Code Set Up Tutorial

Surface Water Identification and Forecasting Tool

Southwest Water Resources| NASA DEVELOP | Summer 2021

Monitoring Surface Water Extents of Remote Stock Ponds in the Southwestern United States Using Earth Observing Systems for Enhanced Water Resources Management

**Tutorial**

Brooklyn Appling (Geoinformatics Fellow)

***Previous Contributors:***

Rainey Aberle (Project Lead)

Rebecca Bernat

Michael Corley

Lukman Fashina

Kyle Paulekas

***Advisors:***

Seungbum Kim (NASA Jet Propulsion Laboratory Radar Remote Sensing)

Keith Weber (Idaho State University GIS Training and Research Center)

***Lead:***  
Brandy Nisbet-Wilcox (Idaho – Pocatello)

**Table of Contents**

[Overview 3](#_Toc190097699)

[Set up & Requirements 3](#_Toc190097700)

[Google Cloud Project 3](#_Toc190097701)

[Methods 3](#_Toc190097702)

[1. Bringing in Code 3](#_Toc190097703)

[2. Bringing in Assets 4](#_Toc190097704)

[2.1 Importing External Files 4](#_Toc190097705)

[2.2 Creating a Custom Polygon 7](#_Toc190097706)

[Conclusion 8](#_Toc190097707)

[Acknowledgements 8](#_Toc190097708)

[Citations 8](#_Toc190097709)

# 

# Overview

This tutorial demonstrates how to set up the code and assets of the Surface Water Identification and Forecasting Tool (SWIFT) inside a Google Cloud project in Google Earth Engine (GEE). The scripts and assets can be found at <https://github.com/nasa-develop/swift>.[1] This tutorial does not demonstrate how to run the code and is not a substitute for the README or SWIFT\_UserTutorial files.

# Set up & Requirements

## Google Cloud Project

Since November 14, 2024, the use of Google Earth Engine (GEE) has been fully shifted to Google Cloud projects. Creating and using Google Cloud projects is free for noncommercial use ([see here for terms](https://earthengine.google.com/noncommercial/)) but requires registration.[2]

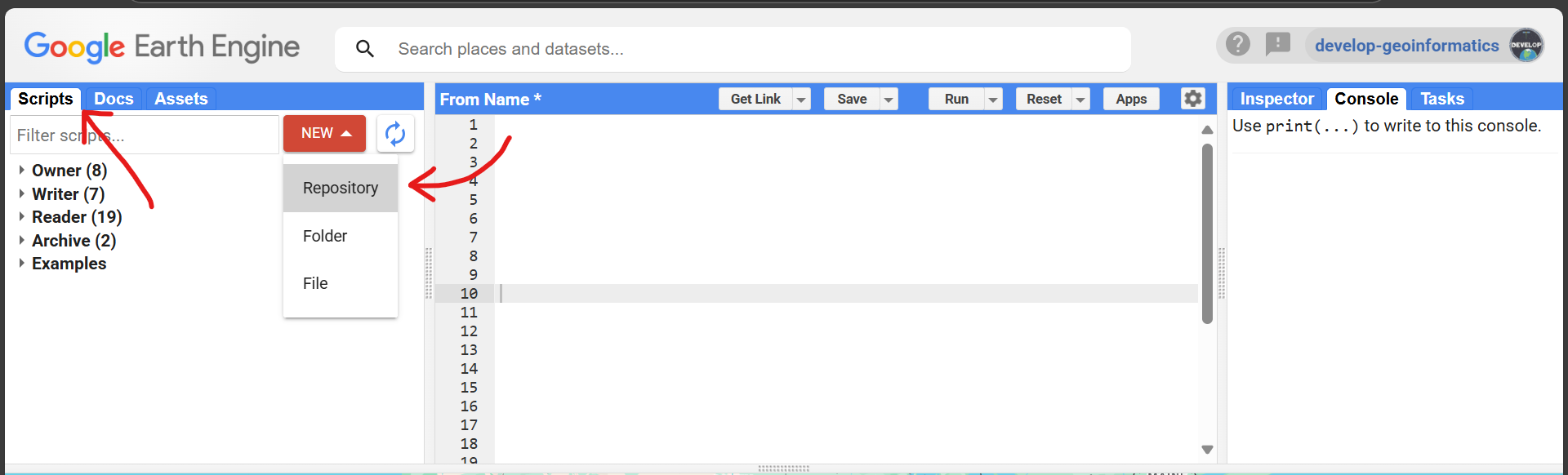
If you do not already have a Cloud project, register for one at <https://code.earthengine.google.com/register>.[3]

After you have a Cloud project, login at <https://code.earthengine.google.com/> to continue with this tutorial.[4]

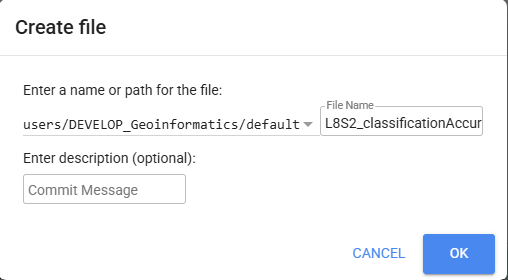
# Methods

## 1. Bringing in Code

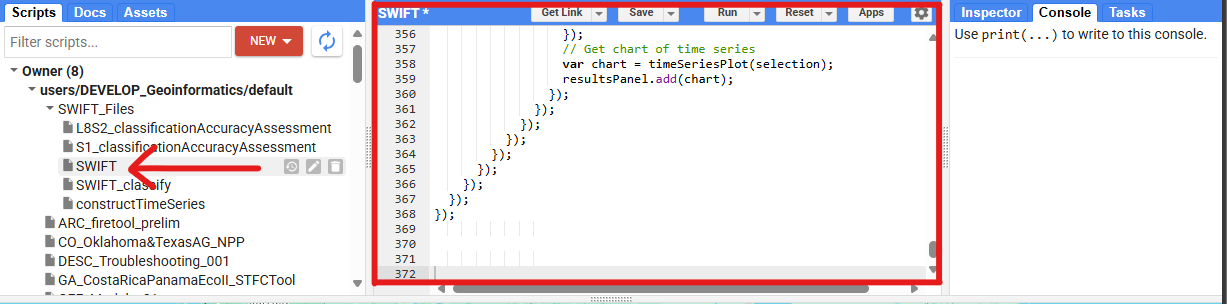
1. Before code can be brought into a Google Cloud project, you will need to create a repository to hold them. This can be done in the GEE Code Editor by going into the “**Scripts**” tab, clicking “**New**” and then “**Repository**”. You can name this repository anything you want.



1. If you wish, you can also create folders within the repository to help organize your files. This is done using the same “**New**” dropdown.
2. Next, you are going to create five files inside your repository/folder. They will be named **L8S2\_classificationAccuracyAssessment**, **S1\_classificationAccuracyAssessment**, **SWIFT**, **SWIFT\_classify**, and **constructTimeSeries**. If you are using a folder, you will need to drag and drop the files into the folder after you create them. This can be done in the “**Scripts**” tab.



1. Next, open each file in the scripts folders. They are saved as .txt files, which cannot be directly brought into GEE. Instead, we are simply going to copy and paste their contents. This can be done very easily by opening the .txt file, selecting everything using ctrl+A, copying the selection with ctrl+C, clicking the script you want to paste into, and pasting everything into the code editor using ctrl+V.



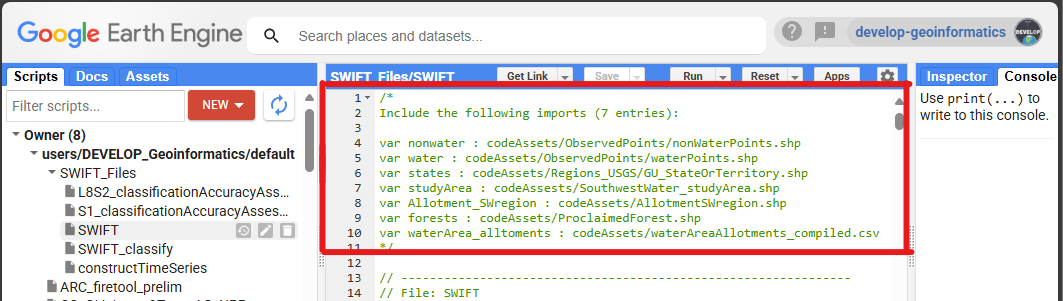
1. Save each file by clicking “**Save**” or using ctrl+S.
2. Lastly, the **SWIFT** file uses tools listed in **SWIFT\_classify**, so you will need to pass it the file location. This can be found in line 332. You will need to modify it by replacing the text in bold:

var classify = require('users/**YourUsername**/**Repository**:**FolderIfAny**/SWIFT\_classify);

## 2. Bringing in Assets

### 2.1 Importing External Files

1. At this point you should have 5 files inside the Google Cloud project environment with code pasted into each of them. However, these files are still not ready to be run because they all rely on external data and files. Open one of these files and at the top you will see a block of commented out code listing imports. It should look like this:



Each of these imports can be broken down into the form “**var name : asset**” consisting of three components:

**var** – This is simply JavaScript for saying that we are declaring a variable. You can ignore this.

**name** – The name you are assigning to the variable and will need to exactly match what is written.

**asset** – The data being stored inside of **name**, in this case external files. Each of these is a file path. For example, “codeAssets/ObservedPoints/waterPoints.shp” can be found by going to <https://github.com/nasa-develop/swift/codeAssets/ObservedPoints> and downloaded the zipped waterPoints.shp file.

All variables not called geometry reference external files found in the NASA DEVELOP SWIFT GitHub. What to do about “**geometry**” will be addressed in section 2.2.

1. There are seven total assets that need to be imported:

Allotment\_SWregion.shp

GU\_StateOrTerritory.shp

ProclaimedForest.shp

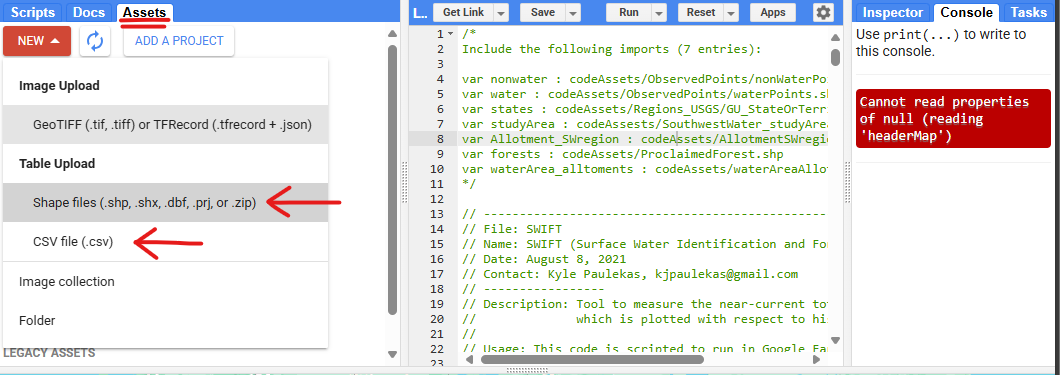
SouthwestWater\_StudyArea.shp

nonWaterPoints.shp

waterPoints.shp

waterAreaAllotments\_compiled.csv

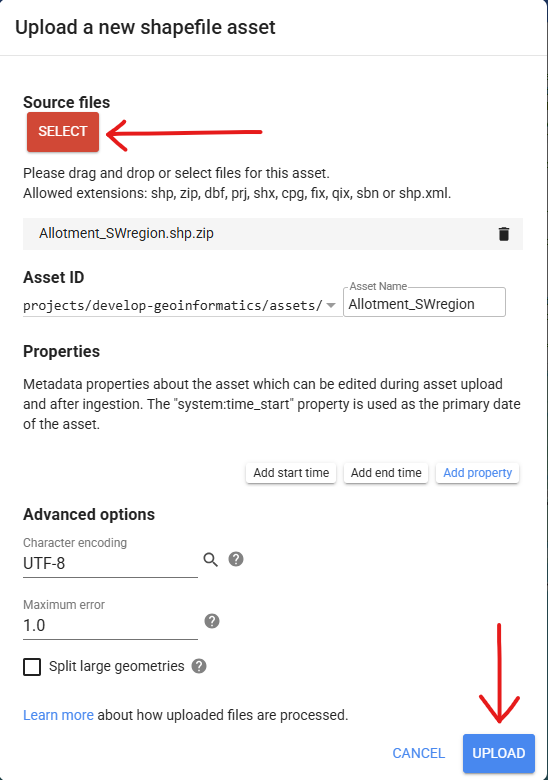
To import an asset go to “**Assets**”, “**New**”, and select either “**Shape files (.shp, .shx, .dbf, .prj, or .zip)**” or “**CSV file (.csv)**” depending on whether the file ends in .shp or .csv.



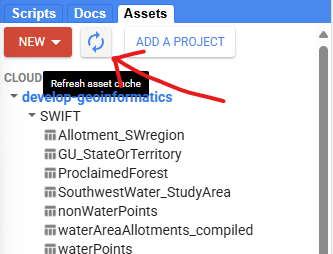
1. Uploading Shapefiles:

Shapefiles (.shp) are actually multiple files stuck together, including .cpg, .dbf, .prj, .sbn, .sbx, .shp, .shx, and sometimes even more. To simplify the upload process, all shapefiles that are used as assets in the Cloud project have been pre-zipped.

After clicking “**Shape files (.shp, .shx, .dbf, .prj, or .zip)**”, a new window should open. Here you will click the “**Select**” and navigate to the location of the zip file on your device. After selecting it and clicking “**Open**”, the name field will populate. All shapefiles use default settings, so you can click “**Upload**” right away.



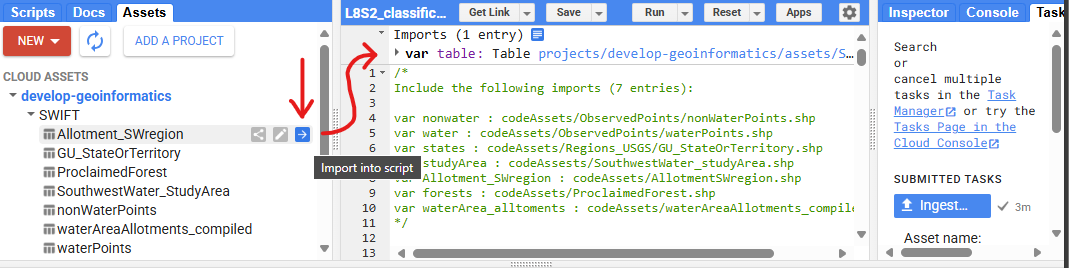
The file will not populate immediately. Instead, there a **Task** will appear in the tasks tab at the top right of your screen. It may take several minutes, but the asset will automatically be added to the **Assets** tab after this task completes. However, you will need to click the refresh icon to see it.



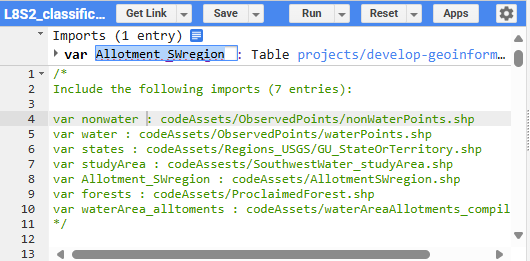
1. Uploading CSVs is the exact same process but within a . There are many more options for .csv files, but again all will be left as defaults.

### 2.2 Adding Assets to Scripts

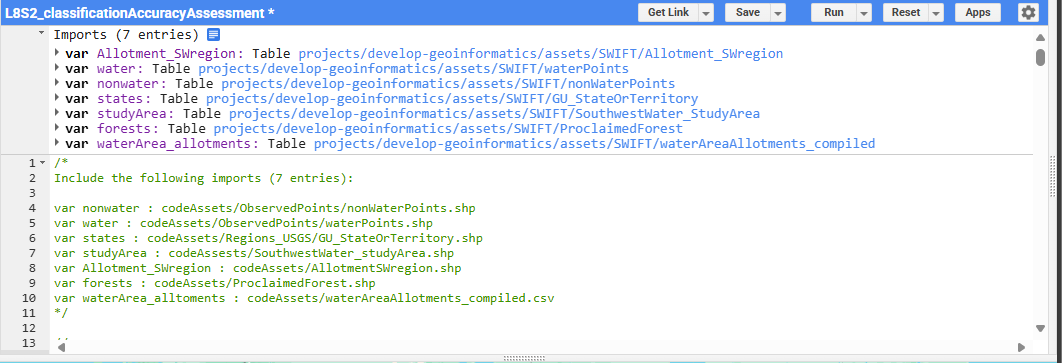
1. Once all seven assets have been uploaded and are visible in the **Assets** tab, they can be added to each script quite easily. First, click “**Scripts**” and navigate to the script you want to add assets to, then open this script.
2. Next, reopen the **Assets** tab. Now in the **Assets** tab you should see the assets you have brought in, while in the code editor you should see your script.
3. Hover over one of your assets. Three icons should appear on the right, with the rightmost being an arrow. Clicking this arrow will bring the asset into the script with the generic name “**table**”.



1. Next, click the name in purple. Write what is listed below in the **name** column for that asset. In this case, “**Allotment\_SWregion**”.



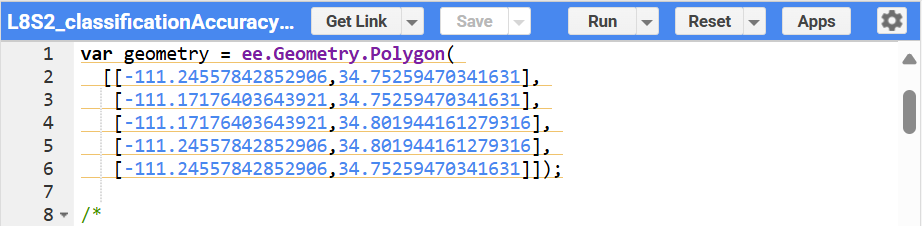
1. Repeat until all non-“**geometry**” imports have been brought in. They should each match an entry listed in the commented block below.



1. Finally, click “**Save**” and repeat for all five scripts.

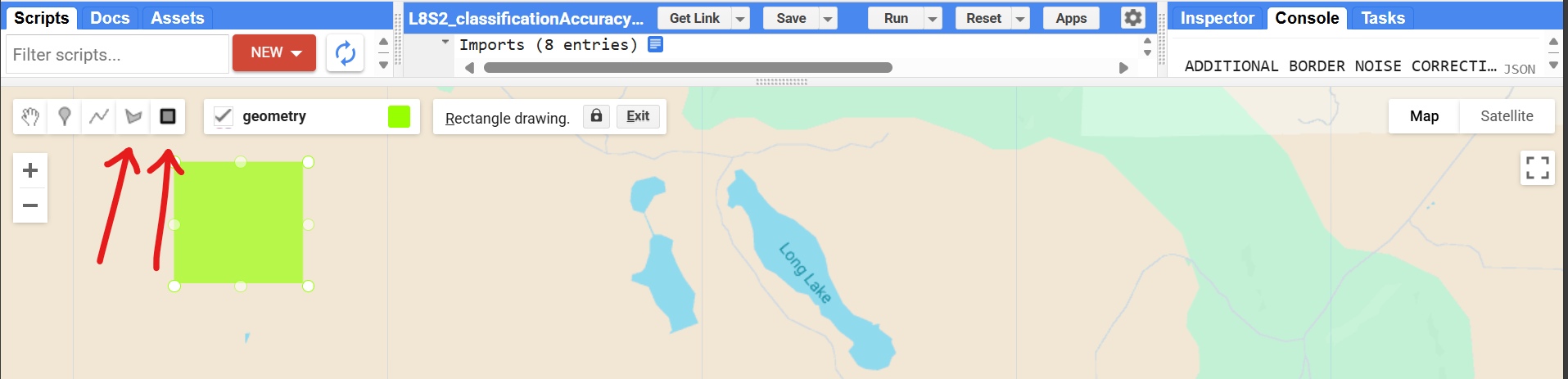
### 2.3 Creating a Custom Polygon

1. Now, all your assets and scripts should have been brought in, with the exception of the “**geometry**” objects. For these, you can still bring in an external .shp or .geoJSON file containing the polygon you want to use, which will roughly follow the steps above. However, there are also two ways to create custom polygons inside Google Earth Engine.
2. The first of these two methods is to add the coordinates manually by passing an **ee.Geometry.Polygon** object to a variable. This is best if you are wanting consistency between multiple different tests or have a precise area to study.

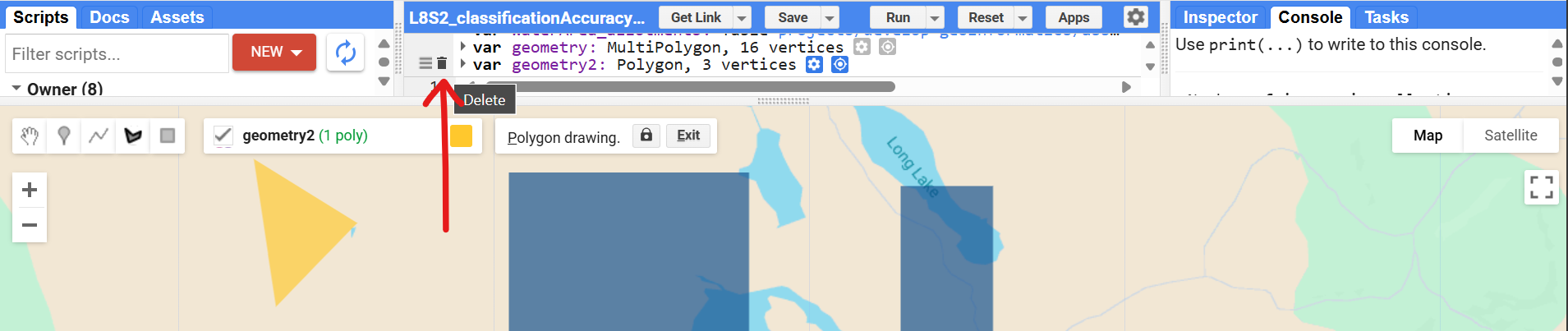


Note: the example above draws a rectangle but has five points. This is because lines are drawn between successive points, and the point where the first line segment begins is also where the last one ends.

1. The second method is to use the custom polygon drawing tools located in the map viewer. There are two of these. The rightmost one (**draw a rectangle**, symbolized by a square) only allows the user to draw squares and rectangles. The one just to its left (**draw a shape**) allows you to draw much more freely, but will not consist of clean 90° angles along cardinal directions.



1. After creating a polygon using the second method, it will automatically appear as an import. If you do not like it, you can hover to the left of where it was imported, and a delete button will appear. Clicking the small arrows next to **var** will expand the import records and can be used to show the coordinates of the polygon.



# Acknowledgements

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

This material is based upon work supported by NASA through contracts NNL16AA05C and 80LARC23FA024.

# Citations

[1] NASA DEVELOP GitHub. (n.d.). SWIFT. [Code Example]. https://github.com/nasa-develop/swift

[2] Google Earth Engine. (n.d.). Earth Engine for Noncommercial and Research Use. https://earthengine.google.com/noncommercial/

[3] Google Earth Engine. (n.d.). Get started using Earth Engine. https://code.earthengine.google.com/register

[4] Google Earth Engine. (n.d.). Earth Engine Code Editor. https://code.earthengine.google.com/