USDA
United States Department of Agriculture

Last Updated: July 2017

EXERCISE 4

Geospatial Programming in JavaScript in Google Earth Engine

Introduction

This exercise focuses on using JavaScript in the Google Earth Engine Code Editor. This exercise quickly introduces tools available in Earth Engine, and builds on much of the information that you learned in exercises 1-3. It is designed to allow you to apply many of the concepts you learned in the earlier exercises and see how to use them in a geospatial programming environment.

Objectives

- Apply and expand on the knowledge you learned in exercises 1-3
- Learn more about Earth Engine

Required Data

- **Google Earth Engine Account**
- **Some knowledge of the Code Editor window to navigate the page (see Appendix)**

Prerequisites

- *Introduction to Geospatial Scripting Exercise 1: Planning a script.*
- *Introduction to Geospatial Scripting Exercise 2: Concepts of Programming, Part 1*
- *Introduction to Geospatial Scripting Exercise 3: Concepts of Programming, Part 2*
- An active Google Earth Engine account.

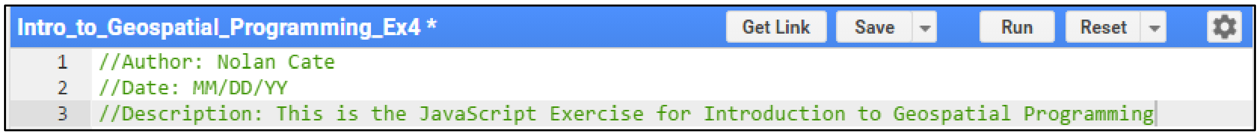
Table of Contents

Part 1: Setting up your Script

You learned in earlier exercises about how to place comments in a script and about the importance of writing a header into your script. This section of the exercise will walk you through that process.

A. Creating a Header and Saving

1. Navigate to https://code.earthengine.google.com/. Make sure you are signed in to your Earth Engine Account. If you need to sign up for an account, or become more familiar with the Code Editor layout, see the Appendix.
2. As you learned in exercise 1, it is a good practice to include a header in your script. On the first three lines **Add comments** with your name, the date, and a description of the script, just like you did in exercise 1. Remember that typing // will comment out a line in JavaScript. Your script should now look similar to the screenshot below:

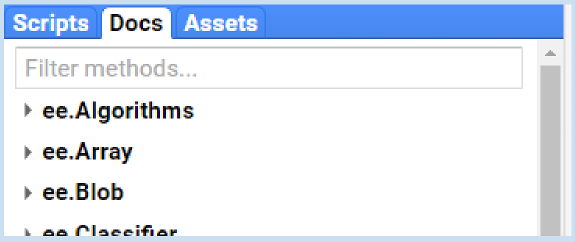


```
//Author: Nolan Cate
//Date: MM/DD/YY
//Description: This is the JavaScript Exercise for Introduction to Geospatial Programming
```

Figure 1: Screenshot of your script at this stage of the exercise. This is an example of a header

3. Now that you have a header, you should save the script. **Click** Save at the top of the page. Give the script a name, something similar to "Intro_to_Geospatial_Scripting_Ex4". You should now see the script in your private repository on the left side of the Code Editor.

Note: *All of the tools used in this exercise and many more are described in the "Docs" tab in the left pane in the Earth Engine Code Editor. As you move through the exercise, it may be very helpful for you to refer to the "Docs" tab for a description of the tools you're using. Furthermore, as you move on to write your own scripts it is always useful to refer to this tab to make sure you're using Earth Engine tools correctly.*



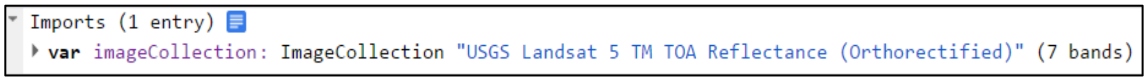
Part 2: Image Collections

Exercises 1-3 guided you through the process of planning a script and introduced basic programming concepts, such as variables, operators, conditional statements, objects and methods. Now you're going to apply many of those concepts to build a script in Earth Engine. It is recommended that you have the previous exercises to refer to if you've forgotten some of the terminology.

A. The Earth Engine Data Catalog

The data catalog is where information about all the imagery in Earth Engine is stored. As you begin using scripting more in your geospatial analyses, much of this information is going to be very useful to you. The Earth Engine library holds petabytes of information, so if you're ever unsure if there is data that you need in Earth Engine, the data catalog is where you will search for it. Today you will be working with imagery from a Landsat sensor. The steps for finding Landsat imagery are outlined below.

1. At the top of the code editor window you will see a search bar that says "Search Places and Datasets". In the search bar **Type**: USGS Landsat 5 TM TOA Reflectance (Orthorectified). Under Rasters, **Click** USGS Landsat 5 TM TOA Reflectance (Orthorectified).
2. You have now opened a small pop-up data page for Top of Atmosphere calibrated Landsat 5 imagery. The main body of the page has information about the imagery and how it was collected. The right side of the page has useful metadata. At the bottom is the ImageCollection ID, which is how you will reference the images in your script. Below the ImageCollection ID is a blue button labeled "Import"
3. **Click** Import. You should see the image collection imported at the top of your script, as in the screenshot below



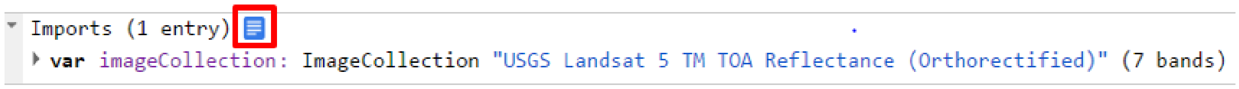
```
Imports (1 entry)
  var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
```

Figure 2: A screenshot showing imports at the top of your code editor window

B. Earth Engine Image Collections

In exercise 3, we learned that objects are sets of data, defined by a class, that have a series of properties. Earth Engine has many of its own unique object types. In Earth Engine Raster data are represented as Earth Engine image objects. Image objects have one or more bands, and they have metadata which come as a series of properties associated with the image object. Image collection objects are similar to image objects in that they function as a single object. However, they hold many images, not just one. This means that they have different methods that work on them. One common Earth Engine mistake is to try using Image object methods on an Image Collection object.

1. In your code editor window you are going to create the image collection variable using the ImageCollection ID you imported in the last step. In the imports section at the top of your code, click the "show generated code" button, seen in the screenshot below:



```
Imports (1 entry)
  var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
```

Figure 3: This screenshot shows the location of the "show generated code" button

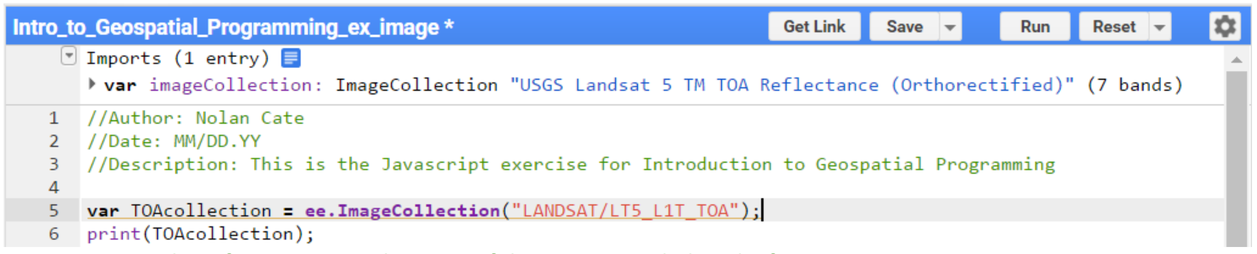
2. In the pop-up window that opens, **Copy and Paste** the text into your code. It should read:

```
var imageCollection = ee.ImageCollection('LANDSAT/LT5_L1T_TOA');
```

Note: *the "ee" prefix in ee.ImageCollection tells us that the variable is an instance of an Earth Engine class. Other Earth Engine classes include ee.Array, ee.Number, ee.Chart, and more. The key thing to remember is that Earth Engine objects are processed on Google servers rather than your local machine, which decreases processing time. For more information, see https://developers.google.com/earth-engine/client_server*

3. Now you have a variable that you can work with in your script. But the name "imageCollection" isn't very descriptive. Change the name of the variable to TOAcollection. The line should now read:
```
var TOAcollection = ee.ImageCollection('LANDSAT/LT5_L1T_TOA');
```
4. To explore it, **Type** Enter to get to line 2, then issue a print statement on this line by **Typing:**
```
print(TOAcollection);
```

You may copy and paste the code from this exercise into the code editor window, but you are encouraged to type it in yourself as it may help you learn the code faster. At this point, your code should look like the image below:



```
Imports (1 entry)
 var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA");
print(TOAcollection);
```

Figure 4: Screenshot of your script at this stage of the exercise, including the first print statement

5. **Click** Run. In the Console pane on the right you will see an image collection line appear, which a processing symbol next to it. GEE will take a while to run, but after some time it will error, telling you that it has accumulated over 5000 elements. Though GEE can perform computations on many more images, it cannot provide metadata on more than 5000 images at once.
6. This highlights the importance of learning to filter the image collections. Because of the large amount of data in EE, this is a very important method.

C. Filtering an Image Collection

1. You're going to filter the image collection by location, but to do so you need to be able to specify the location. **Move** the TOAcollection statement one line down. Then on the now blank line above it, **create** a new variable called GTACpoint
```
var GTACpoint
```
2. Set this variable equal to an Earth Engine Geometry point using ee.Geometry.Point()
```
var GTACpoint = ee.Geometry.Point()
```
3. Now in the parentheses at the end of ee.Geometry.Point, add in the longitude/latitude coordinates -112,40.7, which is in Salt Lake City, Utah, where GTAC is located.

```
var GTACpoint = ee.Geometry.Point(-112,40.7)
```

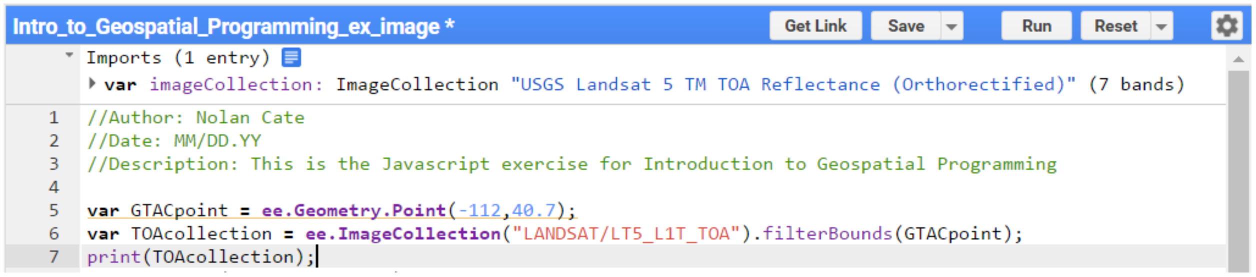
4. Now in your TOAcollection statement, and add the method .filterBounds() after the image collection, before the semicolon. There are a number of ways to filter an image collection, including by date (.filterDate()), by properties (.filterMetadata()), or by location using .filterBounds(), which is what you're doing here. Recall from exercise 3 that methods are like functions that work with a specific type of object. The .filterBounds() method is a method for working with image collection objects. The unfinished statement will now look like this:

   ```
   ee.ImageCollection('LANDSAT/LT5_L1T_TOA').filterBounds(
   ```

5. Now you must give the .filterBounds() method a location to filter the image collection by. In the parentheses after the method, add the variable, GTACpoint, which you just created. This line should now read:

$$\underbrace{\texttt{ee.ImageCollection('LANDSAT/LT5\_L1T\_TOA')}}_{\text{This is the line of code that you copied from your import record in section A, which defines your image collection as Landsat imagery}}\underbrace{\texttt{.filterBounds(GTACpoint))}}_{\text{This method filters the images in your collection by the point variable, which is an Earth Engine Geometry object that you just created.}}$$

6. Click run again. Now GEE will only print images that intersect with the point that you've filtered by. It won't add them to the map (You will learn that command soon), but in the console on the right, you can see information about the image collection. Your code should now look like this:



```
Imports (1 entry)
 var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)

//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
```

Figure 5: Screenshot of your script at this stage of the exercise after filtering the image collection

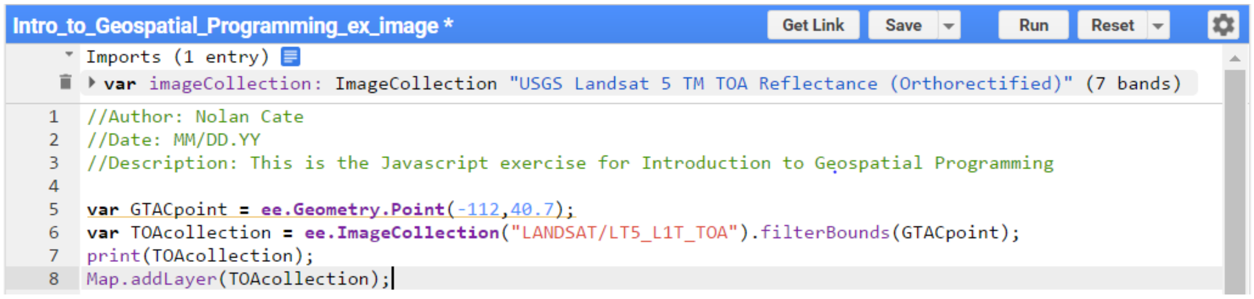
D. Adding a Layer to the Map

Printing a variable simply displays metadata about the object in the console. While this is invaluable information, you will also need to display the images visually.

1. The first line below your header is holding your filtered image collection variable. The next line has a geometry object used to filter the image collection and the line after has a print statement. Now place a statement to add the collection layer to the map. There are a number of Map methods you can use to manipulate your display (adjusts the center and the zoom of the map when you click run). Now we're going to use the Map.addLayer command. **Type:**

   ```
   Map.addLayer(
   ```

2. The code editor will automatically close the parentheses for you. If it doesn't, you need close the parentheses, or the code editor will show a red x to the left of the line that you're writing. You need to pass arguments to Map.addLayer for it to work. In this case the argument will be the variable that we want to add to the map. **Type,** TOAcollection in the parentheses. This line should now read:
   ```
   Map.addLayer(TOAcollection);
   ```
3. Drag the map using your mouse and use the scroll wheel to center your map over Northern Utah.
4. **Click** Run. GEE may take a few moments to run, but it will display an image in your map. You can use the "Layers" tool in the upper right corner of the map window to turn the layer on and off, and adjust the opacity. In the next section you will learn to manipulate the display settings of the image. Your code should now look like this:



```
Imports (1 entry)
 var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)

//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
Map.addLayer(TOAcollection);
```

Figure 6: A screenshot of your script at this stage of the exercise including adding the collection to the map

Part 3: Reducers and Operators

A. Creating a Composite Image and Adjusting the Display

In this section, you will use an image collection method to reduce the image collection into a single composite image. There are a number of ways to do this. In this exercise you are going to take the median value from the image collection. For more information on this reducer and other reducers, see: https://developers.google.com/earth-engine/reducers_intro.

1. You are going to use the .median method to reduce the image collection. This is a very simple, single line statement. **Declare** a new variable called medianImg and set it equal to the collection variable with the median method. See below.

   ```
   var medianImg = TOAcollection.median();
   ```

   This declares a variable and names it medianImg.

   This sets the variable equal to the TOAcollection variable, but it runs the .median() method. The .median() method is a reducer specific to image collections. It takes each image in the collection and calculates the median value of each pixel for each band. The result is not an image collection, but a single image.

2. To examine the metadata of the new image, add another print statement on the line below where you calculated the median image.

```
print(medianImg);
```

3. **Click** Run. In the console on the right you will see the image collection printed out, as you did before. But now you will also see the median composite image from the median image you just calculated using the .median() reducer. Explore the metadata of the image by clicking the dropdown menu on the image in the console, then clicking the bands dropdown. Notice the bands are named “B1”, “B2”, “B3”, etc.
4. Now add the median image to the map, the same way you added the image collection to the map.

```
Map.addLayer(medianImg);
```

5. **Click** Run. Like before, it will take a few moments for the script to run. But now the top layer will be a composite image, made up of the median values of each pixel in the large stack of images in the image collection. Because clouds in images are outlier values, taking the median values over a collection will yield a cloud-free composite, though it still appears dark in the map. Now you will learn to adjust the display settings programmatically.
6. The Map method .addLayer() can take a number of arguments (remember that arguments are what needs to be passed to a method). The only required argument is the layer to display. But you can add arguments to instruct the method about *how* the image should be displayed. Remember from exercise 3 that arguments in a method need to be separated by a comma. In the line with the Map.addLayer() method, begin by **typing** a comma after the layer, medianImg.
7. After the comma, we will add in the bands to display and the minimum and maximum values. All of this information needs to go in curly brackets ({}), so after the comma you just typed, **Type** {} (the brackets should auto complete themselves).

Note: *The information in the curly brackets is called a dictionary.*

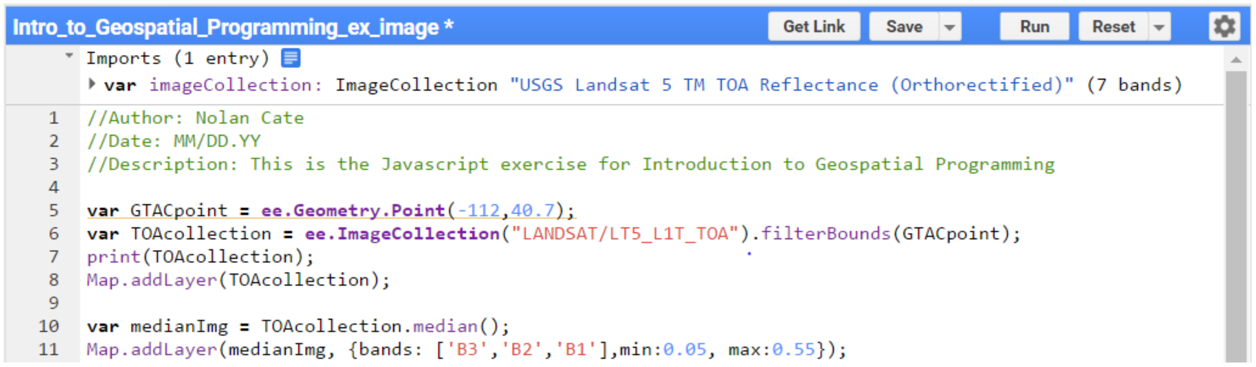
8. Now within the brackets **Type** bands:. This is telling the method which of the Landsat bands you want to display. It is expecting a list so after the colon **type** [‘B3’,’B2’,’B1’]. This is a list of the “True color” Landsat 5 bands. Your line should now read:

```
Map.addLayer(medianImg, {bands: ['B3','B2','B1']});
```

9. Now Earth Engine will be using the appropriate bands for display, but if you click run the image will still appear dark. *After* the last square bracket but *before* the last curly bracket **type** another comma. Now **type** min:0.05. **Type** one more comma and then max:0.55. Because this image collection is showing reflectance values, every pixel value in each band will be between 0 and 1 (a ratio of how much light is reflected). Setting the min and max just adjusts the image display to make it easier to see. Your line will now read:

```
Map.addLayer(medianImg, {bands: ['B3','B2','B1'],min: 0.05, max: 0.55});
```

10. Make sure that you have entered the statement with correct syntax, then **click** Run. The image will display on your map, as before, but should now be much easier to see. Your code should look like the screenshot below.



```
Imports (1 entry)
var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
```

```
//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
Map.addLayer(TOAcollection);

var medianImg = TOAcollection.median();
Map.addLayer(medianImg, {bands: ['B3','B2','B1'],min:0.05, max:0.55});
```

Figure 7: A screenshot of your script at this stage of the exercise showing the .median() reducer and display properties

B. Operators in Earth Engine

So far, you have used a number of Earth Engine methods. These methods are simply functions that are specific to an object type. Now we're going to use some operators (in this case, arithmetic operators) to do some band math on an image. The variable you are going to use is the median image you created in the last section. In this section you're going to use these operators to create a Normalized Difference Vegetation Index (NDVI). To read more about NDVI, visit: http://earthobservatory.nasa.gov/Features/MeasuringVegetation/measuring_vegetation_2.php

1. Begin by **declaring** a new variable called ndvi.

2. Now we will set the ndvi variable to the ndvi equation. See below.

$$ndvi = \frac{NIR - red}{NIR \ + red}$$

To understand this equation you must remember that the image collection we're using contains images from Landsat 5. Landsat 5 TM images have 7 image bands, each of which represents reflectance in a specific region of the electromagnetic spectrum. In the case of Landsat 5 TM, band 4 represents reflectance in the Near Infrared (NIR) wavelengths of light, and band 3 represents red light. To read more about Landsat bands, visit: http://landsat.usgs.gov/best_spectral_bands_to_use.php

3. Start with the top of the equation. We need to get just band 4 and band 3 out of the median image. Recall from the previous section that bands are named "B1", "B2", "B3", etc. To pull out an individual band, you must use the .select method. So in this case, band 4 of the median image, which is NIR in our equation is:
   ```
   medianImg.select('B4')
   ```
   So the numerator of the equation (NIR-red) could be expressed as medianImg.select('B4') - medianImg.select('B3')
4. Before you enter the equation, however, you need to know that Earth Engine, on top of having its own object types, also has its own operators, with syntax like a method. This was touched on in exercise 2. Instead of a subtraction symbol, it will work better in Earth Engine

to use .subtract() and pass an argument of an image to be subtracted. So instead the numerator of the NDVI equation *should* be expressed as:

```
medianImg.select('B4').subtract(medianImg.select('B3'))
```

Now with all of this information, can you script the entire NDVI equation?

5. Set the variable ndvi equal to the NDVI equation programmed into Earth Engine with the following statement:

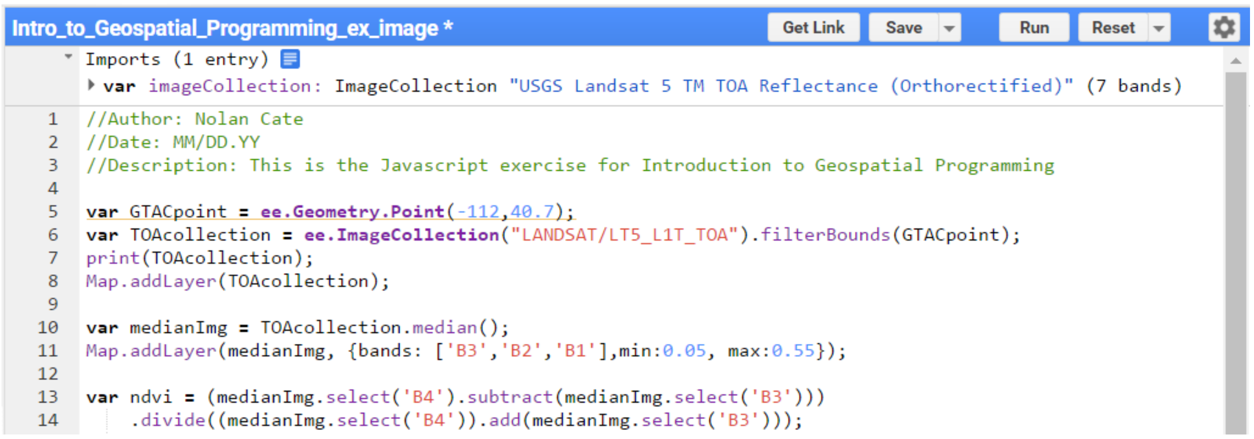
```
var ndvi = (medianImg.select('B4').subtract(medianImg.select('B3')))
     .divide((medianImg.select('B4')).add(medianImg.select('B3')));
```

These operators (.add(), .subtract(), .divide(), and .multiply()) are Earth Engine specific and will not work in other JavaScript programs. For more Earth Engine Specific operators, see the glossary or for the most up to date information, the Earth Engine page, https://developers.google.com/earth-engine/image_math.

7. Now add this image to the map, just as you have done with previous images

```
Map.addLayer(ndvi);
```

Your code should look like the screenshot below.



Intro_to_Geospatial_Programming_ex_image *

```
Imports (1 entry)
 var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)

//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
Map.addLayer(TOAcollection);

var medianImg = TOAcollection.median();
Map.addLayer(medianImg, {bands: ['B3','B2','B1'],min:0.05, max:0.55});

var ndvi = (medianImg.select('B4').subtract(medianImg.select('B3')))
    .divide((medianImg.select('B4')).add(medianImg.select('B3')));
```

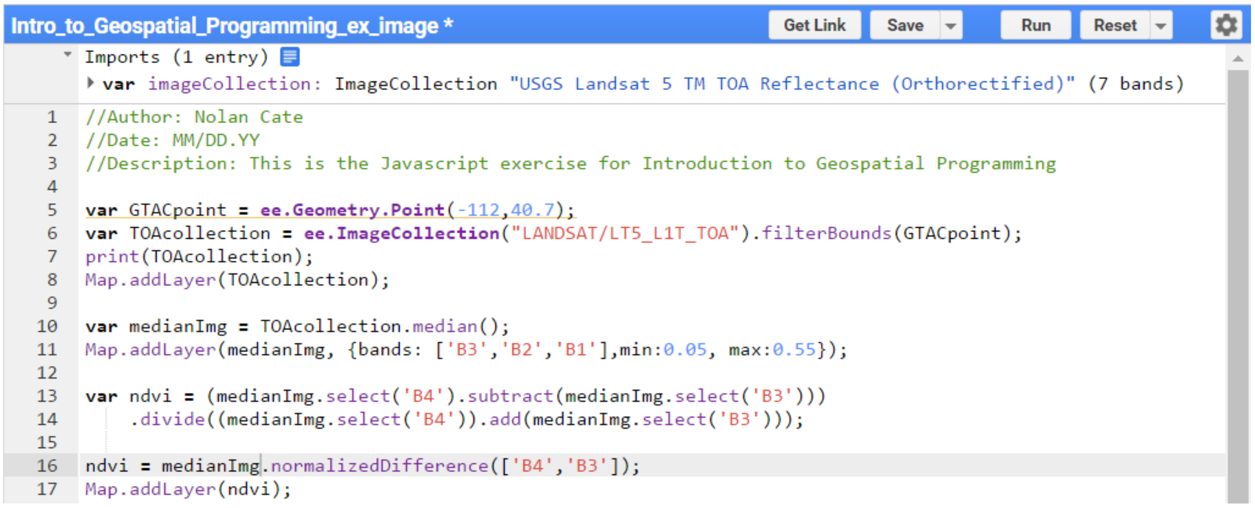
Figure 8: A screenshot of your script at this stage of the exercise showing ndvi being calculated

8. **Click** Run. Watch as a layer highlighting healthy vegetation fills your map screen.
When writing a script, there is usually more than one way to do something. Creating an NDVI layer this way is a good way to learn and practice using Earth Engine operators. But there are several other, cleaner ways, of calculating an NDVI image. One would be to use the .expression() method. For more information on this, see: https://developers.google.com/earth-engine/image_math#expressions.
Before you move on to the next section, try calculating NDVI the easiest way. Instead of entering a long equation, there is a simple method we use that will take two bands of an image and calculate a normalized difference index.
9. On the next line, use the normalizedDifference method to calculate NDVI. See the statement below:

```
var ndvi = medianImg.normalizedDifference(['B4','B3']);
```

This method takes an image (in this case, the image you named medianImg) and performs a normalized difference on a list of 2 bands that you pass it (in this case the list holds the bands 4 and 3). This shows that, like many things in programming, there is not a single correct way of getting something done. But learning more vocabulary of the language will likely help you save time.

At this point, your code should look similar to the screenshot below:



```
Imports (1 entry)
var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
```

```
//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
Map.addLayer(TOAcollection);

var medianImg = TOAcollection.median();
Map.addLayer(medianImg, {bands: ['B3','B2','B1'],min:0.05, max:0.55});

var ndvi = (medianImg.select('B4').subtract(medianImg.select('B3')))
    .divide((medianImg.select('B4')).add(medianImg.select('B3')));

ndvi = medianImg.normalizedDifference(['B4','B3']);
Map.addLayer(ndvi);
```

Figure 9: A screen capture of the code editor window, showing what the code should look like at the end of section 2

Part 4: Creating a Function

This section will give you an opportunity to practice writing a function, which you learned about in exercise 3. A function allows you to wrap up long pieces of code so they can be used again later. Then you can use them multiple times in a script, the same way you use a method.

A. Setting Up a Function

1. In this piece of the exercise you will create a function called cloudMask. The arguments you need to pass this function is just an image to create the cloud mask for. The correct syntax for this function is:

```
function cloudMask (image){
}
```

Note: *Recall from exercise 3 that a JavaScript function must start and end with curly brackets. In GEE the code editor should automatically add an ending bracket when you type a beginning bracket. But if it doesn't, be sure to add it yourself.*

2. What you have now is a complete function. But right now there is no code within the function to run, and remember that the function will not run anyways, unless we explicitly call it. Before we call it, let's fill it in with some code.

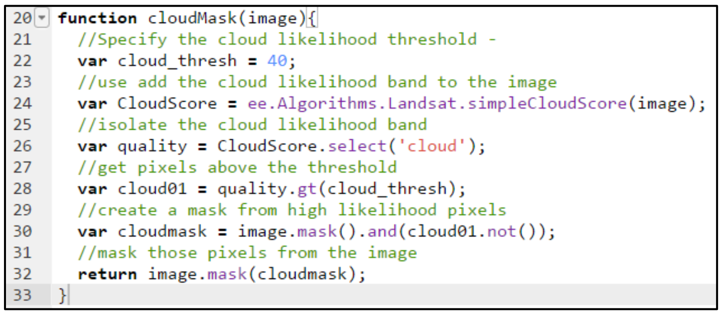
B. Writing the Body of the Function

The cloud masking function you're writing in this section contains a series of complicated steps. Don't worry if you don't completely understand every line of code. The purpose of this section is more to understand how to set up a function in a script. If you are interested in more training on the Earth Engine code in this section, see Part 6: Notes and Other Resources. The following steps guide you through writing the body of the cloudMask function.

1. Begin by specifying a cloud likelihood threshold. On the line after the first curly brace of your function, write the code:

   ```
   //Specify the cloud likelihood threshold -
    var cloud_thresh = 40;
   ```
   Hit Enter
2. On the next line, use a pre-defined Earth Engine Algorithm to add a cloud likelihood band to the image you passed to the function:
   ```
   //use add the cloud likelihood band to the image
    var CloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);
   ```
   Hit Enter
3. On the next line, select the cloud likelihood band, called "cloud" by default. Use the .select() method you just learned shown in the following code:
   ```
   // isolate the cloud likelihood band
   var quality = CloudScore.select('cloud');
   ```
   Hit Enter
4. On the next line, create a variable that gets the pixels greater than the cloud threshold you set in step 1, using the following code:
   ```
   // get pixels above the threshold
   var cloud01 = quality.gt(cloud_thresh);
   ```
   Hit Enter
5. On the next line, use the .mask() method and an image logical operator methods to create a mask using the pixels from the cloud01 variable you created in the last step. Use the following code:
   ```
   // create a mask from high likelihood pixels
   var cloudmask = image.mask().and(cloud01.not());
   ```
   Hit Enter
6. On the last line of the function, have the function return the image that it was passed, with the newly identified cloudy pixels masked, using the following code:
   ```
   // mask those pixels from the image
   return image.mask(cloudmask);
   ```
   Hit Enter

The function that you have just written can be given a Landsat image in Earth Engine and mask out cloudy pixels that it identifies in the image so they they're not used in your analysis. At the end of step 6, your function should look like Figure 4.



```
function cloudMask(image){
  //Specify the cloud likelihood threshold -
  var cloud_thresh = 40;
  //use add the cloud likelihood band to the image
  var CloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);
  //isolate the cloud likelihood band
  var quality = CloudScore.select('cloud');
  //get pixels above the threshold
  var cloud01 = quality.gt(cloud_thresh);
  //create a mask from high likelihood pixels
  var cloudmask = image.mask().and(cloud01.not());
  //mask those pixels from the image
  return image.mask(cloudmask);
}
```

Figure 10: A screenshot of the function written in section B

Again, don't worry if you don't completely understand every line of this function. What is more important in this exercise is that you understand the structure of the function, and could replicate it in a script you write on your own.

C. Using the Function

When we call the function, we will need to pass an argument (in this case, an image) to tell it which image to mask clouds from. Our cloudy images, however, are in the TOAcollection, which is an image collection, not a single image. Try pulling out a cloudy image from the collection using the steps below.

1. On a new line below your function, **Create** a new variable called maskedImage.
2. You want the masked image equal to the output of an image from the cloudMask function. So set the variable equal to cloudMask().
3. In the parentheses of cloudMask you need to pass an image argument. Begin by using ee.Image. So far your line of code should read:

```
var maskedImage = cloudMask(ee.Image())
```

4. In the ee.Image parentheses you're going to pull an image out of TOAcollection by first sorting the collection by cloud cover. Use the .sort() method to sort by the property CLOUD_COVER.

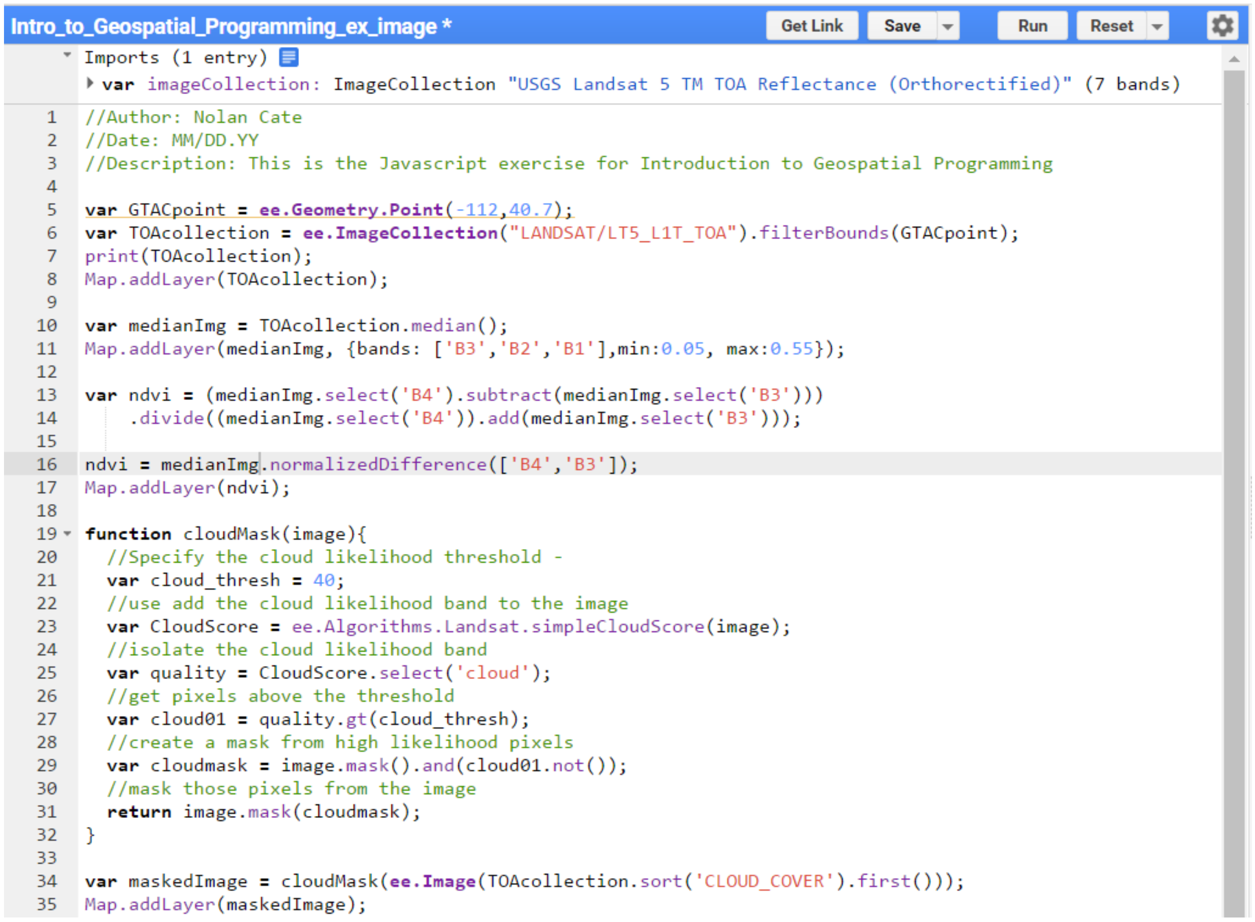
```
var maskedImage =
cloudMask(ee.Image(TOAcollection.sort('CLOUD_COVER')))
```

Notice that the property you're sorting by needs to be a string.

5. Now the TOAcollection is sorted by cloud cover. To pull a single image out, use the .first() method. This method needs no arguments.

```
var maskedImage =
cloudMask(ee.Image(TOAcollection.sort('CLOUD_COVER').first()));
```

6. Now the maskedImage variable is set to the output of a single image from the TOAcollection that has been run through the cloudMask function. To visualize the image, add it your map using the Map.addLayer() command that you used in the previous section. Your code will look like the image below:



```
Imports (1 entry)
▸ var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
```

```
//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
Map.addLayer(TOAcollection);

var medianImg = TOAcollection.median();
Map.addLayer(medianImg, {bands: ['B3','B2','B1'],min:0.05, max:0.55});

var ndvi = (medianImg.select('B4').subtract(medianImg.select('B3')))
    .divide((medianImg.select('B4')).add(medianImg.select('B3')));

ndvi = medianImg.normalizedDifference(['B4','B3']);
Map.addLayer(ndvi);

function cloudMask(image){
  //Specify the cloud likelihood threshold -
  var cloud_thresh = 40;
  //use add the cloud likelihood band to the image
  var CloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);
  //isolate the cloud likelihood band
  var quality = CloudScore.select('cloud');
  //get pixels above the threshold
  var cloud01 = quality.gt(cloud_thresh);
  //create a mask from high likelihood pixels
  var cloudmask = image.mask().and(cloud01.not());
  //mask those pixels from the image
  return image.mask(cloudmask);
}

var maskedImage = cloudMask(ee.Image(TOAcollection.sort('CLOUD_COVER').first()));
Map.addLayer(maskedImage);
```

Figure 11: A screenshot of your script at this stage of the exercise, including the cloudMask function and a masked image

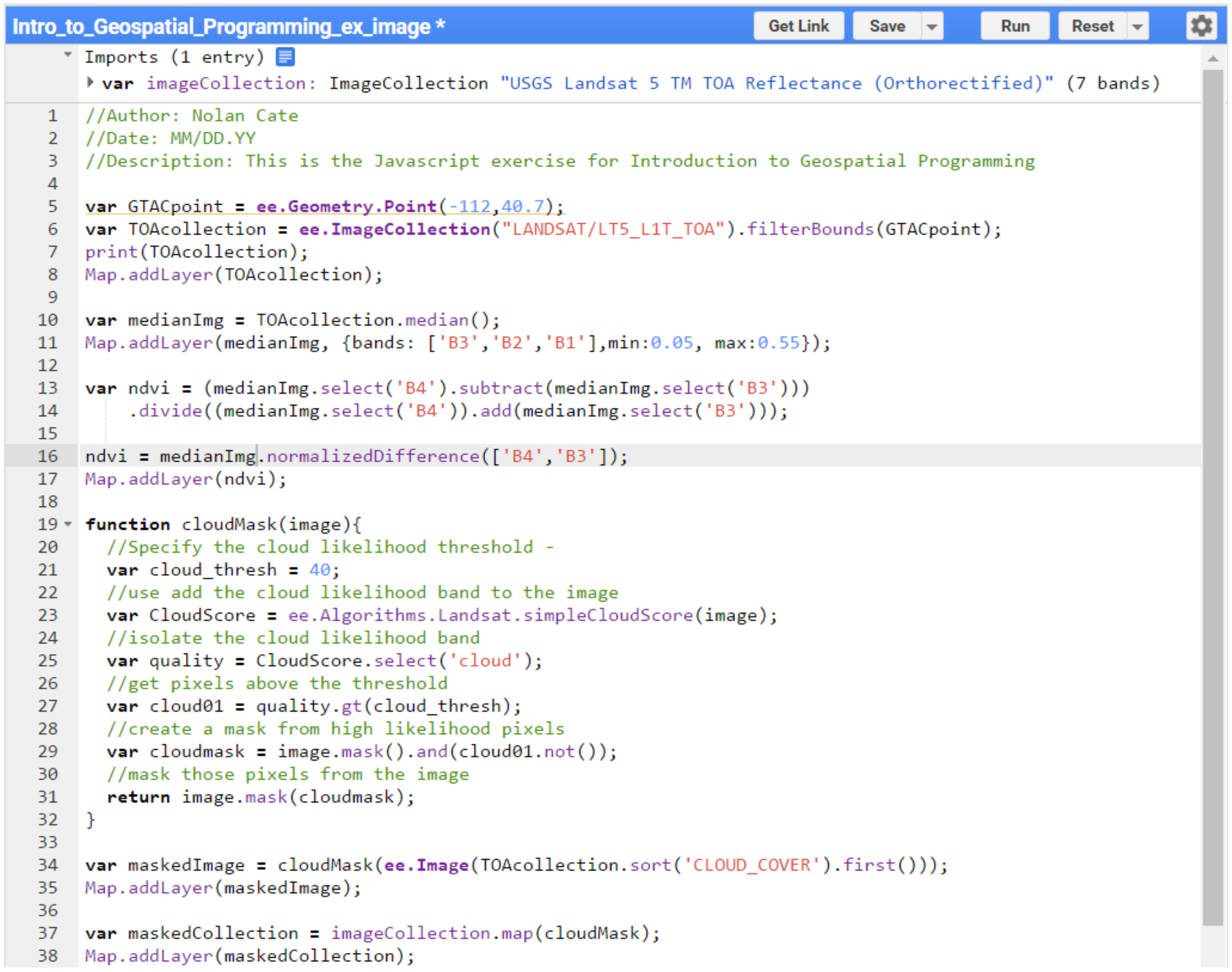
7. **Click** Run. You should now have 4 layers in your map. Notice the image that is added to the map is a Landsat image, but the clouds have been removed from the image.

D. Optional: Mapping the Function Over a Collection

When you call your custom cloudMask function, you will need to pass an image argument, like you did in section C. But what if you want to mask the cloudy images from the entire TOAcollection, which contains many images? Outside of Earth Engine, you could use a loop to run the function on each image in the collection individually. In Earth Engine, loops are frowned upon, because they cannot be run on Earth Engine servers, which is what makes Earth Engine such a powerful processing software. Instead you should use the .map() method.

1. On a line below the function **Create** a new variable called maskedCollection. Set maskedCollection equal to TOAcollection, but don't add a semicolon to the end of the line yet.

2. After collection, add the .map() method. .map() is intended to take an algorithm, like the function you wrote in section B, and apply it to every image in the collection. Because of this, the only argument you'll need to pass to .map() is the cloudMask function. The full syntax for this line is:
   ```
   var maskedCollection = TOAcollection.map(cloudMask);
   ```
3. Now you've created a new variable called maskedCollection, which is the same as the Landsat image collection, but with cloudy pixels masked out.
4. To see the masked layer, add it to the map, using:
   ```
   Map.addLayer(maskedCollection)
   ```
   Your code should look like the image below.



```
Imports (1 entry)
var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)
```

```
//Author: Nolan Cate
//Date: MM/DD.YY
//Description: This is the Javascript exercise for Introduction to Geospatial Programming

var GTACpoint = ee.Geometry.Point(-112,40.7);
var TOAcollection = ee.ImageCollection("LANDSAT/LT5_L1T_TOA").filterBounds(GTACpoint);
print(TOAcollection);
Map.addLayer(TOAcollection);

var medianImg = TOAcollection.median();
Map.addLayer(medianImg, {bands: ['B3','B2','B1'],min:0.05, max:0.55});

var ndvi = (medianImg.select('B4').subtract(medianImg.select('B3')))
    .divide((medianImg.select('B4')).add(medianImg.select('B3')));

ndvi = medianImg.normalizedDifference(['B4','B3']);
Map.addLayer(ndvi);

function cloudMask(image){
  //Specify the cloud likelihood threshold -
  var cloud_thresh = 40;
  //use add the cloud likelihood band to the image
  var CloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);
  //isolate the cloud likelihood band
  var quality = CloudScore.select('cloud');
  //get pixels above the threshold
  var cloud01 = quality.gt(cloud_thresh);
  //create a mask from high likelihood pixels
  var cloudmask = image.mask().and(cloud01.not());
  //mask those pixels from the image
  return image.mask(cloudmask);
}

var maskedImage = cloudMask(ee.Image(TOAcollection.sort('CLOUD_COVER').first()));
Map.addLayer(maskedImage);

var maskedCollection = imageCollection.map(cloudMask);
Map.addLayer(maskedCollection);
```

Figure 12: A screenshot of your script at this stage of the exercise, including mapping a function

5. **Click** Run. You should have 5 layers now in your map.

Part 5: Optional: Visualizing Information

A. Creating a Chart

You are going to create a histogram in the Earth Engine Code Editor. To do this you need to have a vector boundary (a polygon) for which area to chart. This is the area argument that the function needs. In this piece of the exercise, you will create a polygon around the filtered collection layer, then generate a histogram of NDVI values.

1. On a new line, **Create** a new variable called boundary, pull the first image out of your image collection, and take the geometry of that image.

```
var boundary = TOAcollection.first().geometry();
```

2. Now you can use an Earth Engine chart function (ui.Chart). You will need to wrap the chart function in a print statement to display the chart on the console, and you are going to set the ui.Chart to a histogram of an image that we pass to the function. Try using the NDVI image you created. Remember the variable name is ndvi. Add the statement below to your function.

```
print(ui.Chart.image.histogram(ndvi,boundary,500));
```

This creates an Earth Engine chart. Specifically a histogram chart from an image.

This section sends arguments to the ui.Chart function. The first argument is the image used to make the histogram of (ndvi). The second argument is the area over which to calculate the histogram (the boundary that you created in step 1). The last argument is a pixel scale in meters to create the histogram at (here we're using 500 meters).

3. **Click** Run. You should see a chart similar to the one below output in your console pane. It is a histogram of your ndvi image

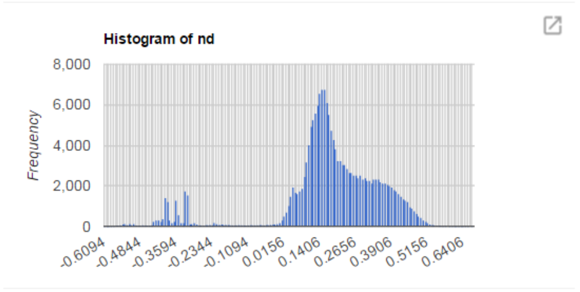


Figure 13: An ee.Chart showing a histogram of NDVI values

Part 6: Notes and Other Resources

Earth Engine has a number of very useful resources that you will find yourself referring to often as you become an avid Earth Engine user.

A. Notes and Resources

1. GTAC offers an Earth Engine Code Editor specific training. The training is similar to the exercise you just completed, but goes into significantly more detail about the software. It is more focused on the contents of the script that you write, rather than the structure of your code. To find the tutorial, visit: http://fsweb.geotraining.fs.fed.us/www/index.php?lessons_ID=3448

2. Google offers a number of tutorials, which are very useful if you need guidance through a particular tool. Browsing the page: https://developers.google.com/earth-engine/ will give you lots of information about the software.
3. In addition to offering the tutorials, there is an online forum specific to Earth Engine. The developers of the software are present on the forum, as well as a very active user community. Sifting through old posts may often yield a solution to your problem. If you're still stumped, this is the perfect place to ask a question anywhere from beginner to advanced. https://groups.google.com/forum/#!forum/google-earth-engine-developers
4. In the code editor page itself there is some documentation explaining all the methods available for working with the Earth Engine objects. In the panel on the left, the docs tab contains a list of all of the methods and their required arguments. If you can't get a tool to work, check what the docs tab has to say about it. You may be missing a key piece of information.

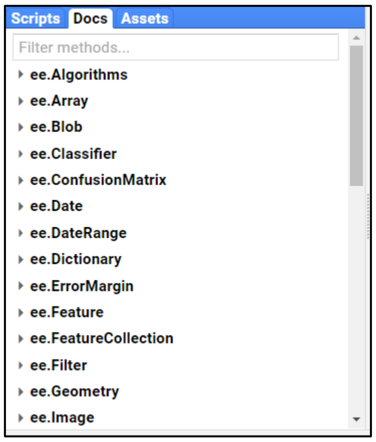


Figure 14: The documents panel on the right pane of a code editor window

Congratulations! You have learned some basic skills about geospatial scripting in Earth Engine. This should have given you the chance to practice some of the skills you learned in earlier exercises and given you some resources to continue learning. Remember that when scripting you are learning a new language, and it will take time. Review this exercise and visit the additional resources provided to keep practicing your new skills.