

Exercise 2: Earth Engine Image Objects and Methods

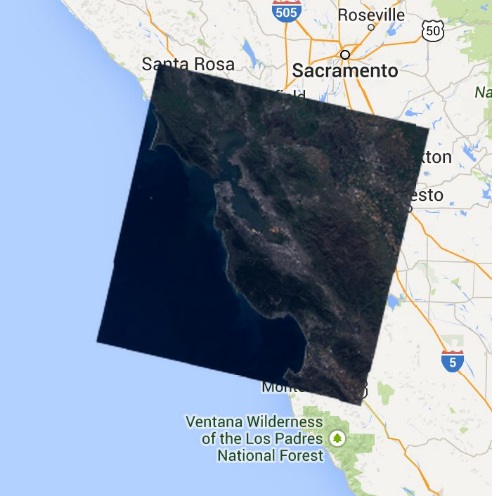
*Concept Note*

**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

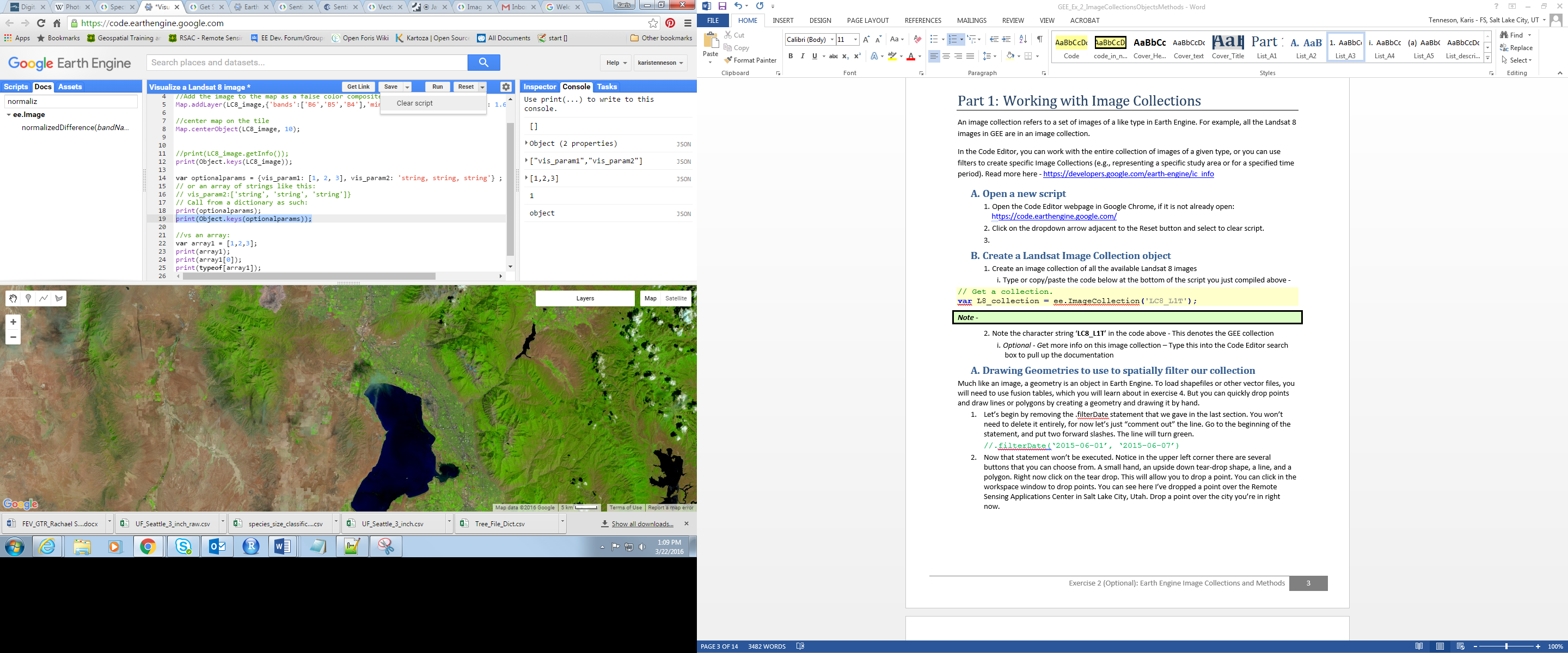
Exercise 2: Earth Engine Image Objects and Methods



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 learn about JavaScript syntax and several key Earth Engine spatial data concepts. The focus of this exercise is on properties and methods, or functions, associated with single raster images in Earth Engine. However, you will also get a brief introduction to other types of Earth Engine spatial objects. This exercise will get you writing a simple JavaScript script. You will also learn about Fusion Tables.

1. Set up your workspace
   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 variable representing a single Landsat 8 image
     1. Use the code in the box below to create a variable representing an ee.Image object for a Landsat 8 image.
        1. Copy and paste the code below into the Code Editor Code Editor.

// Store an image in a variable, lc8\_image.

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

// Display the image in the map window.

Map**.**addLayer**(**lc8\_image**,**   
 **{**min**:**0.05**,** max**:** 0.8**,** bands**:** 'B6, B5, B4'**},**   
 "Landsat 8 Scene"**);**

// Center the map window.

Map**.**centerObject**(**lc8\_image**,** 8**);**

1. Explore Existing Image Processing Functions

A comprehensive collection of tools are readily available in the Code Editor for analyzing and processing the image objects that you have been learning about. These are available as Earth Engine methods and functions.

* 1. Calculate NDVI on your Landsat image
     1. You can calculate the Normalized Difference Vegetation Index (NDVI) on your image using the normalizedDifference() method. Copy the lines below and paste them into the bottom of your script. Click Run. This will calculate the NDVI value in every pixel of your image.

// Create an NDVI image using bands the NIR and red bands (5 and 4).

**var** NDVI **=** lc8\_image**.**normalizedDifference**([**'B5'**,** 'B4'**]);**

// Display the NDVI image with a grayscale stretch.

Map**.**addLayer**(**NDVI**,   
 {**min**:** **-**0.2**,** max**:** 0.5, palette**:** **[**'FFFFFF'**,** '339900'**]},** "NDVI"**);**

* 1. Mask clouds
     1. Clear your script from Part 2 A, except for the lines below.

// Get a Landsat image.

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

// Add the image to the display.

Map**.**addLayer**(**lc8\_image**,**   
 **{**min**:** 0.05**,** max**:** 0.8**,** bands**:** 'B6, B5, B4'**},**   
 "Landsat 8 Scene"**);**

// Center the map on the image.

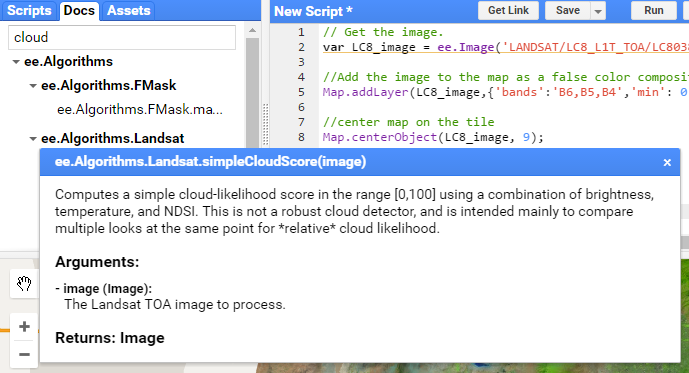
Map**.**centerObject**(**lc8\_image**,** 8**);**

* + 1. Run the script. Notice the clouds that are present on the eastern half of the image? Next you will build a process to remove these from the imagery.
    2. First, you will create a variable, cloud\_thresh. This will store the cloud likelihood threshold value. After you have drafted the script, you can easily change the value of this variable. Change the value and re-run the script to investigate what an appropriate cloud thresholding value is for the study region.

// Specify the cloud likelihood threshold.

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

Next you will use an Earth Engine algorithm that will calculate a simple cloud-likelihood score using a combination of brightness, temperature, and the Normalized Snow Index (NDSI). This likelihood is represented on a scale of 0 to 100, where larger values indicate a greater likelihood of a pixel being clouded. You can read more about it in the Docs tab (or refer to image below).



* + 1. Copy the lines below and paste them at the bottom of your script. This will generate a raster layer that ranges from 0 to 100, the pixels with a higher value are more likely to be clouds.

// Add the cloud likelihood band to the image.

**var** cloudScore **=** ee**.**Algorithms**.**Landsat**.**simpleCloudScore**(**lc8\_image**);**

* + 1. *(Optional)* Add the cloud layer to the map by copying the lines below and adding them to the bottom of your script. Click Run.
       1. What values do the cloudy areas get assigned? (hint: use the inspector to look up values at different locations in the map)
       2. Do you notice the difference between the results of the two Map.addLayer() statements? (hint: use the inspector tab and turn the layers on and off to compare)
       3. After you are done inspecting the cloudScore raster, remove (or comment) these lines from your script.

// Add the cloud image to the map.

// This will display the first three bands as R, G, B by default.

Map**.**addLayer**(**cloudScore**, {},** 'Cloud Likelihood, all bands'**);**

// Since you are interested in only the cloud layer,

// specify just this band to be displayed in the

// parameters of the Map.addLayer statement.

Map**.**addLayer**(**cloudScore**, {**bands**:** 'cloud'**},** 'Cloud Likelihood'**);**

* + 1. The raster you generated from the ee.Algorithms.Landsat.simpleCloudScore() method returns an image with 13 bands: the 12 from the Landsat image, and the 13th band is the new one – the cloud score/likelihood value. The cloud score band is useful to mask clouds in the Landsat image. Copy the lines below and paste them at the bottom of your script. This will set the ‘cloud’ band into a variable called *cloudLikelihood*.

// Isolate the cloud likelihood band.

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

* + 1. Copy the lines below and paste them at the bottom of your script. This code uses the lt() method to create a binary raster that assigns each pixel a value:
       1. One (1) if the cloud likelihood, quality variable, is less than the cloud threshold value;
       2. Zero (0) if the cloud likelihood, quality variable, is greater than the cloud threshold value.

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

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

// Add the image to the map.

Map**.**addLayer**(**cloudPixels**, {},** 'Cloud Mask'**);**

* + 1. Run the code and inspect the values at different locations (hint: use the Inspector tab).
    2. Copy the lines below and paste them at the bottom of your script.
       1. You use the updateMask() method to remove the values that have a high cloud likelihood value from the Landsat image. The updateMask() method removes pixels from the Landsat image where the input image, *cloudPixels*, has a value of zero.
       2. Run the code and inspect the output.

// Create a mask indicating which pixels are likely to be clouds.

**var** lc8\_imagenoclouds **=** lc8\_image**.**updateMask **(**cloudPixels**);**

// Review the result.

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

* 1. Edit the script to mask out haze in addition to clouds
     1. With the original Landsat 8 image toggled off, zoom in to the edge of a cloud in the image. Do you observe the presence of haze in any areas of the masked image?
     2. You can try to reduce the haze by decreasing the cloud likelihood threshold. Locate the variable cloud\_thresh and change this value from 40 to 20. Click Run and review the result.
        1. Lowering the cloud likelihood threshold from 40 to 20 dramatically reduces the presence of cloudy and hazy pixels in the image - the remaining pixels appear clear and bright
        2. Do you think this is an appropriate cloud likelihood threshold?
     3. Use viewer tools to explore the images and try other thresholds if you would like to.

**Note:** Shadows are not eliminated from the image.

1. 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; however, for many applications 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 your GIS analyses on your desktop.

* 1. Load some imagery you want to export

For this export, you will export a subset of the 2005 Canopy Height data derived from spaceborne lidar data from the Geoscience Laser Altimeter System (GLAS) and ancillary geospatial data.

* + 1. Copy and paste the following statements into an empty code editor panel. Click Run to explore the data set.

// Store the canopy height image as a variable, canopyHeight.

**var** canopyHeight **=** ee**.**Image**(**"NASA/JPL/global\_forest\_canopy\_height\_2005"**);**

// Add the data to the map window.

Map**.**addLayer**(**canopyHeight**,** **{**min**:** 0**,** max**:** 36**,** palette**:** **[**'FFFFFF'**,** '00FF00'**]},**

'canopy height'**);**

Map**.**setCenter**(**100.096435546875**,** 13.966054081318301**,** 8**);**

* + 1. Next use the drawing tools to draw a small polygon that represents the region you would like to extract the canopy height data for. Refer to Exercise 3, Part 2 if you need a refresher on how to digitize a geometry. Keep in mind: the smaller the polygon, the faster the download time.
  1. (optional) Generate a histogram
     1. Below is the script to generate a histogram of canopy height over the study region you just digitized. Copy and paste it into your code editor panel to investigate the range of canopy heights in your study region.

// Generate the histogram data.   
var canopyHeightHistogram = Chart.image.histogram(canopyHeight, geometry)  
 .setOptions({title: 'Histogram of Canopy Height'});

// Display the histogram.   
print(canopyHeightHistogram);

* 1. Exporting Data

Exporting data from the Code Editor is possible through the exportfunctions, 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.

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

* 1. Review the documentation for Export.image.toDrive()
     1. Under the Docs tab, navigate to and open the Export.image.toDrive() function documentation, housed under the Export group. Review the documentation.
  2. Create an export task
     1. 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 your Google Drive. Refer to the text box below for a discussion of the parameters specified here.

// Export the image to your Google Drive.

Export**.image.**toDrive**({**

**image:** canopyHeight**,**

description**:** "MyFirstExport"**,**

maxPixels**:** 1e8**,**

region**:** geometry**,**

crs**:** 'EPSG:32647'**,**

scale**:** 1000

**});**

**Note –** In this example, you have specified a few of the optional arguments recognized by **Export.image()**. Though this function takes several optional parameters, it is valuable to familiarize yourself with these:

**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. “1e8” is 10 to the 8th power (108).

**region –** By default, the viewport of the Code Editor is exported 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 the canopy height data set is 30 arc-seconds or approximately one kilometer.

* 1. Run the Task to export an image to Google Drive

**Google Drive note:** Google Drive is a temporary holding spot for any data you may download.

* + 1. Run the script. After a moment, the Tasks tab in the upper right of the Code Editor should be highlighted.

Highlighted tasks tab

* + 1. Click on the Tasks tab. Then click the blue Run button (shown below) to export the data to your Google Drive.

Run your first export.

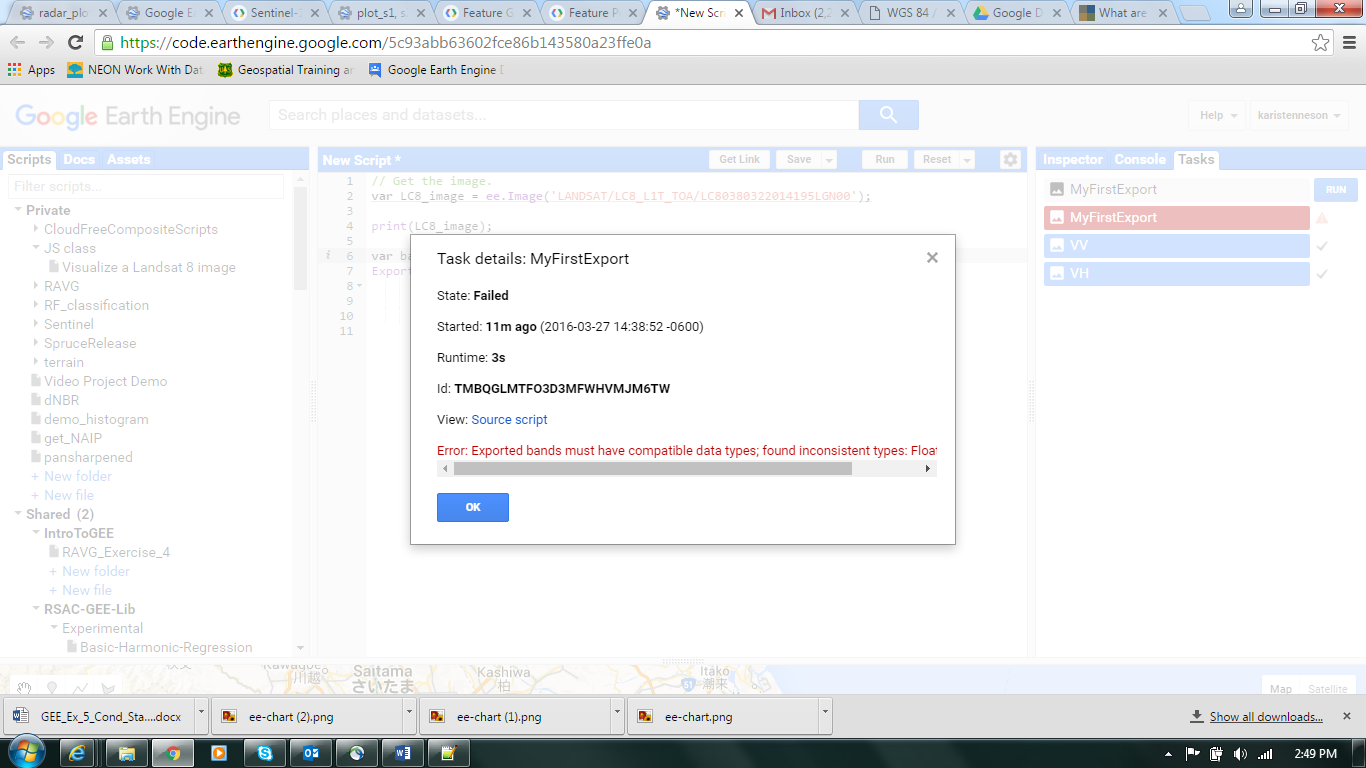
* + 1. Review the information in the Initiate export window that appears (below). Then click Run to begin.

**Note:** This task will be exported to your Google Drive under your Google Account. This will be a 1000-meter resolution GeoTiff image with the root name **MyFirstExport**.

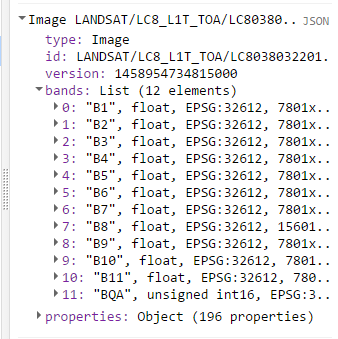
* + 1. After you click Run at the bottom of the Initiate export window to start the export, your screen will show the export is processing. The task under the Code Editor Tasks tab should now have a spinning GEE icon next to it. It can take some time to export the task. When it is completed, this icon will disappear and the task name will turn blue. Look to see if yours turned blue.



**Note:** If you try exporting Landsat scene without first sub-setting the image to get just the bands of interest, you will get an error message.



This is because the bands in the Lansdat image are not all saved as the same data type. Most of the bands are 32 bit float (float 32), but the BQA is an unsigned 16 bit integer (UInt16). You can look up the data type of each band using the print function (see image below). You can either export the first 10 bands separately or convert the mismatched band, BQA, to the data type to match the other bands (e.g., UInt16 to float32).



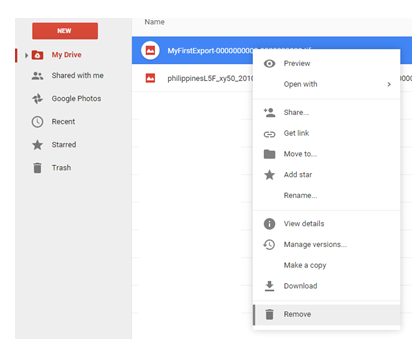
* 1. Review the result
     1. When the export has completed, navigate to your Google Drive using the link below:

<https://drive.google.com/drive/my-drive>

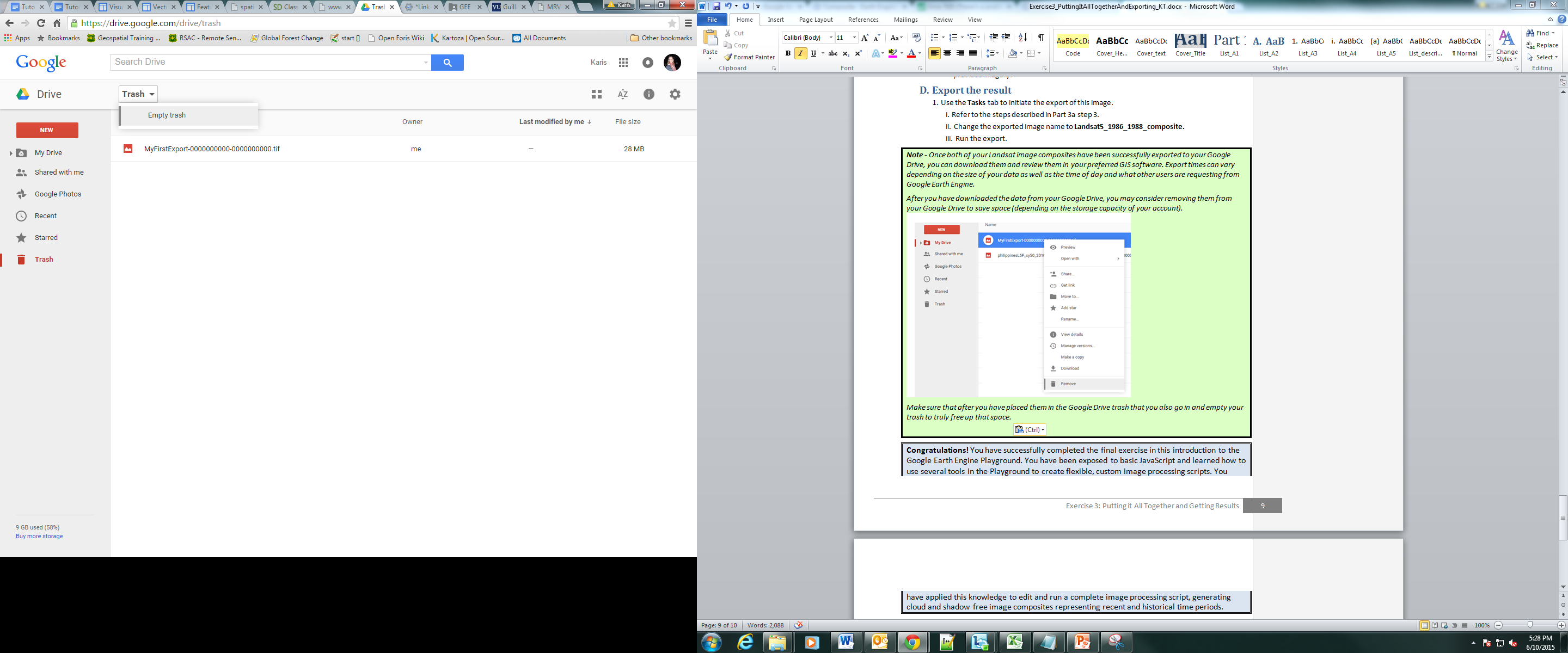
* + 1. Locate the new images, called something similar to MyFirstExport-0000000000-0000000000.tif, in your Google Drive.
    2. *Optional –* Download one of the images and view it in your preferred GIS software (e.g., ArcMap or QGIS).

***Note -*** *Once both of your Landsat image composites have been successfully exported to your Google Drive, 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.*

***Google Drive Data Management Note -*** *After you have downloaded the data from your Google Drive, you may consider removing them from your Google Drive to save space (depending on the storage capacity of your account).*



*Make sure that after you have placed them in the Google Drive trash that you also go in and empty your trash to truly free up that space.*



* 1. Save the complete script to your GEE account
     1. Click the Save button in the Code editor window.
     2. Enter a new name or accept the default name and click the okay button.

**Note –** The above example highlights a task that will be exported to your Google Drive under your Google Account. Each download will be a 30-meter resolution GeoTiff image. For the Palawan composite area each exported composite will be approximately 370 MB. Be sure you have room on your Google Drive! If necessary delete and empty trash to make more room.

1. *(optional)* Accessing Metadata

Understanding how to access metadata of your imagery is important when you are creating your scripts.

**Note:** these examples are from the Earth Engine Documentation, available here: <https://developers.google.com/earth-engine/image_info>

* + 1. Clear the previous script from your code editor panel.
    2. Examine the statements below. Then copy and paste the following into your code editor. Run the script and investigate what is returned in each print statement.

// Get an image.

**var** lc8\_image **=** ee**.**Image**(**'LANDSAT/LC8\_L1T\_TOA/LC81290502015036LGN00'**);**

//Add the image to the map as a false color composite.

Map**.**addLayer**(**lc8\_image**,  
 {**bands**:** 'B6, B5, B4'**,** min**:** 0.05**,** max**:** 0.8**,** gamma: 1.6**},**   
 'Landsat8scene'**);**

// Center the map on the image.

Map**.**centerObject**(**lc8\_image**,** 9**);**

// Retrieve information about the bands as an Earth Engine list.

**var** bandNames **=** lc8\_image**.**bandNames**();**

print**(**'Band names: '**,** bandNames**);**

// Get projection information from band 1.

**var** b1proj **=** lc8\_image**.select(**'B1'**).**projection**();**

print**(**'Band 1 projection: '**,** b1proj**);**

// Get scale (in meters) information from band 1.

**var** b1scale **=** lc8\_image**.select(**'B1'**).**projection**().**nominalScale**();**

print**(**'Band 1 scale: '**,** b1scale**);**

// Note that different bands can have different projections and scale.

**var** b8scale **=** lc8\_image**.select(**'B8'**).**projection**().**nominalScale**();**

print**(**'Band 8 scale: '**,** b8scale**);**

// Get a list of all metadata properties.

**var** properties **=** lc8\_image**.**propertyNames**();**

print**(**'Metadata properties: '**,** properties**);**

// Get a specific metadata property.

**var** cloudiness **=** lc8\_image**.**get**(**'CLOUD\_COVER'**);**

print**(**'CLOUD\_COVER: '**,** cloudiness**);**

// Get the timestamp and convert it to a date.

**var** date **=** ee**.**Date**(**lc8\_image**.**get**(**'system:time\_start'**));**

print**(**'Timestamp: '**,** date**);**

1. *(Optional Read)* Some Background on Objects
   1. Objects

What is a JavaScript Object?

“JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method. In addition to objects that are predefined in the browser, you can define your own objects.” (From the Mozilla Developer Network JavaScript Guide, [link](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects))

Read more about JavaScript objects here: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects> or here: <http://eloquentjavascript.net/06_object.html>.

There are also Earth Engine Objects. These are objects that have meaning in Earth Engine, but not general JavaScript applications. Examples of Earth Engine objects include images (e.g., a Landsat scene) and image collections (a collection of Landsat scenes). Here, the focus is on Earth Engine objects and their associated geoprocessing methods. However, it is important to distinguish between Earth Engine objects and JavaScript objects so that you don’t mistakenly apply methods for one type of object to the other.

* 1. Object properties and their associated methods

**Properties**: objects have properties. Each property has a name and a value (which might be null). The name and value pairs tell you something about the individual instance of the object. For example an image object has properties specific to that object. A Landsat scene is an image object, with 12 bands, and multiple properties that represent things like the time at which the scene was acquired (system:time\_start) and scene metadata set by the provider (USGS).

**Methods:** functions that are specific to object types are called methods. They can retrieve data, sort data, update values of an object’s properties, etc. For example, earlier you used the NormalizedDifference which is an image-object method.

* 1. Earth Engine Image Objects

In the GEE Code Editor, raster data can be represented by two types of objects: an Image object or an ImageCollection.

**Image:** raster data are represented as images objects in Earth Engine. An image object represents a single raster image, such as a single Landsat scene collected on a given day, a Landsat median composite, or a topographic dataset (DEM). Images are composed of zero or more bands, where each band has a name, data type, pixel resolution and projection. Each image also has metadata stored as a dictionary of properties. Read more here - <https://developers.google.com/earth-engine/image_info>

**Image collection:** a set of images. For example the Landsat 8 TOA Reflectance collection (LANDSAT/LC8\_L1T\_TOA) includes all of the imagery Landsat 8 has collected since April 11, 2013, orthorectified and corrected to Top of Atmosphere reflectance. Collections are useful for temporal analysis, or creating cloud free composites that include imagery from several acquisitions.

* 1. Earth Engine Vector Objects

Earth Engine uses the Geometry data type (as a GeoJSON or GeoJSON GeometryCollection) to store vector data; these include points, line strings, linear rings, and polygons. You can interactively create geometries using the draw tools or with a list of coordinates. Features are composed of a geometry and, like images, a dictionary of properties.

You can create an object with a geometry in Earth Engine, a GeoJSON Feature, or a collection of features. Shapefiles can be converted to [Fusion Tables](https://support.google.com/fusiontables/answer/2571232), then accessed in Earth Engine as a FeatureCollection.

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.