**App 1-5:**

I want you to get the 5 temperatures from the user and then the script calculates the average of the 5 values inputted from the user (so, it will be different each time).

Now it works. Keep your spacing consistent!

**App 1-6:**

Same issue as App 1-5

Also, you are not outputting the commas between the numbers. It is a small detail but it is good practice because it increases the complexity of scripting the output statement (because you have to do much more switching between string output and variable output)

Still don't have commas in the output…

**App 1-7:**

You have a logical issue here. You have an embedded ***if*** condition that will always be FALSE because to get to that condition (temp > 80) you must pass the first condition (temp < 30) – and if you pass the first condition, you cannot pass the second.

if(maxtempToday < 30)

{

cat ("brrrrrrrrrr");

# the if statement is embedded in the previous if statement

if(maxtempToday >= 80) # this condition can never be met here!

{

cat ("enjoy the sunshine");

}

}

You have the exact same issue with your other ***if*** statement. I would first do this script without using the embedded ***if*** statements. You basically need four ***if*** statements. After you figure that out, I would challenge you to do the equivalent script with embedded ***if*** statements.

if(some condition)

{

}

if(some other condition)

{

}

Also, the curly brackets on lines 15 and 16 are not needed. There are not hurting anything (sort of like extra parentheses in an algebraic formula) but they are confusing because they hint that your code is sort of split at line 15 (and it is not)

Works now… this was not taught until 1-8 but you could do line 11 like this:

else if(maxtempToday >= 80)

{

cat ("enjoy the sunshine");

}

It is better this way because maxTempToday >= 80 is mutually exclusive with maxTempToday < 30, so there is no point in executing line 11 if line 6 evaluated to TRUE.

**App 1-8 (part 1):**

**Keep your spacing consistent**

Lines 11 and 25-28 need to be adjusted. Yes, I am a tyrant about spacing – it is a life saver as your code gets longer and more complex.

**Small fix to make:**

If you type in ***-20***, you get no response from your script.

**And a challenge:**

Try to create an ***if-else-if*** structure with the same functionality that only has 1 check for an invalid statement (you currently have 2).

Looks good – including the challenge!

**App 1-8 (part 2):**

Works. Another way to do the code:

if (grep(x=grade, pattern="a|A"))

{

cat ( "90-100");

}

I only point this out to whet the appetite for ***grep()***, which is incredibly powerful. ***grep()*** is the answer to the question: How do I deal with text input where people are not consistent with how they type things in?

**App 1-9:**

I want you to rethink this script and put all the ***readline()*** inputs from the user at the top of the script. After the script gets the inputs, then it does the checks on the inputs.

In general, when you design a script it is best to put the variables you are inputting (whether from a user, database, csv file, or another package) at the top of your script. The variables are like resources for your script and it is easier to read and tweak code if one can see all the resources the code needs at the top of the script.

This is sometimes impossible but in this it is not.

Also, it looks like you have two scripts, one starting at line 1 and the second at line 47. You should only have one. I know that the typical R programmer breaks scripts up into blocks that they individually execute. I would love to break R programmers of this habit – it makes code harder to share and debug.

The code works but I would change one thing (parentheses in red)…

if( (fishWeight >20 && fishWeight <100) &&

(fishlocation == "north" || fishlocation== "south") )

{

cat ("\nbonus fish");

}

In this case the order of operation worked the way you wanted it but the rules when it comes to && and || is not always intuitive so it is best to be explicit with added parentheses (just like algebra)

**App 1-10:**

The error statement (fishlocation != "north" && fishlocation != "south"), in this case, would be best as the first check in the ***if-else*** structure. This is because there is no point in checking ***fishage*** if the ***fishlocation*** is not valid.

Keep you spacing consistent (here I go again…):

***fishAge*** and ***fishLocation*** (lines 3 and 4) should be visually at the same level as the ***rm()*** above and the ***if()*** below. Also, the last ***else if*** should be at the same level as the else if above.

Use the more standard variable naming conventions: ***fishAge*** or ***fish\_age*** instead of ***fishage***

The extra parentheses in line 7:

if ( (fishage >3 && fishage <5) && (fishlocation == "north"))

are not needed, but I am not complaining. I think it makes it easier to read what is going on.

You are missing semicolons on two lines. Yeah, R works without the semicolons, but I feel it is important to understand where end-of-statements are. If you move to C programming, then it is necessary.

**Semicolon usage (kind of tricky…):**

else if ((fishage >=5) && (fishlocation =="north"))# no semicolon here

{

cat("category III"); # semicolon here

}

Just extra material -- pointing out something I have not covered in my lessons. You do not put semicolons at the end of ***if/ else-if*** statements (or ***for*** loop). This is because they have curly brackets to indicate an attached codeblock and, hence, is not an end-of-statement. Basically, if a statement has a curly bracket it does not end in a semicolon

**Challenge:** You can combine the two ***else if*** statements on lines 15 and 19 into one ***else if*** statement. This is a bit tricky, but I want to see if you can see it. Note: this becomes easier if you move the location error statement up to the top of the ***if-else*** structure

**App 2-1:**

Line up your curly brackets { } vertically so the close bracket is at the same horizontal position as the open one.

**App 2-2:**

Make it so your ***Console*** output is all lined up (you have one extra space in there…

Every line of this script is at the same level so they should all start at the same level – you are only one space off!

Put your semicolons in!

**Subsetting a data frame:**

You could do line 5 as:

precipitation=weatherData$precipitation; # new notation

precipitation=weatherData[ ,4]; # old notation

precipitation=weatherData[ ,"precipitation"]; # alt old notation

I am changing my lessons to move to the new notation for two reasons:

1. The old notation does not work in tidyverse dataframe (called tibbles) – and you never know when you are using tidyverse tibbles (in this class you don’t, but they are used quite a bit)
2. RStudio gives very useful suggestions as soon as you type the dollar sign ($).

I do like the row/column visual of the old notation but that does not offset the two advantages above.

**App 2-3:**

The script works but I want you to design the script file so that it looks like one script. If you want to show that a section of code is for a particular question, then use comments instead of curly brackets.

Your 2-3 looks like 3 scripts (starting at lines 1, 12, and 23). Also, remove any lines that are repeats.

Also, you might want to get used to subsetting data frames like this:

weatherData$precipitation[i] # precipitation is column 4

instead of

weatherData[ i,4]

There are a few reasons:

* the first method is the only one the works on tidyverse data frames (called tibbles)
* RStudio gives suggestions as soon as you type ***$***
* using the column name instead of number is safer if the columns change order

This is a change in my philosophy since I did this class, and the next version of this class will use the first method.

**App 2-4:**

This had widely been considered the hardest lesson in this class, but it is important to learn how to effectively do for loops with if statements inside.

In your script, your for loops is not doing much – it only consists of one line

for(i in 1: lengthVector)

{

numberTotal=numberTotal +numbers[i]; #the only line in the loop

} # this ends the for loop

So, all the if statements underneath are independent of the for loop (i.e., they are being executed only once.

The value of i after the for loop is the last value it was in the for loop. In the environment Window you can see the value of i = 6. This means that the if statements are only checking the 6th value in the vector.

I would first try to use a for loop to handle individual tasks (find the mean, find the max…) then see if you can combine them into one for loop.

**App 2-5:**

Looks good, another way to write line 14:

weatherData[ ,"deltaHighTemp"]=changeinHighTemp;

is

weatherData$deltaHighTemp=changeinHighTemp;

**App 2-6:**

I actually meant for you to remove the highTemp column and save the rest of the data frame to a new data frame (not create a data with only the highTemp column).

So, you were trying something a little harder! Way to overachieve! (I should also better phrase the question…)

The reason it did not work was because R assumed, because there was only one column, that you just wanted a vector instead of a data frame.

This creates a vector with the values from the 4th column

weatherDataHighTemp=weatherDataHighTemp[,4];

If you want to create a one-column data frame you need to be explicit:

weatherDataHighTemp=as.data.frame(weatherDataHighTemp[,4]);

**App 2-7:**

Script works. Now I would like you to start adding more vertical space in your code to distinguish blocks of code – in other words, don't put code on every line. This is definitely more of an art than a science!

So, I wrote this before doing my GGPlot class and I now believe that GGPlot is the way to go for many, many reasons. The biggest reason is that GGPlot does a much better job of structuring a plot and this becomes vital when you get into advanced topics like Animations and RMapping.

So, my next class will switch R Base to GGPlot. You will have access to that… or you can take my GGPlot class!

**App 2-8:**

Your mass conversion function definitely uses a brute-force method!

Try redoing the function where there are only 3 inputs:

* the value to convert (a numeric value)
* the unit it is it (a string value)
* the unit to convert it to (a string value)

For the other function… I learned something new here!

for ( i in 1:length(highTemp))

{

changeinHighTemp[i] = highTemp[i]-highTemp[i-1];

}

The first time through the for loop, i=1 so it is:

changeinHighTemp[1] = highTemp[1]-highTemp[0];

But, highTemp[0] is not valid!

Yet, R proceeds anyways without an error – it just marks the value as NA. I wonder if this is a change between R v.3 and R v.4 because I do not remember this.

Anyway, your code works but I would rather see your code explicitly deal with the 1st value – so as to not leave room for error!

One way to do this is to set the first value in the vector to NA before the for loop and just have the for loop run from **2** to **length(highTemp)**

**App 2-9:**

**App 2-10:**

The reason you are getting multiple invalid condition statements is because you are executing it every time you go through the for loop, which is for every value in the vector you are checking:

for(val in 1:vecLength) # for each value in the vector

{

if(!(conditionalOp == "==" ||conditionalOp == ">" ||

conditionalOp == "<" ||conditionalOp == ">=" ||

conditionalOp == "<=" ||conditionalOp == "!="))

{

cat("sorry invalid condition"); # will repeat for each value in the vector

}

…

}

There are two way to fix the problem

**The cheap way (add a *break* statement):**

for(val in 1:vecLength) # for each value in the vector

{

if(!(conditionalOp == "==" ||conditionalOp == ">" ||

conditionalOp == "<" ||conditionalOp == ">=" ||

conditionalOp == "<=" ||conditionalOp == "!="))

{

cat("sorry invalid condition");

break; # break completely out of the for loop

}

…

}

**The robust way:**

Check the ***conditionOp*** ***before*** you enter the for loop. In other words, the for loop will not even execute if the ***conditionalOp*** is not valid (so, the for loop is inside the ***if*** statement). This is the far better way to go but also more challenging. I'll let you try it out if you want – this is an important to learn that is slightly beyond the scope of this class. Feel free to ask more questions about this.

**App 3-1:**

I wish R had a strict mode that forced the use of semicolons because they give you better understand of program flow – and they are mandatory in other languages (and embedding C in R can be very useful). You are pretty close on your use of semicolons. The last line could use one.

I do not like the fact that R determines whether an end-of-line (i.e., Enter) ends a statement or not. It is better for the programmer to decide end-of-statement with a semicolon (even if they get it wrong sometimes)!

OK, that was an opinion piece.

**App 3-2:**

Something I will change in my next version of the class:

You can subset a List (e.g., ***lansingWeather***) using ***$*** just like you would subset a column from a data frame.

So, the following are equivalent:

lansingDecWeather=lansingWeather[["data"]];

lansingDecWeather=lansingWeather$data;

Another option is to use single-bracket notation:

lansingDecWeather=lansingWeather["data"];

The single bracket notation means you want to preserve the List. I am not sure how useful that is… but it is there.

**App 3-3:**

On line 71:

dailyMaxAvgWind[x]=max(lansWindMat[ ,x],na.rm=TRUE)

***x***  represent the days and is a value between **1:31**. So, ***x*** represents the rows. But, you have ***x***  representing the column in the line above.

The reason you are getting a subscript out of bound error is that there are only 5 columns in ***lansingWeather*** and you are trying to access columns up to **31** (actually, the error comes as soon as you access column **6**).

After you get that fixed, sending the information to the Console can be as easy as:

cat("The max values are", dailyMaxAvgWind);

But, if you really want to format it nicely, then you might need to send the induvial values:

cat("max for day", x, "is", dailyMaxAvgWind[x]); # inside for loop

Note: still a lot of formatting to do!

**App 3-4:**

Just added some GGPlot equivalents to it so you could see the difference.

**App 3-5:**

***rnorm()*** generates random, but weighted, numbers centered around your mean value.

So,

normalDWind = rnorm(n=200, mean=20, sd=5);

generates 200 random numbers weighted around the mean (20) -- and how close the random values are to the mean is dependent upon the variance.

The reality is that R is not picking numbers randomly (which is something a program cannot do), instead R is using a very complex formula that looks random to the user. Since there is a formula, we can control the "random" output. The main reason to control the output is so that the same "random" numbers are picked every time you execute the program.

***set.seed()*** is used to control the random output (i.e., the same random numbers picked every time).

The ***seed*** number is the starting point.

So, ***set.seed(seed=10)*** and ***set.seed(seed=20)*** will pick a different set of random numbers.

Try running just this code:

set.seed(seed=10);

normalDWind = rnorm(n=200, mean = 20, sd=5);

Each time you run the two lines, ***normalDWind*** will be the same.

Now, switch the seed number:

set.seed(seed=20);

normalDWind = rnorm(n=200, mean = 20, sd=5);

Each time you run the two lines above, ***normalDWind*** will be the same – but different from the previous seed.

If you remove the seed (or never set it to begin with):

set.seed(seed=NULL);

normalDWind = rnorm(n=200, mean = 20, sd=5);

Then ***normalDWind*** will be different every time you execute the two lines above.

**Resampling**

Resampling broadly involves the act of picking random values from a sample of a population. There are many, many, many different methods used in resampling but if you want your script to produce the same "random" values every time then you need to use ***set.seed()*** at the beginning of your script.