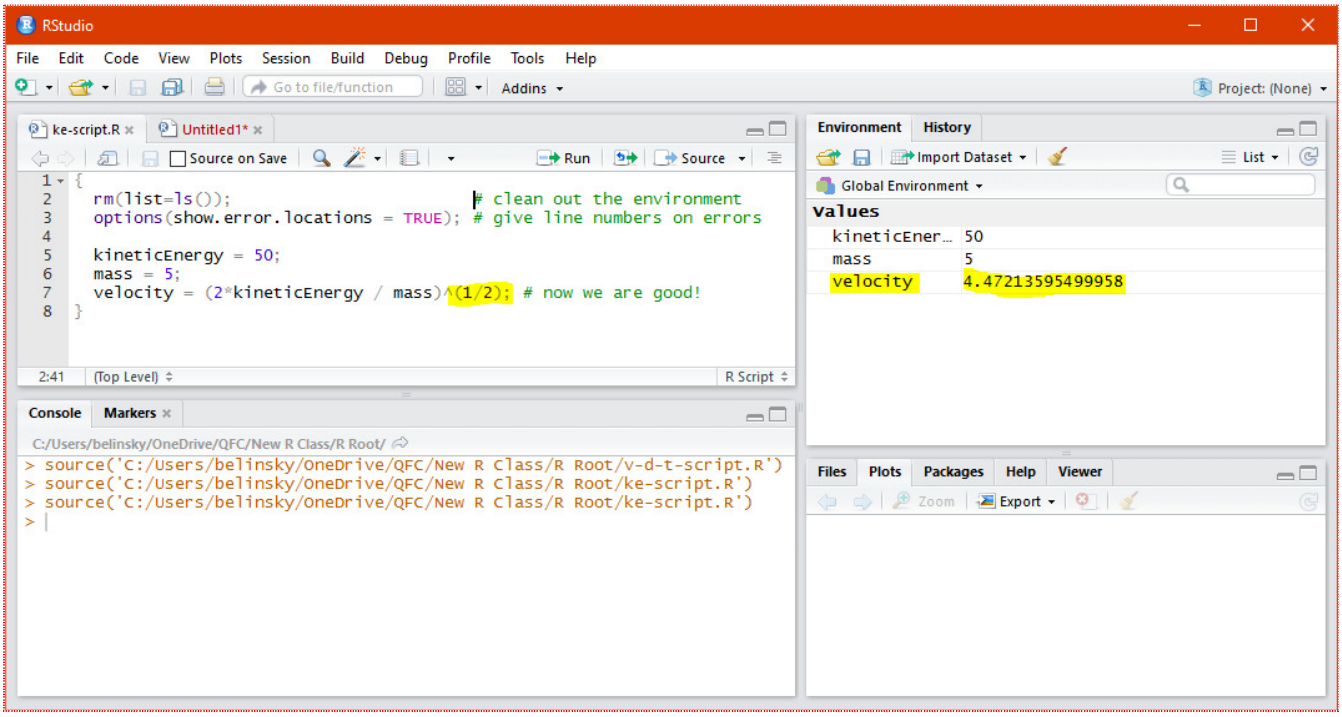

*Fig 4: Correct answer for **velocity***

This style will work for all powers and roots:

```
{
  rm(list=ls()); # clean out the environment
  options(show.error.locations = TRUE); # give line numbers on errors

  kineticEnergy = 50;
  mass = 5;

  test1 = (2*kineticEnergy / mass)^(1/3);  # third root
  test2 = (2*kineticEnergy / mass)^(5);    # fifth power
  test3 = (2*kineticEnergy / mass)^(5/3);  # mixed root and power
  test4 = (2*kineticEnergy / mass)^(3.17); # decimal power
}
```

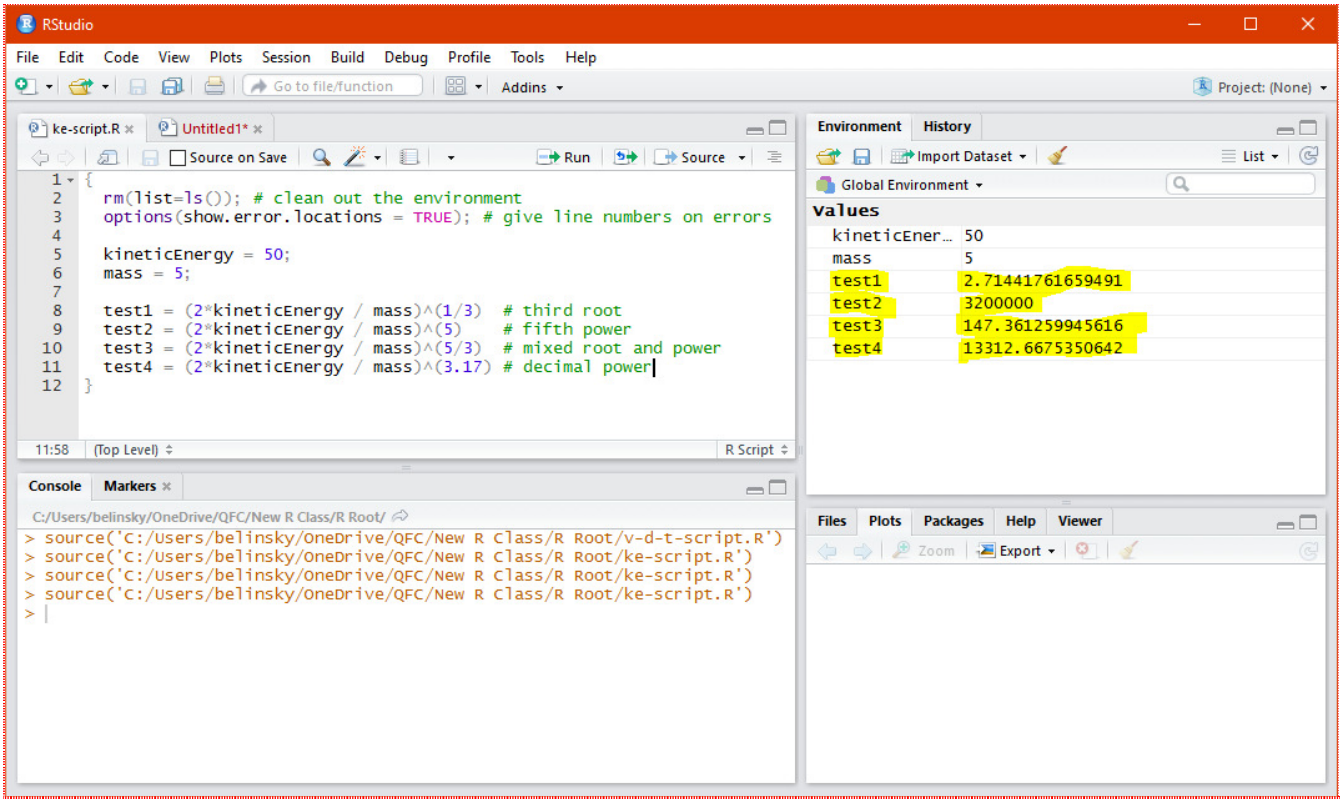

Fig 5: Testing different powers in R

6 - Assignment

For the following five fish length values (in centimeters): 25, 26, 20, 22, 32

Trap: Units, or lack thereof, in programming

A) Calculate the (a) mean, (b) variance, and (c) standard deviation of the five values

B) Make sure the 5 values, their mean, their variance, and their standard deviation appear in the Environment Window after the script is executed.

Visit this page if you need a reminder about how to calculate mean, variance and standard deviation

Note: there are quicker ways to solve for mean, variance, and standard deviation in R, which we will learn later.

7 - Traps: Forgetting Multiplication Symbol

Let's say you are solving for kinetic energy:

$$E_k = \frac{1}{2}mv^2$$

And you have a value for velocity (***v***) and mass (***m***)

```
1 {
2   rm(list=ls());                        # clean out the environment
3   options(show.error.locations = TRUE);  # give line numbers on errors
4
```

```
5   m = 100;
6   v = 10;
7   KE = ????;
8 }
```

If you make line 7:

```
7   KE = 1/2*mv^2;
```

Then you will get the error: "*object 'mv' not found*" in the Console Window because R does not realize you want to multiply the variables ***m*** and ***v***, it thinks you are trying to use a variable named ***mv***, but the variable ***mv*** does not exist.

If you make line 7:

```
7   KE = (1/2)m*v^2;  # same error arises if you do 1/2m*v^2
```

you will get the error: "*unexpected symbol*" (*Fig 6*) and the Console Window will point to the ***m***

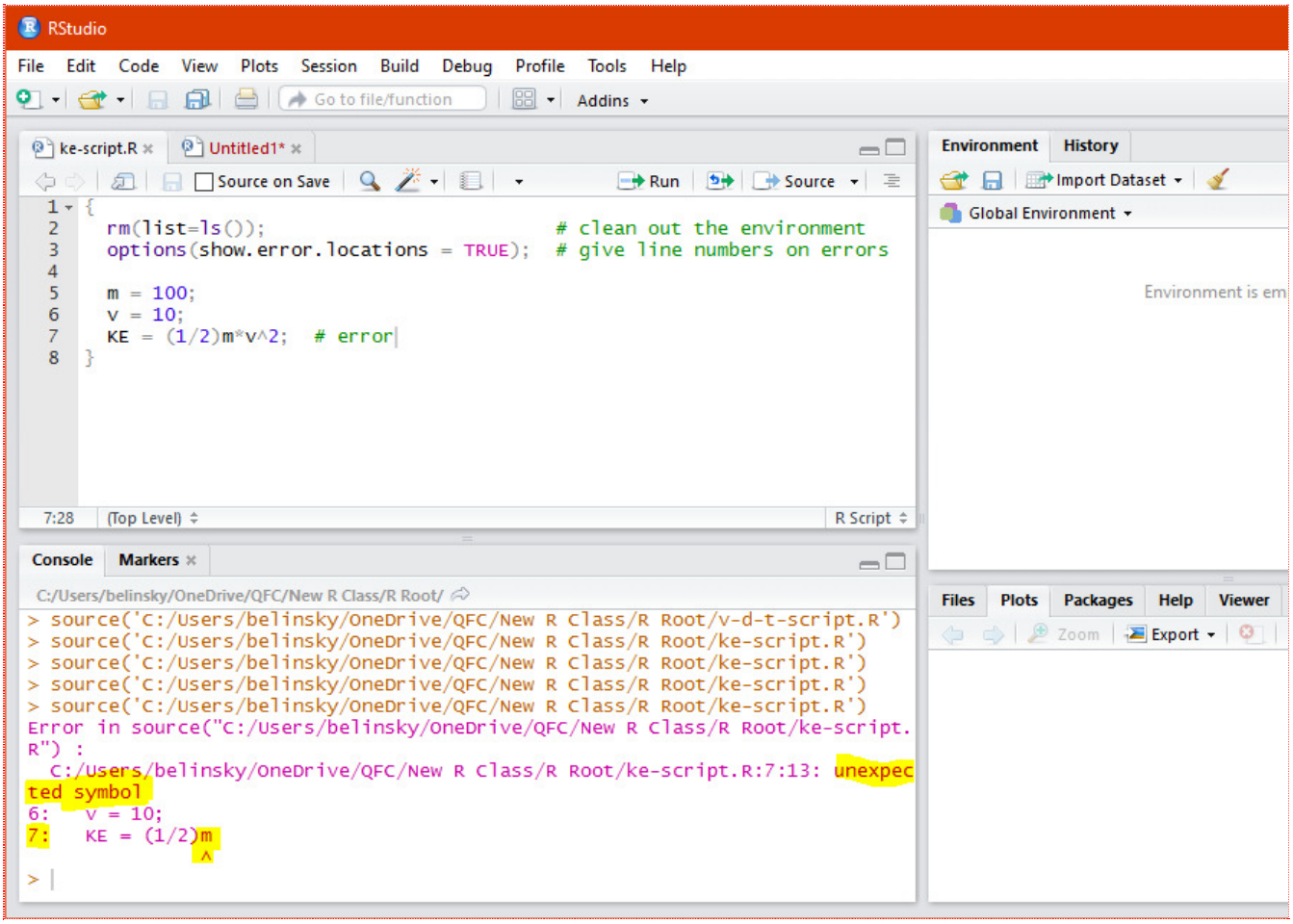

Fig 6: Unexpected symbol error

In this case, R is expecting an operation after **(1/2)** that is represented by a symbol. ***m*** is a variable, not a symbol, hence the error.

8 - Extension: Alternate way to compute square roots

In R, a square root can be computed by raising something to the one-half power:

```
1  # the generic way to compute powers and roots
2  velocity = (2*kineticEnergy / mass)^(1/2);
```

There is also an explicit R function that does square root:

```
1  # the standard way to do powers and roots
2  velocity = sqrt(2*kineticEnergy / mass);
```

This works and you will see this method often used in R Scripts. However, this method only works for the case of square root so I prefer using the (**^**) operator, which is a much more robust operation than ***sqrt()***.

9 - Trap: Units, or lack thereof, in programming

One problem that crops up quite often in programming is that none of the numbers used in calculations have units. So we often have lines of code without any mention of units like this:

```
1 {
2   # find an average of the following three weights
3   weight1 = 175;
4   weight2 = 200;
5   weight3 = 210;
6
7   aveWeight = (weight1 + weight2 + weight3) / 3;
8 }
```

And if we add units to the number...

```
1 {
2   # find an average of the following three weights
3   weight1 = 175lb;  # causes "unexpected symbol" error
4   weight2 = 200lb;
5   weight3 = 210lb;
6
7   aveWeight = (weight1 + weight2 + weight3) / 3;
8 }
```

We get the error *"unexpected symbol"* on line 3 because R is expecting some sort of operation after the number **175** and **lb** is not a valid operation. Lines 4 and 5 would also cause an *"unexpected symbol"* error but R ceases executing at the first error.

It is best to mention the units somewhere in the comments especially if your script is large or others are using your script.

```
1 {
2   # find an average of the following three weights (all in pounds)
3   weight1 = 175;
4   weight2 = 200;
5   weight3 = 210;
```

```
6
7   aveWeight = (weight1 + weight2 + weight3) / 3;
8 }
```

Otherwise, you risk a situation like the Mars Climate Orbiter crash, which could have easily been avoided with proper comments.