Scratch Data Tools General Testing Plan

These steps should be taken any time a new feature is added or bug is fixed to ensure that the system is still functional before merging code into the master branch. This will go through some of the basic functionality of the program as well as line out some steps for testing the core functionality of Map, Filter and Reduce. The user and developer documentation located in the DataTools-scratch-vm repository also contain detailed information on the intended use and behaviors of the Data tools extension blocks, the popups and menus created as well as details on how to start and run the program.

Starting the Program

At this point you should have pulled the code on a feature branch that you are testing, the developer documentation has more information on where to get the source code. If at some point there is an error related to the access of certain files, you will want run the commands again with sudo or as a super user and continue to do so. The three different directories should be set up in this order to keep consistency.

Scratch-blocks → If you have not run scratch on your system at all or a new version of scratch-blocks is available on master then you must follow these steps, otherwise you do not need to repeat these steps each time.

1. Navigate to the DataTools-scratch-blocks directory
2. run the command: “npm install”
3. run the command: “python build.py” (this will take some time to run)
4. run the command: “npm link”

scratch-vm → The vm is where the core functionality behind the blocks and extension lives so it’s important to make sure this is up to date. The run watch command is important here because it allows us to edit the vm and have the program running, when a file is saved the site will then auto refresh which makes making changes quick and easy.

1. Navigate to the DataTools-scratch-vm directory
2. run the command: “npm install”
3. run the command: “npm link”
4. run the command: “npm run watch”

scratch-gui → The gui is where the actual site is hosted from and links together both the vm and blocks.

1. Navigate to the DataTools-scratch-gui directory
2. run the command: “npm install”
3. run the command: “npm link scratch-vm scratch-blocks” (check that the location of the files printed here are the correct file paths for your desired version)

4. run the command: “npm start”

Once all of the command have been run and everything is compiled the site should be available at localhost:8601.

Automated Tests

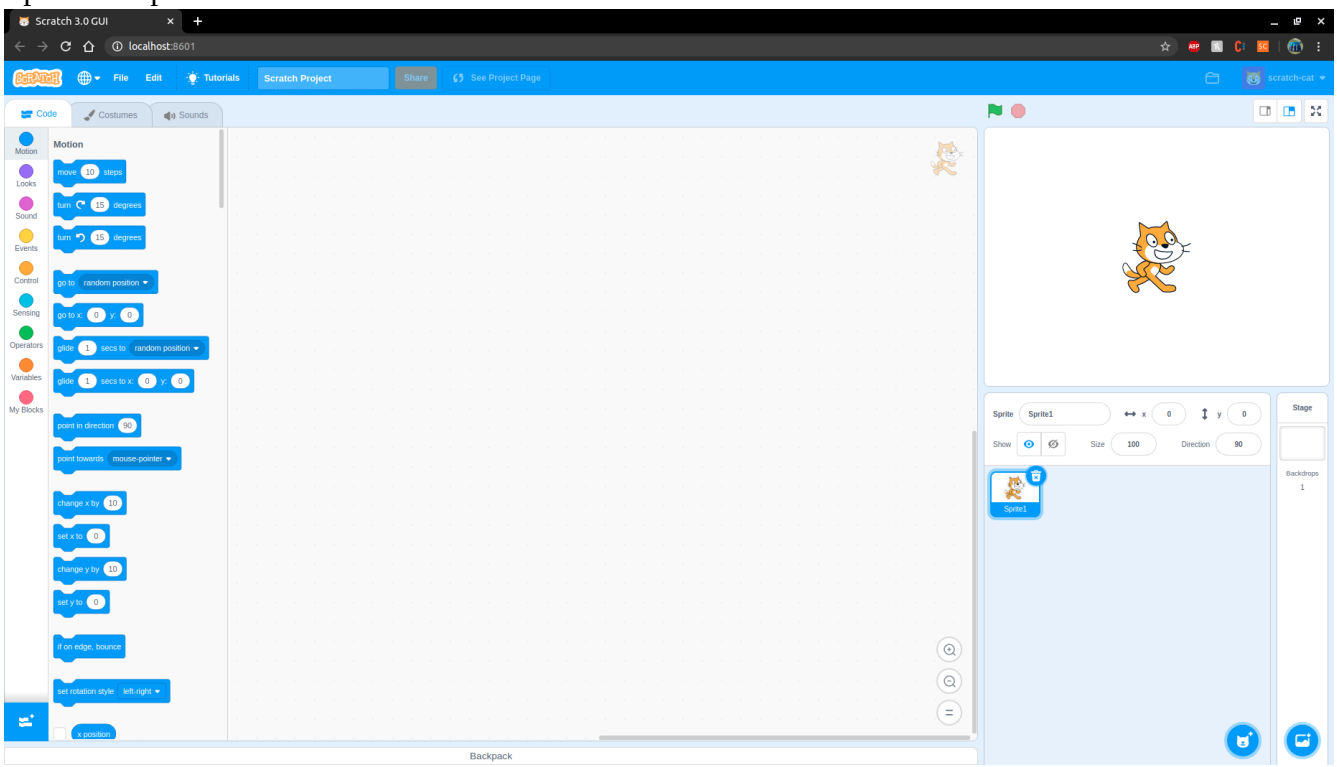
There are some automated tests that have been written for the core functionality of the Data Tools Extension, they are located in DataTools-scratch-vm/test/unit/extension_datatools.js. There are examples of how to write tests if the function being added can be tested thoroughly with automated testing. The steps to run these tests are as follows:

1. Navigate to the DataTools-scratch-vm directory
2. run the command: “npm install”
3. run the command “npm run-script build”
4. run the command “node test/unit/extension_datatools.js”

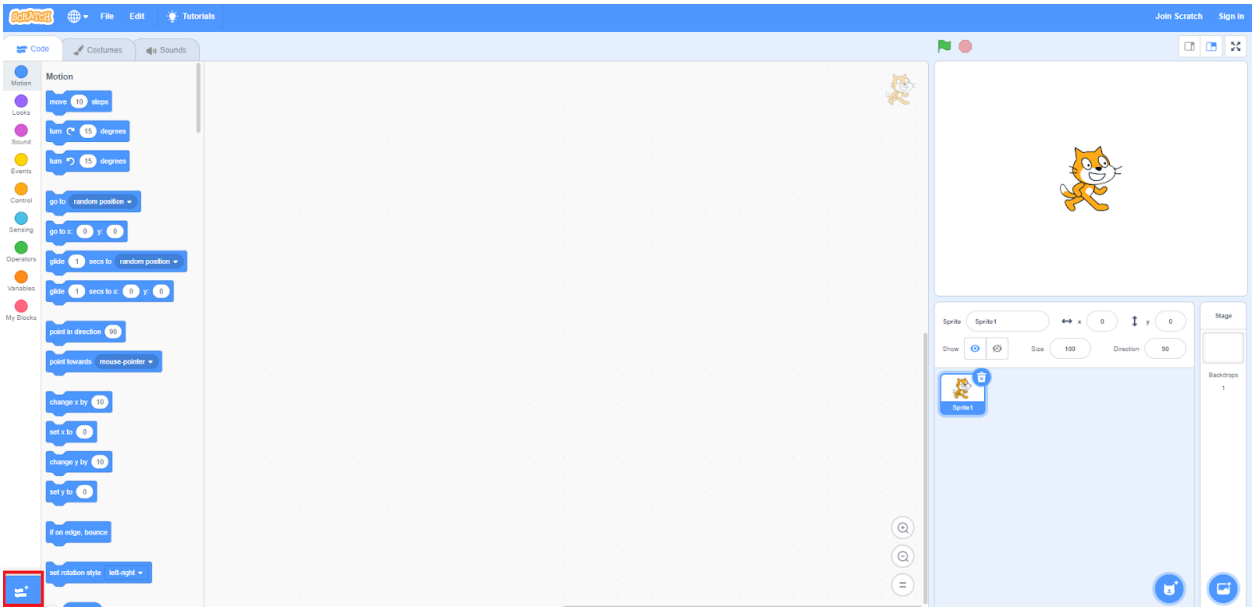
The tests will run and the results should be printed to the console.

General Things to Look For

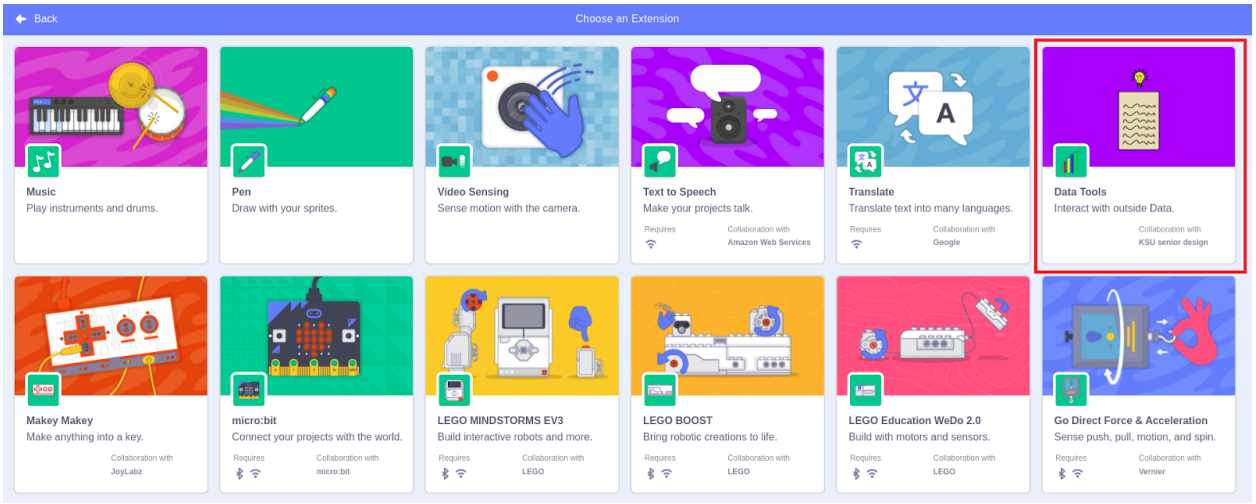
Upon startup the site should look like this:



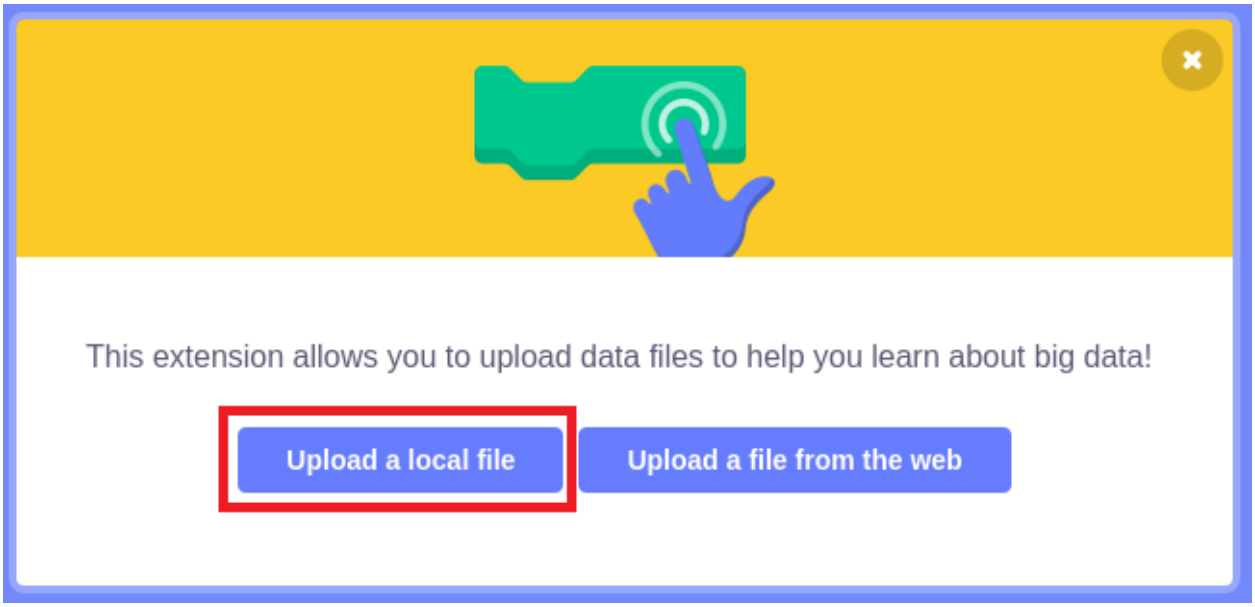
First we want to make sure that the extension is still visible and can be loaded, if there is an error with one of the blocks the extension will not be able to be loaded and the error will appear in the console of the web browser. Load the extension by selecting the blue button in the bottom left hand corner here:



Then select the Data Tools extension from the menu to add the extension:



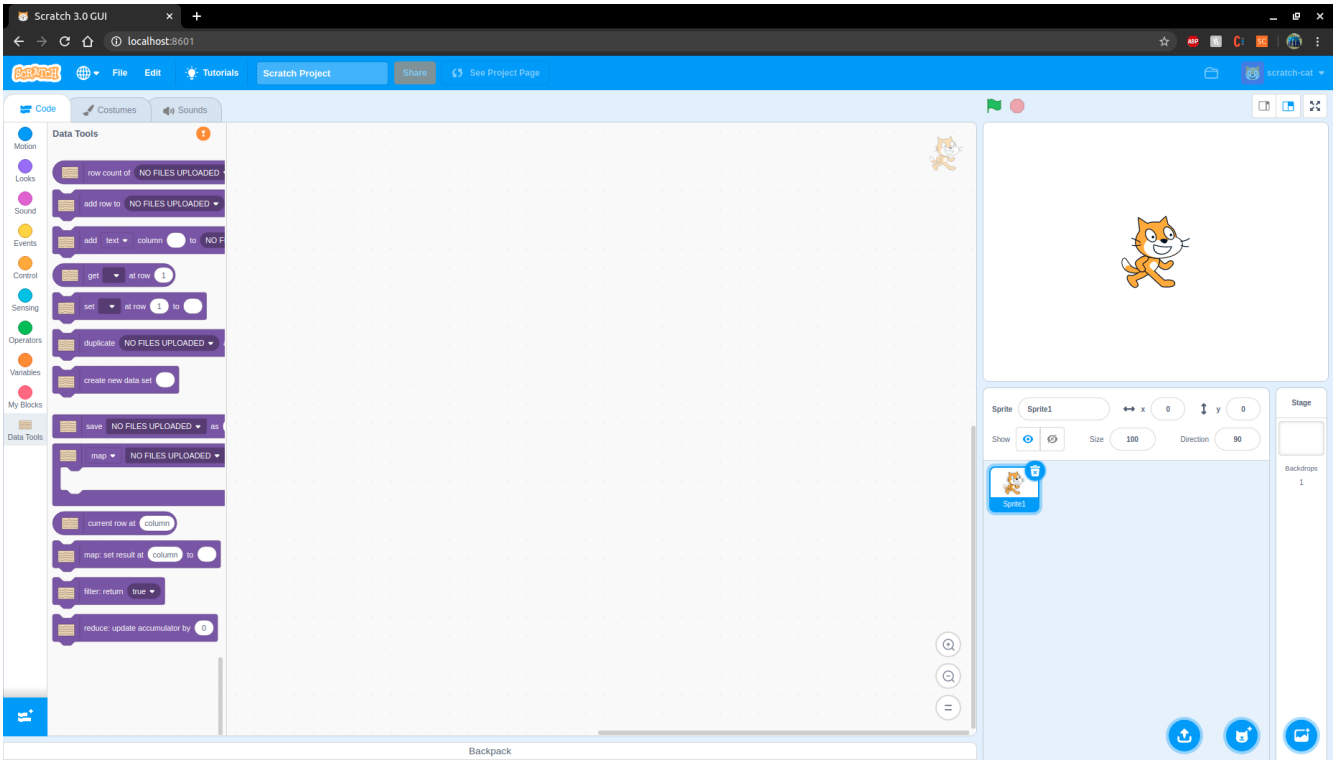
When the extension is first loaded a popup should appear prompting the user to upload a file like so:



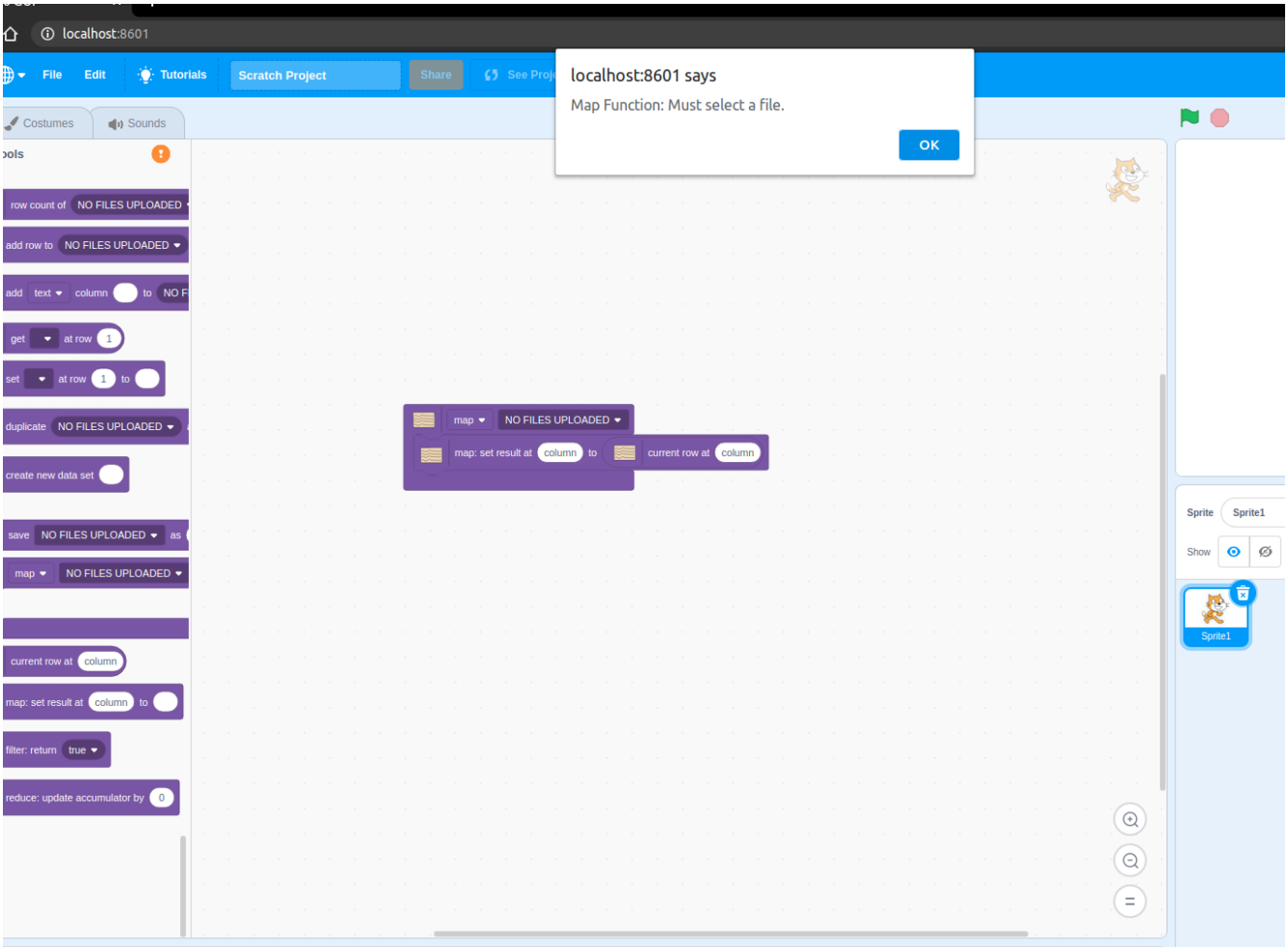
At this point there are two ways to go, either exit out of this window or upload a file. Now there are three general file states that the blocks should handle:

- NO FILES UPLOADED – there are absolutely no files in the system either created by a user or uploaded by a user
- Empty data set – an empty data set is created using the “create new data set” block. This data set has no columns and no rows
- A file with data – A file is either uploaded by the user or created by the user and there is data in the rows and columns of the data set. This also covers the case where the map/filter block is used as a parameter for another block as these blocks return a data set.

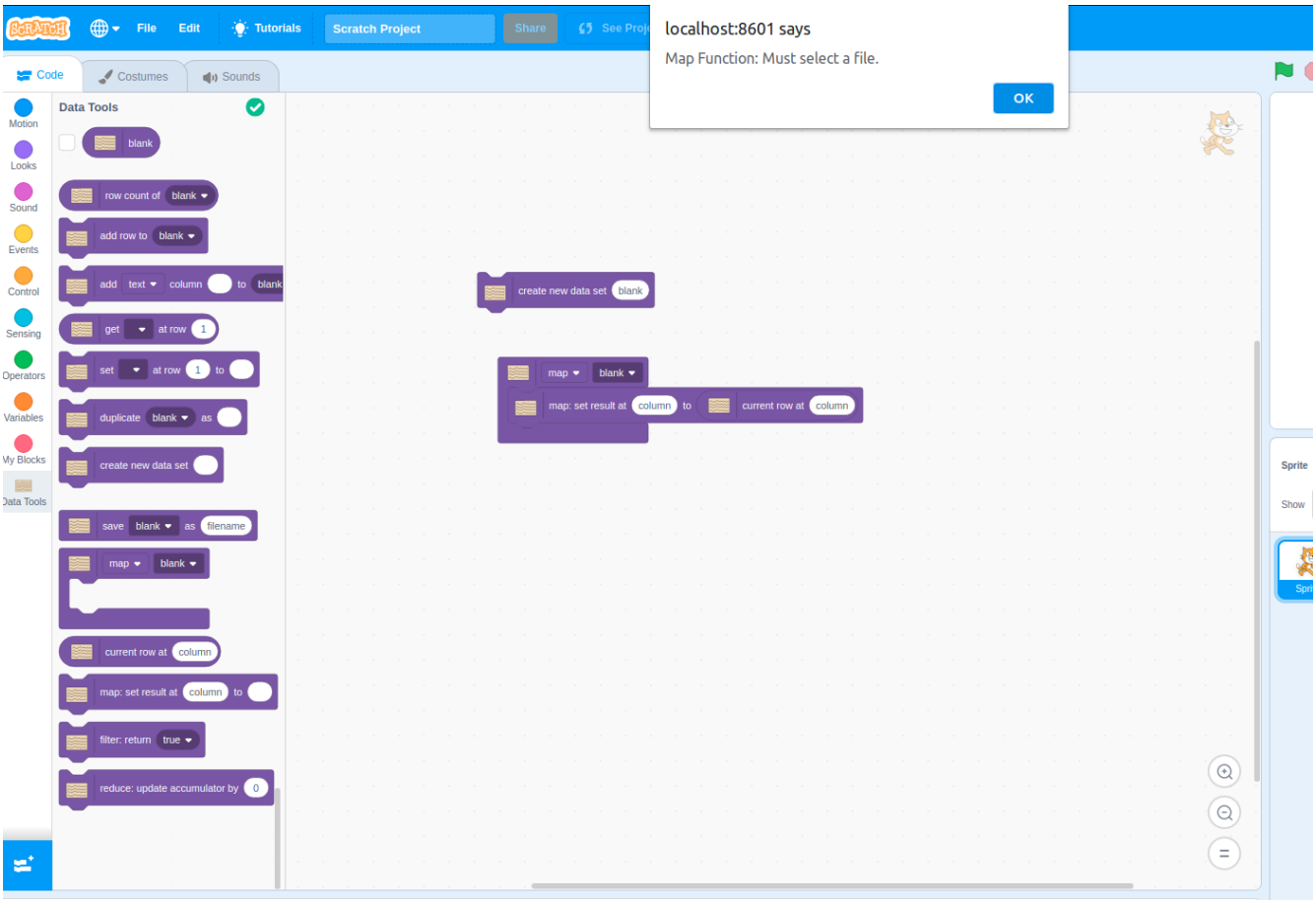
All blocks should be able to handle all three of these file states so it’s usually best to test the first two states first as they are the easiest. Do this by exiting out of the popup without uploading a file. The site should look like this:



Notice the orange circle with an exclamation point above the blocks, this indicates that no files are uploaded. The File menus on the blocks will also read “NO FILES UPLOADED”. Below is a sample test done with map when no files are uploaded and the result of the test:



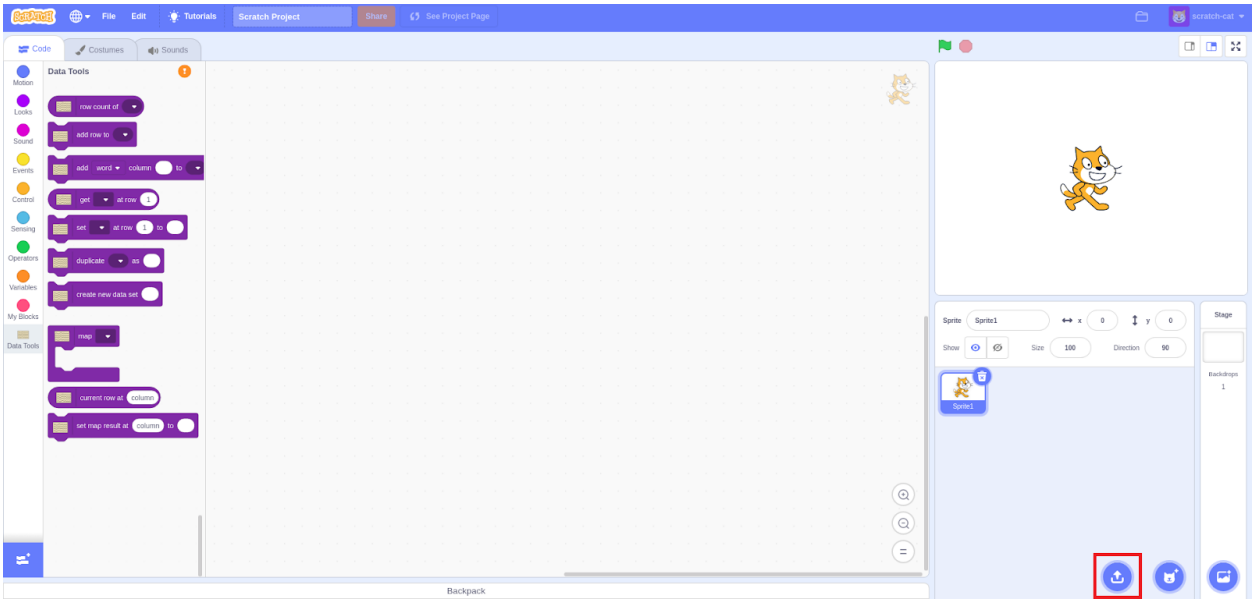
A similar test is done after creating an empty data set, naming it blank, and then trying to perform a map on the data set:



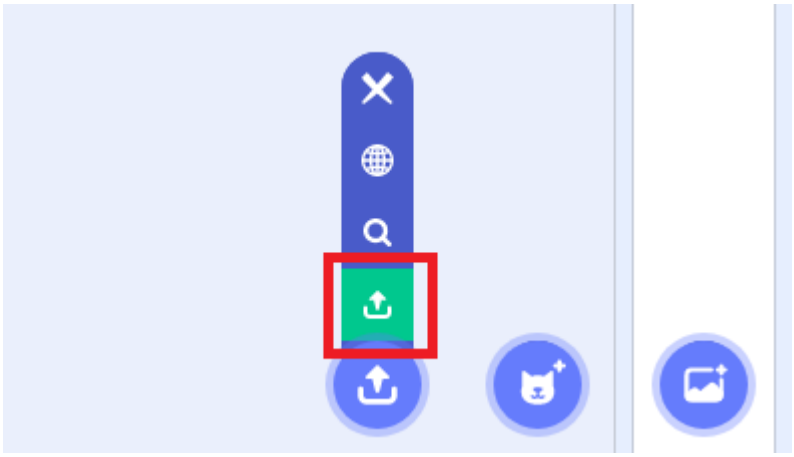
If an error does occur during these two states it typically will cause the system to not allow any other blocks to be ran and an error will be outputted to the console of the browser. There could also be unexpected behavior as well, like the map function above executing when there are no rows or columns to execute on. We will use the Map/Filter/Reduce block to demonstrate how to test the system using a data file.

Specific Testing for Map/Filter/Reduce

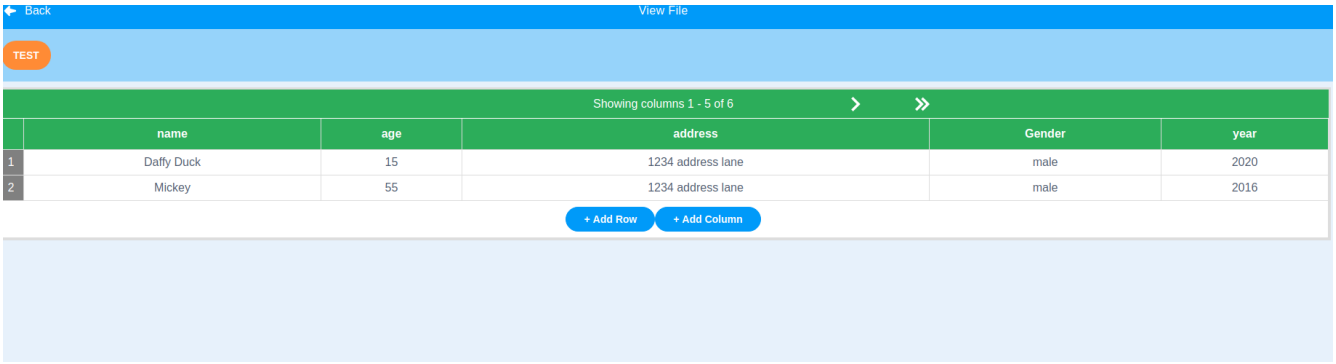
The Map/Filter/Reduce blocks go much deeper than the other blocks in terms of their complexity and functionality so we have elected to test them manually rather than testing them through automated testing. The functionality of all three are housed in the same block using a drop down menu for the user to decide which function they would like to use. We will start by reloading the program and uploading a test file. This can be done by either selecting “Upload a Local File” from the popup that appears when the extension is loaded or selecting “Upload Local File” from the menu that is loaded in the bottom right hand corner seen here:



and then here:



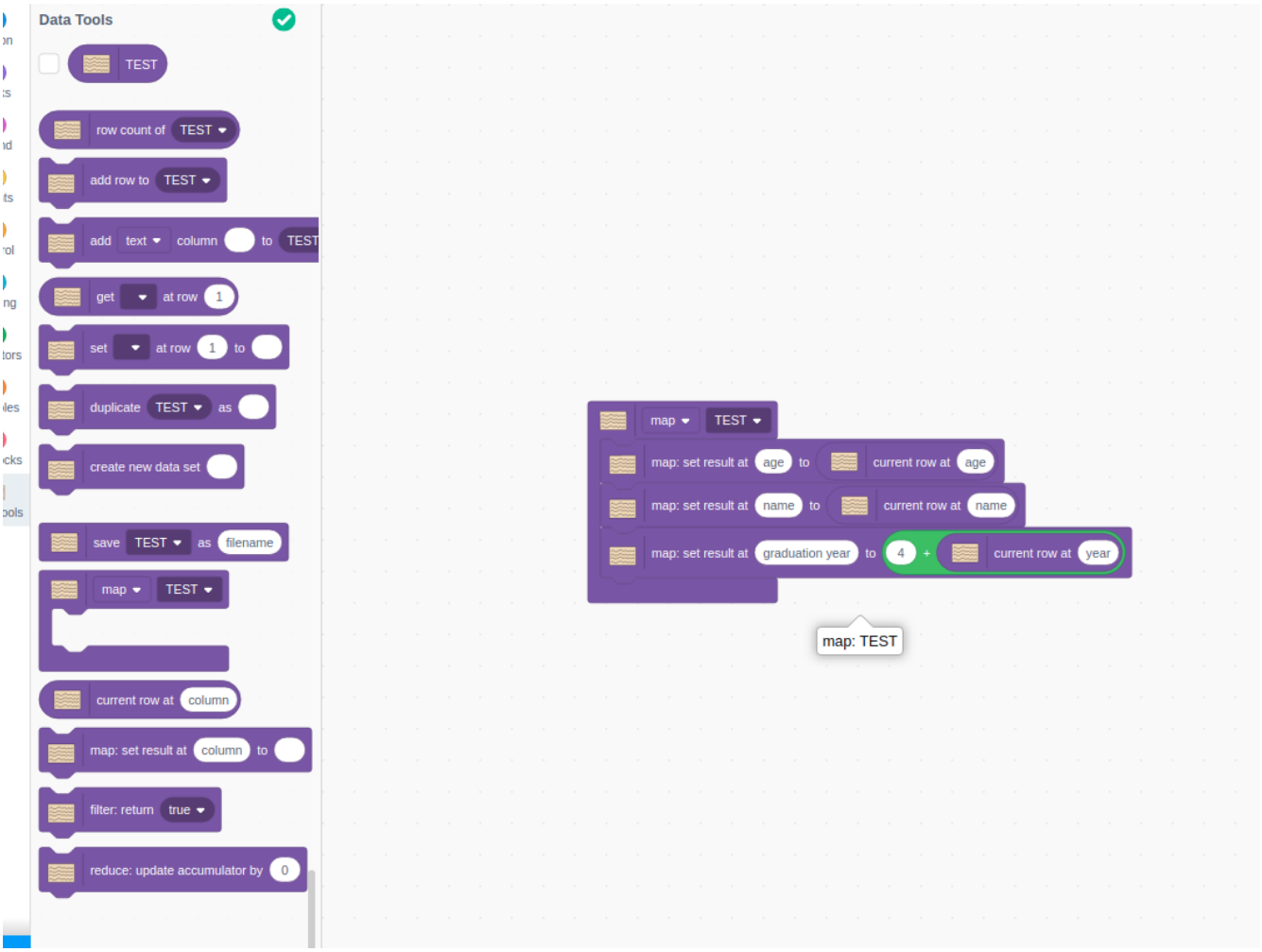
This will open a file dialogue that will allow you to upload a .csv, .xml or .json file that has data inside of it. Typically at this point it is a good idea to view the file using the file viewer by selecting the magnifying glass from the same menu as used above, giving you a view of the file like so:



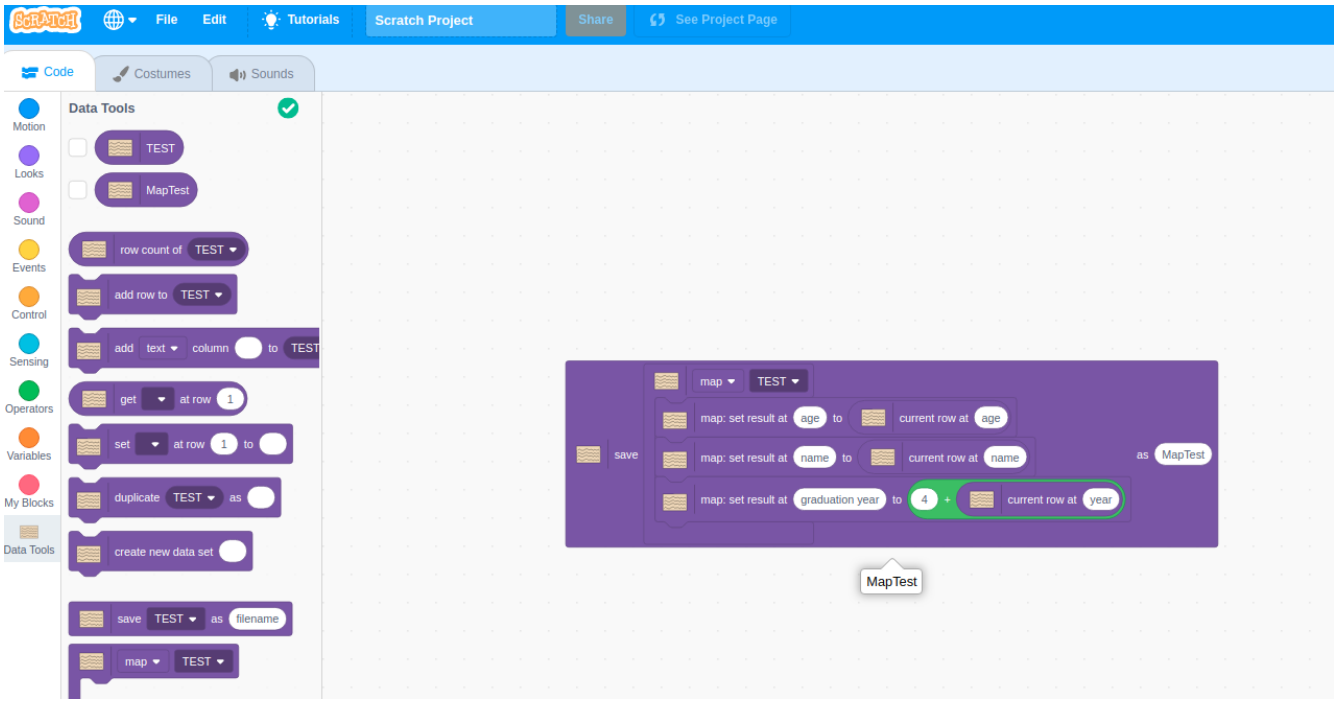
This is just to make sure that the file was uploaded properly and the data is stored correctly. We will start by testing out the Map functionality.

Map

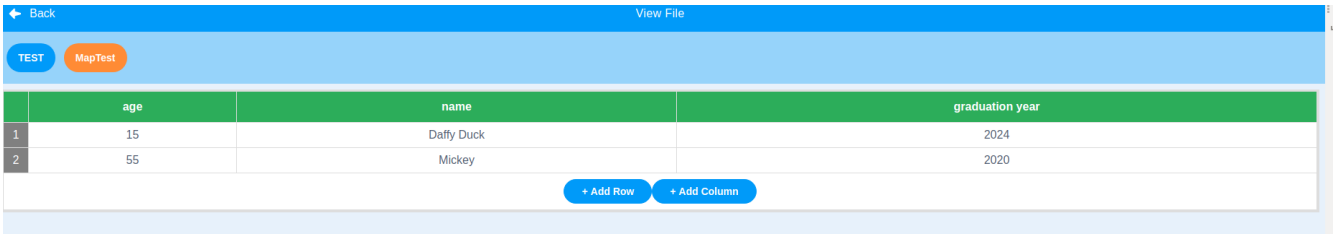
Maps general purpose is to match 2 data fields to each other and return the resulting data set. The default functionality of the Map block is to return this data set but not save it in the system unless the map block is placed inside of a save block. An example of the map block without saving is below:



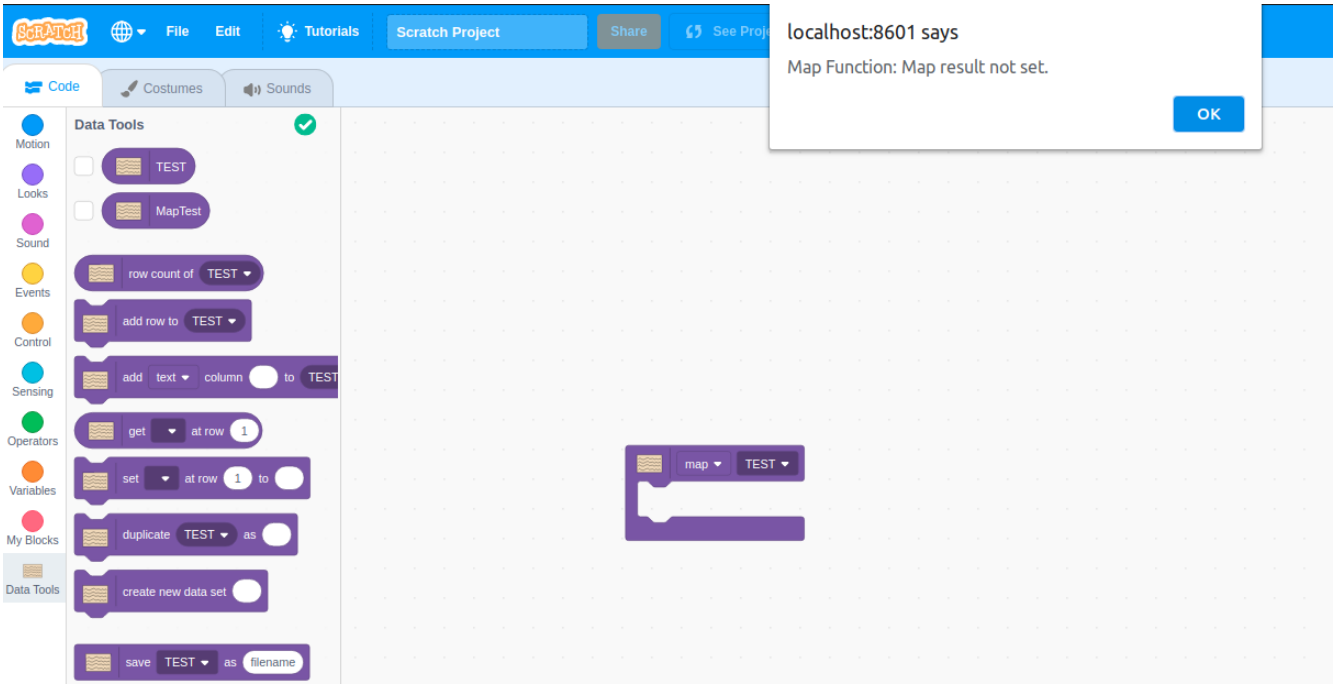
The key thing to notice here is that after running the block, a new data set is not added to the list of data sets above the blocks. However, if we put the map block into the save block, we get a result that we can view and verify:



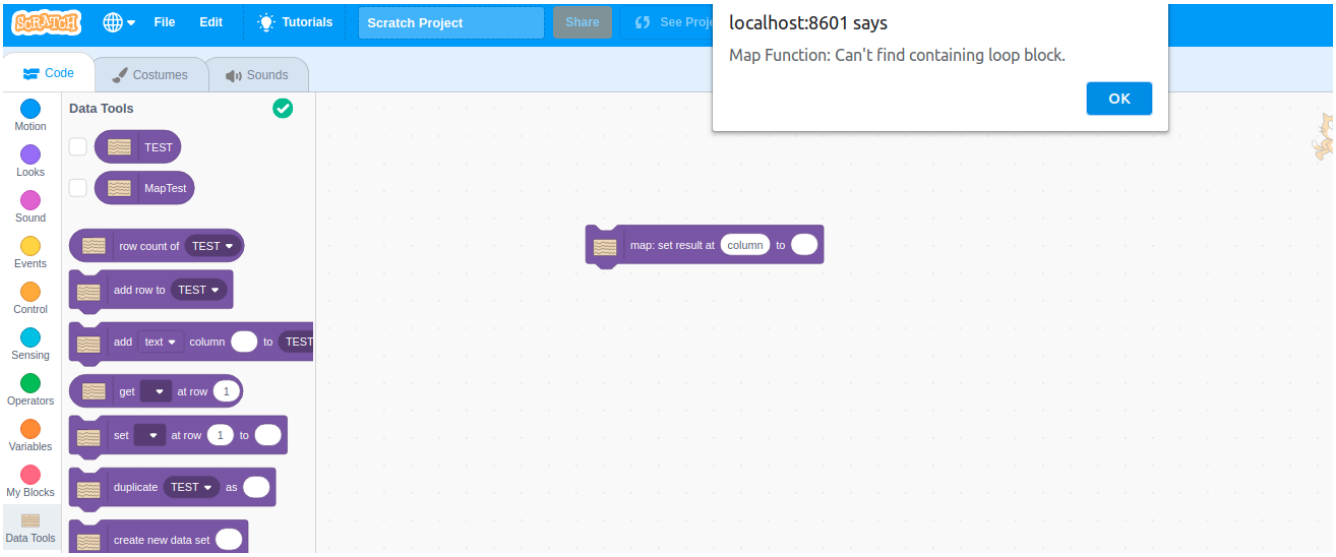
We can then view the resulting map from the file viewer menu like we did before, finding the result of the map:



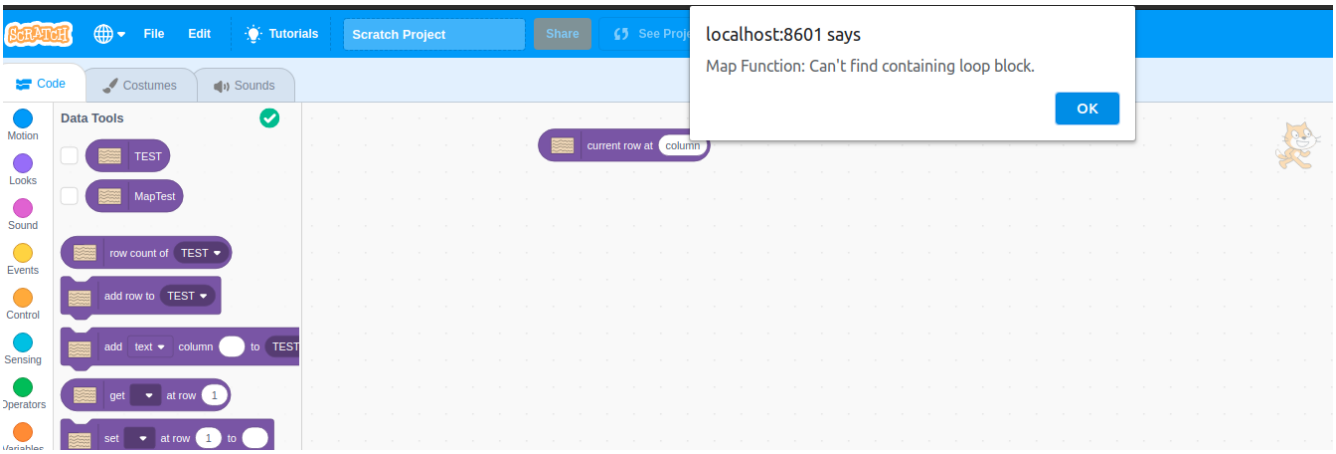
We can see that the data was properly mapped and even transformed with the graduation year. There are some edge cases with this block as well one is what happens when there are no blocks put into the map:



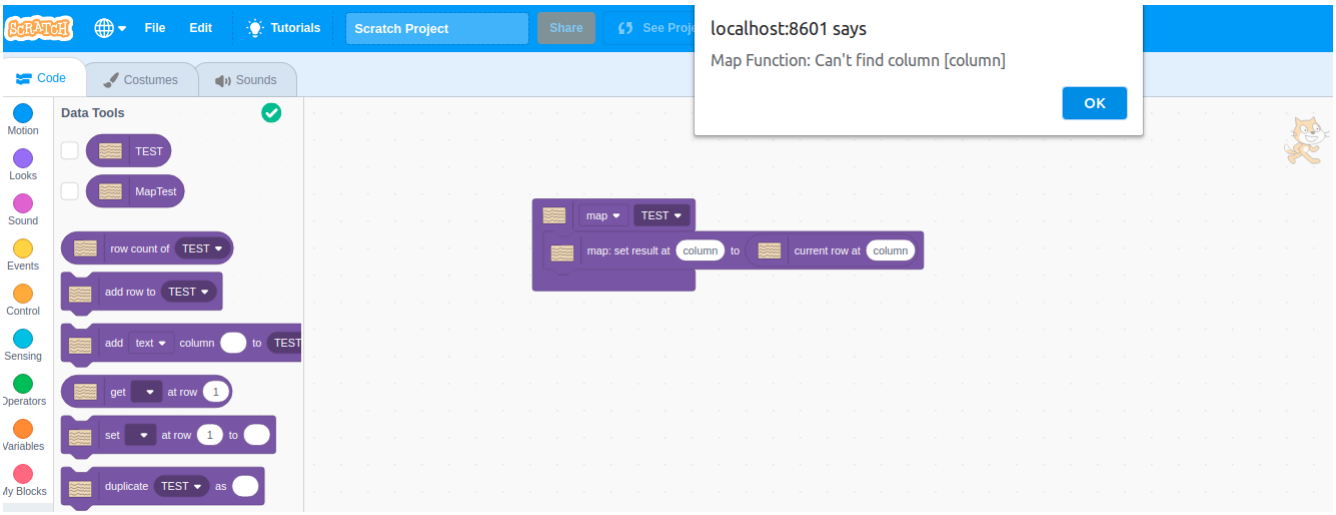
The system handles this case by alerting the user and stopping the execution of the block. If a “map: set result at” block is used outside of a map containing block the system also halts execution and then alerts the user of the mistake:



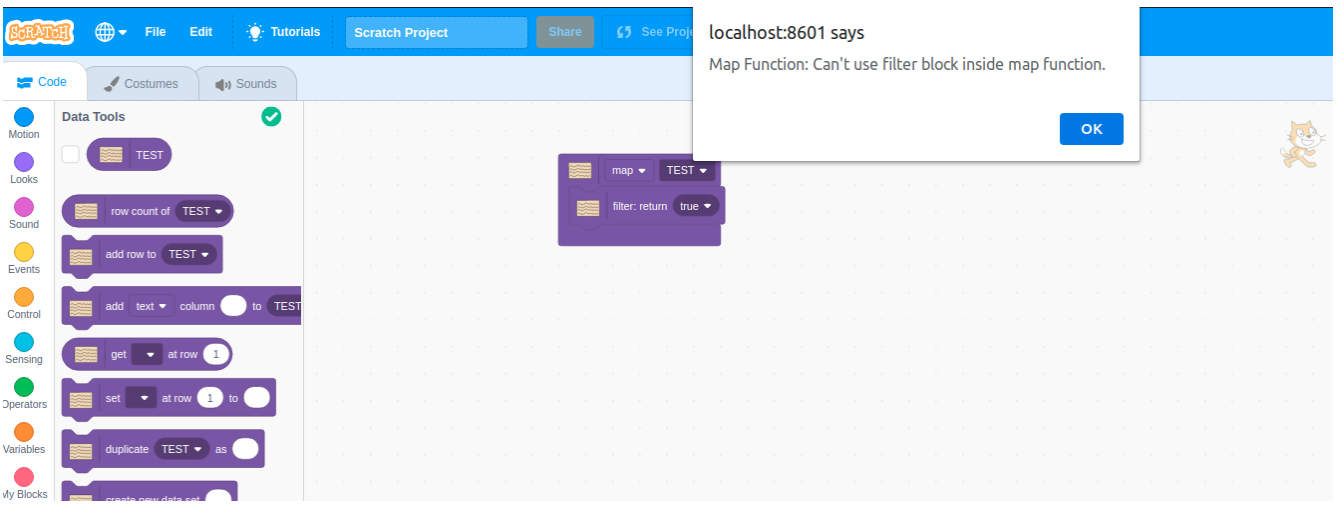
The same happens when the “current row at” reporter block is used outside of the map containing block:

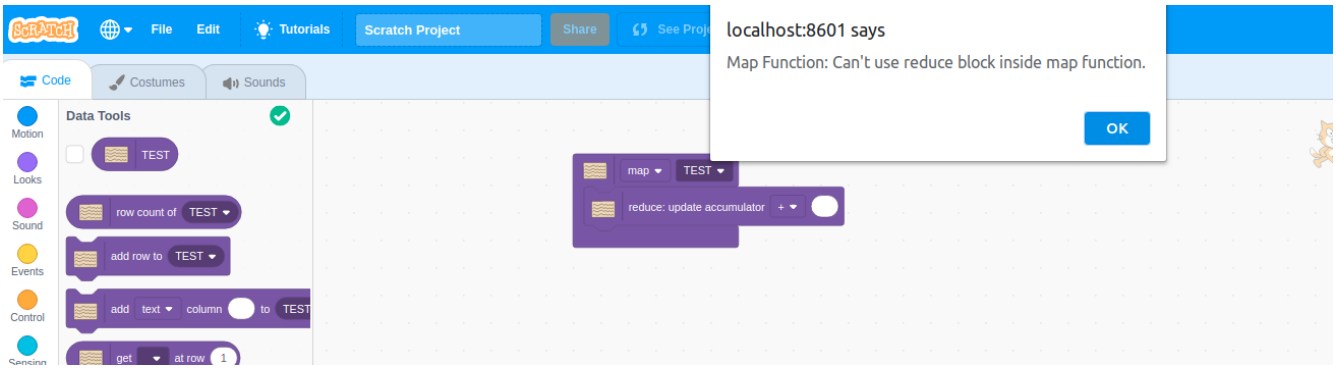


If the user is trying to access a column that does not exist in the selected data set, an alert is shown and execution is stopped:

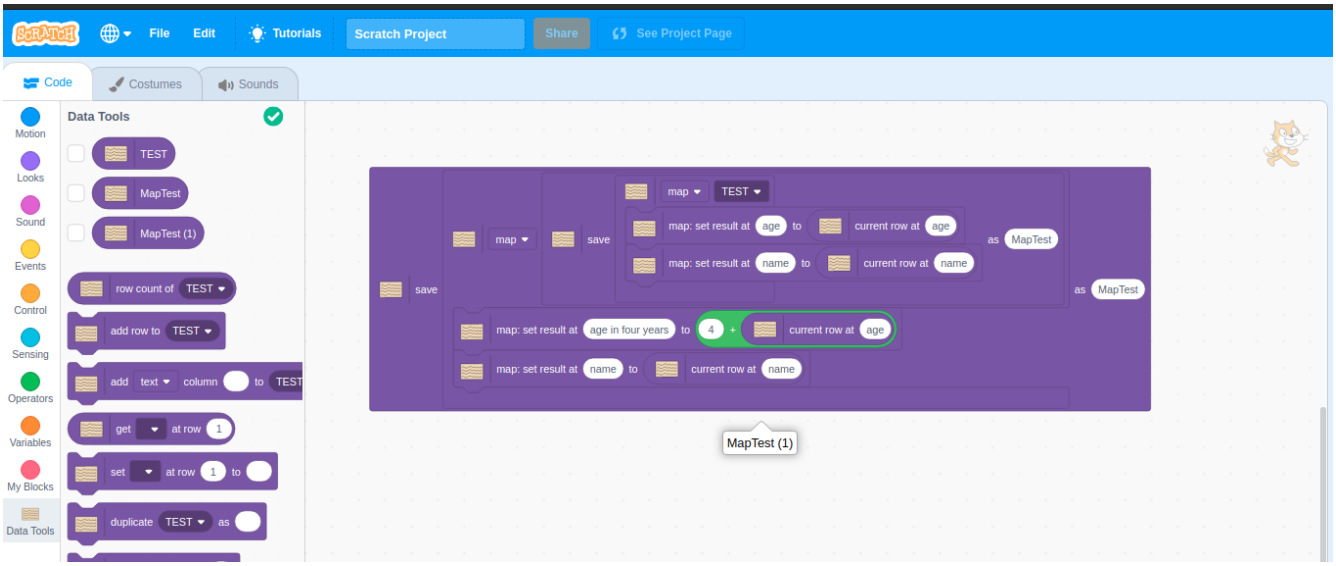


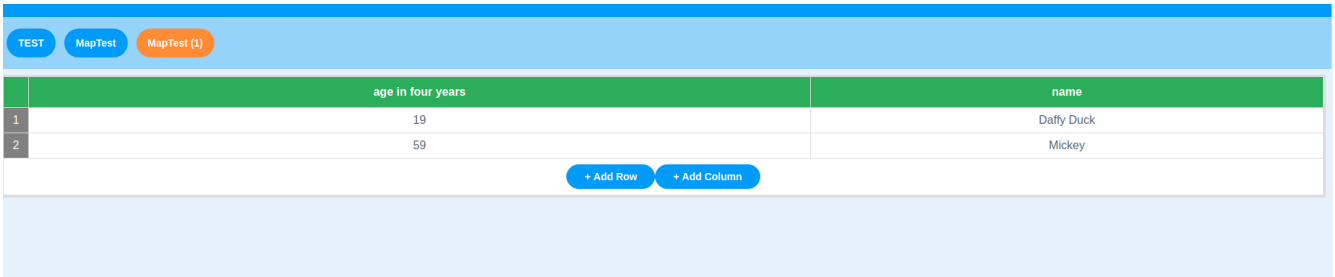
The filter and the reduce blocks cannot be used if the map option is selected in the function block, the system alerts the user of the mistake and then halts execution:





Map function blocks can be used as a parameter in another map function block thus effectively stacking two maps together like so:

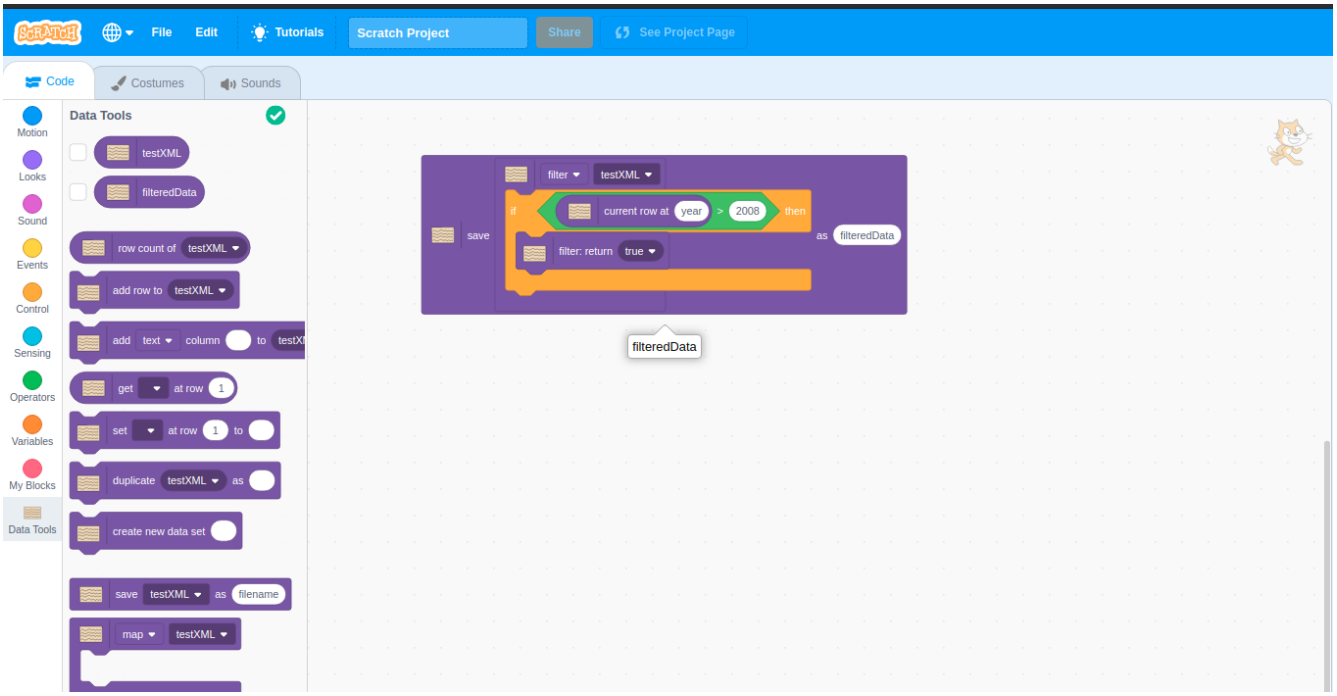




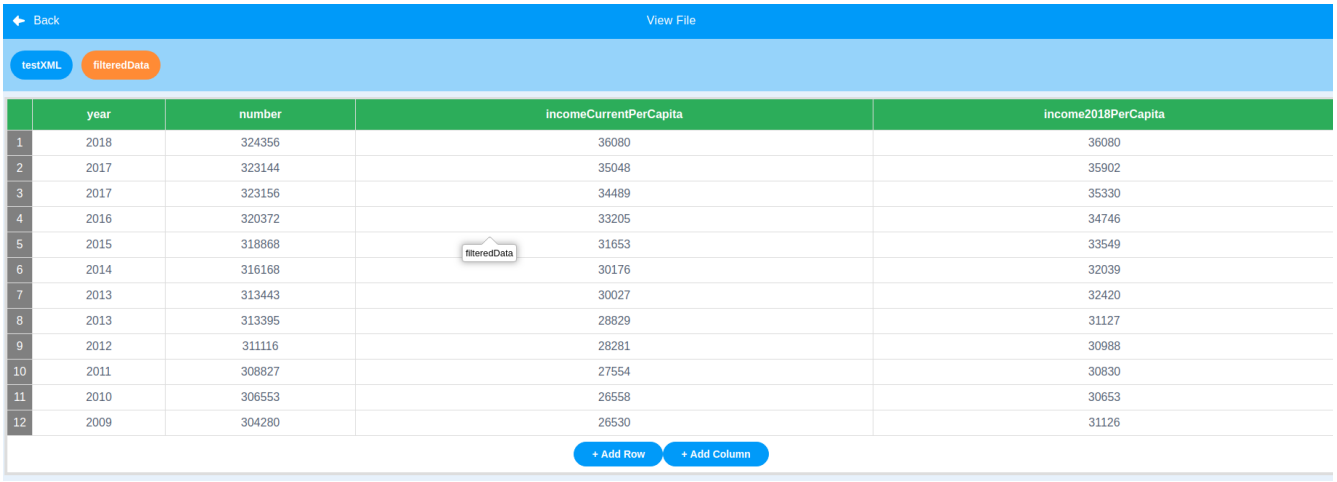
Map blocks can effectively be stacked upon one another to create a chain of maps. Many of the cases gone through above are similar if not the same for both the filter and reduce functions, for example running the “file: return” block outside of the function block and putting a “map: set result at” block inside of the function block when reduce is selected. The functionality of saving the data sets resulting from these functions is the same for all three functions.

Filter

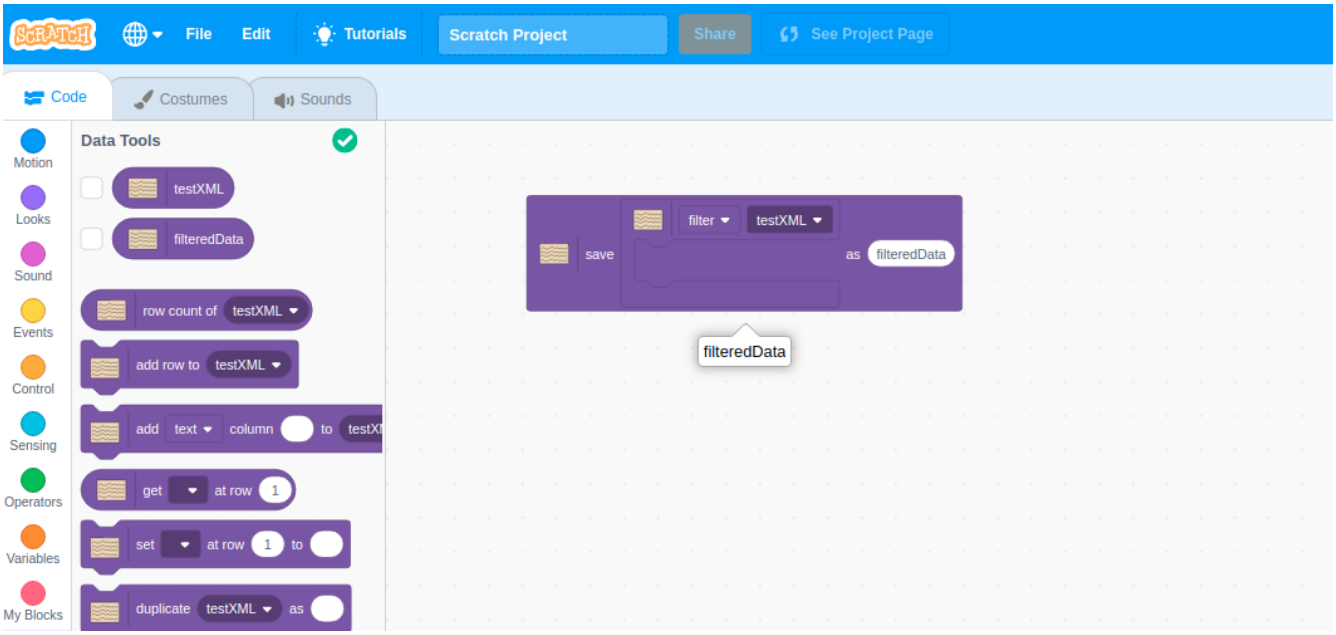
Filter was created to go through an existing data set and return a data set that does not have any entries that the user does not want to include. For example if a user wants to only look at data from the last 10 years it would look something like:



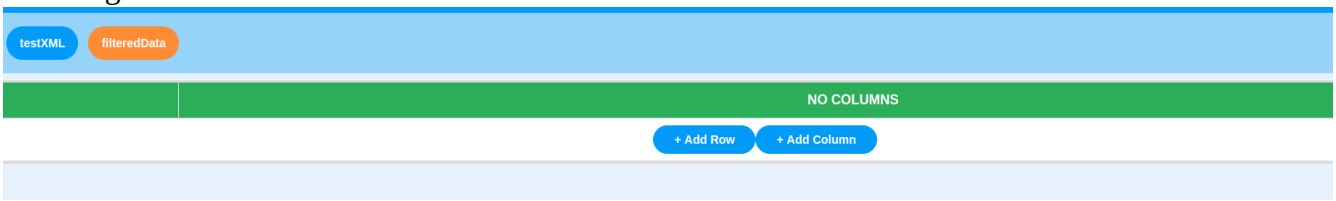
returning a data set that looks like so:



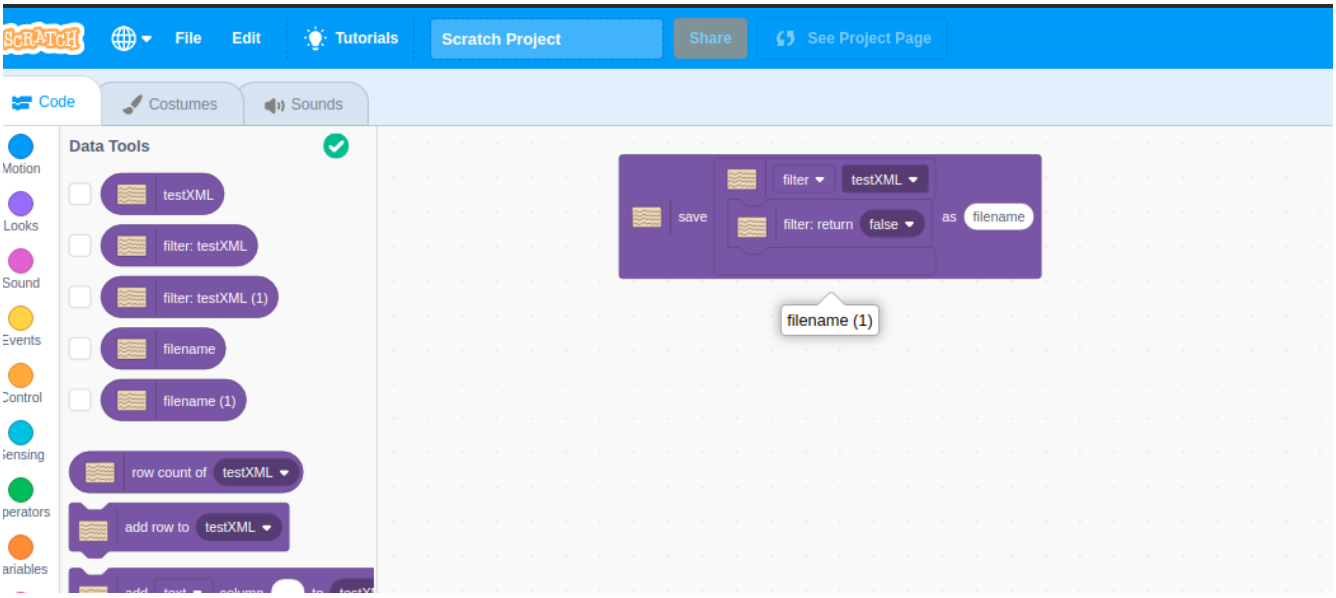
A key difference between filter and map is that if the “filter: return” block is not inside of the filter function block, no error is returned, rather it is assumed that all entries should be removed, thus returning an empty data set:

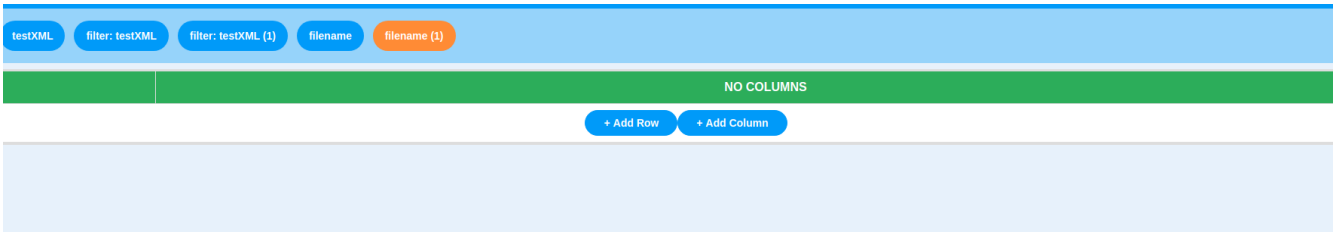


returning a data set like so:

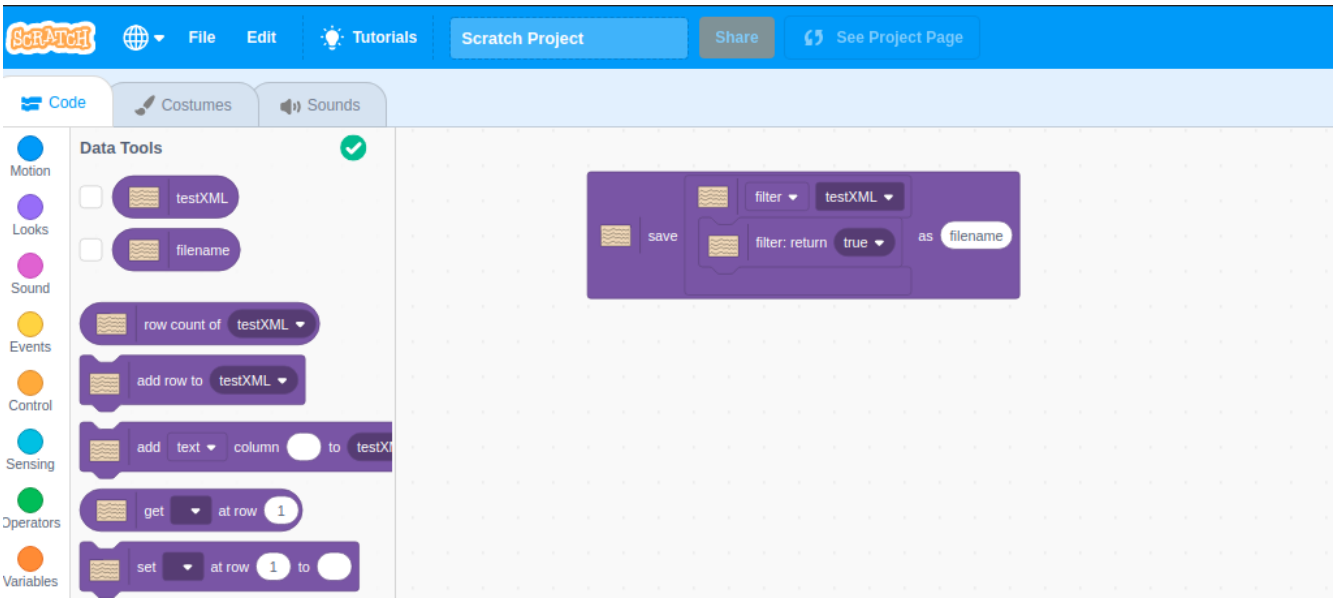


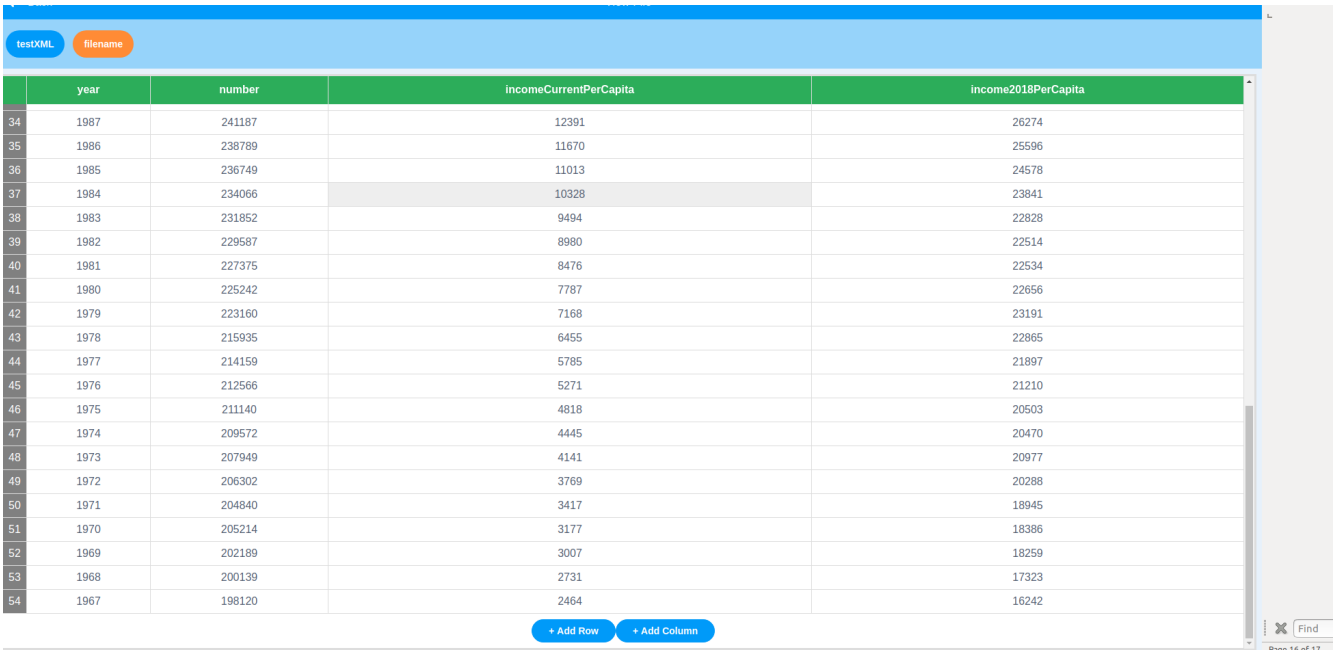
An empty data set will be returned if no “filter: return true” block is hit in the filter function block, this includes the case where only a “filter: return false” block is inside the filter function block:



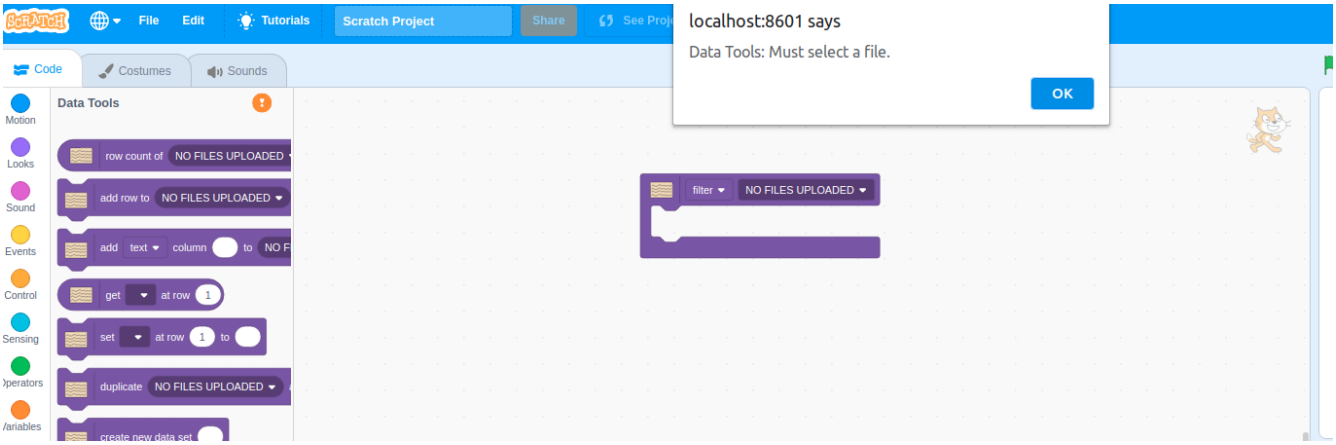


If there is no conditional surrounding the “filter: return true” block then the original data set will be returned:

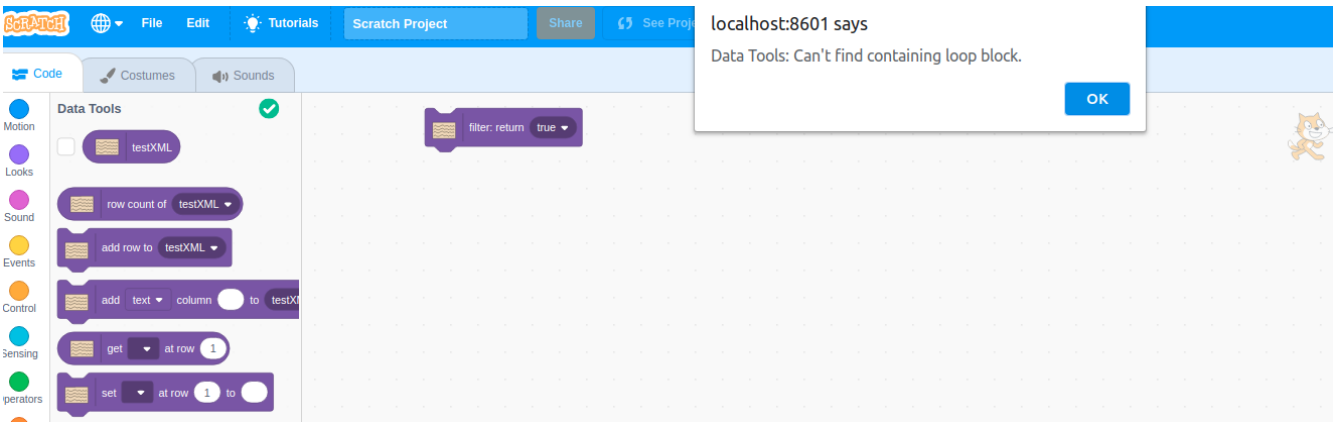




Filter behaves the same way that Map does when no files are uploaded in the system, alerting the user and then stopping execution:

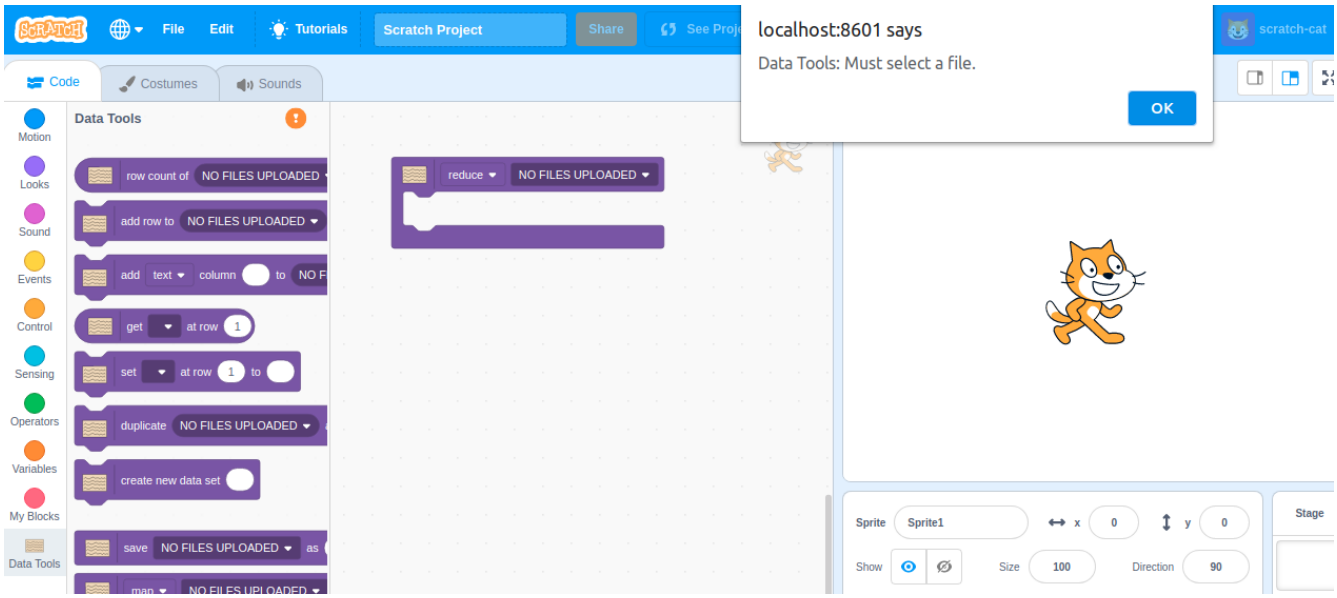


The “filter: return” block also must be put inside of the filter function block in order for it to execute, if it is not it will alert the user and stop executing:

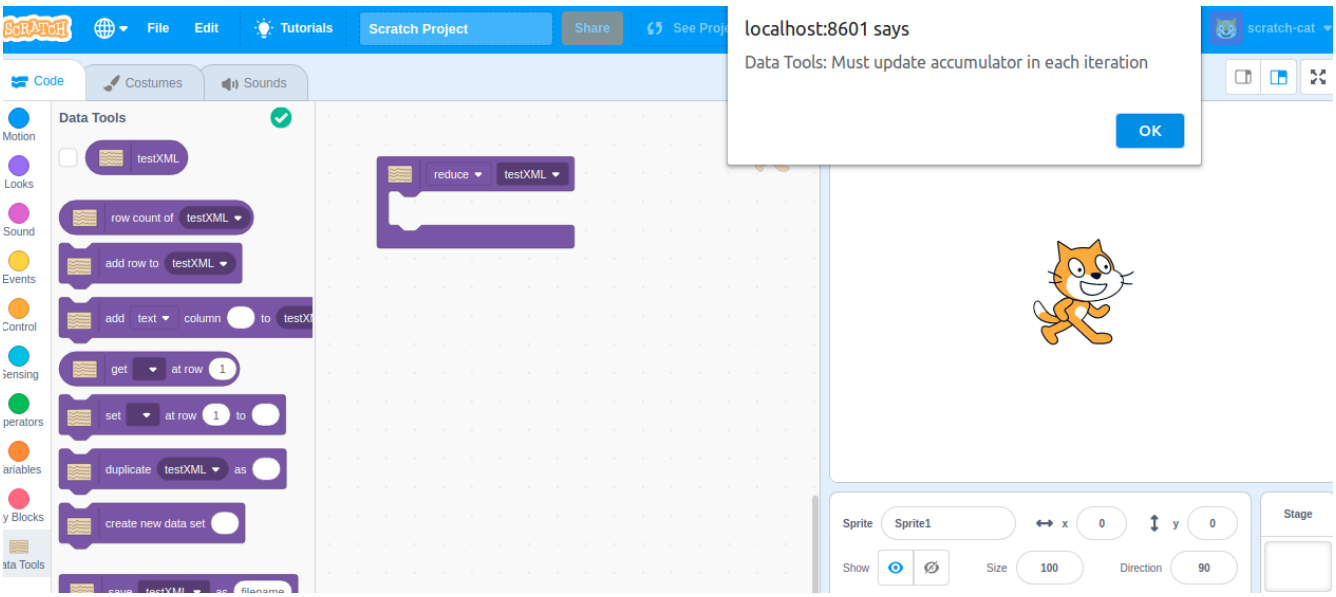


Reduce

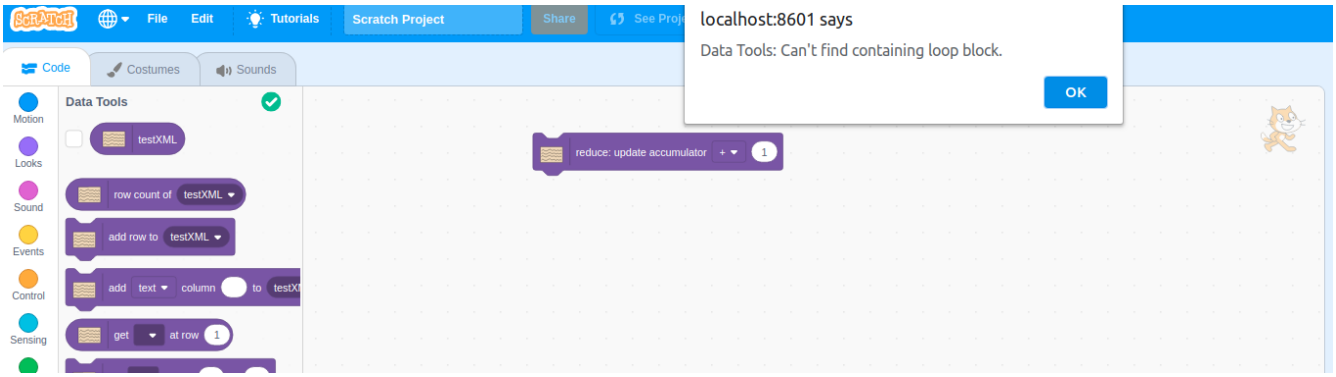
Reduce was added to give the user the ability to iterate through a data set and return a single value based off of the users desired operation. This makes doing things like taking the average easy. The Reduce function block has a similar functionality to the map block when NO FILES UPLOADED, alerting the user of the problem and stopping execution:



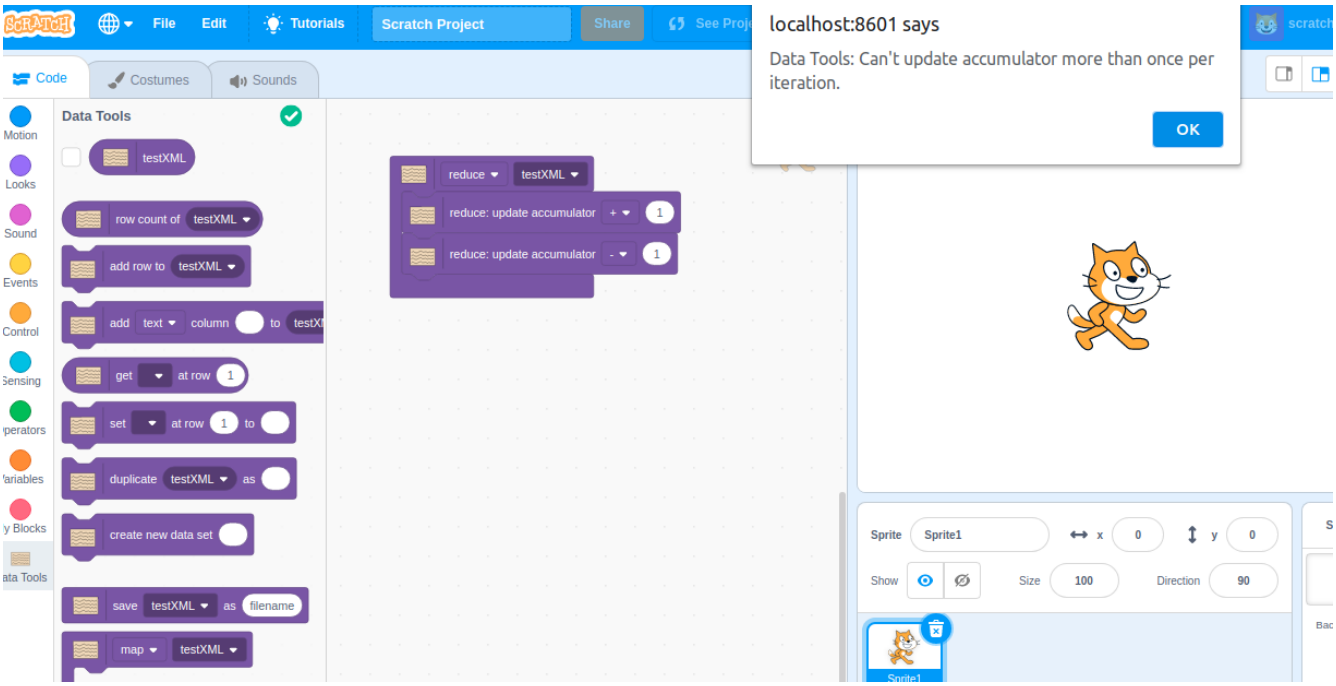
The accumulator block was added to give a way for the user to update the value they would like to return. The block gives the option to add, subtract, multiply or divide from a numeric accumulator or append strings together. If the accumulator block is not present in the reduce function block, the user will get an alert and the block will stop executing like so:



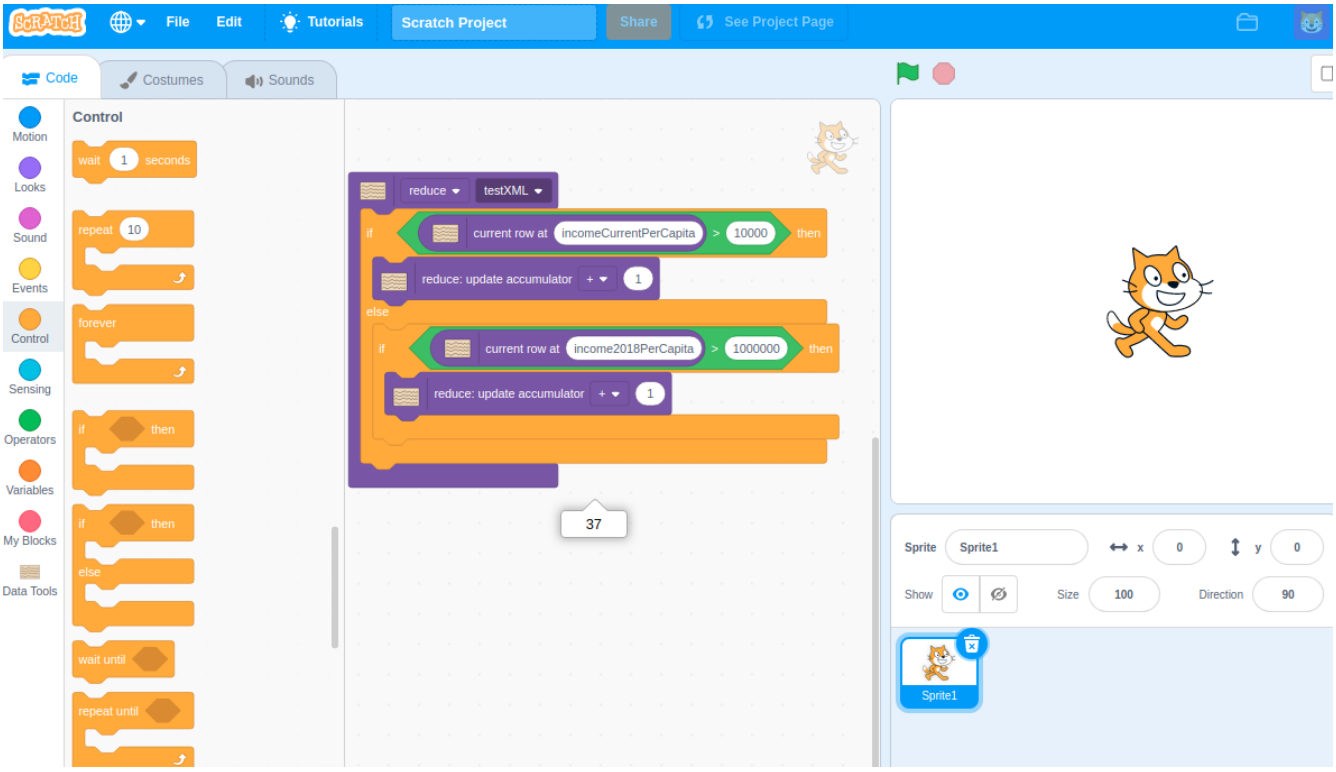
The accumulator block must be placed inside of a reduce function just like the filter and map blocks, otherwise an alert will show and the block will stop executing:



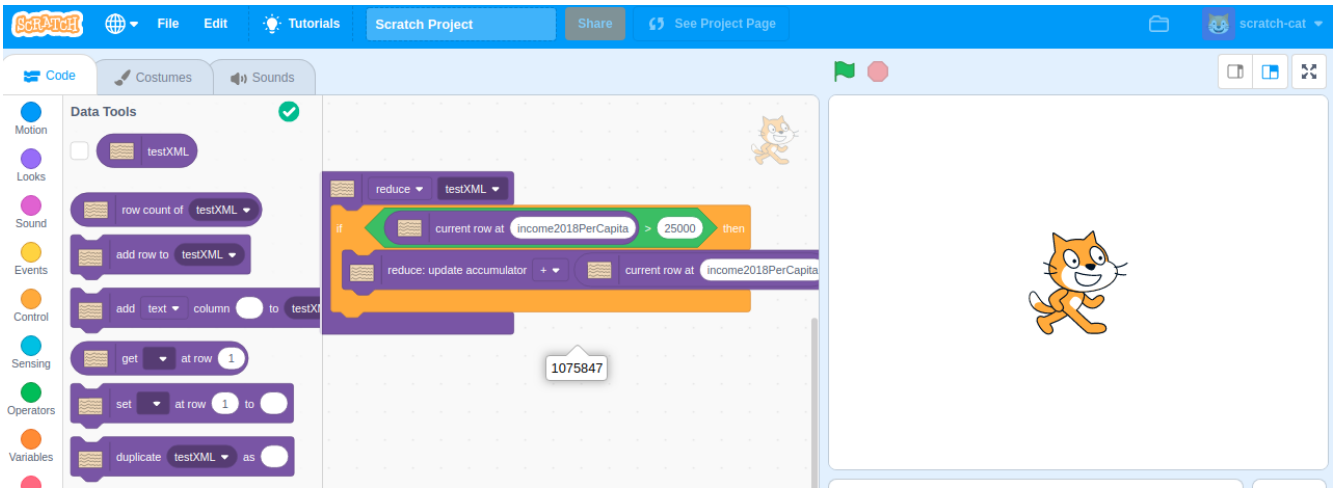
The accumulator block cannot be updated twice in one iteration so putting to of them together will cause an alert and the block to stop execution:



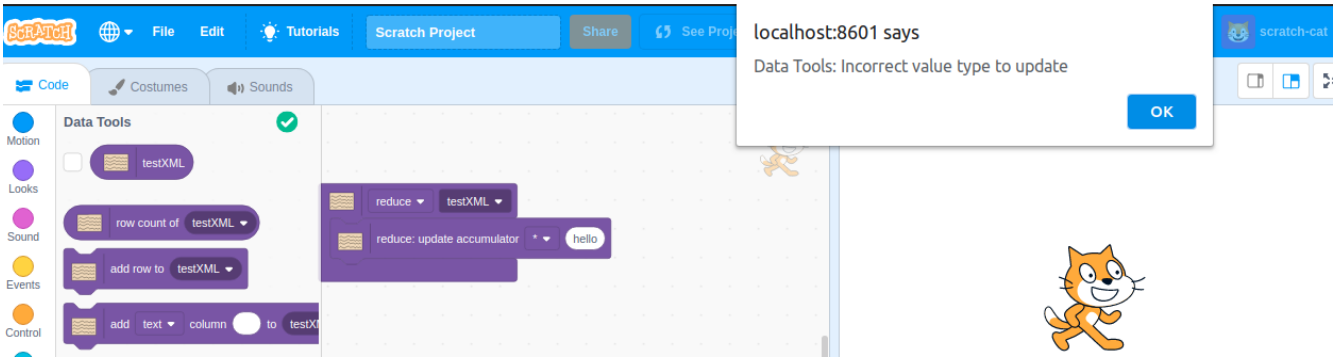
However, you can still include multiple updates so long as the do not both happen in a single iteration, like so:



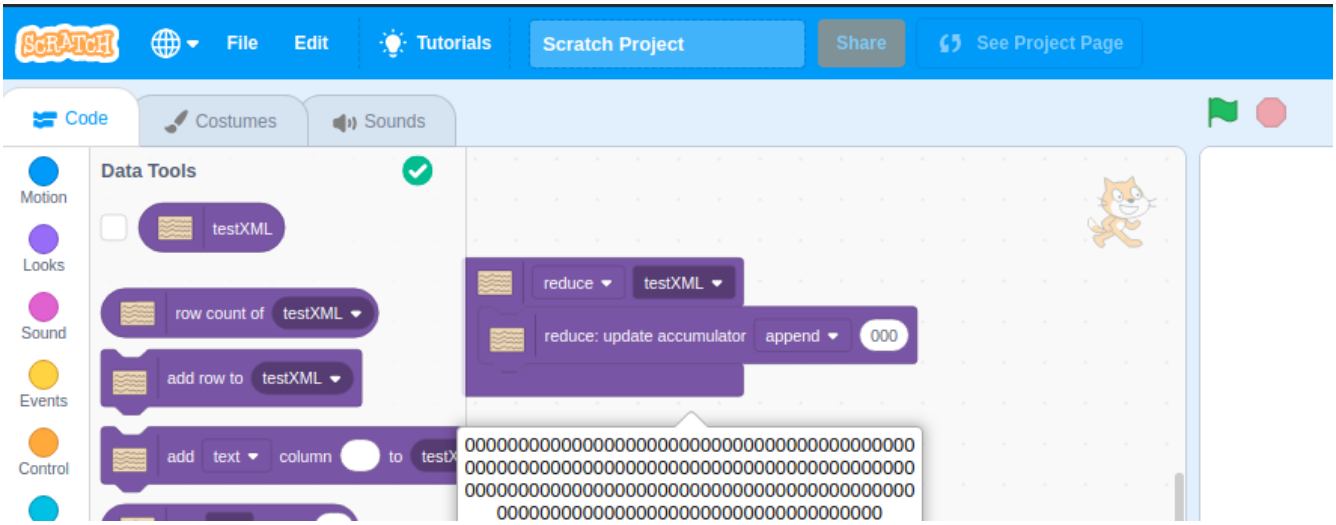
So long as the accumulator is updated at least once during the process an error will not happen:



The accumulator block only allows for numerical values when a numeric operator is selected, for instance it will show an alert and stop executing if a user tries to multiply a string:



But it will append numeric values as if they were string:



Finally an advanced example, the system should be able to handle the combination of all of three of the function blocks: