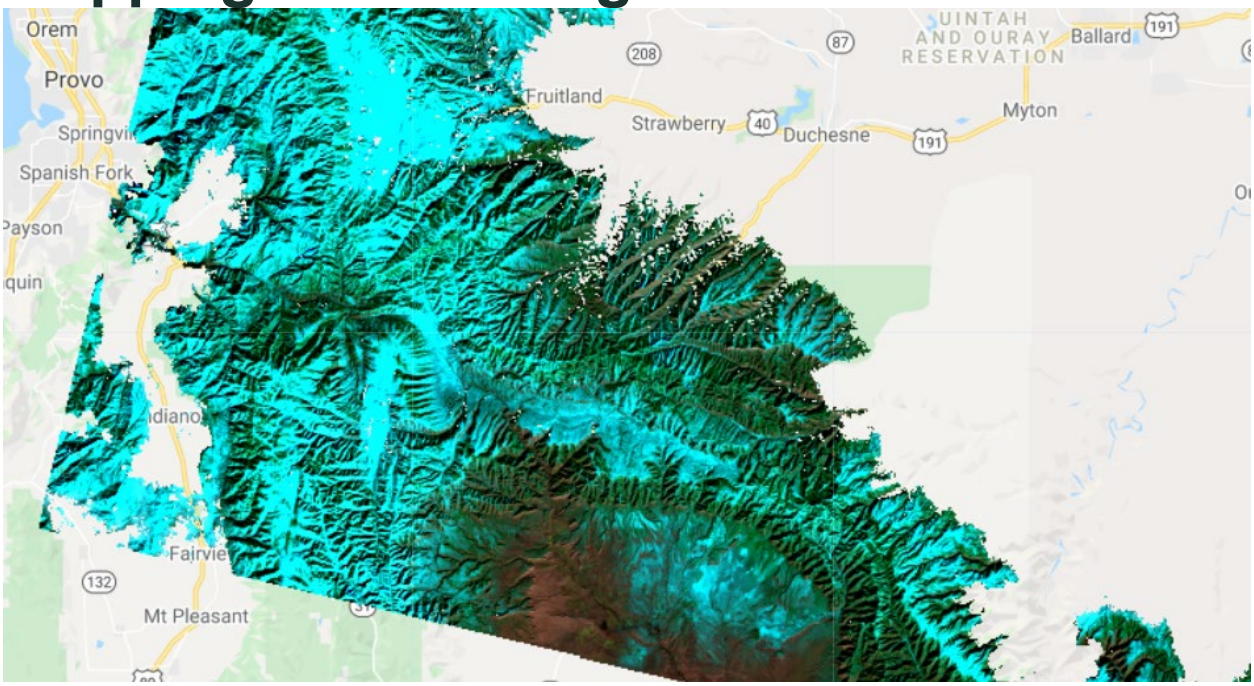
Last Updated: February 2022
Version: Google Earth Engine

Exercise 3: Custom Functions and Mapping across Image Collections



Introduction

In this exercise, you will learn some new Earth Engine spatial data concepts. As you've already seen it is essential to be able to write efficient functions so that you can repeatedly use code you write.

Objectives

- Practice writing functions
- Learn how to mask clouds
- Map a function over an image collection
- Export an image

Prerequisites

- You have a Google Earth Engine account
    - Register for a Google Earth Engine account using your USDA email. Fill out this Microsoft Form, which will tell GTAC staff to create an account for you. This process may take 1-3 business days.
- Completion of Exercise 2

Table of Contents

Part 1: Create a cloud mask function

Here you will write up a function for masking clouds that you can call later in the script, instead of having it as a standalone.

A. Load your script from the previous exercise

1. In the last exercise you loaded an image collection, filtered it spatially and temporally, and reduced it using a median reducer. Load the script from the previous exercise. You can also access a copy of the code in the GTAC Training Repository.
2. Update and simplify your script to remove some of the extraneous operations that we performed in the last exercise. We won't need this information for our next exercise. Delete the comments and code for the following actions: **Add collection to map and center the display on the study region** (Lines 17-21), **Print information about the filtered image collection** and **Count and print the number of images** (Lines 23-28), and **Count the number of images** (Lines 33-35). Your new script should look like the code below.
3. Save your new script as **Ex3_CloudMasking.**

```
// Get an image collection
var landsat8_collection = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2");

// 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_collection = landsat8_collection.map(applyScaleFactors);

// Filter to the 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');

// 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:'SR_B6,SR_B5,SR_B4'}, "Median Composite 2015");
```

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

B. Write a cloud masking function

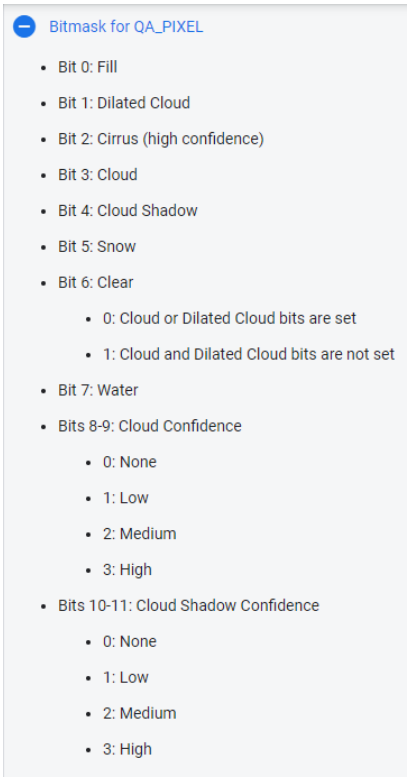
The content below will teach you one method to mask clouds. There are many cloud masking methods available. This one will work well for this task. You're going to wrap the cloud masking routine in a function. Best practice is to include the functions in your script at the *top*.

1. The function that you're going to write will only mask clouds out of a single image, not an image collection. Later you'll learn to map the function over the image collection, so now you'll write the function to accept an image. Write the first line of the function as:

```
// Create the cloud masking function.
var maskClouds = function(image){
```

Next, you will use the **pixel_qa** band in the Landsat 8 image to obtain a cloud mask and cloud shadow mask for your image. Landsat 8 Collection 2 images are pre-processed using the CFMask algorithm, which uses decision trees to label pixels as clouds, water, and snow. It also estimates where cloud shadows appear. The algorithm stores the outputs of its calculations in the **QA_pixel** band. Different bits, or data pieces at different locations in the band, store each of the outputs of CFMask.

QA_pixel is a 16-bit band. You don't need to know everything about how bits work for the purposes of our function today; what's important here is that bits 3 and 4 contain the information that we need to mask out clouds and cloud shadows. You can learn more about bit masks at this resource.



2. Add the lines below to the next line of the function. This will select the QA_pixel band for your image.

```
// Get the pixel QA band.
  var qa = image.select('QA_PIXEL');
```

3. The raster generated from this selection is now a local variable in the function, an image called **qa.** This variable contains the bits that we need to create a cloud mask. Now the variable **qa** will be a new raster image, which contains all the bits that we need to maks clouds and cloud shadows.

4. Next, we need to isolate the bits we need to perform a cloud mask. We tell the script the operation that we need to perform in order to isolate the cloud mask bit from the image: We convert bit 3 into a single digit – 0 or 1. We do the same for bit 4.

```
// Get bit 3: cloud mask
  var cloudsBitMask = (1 << 3);
// Get bit 4: cloud shadow mask
  var cloudShadowBitMask = (1 << 4);
```

5. Next we create a mask by identifying by where both the cloudMask and the cloudShadow mask equal zero.

```
// Identify where both the cloud mask and cloud shadow mask equal 0,
// indicating clear conditions
var mask = qa.bitwiseAnd(cloudsBitMask).eq(0)
                 .and(qa.bitwiseAnd(cloudShadowBitMask).eq(0));
```

The output **mask** will be a binary raster, an image where all pixels with a value of 1 are "not cloud" and all pixels with a value of 0 are "cloud". A binary raster like this is usually referred to as a mask. Add the lines below to the function.

6. Now you can use the **updateMask()** method to remove the values that have clouds or cloud shadows from the Landsat image. The updateMask() method removes pixels from the Landsat image where the input image, *mask*, has a value of zero. Add the lines below to the function. Run the code and inspect the output.

```
  // Apply mask to remove clouds from input image
  var image_noclouds = image.updateMask(mask);
```

7. The last thing that you need to do is specify the variable that the function is going to return. The variable you want is the one you made in the last step. To return it, simply add a return call to the last line of the function.

```
  return image_noclouds;
}
```

8. Make sure that your function ends with a closed curly bracket. The function should now look like the code in the image below:

```
Imports (2 entries)
  var imageCollection: ImageCollection "USGS Landsat 8 Level 2, Collection 2…
  var studyArea: Polygon, 4 vertices
// Create the cloud masking function
function maskClouds(image) {
  // Get the pixel QA band.
  var qa = image.select('QA_PIXEL');
  // Get bit 3: cloud mask
  var cloudsBitMask = (1 << 3);
  // Get bit 4: cloud shadow mask
  var cloudShadowBitMask = (1 << 4);
  // Identify where both the cloud mask and cloud shadow mask equal 0,
  // indicating clear conditions
  var mask = qa.bitwiseAnd(cloudsBitMask).eq(0)
                 .and(qa.bitwiseAnd(cloudShadowBitMask).eq(0));
  // Apply mask to remove clouds from input image
  image_noclouds = image.updateMask(mask);
  return image_noclouds;
}
```

9. In this section you didn't get to see any masked images added to the map. This is because everything you wrote is wrapped in the function, and all the variables are local to the function. Your function is written to take in a single image to mask clouds. But the image collection is not a single image, so how are you going to apply the function to the entire collection? The answer is you map the function over the collection. You'll learn to do this in the next part, and you can also visualize a masked image.

Part 2: Mapping Functions Across Image Collections

Frequently when scripting this would be when you would use a loop to cycle through the image collection. However, because of the architecture behind Earth Engine loops are frowned upon. The ideal way to use this function over an image collection is to *map* it across the collection.

Note: *Mapping a function is when a function is applied to every element of an iterable object individually. In the context of an Earth Engine ImageCollection() object this means that a function will be applied to every image that makes up the collection. It will return the image collection, but with each image processed by the function. The Earth Engine FeatureCollection() class also has a map() method where each feature of the collection can have a function applied.*

A. The .map() Method

1. Now at the bottom of the script add a statement that maps the function across the image collection.

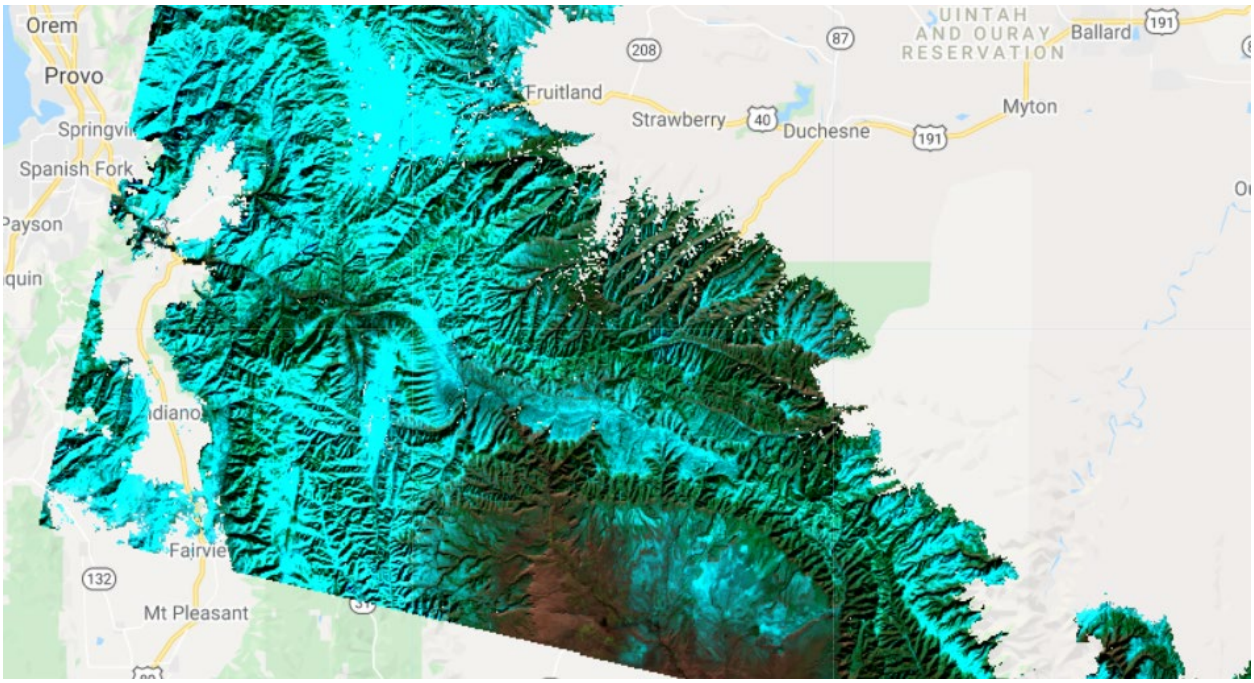
```
// Mask clouds for all images in the image collection
var landsat8_SA_2015_cloudMasked = landsat8_SA_2015.map(maskClouds);
```

   The .**map()** method applies the function to every element (image) of the image collection you apply it to.

2. Now you can add the image collection that you've removed clouds from to the map the same way you've added image collections in the past.

```
// Add the first masked image in the collection to the map window.
Map.addLayer(landsat8_SA_2015_cloudMasked.first(),{min:0.05, max:0.8,
bands:'SR_B6,SR_B5,SR_B4'}, 'first image with clouds masked');
```

3. You should zoom and pan your map to the first image that is displayed and experiment with the transparency slider on the layers to see where clouds have been removed from the image.



Note: *You've applied the maskClouds function but depending on the date range, threshold, and other layers you have turned on it may be hard to see where the clouds have been masked. You are strongly encouraged to experiment with this function by changing the data range in the .filterDate() method and toggling layers on the map on and off to get a better idea of exactly how the maskClouds function performs.*

Also note that there is snow present in this image—showing up in blue. How could we use the information we have above, about the bits in the QA_pixel band and creating a bit mask, to mask snow from our images as well?

Part 3: Exporting Data

A major advantage of cloud computing is the ability to accomplish complex tasks with the computational load and data storage shifted to the cloud. For many applications, however, you will want to export and save your results at some point. In this exercise, you will learn how to use the Code Editor to export results that you can save, share, and use in analyses on your desktop.

A. Prepare imagery you want to export

1. When you export data, it is easiest to have the image object in its final form before you export it. Here you will prepare the cloudmasked median image for export in true color format.
2. Use a **median reducer** on the `landsat8_SA_2015_cloudMasked` image collection that you just created to aggregate the information from all the images in the collection and create one single image that you will export. We'll also clip the image to just our Study Area, rather than all of the images that intersect the study area. See example code below.

```
// Reduce the collection to the median value per pixel
var median_L8_2015_cloudMask = landsat8_SA_2015_cloudMasked.median();

// Clip to study area
median_L8_2015_cloudMask = median_L8_2015_cloudMask.clip(studyArea);

// Inspect information about the median image
print(median_L8_2015_cloudMask, "Landsat 8 2015 Cloud Masked Median
Composite");

// Display median composite in the Map window
Map.addLayer(median_L8_2015_cloudMask,{min:0.05, max:0.8,
bands:'SR_B6,SR_B5,SR_B4'}, "Median Composite 2015");
```

B. Exporting Data

Exporting data from the Code Editor is possible through the export functions, which include export options for images, tables, and videos. You will focus on Export.image.toDrive() to download your imagery data sets. You can also export your images as an asset or to Google Cloud storage, but these won't be covered in this exercise.

Export methods take several optional arguments so that you can control important characteristics of your output data, such as the resolution and projection.

1. Under the Docs tab, navigate to and open the **Export.image.toCloudStorage()** function documentation, housed under the Export group. Review the documentation.
2. **Add the statements below** to the bottom of your script. This will create a task in the task tab that you can use to export your image. Images export into the Cloud Storage Bucket that you created in Exercise 1. Make sure to update the **bucket** parameter to reflect the name of *your* bucket!
3. Refer to the text box below for a discussion of the parameters specified here.

```
// Export image to Cloud Storage
Export.image.toCloudStorage({
  image: median_L8_2015_cloudMask,
  description: 'Landsat8_Median_Composite_2015',
  bucket: 'lleatherman-gtactrainingstudents',
  fileNamePrefix: 'Landsat8_Median_Composite_2015',
  region: studyArea,
  scale: 1000,
  crs: 'EPSG:3857',
  maxPixels: 1e13});
```

Note – *In this example, you have specified a few of the optional arguments recognized by* ***Export.image.toCloudStorage()****. Also notice that all of the parameters have been passed through a dictionary (parameters in the curly brackets). Though this function takes several optional parameters, it is valuable to familiarize yourself with these:*

description *– the name of the export in the Tasks pane*

bucket *- basically, a folder in the Google Cloud Project you have selected*

region – *By default, the viewport of the Code Editor is exported (everything you see on your map) but you can also specify a geometry to change the export extent.*

crs – *The coordinate reference system for the output image. This is specified using the EPSG code. You can look up the* ***EPSG*** *code for your desired spatial projections at http://spatialreference.org.*

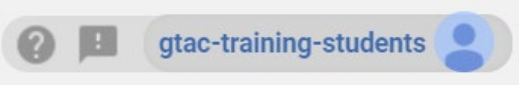
scale – *The resolution in meters per pixel. The native resolution for these data set is 30 meters. In this exercise it is set to 1000 just to speed things up.*

maxPixels *– This restricts the numbers of pixels in the exported image. By default, this value is set to 10,000,000 pixels. You can set this argument to raise or lower the limit. "1e13" is 10 to the 13th power (10^{13}).*

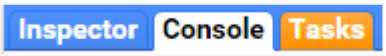
C. Run the Task to export an image to Cloud Storage

Cloud Storage note: *The GTAC-Training-Students Cloud Project is wiped monthly and should be treated as a temporary holding spot for any data you may download. For other projects outside this course, you should make a bucket in your regional projects folder, following the instructions in Exercise 1.*

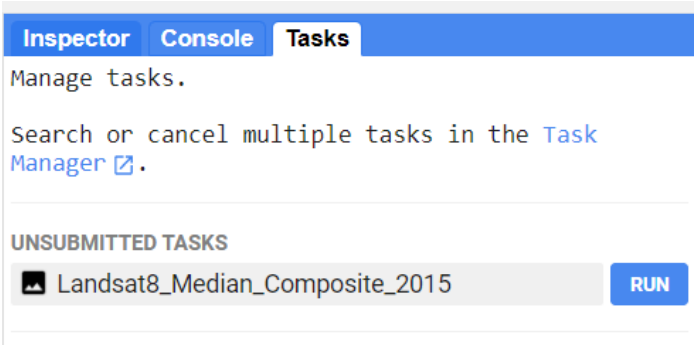
1. Make sure that the gtac-training-students Cloud Project is selected in the upper right corner of the Code Editor window.



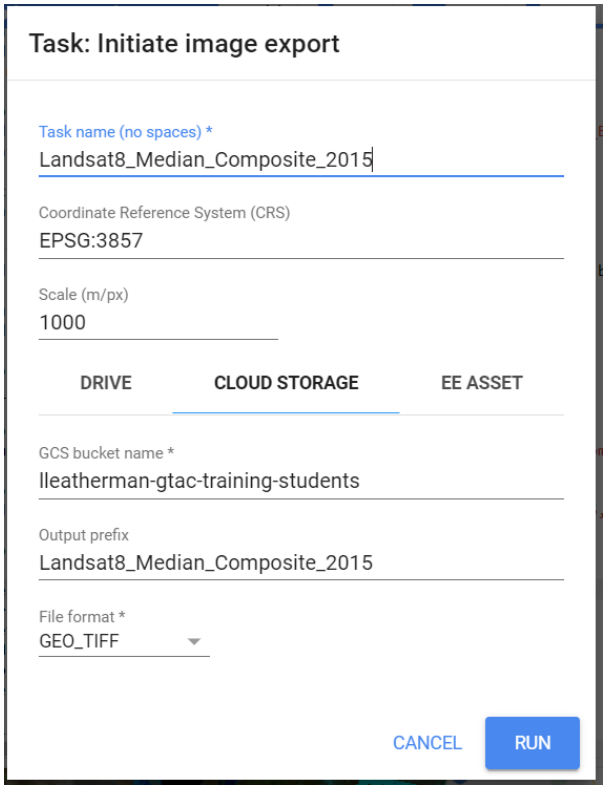
2. **Run** the script. After a moment, the Tasks tab in the upper right of the Code Editor should be highlighted.



3. Click on the **Tasks** tab. Then click the blue Run button (shown below) to export the data to Cloud Storage.



4. Review the information in the Initiate export window that appears (below).



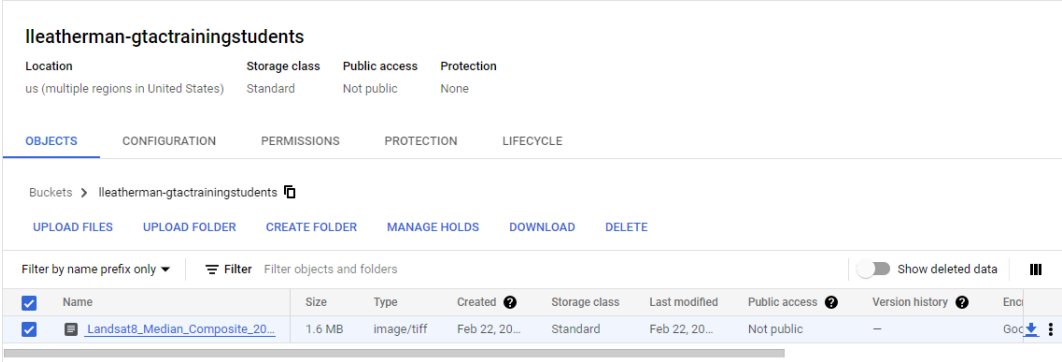
5. Then click **Run** to begin.

Note: *This task will be exported to the Cloud Storage Bucket that you created. This will be a 1000-meter resolution GeoTiff image.*

6. After you click Run at the bottom of the Initiate export window to start the export, your screen will show the export is processing. The task under the Code Editor Tasks tab should now have a spinning GEE icon next to it. It can take some time to export the task. When it is completed, this icon will disappear, and the task name will turn blue. Look to see if yours turned blue.

D. Download composites from Google Cloud Project

1. Go to the Google Cloud Project Browser for your Google Cloud Project. Here, we'll be working in the gtac-training-students cloud project.
2. You should see a list of all the buckets in the project. Navigate to the bucket you created earlier and click on it to open it.
3. In your bucket, you should see the composites you just exported.
4. Click the check box next to the first composite to select it, and then click the blue download option. View it in your preferred GIS software (e.g., ArcMap or QGIS).



Note - Once your data have been successfully exported to Google Cloud Storage, you can download them and review them in your preferred GIS software. Export times can vary depending on the size of your data as well as the time of day and what other users are requesting from Google Earth Engine.

Congratulations! You've completed this exercise and learned how to map a cloud masking function over an image collection. All of the skills you've now learned will be applied in the next exercise where you'll write a RAVG script.