

Exercise 3: Writing Custom Functions and Mapping across Image Collections

*Concept Note*

**Google Earth Engine Training**

*Developed, delivered and supported by:*

The USAID & NASA supported SERVIR-Mekong Program, United States Forest Service – Geospatial Technology and Applications Center (USFS-GTAC), ADB – Core Environment Program, SilvaCarbon, and Google Earth Engine

Exercise 3: Writing Custom Functions and Mapping across Image Collections



Introduction

The Code Editor offers access to the full power of Earth Engine; however, a basic understanding of the fundamentals of coding and JavaScript is required. In this exercise, you will continue to learn about JavaScript syntax and some new Earth Engine spatial data concepts. You will build on what you learned in exercise two about image objects; however you will now turn your focus to working with collections of images, or stacks of similar image objects. In this exercise, you will focus on basic concepts and methods associated with image collections in Earth Engine.

Objectives

* Practice writing and running code in the Earth Engine Code Editor to access raster image collections, filter them temporally (by a date range) and spatially (e.g., by your study region), work with geometries, and explore some image processing functions

Required Materials

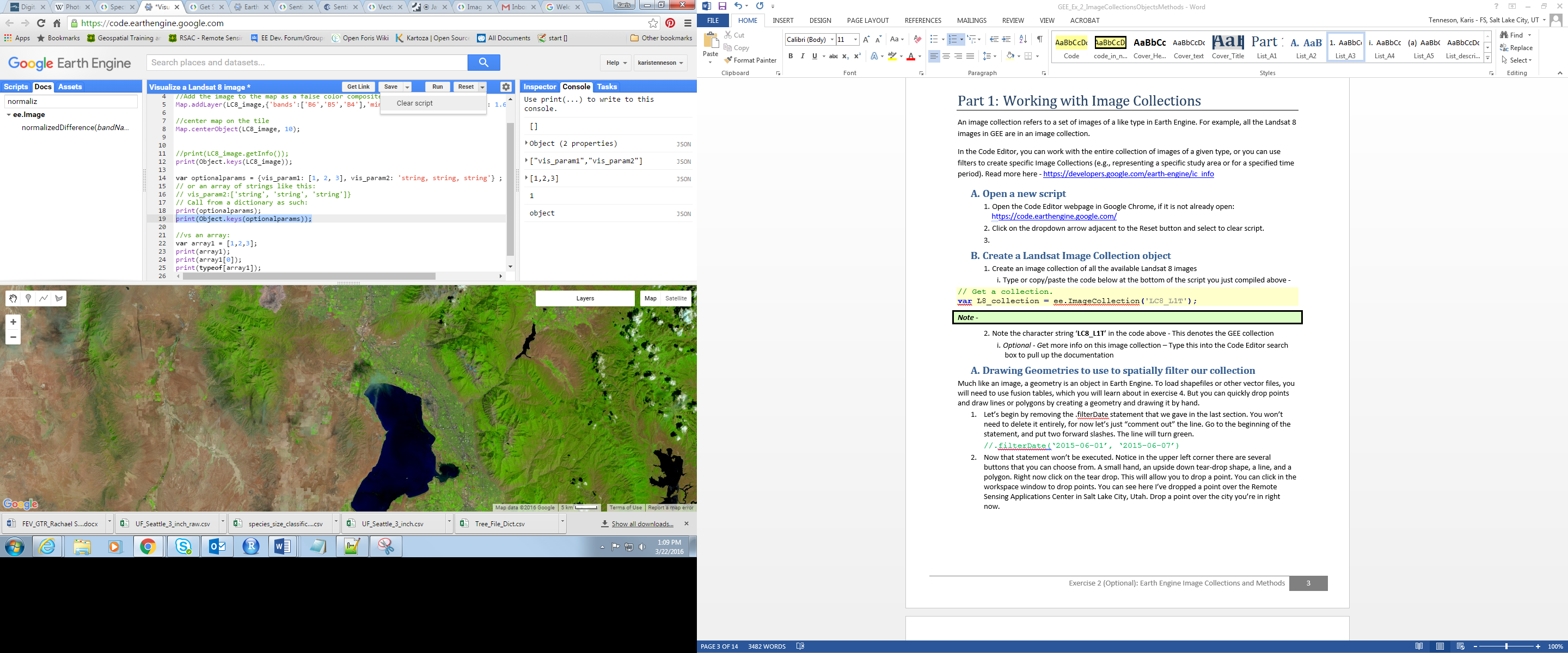
* An approved Google Earth Engine Account
* *Suggested:* Google Chrome installed on your computer

1. Working with Image Collections

An ImageCollection refers to a set of images in Earth Engine. For example, all the Landsat 8 images in GEE are in an ImageCollection.

In the Code Editor, you can work with the entire collection of images, or you can use filters to create subsets of the ImageCollection (e.g., representing a specific study area or for a specified time period). Read more here - <https://developers.google.com/earth-engine/ic_info>

* 1. Open a new script
     1. Open the Code Editor webpage in Google Chrome, if it is not already open: <https://code.earthengine.google.com/>
     2. Click on the dropdown arrow adjacent to the Reset button and select Clear script.



* 1. Create a Landsat Image Collection object
     1. Make a variable referencing the image collection of all the available Landsat 8 images by typing or copy/pasting the code below into the code editor.

// Get an image collection.

**var** landsat8\_collection **=** ee**.**ImageCollection**(**'LANDSAT/LC8\_L1T\_TOA'**);**

* + 1. To add this collection to the map, copy the code below and paste it into the code editor. Then click Run to execute all the lines.

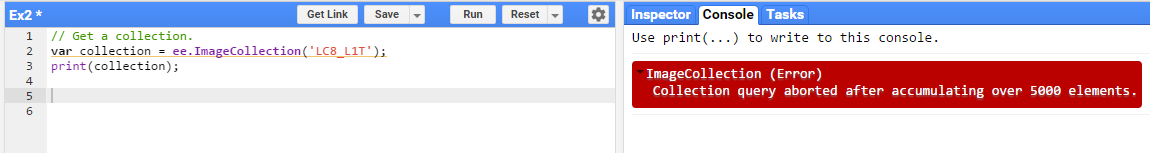
// Get an image collection and center the display.  
Map**.**addLayer**(**landsat8\_collection**,** **{** min: 0.05, max: 0.8, bands: 'B6,   
 B5, B4' **},** 'Landsat Collection'**);**

Map**.**setCenter**(**100.56**,** 13.94**,** 7**);**

* + - 1. What is mapped here? The image collection has a number of images, but not all are mapped. When you use the Map.addLayer function to add an image collection to the map window, by default the most recent pixel is shown.
    1. *(Optional)* If you add a print statement, you can determine how many images are in the collection. Copy and paste the following lines into the bottom of your script.

// Print the information about the image collection.   
print**(**landsat8\_collection**);**

* + - 1. However, since the image collection is rather large, an error message is returned in the Console. The print Collection query aborted after accumulating the metadata for over 5,000 elements. Next you will filter the collection down by space and time parameters and learn a better way to get a count of the number of images in the image collection.



* + - 1. Comment out the print statement. You will return to this later.

// Print the information about the image collection. // print(landsat8\_collection);

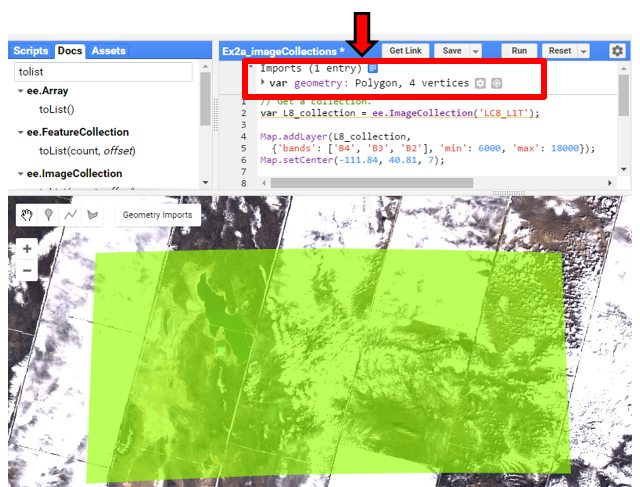
1. Filter Image Collection with a Spatial Boundary
   1. Draw a geometry to use later as a geographic filter (spatial boundary)

A geometry is another geospatial object type in Earth Engine. To load shapefiles or other vector files, you will need to use Fusion Tables, which you will learn about in exercise 4. You can quickly draw lines or polygons by hand or drop points to create a geometry.

* + 1. In the upper left corner of the map window, there are several buttons that you use to draw geometries. These include a small hand (for panning around an image), an upside down tear-drop shape, a line, and a polygon. Click on the polygon.

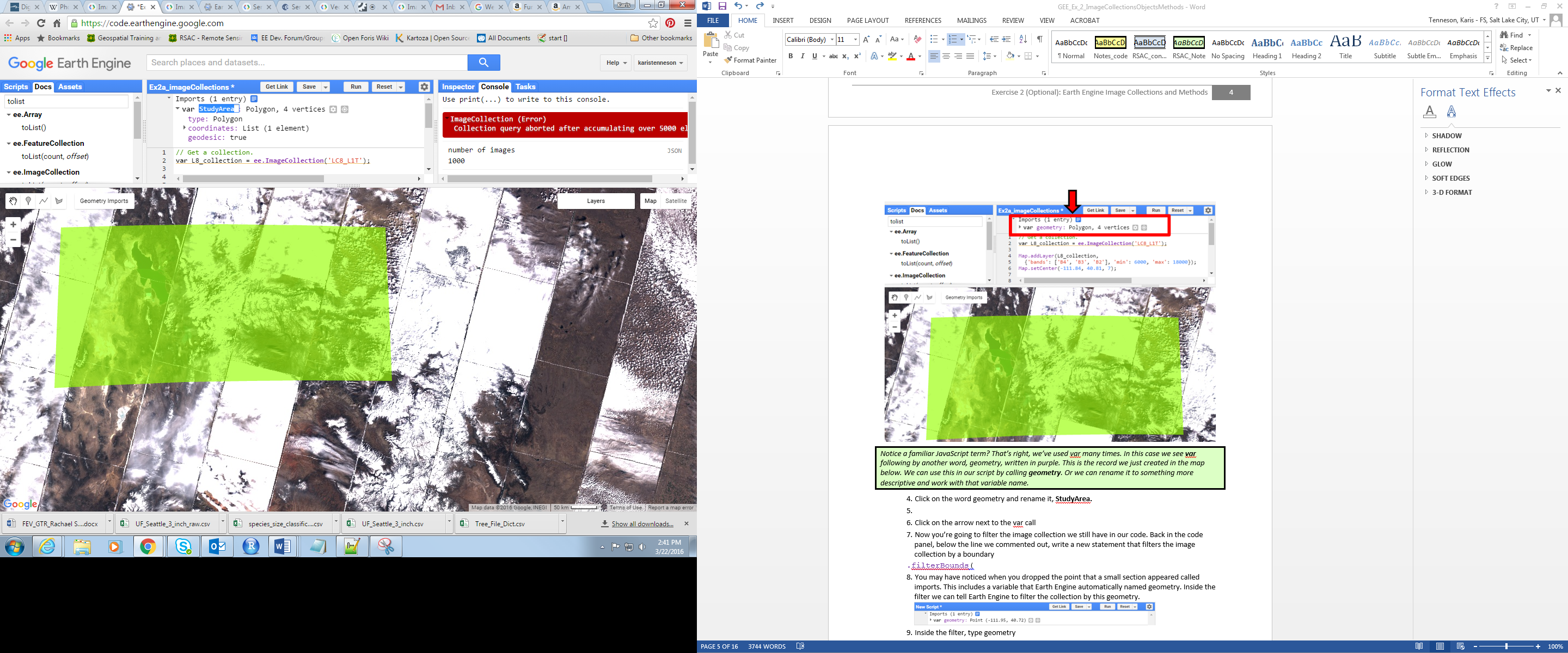


* + 1. This will allow you to draw a geometry that represents your study area of interest. Click in the map display window to create a polygon around an area of interest to you (e.g., around Bangkok or your home town). Remember you can toggle the Landsat image on and off to see the base layer below. Double click to close the polygon.
    2. After you have closed your polygon, there will be an Import record at the top of the code editor panel. Refer to the red arrow and box in the image below.



Notice a familiar JavaScript term? That’s right, you’ve used var many times. In this case **var** is followed by another word, geometry, written in purple. This is the record you just created in the map below. You can use this in your script by referencing the variable named geometry. Or you can rename it something more descriptive by clicking on the variable name in the imports.

* + 1. Click on the word geometry. Change the name from geometry to, studyArea.



* + 1. Click on the arrow next to the var studyArea line to see the details of the geometry you created (see preceding image for an example).
       1. You can show generated code by clicking on the blue box next to the Import line. This code can be copied and pasted into your script below, or any script.
       2. If you hover over the var studyArea line, a trashcan icon appears to the left. This can be used to delete your imported geometry.
  1. Filter image collection by geometry
     1. Now you are ready to filter the image collection, landsat8\_collection. Back in the code panel, copy the lines below and paste them into the code editor. Make sure you copy them below the statement that initially creates the image collection, landsat8\_collection.

// Filter to the scenes that intersect your study region.

**var** landsat8\_studyArea **=** landsat8\_collection**.**filterBounds**(**studyArea**);**

* + 1. Change the variable in the Map.addlayer statement to landsat8\_studyArea (adapted statement is included below). Move it below the statement that creates the landsat8\_studyArea variable.
    2. Change the Map.setCenter statement to map.centerObject and update the input parameters. This will center the map window on the geometry you created (example included below). Move it below the statement that creates the landsat8\_studyArea variable.

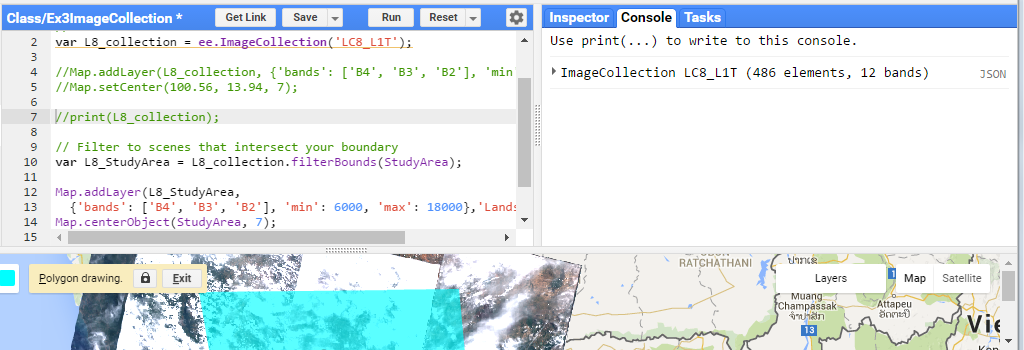
// Display the image.  
Map**.**addLayer**(**landsat8\_studyArea**,   
 {** min: 0.05, max: 0.8, bands: 'B6, B5, B4'**},** 'Landsat 8 in study region'**);**

// Center the display on the study region.

Map**.**centerObject**(**studyArea**,** 7**);**

* + 1. Click Run again. Now the image collection will be filtered to only include images that intersect the polygon that you drew.
    2. *(Optional)* modify the print statement to print the details of your new image collection, landsat8\_studyArea (make sure to move this statement below the creation of the landsat8\_studyArea variable). If your image collection has fewer than 5000 images, information about the collection is printed to the console. The size of your image collection varies depending on the size of the geometry you’ve digitized.

// Print information about the filtered image collection.  
print**(**landsat8\_studyArea**);**



* + 1. You can also use the image collection size() method to determine how many images are in the collection. Copy the following statements and paste them at the bottom of your script. Then Run the script.

// Count and print the number of images.

**var** count **=** landsat8\_studyArea**.**size**();**

print**(**'Count of landsat8\_studyArea: '**,** count**);**

1. Temporally Filter Image Collection
   1. Create a date-limited image collection
      1. Next add a statement that filters your image collection by a length of time, using the filterDate() method (see example script below). filterDate() allows us to specify a start and end date as parameters to reduce the size of the collection to meet your project goals. The new lines add a filter to the data collection, landsat8\_studyArea, based on the date the images were taken.
      2. Next, add a count and print statement to see how many images are in the new image collection.
      3. The complete script is copied below for your reference. Modify your script to match and Run the code.

// Get an image collection.

**var** landsat8\_collection **=** ee**.**ImageCollection**(**'LANDSAT/LC8\_L1T\_TOA'**);**

// Filter the collection to scenes that intersect your study region.

**var** landsat8\_studyArea **=** landsat8\_collection**.**filterBounds**(**studyArea**);**

// Filter the collection to a time period of interest.

**var** landsat8\_SA\_2015 **=** landsat8\_studyArea**.**filterDate**(**'2015-01-01'**,**   
 '2015-12-31'**);**

// Display the image collection and   
// center the map on the study region.

Map**.**addLayer**(**landsat8\_SA\_2015**,**

**{** min: 0.05, max: 0.8, bands: 'B6, B5, B4'**});**

Map**.**centerObject**(**studyArea**,** 7**);**

// Count the number of images.

**var** count **=** landsat8\_studyArea**.**size**();**

print**(**'Count of landsat8\_studyArea: '**,** count**);**

// Count the number of images.

**var** count15 **=** landsat8\_SA\_2015**.**size**();**

print**(**'Count of landsat8\_SA-2015: '**,** count15**);**

* + 1. *(Optional)* Below are some additional methods you can use to explore your image collection. These are taken from the following page in the user documentation: <https://developers.google.com/earth-engine/ic_info>

// Get statistics for a property of the images in the collection.

**var** sunStats **=** landsat8\_studyArea**.**aggregate\_stats**(**'SUN\_ELEVATION'**);**

print**(**'Sun elevation statistics: '**,** sunStats**);**

// Sort by a cloud cover property, get the least cloudy image.

**var** **LoCloudimage** **=**  
 ee**.**Image**(**landsat8\_studyArea**.**sort**(**'CLOUD\_COVER'**).**first**());**

print**(**'Least cloudy image: '**,** **LoCloudimage);**

// Limit the collection to the 10 most recent images.

**var** recent **=** landsat8\_studyArea**.**sort**(**'system:time\_start'**,**   
 **false).**limit**(**10**);**

print**(**'Recent images: '**,** recent**);**

1. Introduction to Reducers
   1. Image Collection Reducers, median pixel value

This section focuses on an important type of object in Earth Engine, reducers. Reducers work on image collections by calculating statistics, such as the mean value for each pixel. The output is an Image object (single raster layer) that characterizes some quality of the complete image collection. To learn more about reducers, visit the User Guide: <https://developers.google.com/earth-engine/reducers_image_collection>.

* + 1. First, simplify the script that you are working with. Modify the script in your Code Editor to match the lines below (or start with a clear script window and copy the lines below into the code editor – just make sure to redraw your study region). Then Run the script.

// Get a collection.

**var** landsat8\_collection **=** ee**.**ImageCollection**(**'LANDSAT/LC8\_L1T\_TOA'**);**

// Filter to scenes that intersect your boundary.

**var** landsat8\_studyArea **=** landsat8\_collection**.**filterBounds**(**studyArea**);**

// Filter to scenes for time period of interest.

**var** landsat8\_SA\_2015 **=** landsat8\_studyArea**.**filterDate**(**'2015-01-01'**,**   
 '2015-12-31'**);**

print(landsat8\_SA\_2015, 'landsat8\_SA\_2015');

* + 1. Now add some statements to create and display a very simple median composite, using the median() image collection reduce method. This will create an Imageobject (single image) representing the median value of each band of all the images in your filtered collection.

// Reduce the ImageCollection to get the median in each pixel.

**var** median\_landsat8\_2015 **=** landsat8\_SA\_2015**.**median**();**

print(median\_landsat8\_2015, 'median\_landsat8\_2015');

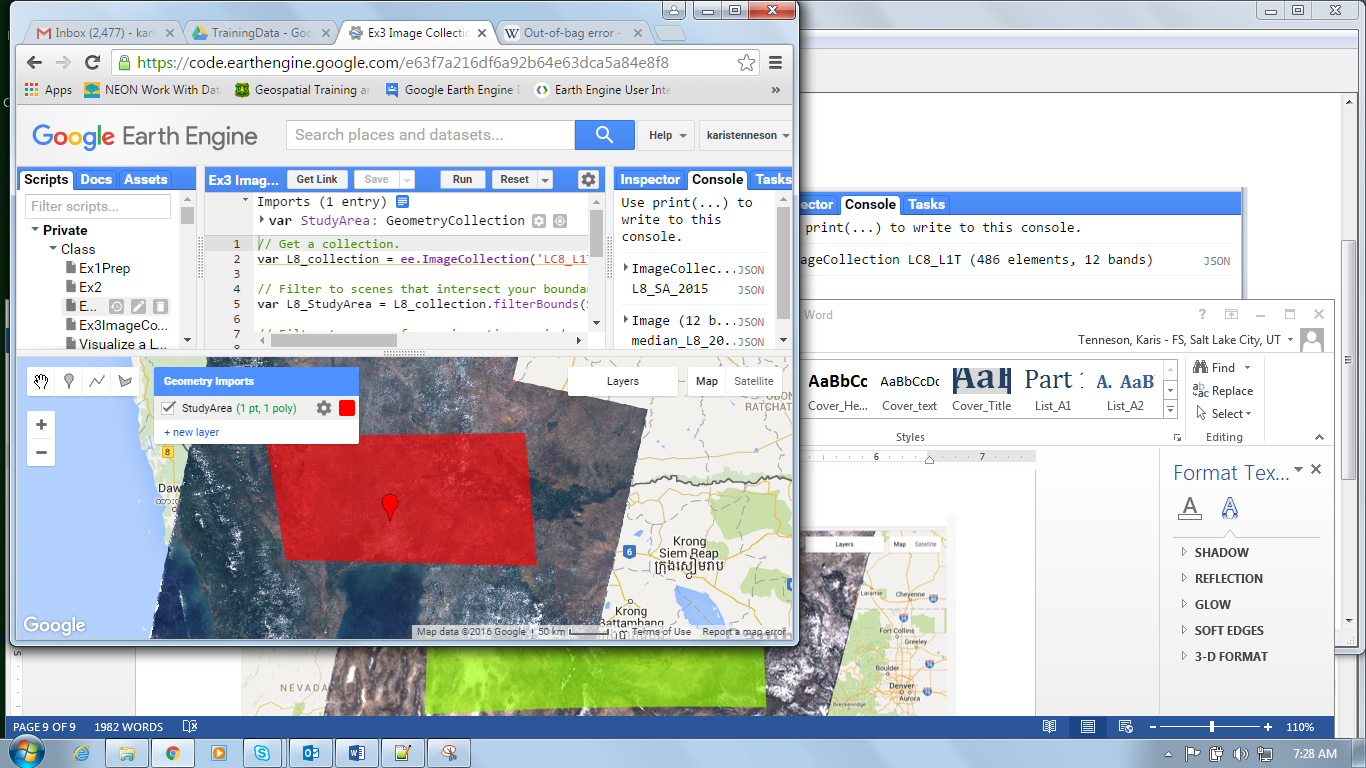
// Display the result and center the map on the study region.

Map**.**addLayer**(**median\_landsat8\_2015**,**

**{** min: 0.05, max: 0.8, bands: 'B6, B5, B4'**});**

Map**.**centerObject**(**studyArea**,** 7**);**

* + 1. Review the result and note the following:
       1. Look at the output from the print statements. Do you see the difference between landsat8\_SA\_2015 and median\_landsat8\_2015? Hint: one is an image collection object, the other is an image object (single raster).
       2. You can turn on the display of your study region by toggling the layer off in the Geometry Imports box in the upper left hand portion of the map screen (refer to following image).
       3. The composite image contains only the Landsat path rows that intersect your study area (zoom out to see the full extent).
       4. The image is also relatively cloud-free since you are viewing median pixel value for all the images in the collection. Are there still clouds present?



Even with running a median reducer, there still might be some pixels that look like clouds. It is better to mask the clouds in each image in the image collection before running the median reducer. In order to do this, you will map a function over the image collection.

* + 1. Save your script, name it ‘Ex 3 Image Collection Part 5’. You will return to it later in the exercise.

1. Functions Primer

As you develop your script, it can start getting quite long. To keep it organized and create more efficient code, you can re-use pieces you’ve built by taking advantage of functions. Functions break apart your code into separate pieces, wrap these pieces up, and give them a name that you can call to apply later when needed. Functions are essentially modular pieces of re-useable code. For example, you might have a function to get the data set(s) of interest, one to analyze them, and finally another to export them. You can break the code up into these three parts – each wrapped up as a function.

Functions are also highly useful when you have a series of statements that you would like to repeat many times in the code. For example, suppose you wanted to calculate the mean of the data or the normalized difference vegetation index (NDVI) of a series of rasters. You could create a function that you could call each time you wanted to execute either of these in your code, rather than have to re-write those pieces each time.

* 1. Structure

Here’s how functions are built:

1. **var**: Declare a variable to store the function in.
2. **functionName:** The word function is followed by the name you would like to call the function. You can name the function anything you like, the same rules and style suggestions of how to name variables apply to naming functions (can’t start with a number, should be descriptive of what it does, etc). Follow this with an equals sign.
3. ***= function*:** Indicate the new variable is a function object. Write the JavaScript word **function**, all in lower case.
4. **Input parameters:** After the word function, add an open and closed parenthesis. These will be filled with the set of input parameters that the user passes to the function, or left empty for functions that don’t have parameters. An input parameter is a piece of information (it can be stored in a variable) that is passed into the function when it is called. If you have many parameters that you would like to include, separate them with commas. To create a function to calculate the NDVI of an image, set the image on which you are computing NDVI as an input parameter.
5. **Curly brackets:** After the parentheses that contain the input parameters, add an open and closed curly bracket.
6. **Code:** Write the code you want to execute within your function (one statement or hundreds of lines) in between the open and closed curly brackets.
7. **Call the function:** Once you’ve created a function, it doesn’t execute unless you call it. To use the function in your code, write the name of the function, followed by parenthesis, with the required input parameters inside the parenthesis, and end the statement with a semicolon.

**var** functionName = **function(**parameter\_1, parameter\_2**){**

// code to execute

// more code to execute

// …

**}**

// call the function

**var** storeOutput = functionName**(**input\_1, input\_2**);**

This is the basic structure of a function. If it sounds complicated, don’t worry, it will all make sense when you write one yourself. Below you’re going to practice by making a simple calculate\_sum function.

**Note:** Common convention declares and specifies (all) the functions first, then calls them later in the script. It’s not required, but it makes for code that is easier to read.

* 1. Create a function to calculate the sum of numbers
     1. Open a new space in which to work by clicking the down arrow next to Reset. Then select Clear Script. Or just open a new tab and go to code.earthengine.google.com.
     2. The code below is an example of a function that calculates the sum of two values.
     3. JavaScript uses a “return” statement to return a local value back to the main program.
     4. By declaring a variable and setting it to the function return value, you create a variable that can be used elsewhere in the script.

**var** calculate\_sum = **function(**in\_value1**,** in\_value2**)** **{**

// Calculate the sum.

**var** sum **=** ee**.**Number**(**in\_value1**).**add**(**ee**.**Number**(**in\_value2**));**

// Return the sum.

**return** sum**;**

**}**

// Now declare a variable and

// set it to the value the function returns.

// Include two numbers to sum as the input parameters.

**var** sum\_test **=** calculate\_sum**(**75**,** 82**);**   
print**(**sum\_test**);**

* + 1. Once a function has been created, you can call the function as many times as you like. Call it again with new parameters. This time add two numbers of your own choice and print the result.

**var** sum\_test2 **=** calculate\_sum**(**790**,** 1.555**);**print**(**sum\_test2**);**

**var** sum\_test3 **=** calculate\_sum**(**133**,** 765**);**print**(**sum\_test3**);**

1. Create a cloud mask function

Recall in Exercise 2, you learned about masking clouds. Now you will write this up as a function that you can call later in the script.

* 1. Add in cloud masking statements from Exercise 2
     1. Copy and paste the lines below into an empty Code Editor window.
        1. The content should look familiar, as it is the process you learned in Exercise 2 to mask clouds.
        2. The var keyword and variable name save the function as an object that you can refer to later, maskClouds.
        3. function()indicates the variable, maskClouds, is a function object type.
        4. Indent all the lines within the function to make it easier to read the code.

// Get an image.

**var** lc8\_image **=** ee**.**Image**(**'LANDSAT/LC8\_L1T\_TOA/LC81290502013110LGN01'**);**

// Specify the cloud likelihood threshold.

**var** cloud\_thresh **=** 40**;**

// Create the cloud masking function.

**var** maskClouds **=** **function(image){**

// Add the cloud likelihood band to the image.

**var** cloudScore **=** ee**.**Algorithms**.**Landsat**.**simpleCloudScore**(image);**

// Isolate the cloud likelihood band.

**var** cloudLikelihood **=** cloudScore**.select(**'cloud'**);**

// Compute a mask in which pixels below the threshold are 1.

**var** cloudPixels **=** cloudLikelihood**.**lt**(**cloud\_thresh**);**

// Mask these pixels from the input image.

// Return the masked input image.

**return** image**.**updateMask**(**cloudPixels**);**

**}**

// Run the function on the lc8\_image and save the result.

**var** lc8\_imageNoClouds **=** maskClouds**(**lc8\_image**);**

// Review the masked image and assess the output.

Map.addLayer**(**lc8\_imageNoClouds**,**   
 **{**bands: **[**'B6'**,** 'B5'**,** 'B4'**],** min: 0.1**,** max: 0.5**},** 'Landsat8scene\_cloudmasked'**);**

* + 1. Save your script as “Ex3\_CloudMaskFunction”.

1. Mapping Functions Across Image Collections

Now that you know the function works, apply it to the image collection that you created in the first half of this exercise using *map*.

* + 1. Replace the *lc8\_image* with the filtered image collection into the script. See full example below.

// Store the Landsat 8 image collection in a variable.

**var** landsat8\_collection **=** ee**.**ImageCollection**(**'LANDSAT/LC8\_L1T\_TOA'**);**

// Filter to scenes that intersect your study region.

**var** landsat8\_studyArea **=** landsat8\_collection**.**filterBounds**(**studyArea**);**

// Filter to scenes for your time period of interest.

**var** landsat8\_SA\_2015 **=** landsat8\_studyArea**.**filterDate**(**'2015-01-01'**,**   
'2015-12-31'**);**

// Specify the cloud likelihood threshold.

**var** cloud\_thresh **=** 40**;**

// Create the cloud masking function.

**var** maskClouds **=** **function(image){**

// Add the cloud likelihood band to the image.

**var** cloudScore **=** ee**.**Algorithms**.**Landsat**.**simpleCloudScore**(image);**

// Isolate the cloud likelihood band.

**var** cloudLikelihood **=** cloudScore**.select(**'cloud'**);**

// Compute a mask in which pixels below the threshold are 1.

**var** cloudPixels **=** cloudLikelihood**.**lt**(**cloud\_thresh**);**

// Mask these pixels from the input image.

// Return the masked input image.

**return** image**.**updateMask**(**cloudPixels**);**

**};**

* + 1. Now add a statement that maps the function across the image collection. Then add the image collection with clouds masked to the map display. See example map statement below.

// Mask the clouds from all images in the image collection   
// with the map function.

**var** landsat8\_SA\_2015NoClouds **=** landsat8\_SA\_2015**.**map**(**maskClouds**);**

// Add the first masked image in the collection to the map window.

Map**.**addLayer**(**ee**.**Image**(**landsat8\_SA\_2015NoClouds**.**first**()),**

**{**min**:**0.05**,** max**:** 0.8**,** bands**:** 'B6, B5, B4'**},**   
 'first image with clouds masked'**);**

// Center your map.

Map**.**centerObject**(**studyArea**,** 7**);**

* + 1. Use a median reducer on the landsat8\_SA\_2015NoClouds image collection that you just created to aggregate the information from all the images in the collection. See example code below.

// Reduce the collection to the median value per pixel.

**var** median\_L8\_2015 **=** landsat8\_SA\_2015NoClouds**.**median**();**

// Print the information of the reduced image.

print**(**median\_L8\_2015**,** 'median\_L8\_2015'**);**

// Display reduced image in the map window.

Map**.**addLayer**(**median\_L8\_2015**,**

**{**min**:** 0.05**,** max**:** 0.8**,** bands**:** 'B6, B5, B4'**},**   
'median composite of cloud free images'**);**

Please contact us at [rsac\_gee@fs.fed.us](mailto:rsac_gee@fs.fed.us) with any questions.



This document is a work of the U.S. federal government, such work is in the public domain in the United States. See the USDA [website and copyright policy](https://www.usda.gov/wps/portal/usda/usdahome?navid=POLICY_LINK) for more information.