EXERCISE 4
Google Earth Engine

Introduction

This exercise quickly introduces tools available in Google Earth Engine (GEE). 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 Google Earth Engine (GEE)
- Differentiate an Image and an Image Collection

Prerequisites

- Active Google Earth Engine (GEE) Account
- Completed Exercises 1-3 in Intro to Geospatial Scripting

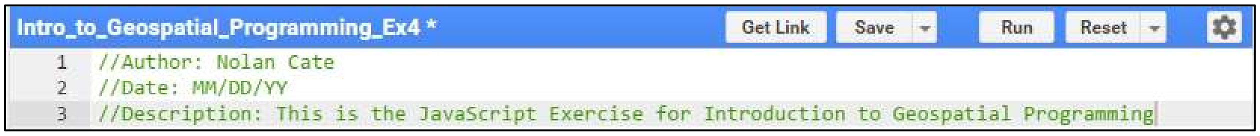
Table of Contents

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 Save the Script

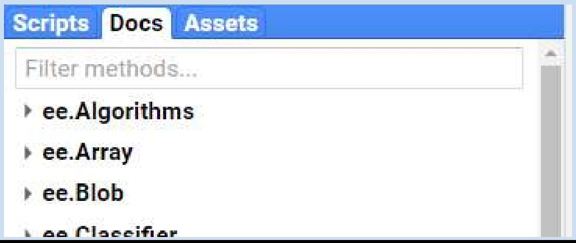
1. Add comments with your name, the date, and a description of the script. Remember typing **//** will comment out a line in GEE. Your script should now look like the screenshot:



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

2. Save the script and name it "Intro_to_Geospatial_Scripting_Ex4". You should now see the script in your private repository on the left side of the Code Editor. You can find an example of this code on the course repository.

Note: All the tools used in this exercise are described in the **Docs** tab in the left pane in the Earth Engine Code Editor. Refer to the **Docs** tab for a description of the tools you're using. Use this tab to make sure you're using Earth Engine tools correctly.

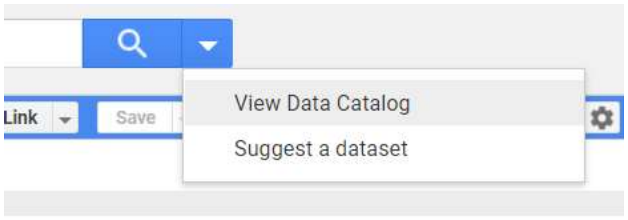


Part 2: Image Collections

A. The Earth Engine Data Catalog

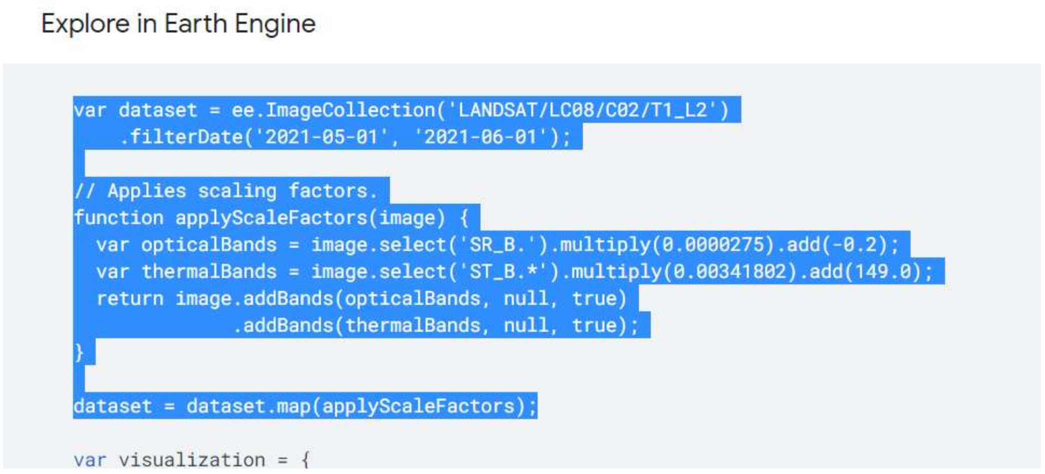
The data catalog is where you can find information about all the imagery in Earth Engine. If you're ever unsure if there is data that you need in Earth Engine, the data catalog is where you will search for it. Today you will be working with imagery from Landsat.

1. To the right of the search bar hover over the down arrow and select **View Data Catalog**



2. Scroll down and select **Explore Landsat Imagery**
3. Read about the different collections, we want to use Collection 2 Surface Reflectance Data. Click on **Landsat 8 Surface Reflectance** scroll down and read the **Description** and **Bands**

4. Scroll down to Explore in Earth Engine and look at the code. Copy from the first line down to the line that starts with dataset. Paste the code into your GEE script



Explore in Earth Engine

```
var dataset = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
    .filterDate('2021-05-01', '2021-06-01');

// Applies scaling factors.
function applyScaleFactors(image) {
  var opticalBands = image.select('SR_B.').multiply(0.0000275).add(-0.2);
  var thermalBands = image.select('ST_B.*').multiply(0.00341802).add(149.0);
  return image.addBands(opticalBands, null, true)
              .addBands(thermalBands, null, true);
}

dataset = dataset.map(applyScaleFactors);

var visualization = {
```

5. The variable name **dataset** is not very descriptive, rename the **dataset** variable to **Landsat8**

B. Filter GEE Image Collections

In GEE raster data is stored as either an **Image** or an **Image Collection**. **Image** objects have one or more spectral bands. An **image collection** is a group of image objects. A common mistake is trying to use **Image** object methods on an **Image Collection**.

1. **Print** the **Landsat8** variable to learn more about it. Click **Run** or **cntl+Enter**. In the **Console** pane on the right, you will see an image collection line appear with a processing symbol next to it. GEE will take a while to run then you will get an error, telling you that it has accumulated over 5000 elements. GEE is trying to give you information about all Landsat 8 images covering the entire world. To fix this error we need to filter the image collection.
2. Filter an image collection by location
   i. Create a variable to define our area of interest (AOI) before the **Landsat8** variable
   ```
   var AOI
   ```
   ii. Set this variable equal to a Geometry point using **ee.Geometry.Point**()
   ```
   var AOI = ee.Geometry.Point()
   ```
   iii. Add in the longitude and latitude coordinates: -112,40.7 which is in Salt Lake City, Utah, where GTAC is located.
   ```
   var AOI = ee.Geometry.Point(-112,40.7)
   ```
3. Filter the Landsat8 variable by location by using **.filterBounds()**
   ```
   var Landsat8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
       .filterDate('2021-05-01', '2021-06-01')
      .filterBounds(AOI);
   ```
   Now your image collection only contains **Landsat8** images between the dates defined in **.filterDates()** and imagery that contains the AOI point by **.filterBounds()**
4. 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, but in the console on the right, you can see information about the image collection.

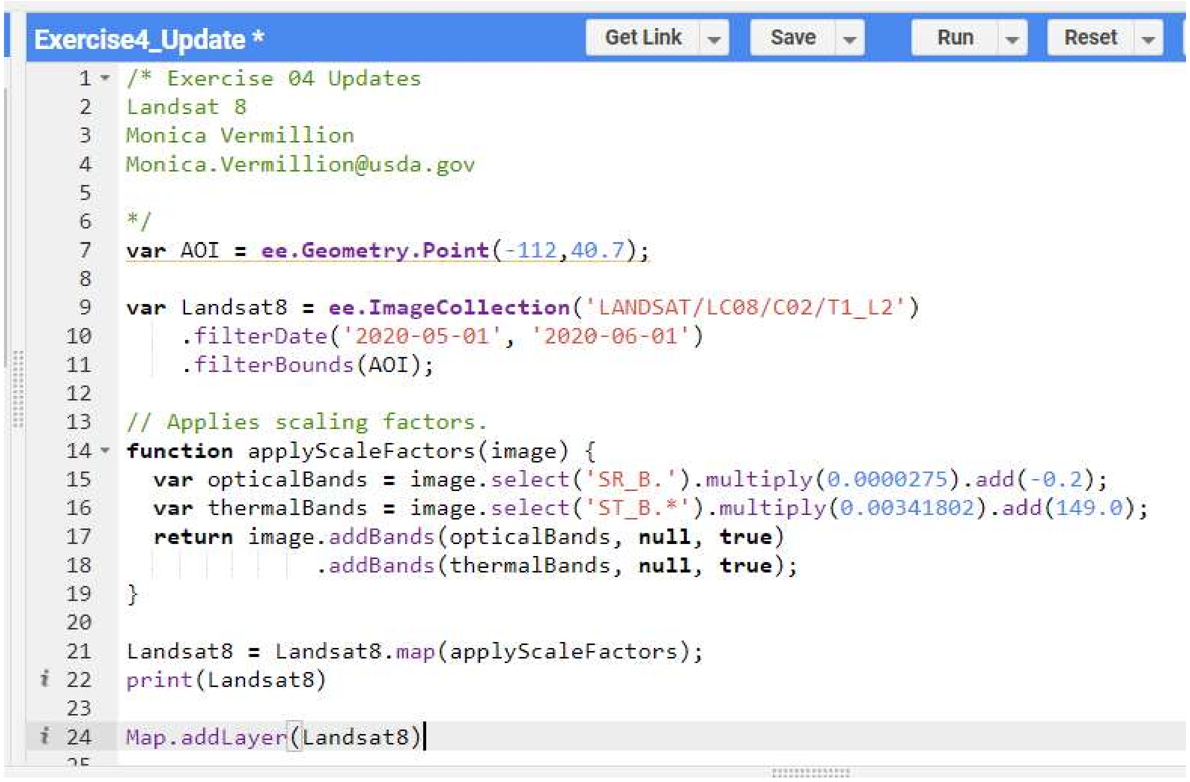
C. Adding a Layer to the Map

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. Use the **Map.addLayer()** command to add the imagery to the map
   `Map.addLayer()`
2. We must tell the computer which variable we want to add to the map - **Landsat8**
   `Map.addLayer(Landsat8);`

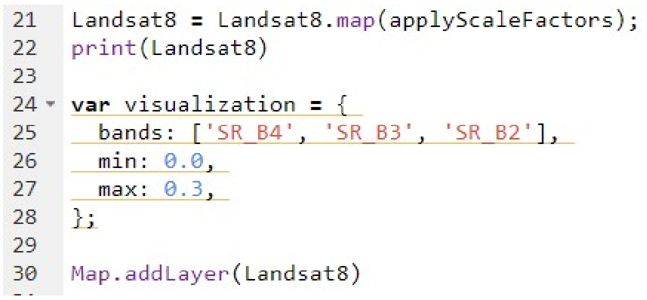
   Drag the map using your mouse and use the scroll wheel to center your map over Northern Utah then Run the script

3. 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. Your code should now look like this:



```
/* Exercise 04 Updates
Landsat 8
Monica Vermillion
Monica.Vermillion@usda.gov

*/
var AOI = ee.Geometry.Point(-112,40.7);

var Landsat8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
    .filterDate('2020-05-01', '2020-06-01')
    .filterBounds(AOI);

// Applies scaling factors.
function applyScaleFactors(image) {
  var opticalBands = image.select('SR_B.').multiply(0.0000275).add(-0.2);
  var thermalBands = image.select('ST_B.*').multiply(0.00341802).add(149.0);
  return image.addBands(opticalBands, null, true)
              .addBands(thermalBands, null, true);
}

Landsat8 = Landsat8.map(applyScaleFactors);
print(Landsat8)

Map.addLayer(Landsat8)
```

4. The image looks really dark so we need to tell GEE how to display the image. Go back to the Landsat 8 page in the GEE Data Catalog. Scroll down to the code section and copy the **visualization** variable and past it into your script above the **Map.addLayer()**



```
Landsat8 = Landsat8.map(applyScaleFactors);
print(Landsat8)

var visualization = {
  bands: ['SR_B4', 'SR_B3', 'SR_B2'],
  min: 0.0,
  max: 0.3,
};

Map.addLayer(Landsat8)
```

The bands display the band in red, green, and blue. The min and max scale the colors

5. Add the visualization to the **Map.addLayer()** so the Landsat image displays in true color

```
Map.addLayer(Landsat8,visualization,”True color”);
```

The last argument in the code “True color” is the name of the layer as it appears on the map

Part 3: Reducers and Operators

A. Creating a Composite Image

In this section, you will use an image collection method to reduce the image collection into a single composite image. 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 the Reducer Overview

1. You are going to use the **.median()** method to reduce the image collection. Create a new variable called **medianImg** and set it equal to median of the **Landsat8** image collection

```
var medianImg = Landsat8.median();
```

2. Print the median image and run the script. **medianImg** is now an Image where **Landsat8** is an Image Collection, you can see this displayed in the right console

```
print(medianImg);
```

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.

3. Now add the median image to the map, the same way you added the image collection to the map, except name the layer “Median Image”

```
Map.addLayer(medianImg,visualization,“Median Image”);
```

B. Operators in Earth Engine

We're going to use operators to do some band math on an image using is the median image you created in the last section. We will use these operators to create a Normalized Difference Vegetation Index (NDVI). Read more about NDVI

1. We will set the the NDVI variable equal to the NDVI equation

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

Landsat 8 images have 7 image bands, each of which represents reflectance in a specific region of the electromagnetic spectrum. In Landsat 8 Band 5 represents reflectance in the near-infrared (NIR), and Band 4 represents red

4. Start with the top of the equation, the numerator. We need to get Band 4 and Band 5 out of the median image. If you look at the printed image bands are named "B1", "B2", "B3", etc. To pull out an individual band use **.select()** method. The code below selects Band 4

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

The numerator of the NDVI equation should be expressed as:

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

Now with all this information, can you script the entire NDVI equation? You can set the numerator as a variable, then the denominator and then create an NDVI equation that divides the numerator by the denominator.

5. The operators will do math on each pixel per band. Below is the calculations for NDVI in one statement:

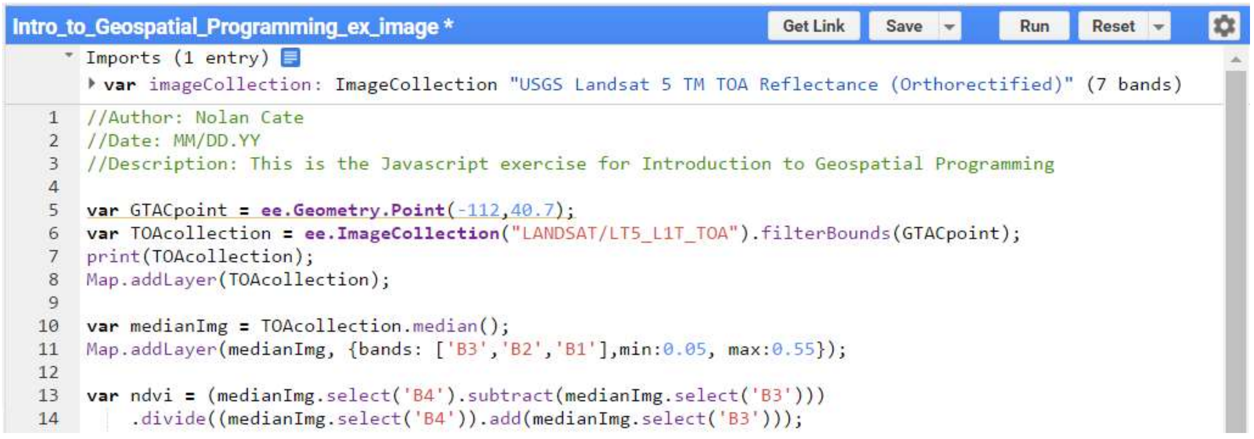
```
var ndvi = (medianImg.select('B4').subtract(medianImg.select('B3')))
     .divide((medianImg.select('B4')).add(medianImg.select('B3')));
```

These operators (.add(), .subtract(), .divide(), and .multiply()) are Earth Engine specific

6. 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.



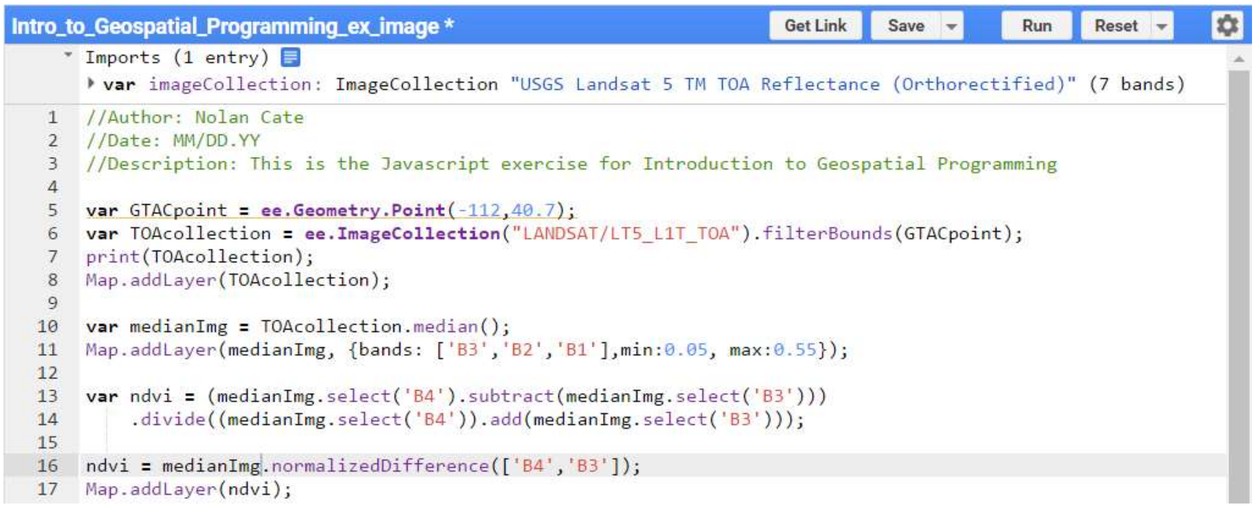
```
Imports (1 entry)
var imageCollection: ImageCollection "USGS Landsat 5 TM TOA Reflectance (Orthorectified)" (7 bands)

//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')));
```

6. 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. Instead of entering a long equation, Google Earth Engine has a .normalizedDifference() method to calculate NDVI:

```
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 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});

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);
```

Part 4: Creating a Function

This section will give you an opportunity to practice writing a function. 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.*

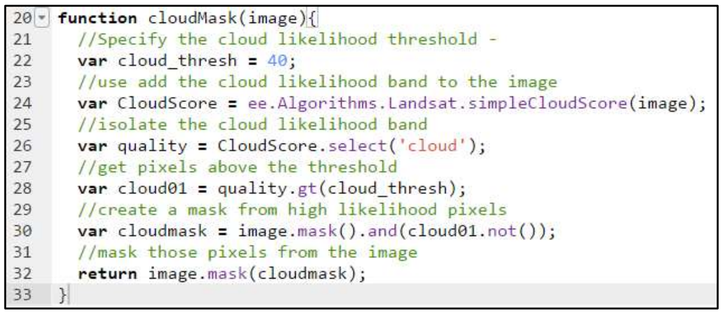
2. What you have now is a complete function. But right now there is no code within the function to run, and remember that the function will not run anyways, unless we explicitly call it. Before we call it, let's fill it in with some code.

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 1: 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')))
```

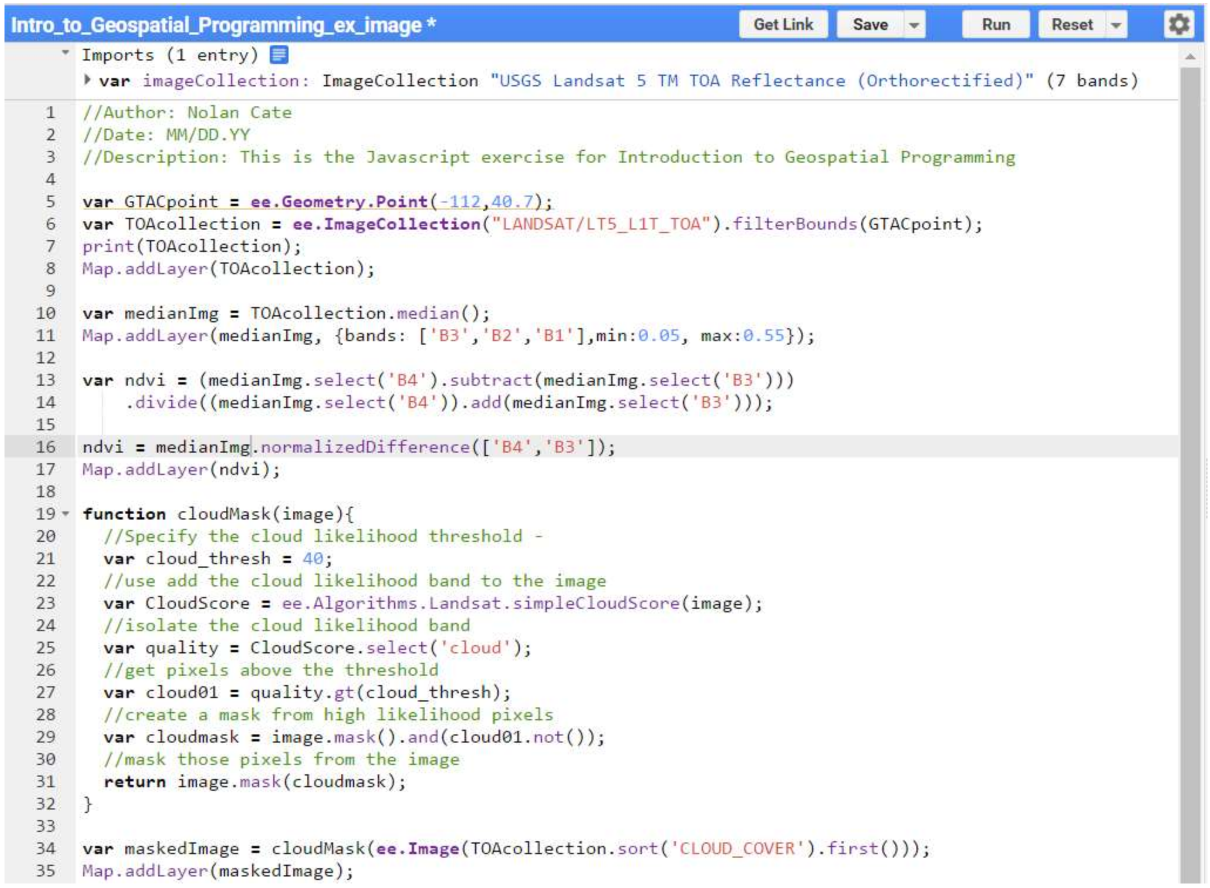
Notice that the property you're sorting by needs to be a string.

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 2: 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,

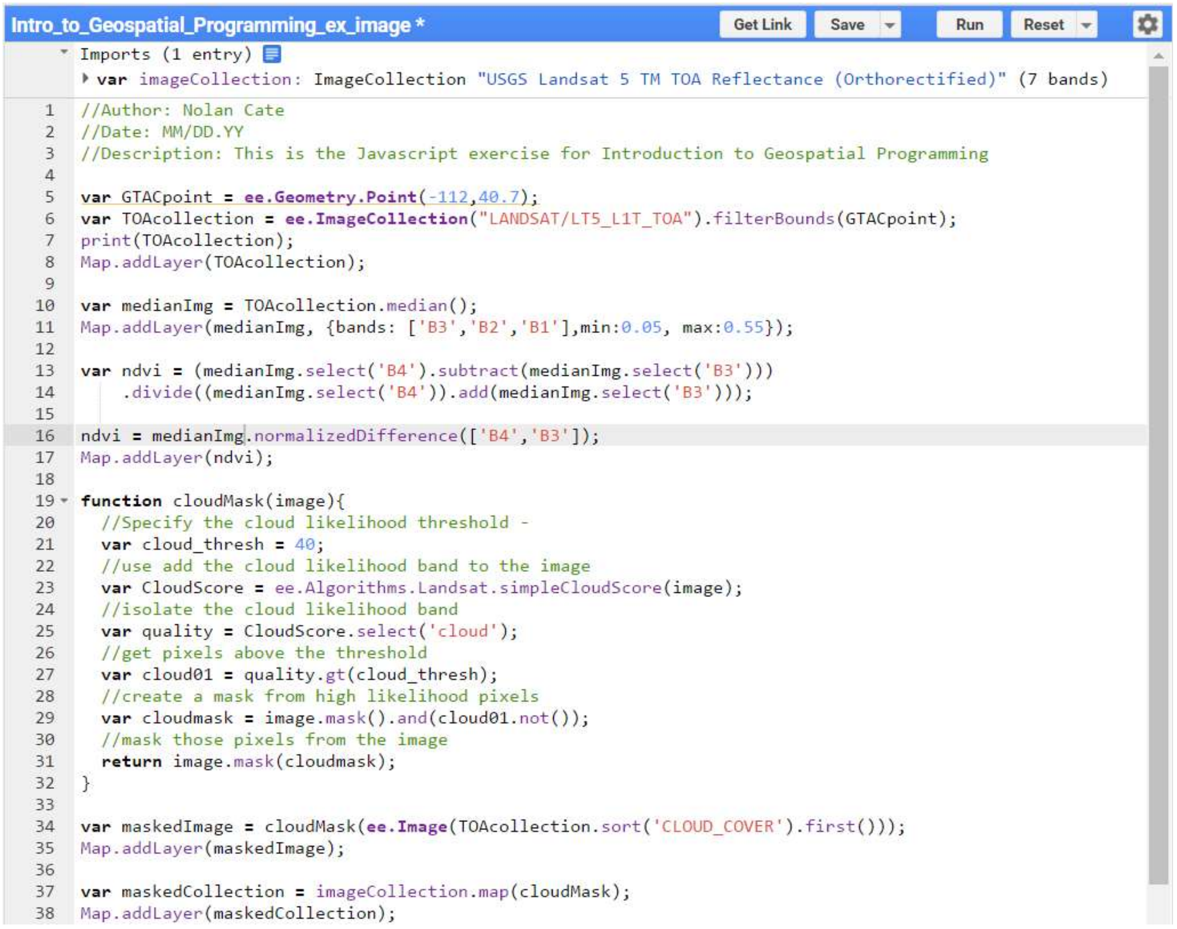
the only argument you'll need to pass to .map() is the cloudMask function. The full syntax for this line is:

```
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);
```

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

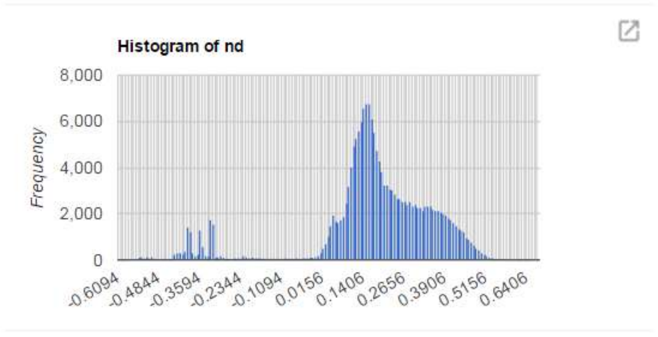
Part 5: Optional: Visualizing Information

A. Creating a Chart

You are going to create a histogram of the pixel values in your NDVI image 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. Create a new variable called **boundary**, pull the first image out of your image collection using the **.first()** method, 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. 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 like the one below output in your console pane. It is a histogram of your NDVI image values.



You can find an example of this completed script on the course repository, GeospatialScripting_ex4.

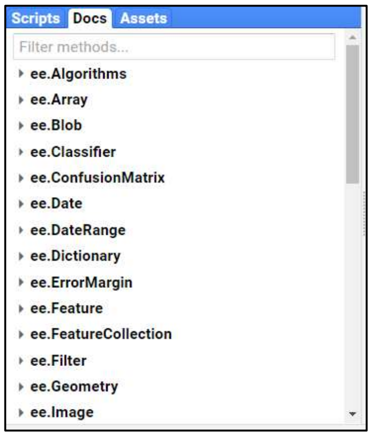
Part 6: Notes and Other Resources

Earth Engine has several 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 several tutorials

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 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.



ee.Algorithms
ee.Array
ee.Blob
ee.Classifier
ee.ConfusionMatrix
ee.Date
ee.DateRange
ee.Dictionary
ee.ErrorMargin
ee.Feature
ee.FeatureCollection
ee.Filter
ee.Geometry
ee.Image

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.