Module 2: Introduction to Google Earth Engine and Cloud Free Composites

Google Earth Engine (GEE) is a cloud-based platform for massive-scale analysis of geospatial data. GEE allows users access to a petabyte-scale archive of publicly available remotely sensed imagery, ancillary data and computational tools to accomplish a myriad of remote sensing and geospatial tasks at unprecedented speeds and scales. GEE is free for non-commercial use provided users sign up for a GEE account. Since its release in 2010, it has steadily developed toward becoming an established tool for large-scale geospatial analysis. In this module, you will focus on acquiring and preparing data for the analysis. You will learn how to use GEE for acquiring and preparing Landsat image data to be used in the land cover and change mapping workflow.

Exercise 1: print 'Hello World' and Exploring Data Archive

Introduction

Google Earth Engine is a cloud-based geospatial processing platform. Earth Engine is available through Python and JavaScript Application Program Interfaces (APIs). The JavaScript API is accessible via a web-based Integrated Development Environment (IDE) called the Code Editor. This platform is where users can write and execute scripts to share and repeat geospatial analysis and processing workflows. The Code Editor offers access to the full power of Earth Engine. In this exercise, you will learn about the Code Editor platform and explore some basic programming concepts, in JavaScript. Some basic coding and JavaScript knowledge is required to use the Earth Engine Code Editor.

Part 1: Introduction to the Code Editor IDE

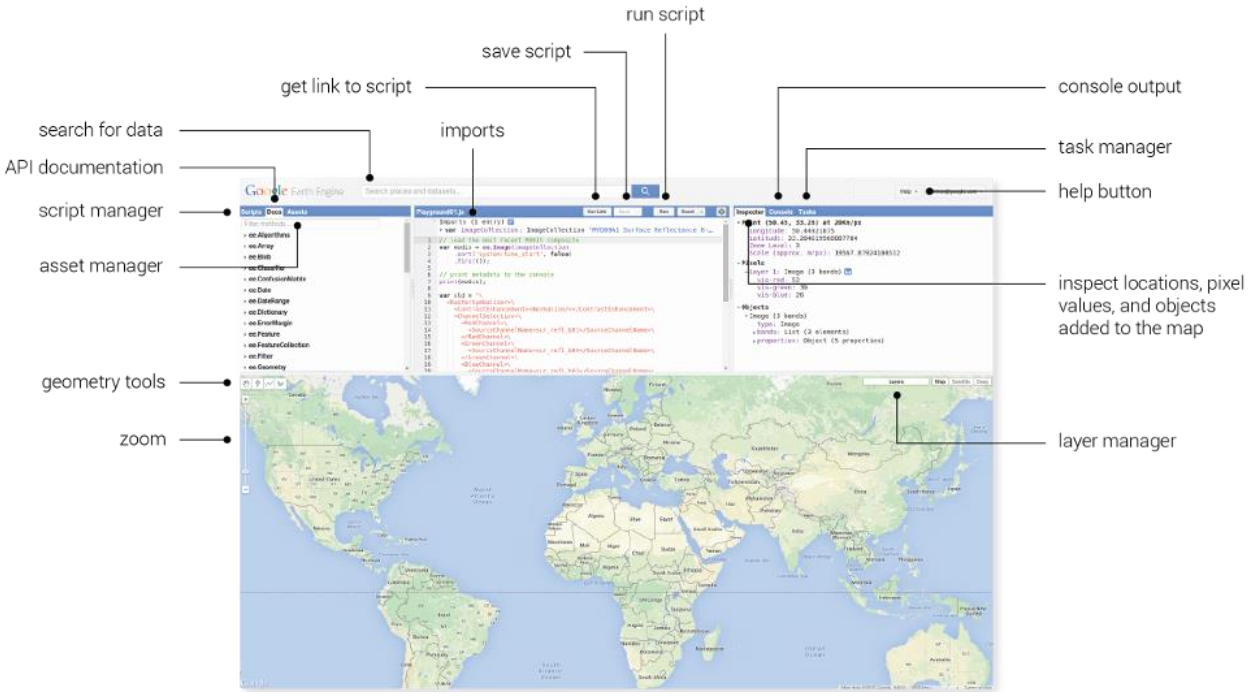
In this exercise, you will work in the Google Earth Engine Code Editor. This platform offers significantly more flexibility than the GUI based Explorer platform. You have the ability to create complex and customized analysis workflows. In the Code Editor you will write JavaScript code to access and analyze imagery.

A. Explore the JavaScript Code Editor

1. In your Google Chrome web browser navigate to the following URL: https://code.earthengine.google.com/

i. When/if prompted, allow the EE Code Editor to access your Google Account.

ii. This should take you to the Code Editor interface shown below.



2. Use the graphic above to guide you and click through the tabs in the upper left hand panel, the Scripts and Documentation Panel.

   i. Under the Scripts tab, note the wide variety of preloaded example scripts that demonstrate capabilities and offer code that you can use for your analyses. You can take a look at these to start to learn about what kinds of things Earth Engine can do. After you create and save a script later in the day, it will be available here in your Private repository.

   ii. Under the Docs tab, there is a searchable list of documentation for the predefined GEE object types and methods. Note these are grouped and organized by type. Briefly explore what kinds of methods (functions) are available in GEE for different object types.

      (a) Select one of interest and click on it to see the information window with a description of the methods and associated arguments (required and optional). Any optional arguments are italicized. (The example scripts include examples of many of these methods, try searching for them using the scripts search bar.)

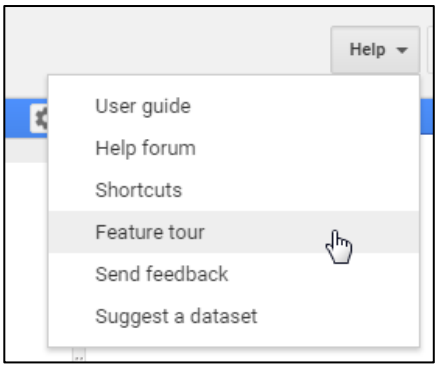
3. Using the graphic above click through the tabs in the upper right hand panel where the Inspector, Console, and Tasks tabs are located.

   i. We will use the Inspector (similar to the identify tool in ArcMap) to easily get information about layers in the map at specified points (specified by clicking in the Map Panel).

   ii. The Console is used to return messages as the scripts run and print information about the data, intermediate products and results. It also records any diagnostic messages, such as information about runtime errors.

iii. The Tasks tab is used to manage the exporting of data and results.

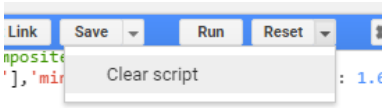
4. Click on the Help button in the upper right and select Feature Tour to learn more about each component of the API.
   i. Click through the options in the Feature tour to become more familiar with each component of the Code Editor.



Part 2: Working with Images

A. 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.



B. 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 2014 Landsat 8 image.
   i. Copy and paste the code below into the Code Editor Code Editor.

```
// Get the image.
var lc8_image = ee.Image('LANDSAT/LC8_L1T_TOA/LC81290502015036LGN00');
```

Notes about JavaScript syntax: *There's a lot going on in just two lines. Take a closer look at the pieces of this statement you're loading into GEE (refer to the Appendices to learn more about JavaScript syntax and some basic programming concepts).*

*1) Double forward slashes, **//**, are comment characters in JavaScript. These prevent text on that line from executing. These are useful for creating notes in your code.*

*2) Variables are declared in JavaScript using the keyword **var**. Variables can be numbers, strings, objects, object collections, etc. Variables are used to store information for use later on in the script. In the case of the statement above, you are naming the variable **lc8_image** and using it to refer to the raster dataset you are interested in analyzing.*

*3) **ee.Image()** is a GEE function that tells GEE that you want to load an image as an object (and in this case, save it as a variable called 'lc8_image'). In JavaScript, functions start with a letter and have a pair of parentheses at the end. Functions often include **inputs or parameters,** which tell the function what to do, and are specified inside of the parentheses. In this case, the parameter you are specifying inside the parentheses is the image ID.*

*A generalized version of the statement above is: **ee.Image('image_id')**. The 'image_id' is the image that you would like to load ('LANDSAT/LC8_L1T_TOA/LC81290502015036LGN00') and reference with the variable (lc8_image).*

*4) The syntax for specifying the image ID in this function (**ee.Image**) is to surround the string of characters (the image ID, 'LANDSAT/LC8_L1T_TOA/LC81290502015036LGN00') in quotes. The image id is in quotes because the collection and image name together is a **string**. **Strings** are sets of characters that in this example, name the specific dataset.*

a. Landsat image ids for individual Landsat scenes can be found at glovis.usgs.gov. You will work with this more in Exercise 3.

5) JavaScript statements end with a semicolon.

2. Run the code and observe the result.
   i. Click the Run button and note that nothing happens in the map or the console. This code merely creates the variable, nothing is printed or displayed.

C. Add the image to the Code Editor map

1. Copy and paste, or type, the code below into your script. These additional lines will add the Landsat image to the map panel. Add these lines *below* the code from the previous step. GEE will execute the code (lines) sequentially when you hit Run.

```
// Add the image to the map.
Map.addLayer(lc8_image);
```

2. Run the code and review the result.
   i. Click on the Run button. This time an image will load in the Map Output window. If you are not zoomed into the USA, centered on Thailand, you won't see anything.
   ii. Use your cursor to navigate (left click and drag) in the map view to Thailand and find the image you called. It would be nice if the script did this for us, next you will add that statement into your script.

D. Center and Zoom the map window

Next you will add a statement to set the zoom factor and location on which to center the map output window. Map.centerObject() is a function that tells GEE where to position the map output window.

1. Copy and paste the two lines of code (below) underneath the four lines you already have in the GEE code editor window. Click Run.

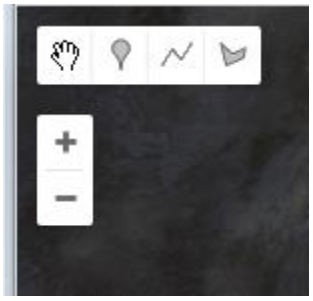
```
// Center the map display on the image.
Map.centerObject(lc8_image, 8);
```

2. To zoom out, decrease the second input to a number less than 8. To zoom in more, increase the second input parameter (try 10). Modify your statement so it looks like the two lines below and click Run. What happened?
3. In the panel in the upper left, switch from the Scripts to the Docs tab. Type Map.centerObject() into the Docs search bar. What is the range of the zoom parameter?

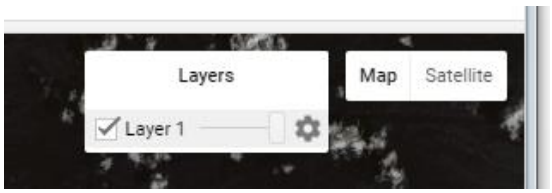
```
// Center map display on the image.
Map.centerObject(lc8_image, 10);
```

E. Explore the map window tools

1. Explore this image, the lc8_image ('LANDSAT/LC8_L1T_TOA/LC81290502015036LGN00'), with the Code Editor map viewer tools.
   i. You can zoom and pan using the tools on the left of the map output panel (shown in the following graphic).



   ii. Use the Layers tool (on the right of the map output panel) to turn the image (Layer 1) on or off (shown in following image).

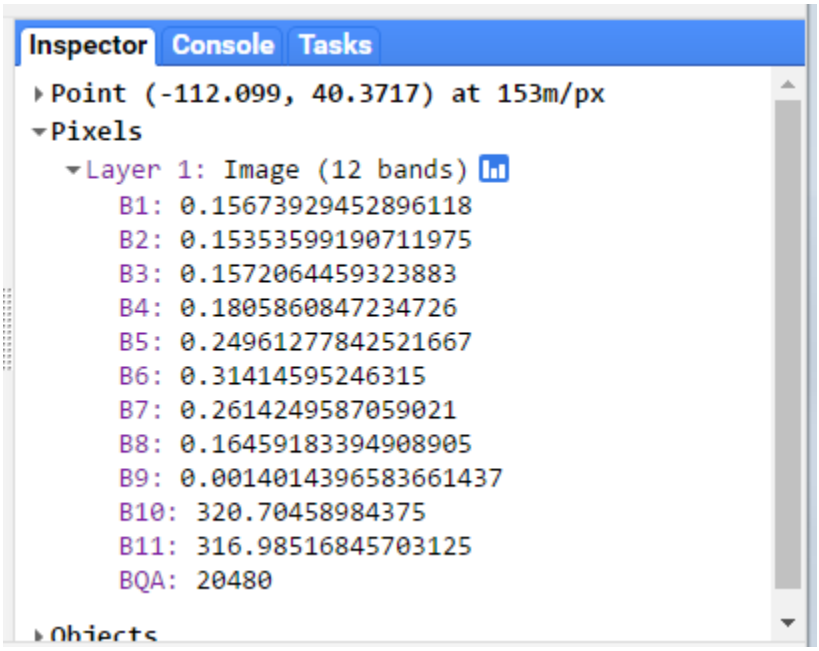


Note: *Even though you saved the* ***LANDSAT/LC8_L1T_TOA/LC81290502015036LGN00*** *image as a variable named* ***lc8_image****, the name of the image is labeled as* ***Layer_1*** *by default in the Layers Legend in the output map window. As you will see later, it is possible to change the name that appears in the Layers tool to something that is more descriptive of the data being displayed.*

iii. Swipe the transparency lever (the sliding bar to the right of the layer name in the preceding image). This will make the 'Layer 1' transparent, revealing the base map underneath.

F. Explore the Inspector window

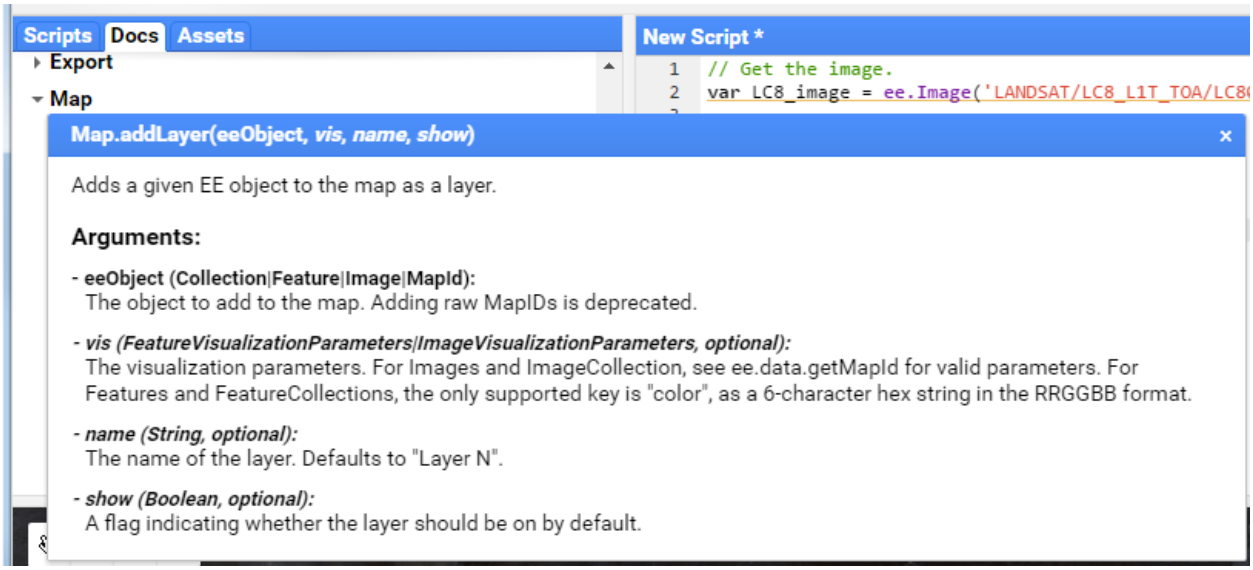
1. Click on the Inspector Tab in the upper right hand corner of the Earth Engine Code Editor interface. Your cursor will now change to a cross hairs when you place it in the map window.
2. Now click anywhere on the map using the Inspector, the cross hairs, to identify pixel values for each band in your image at the selected location (refer to following graphic for example output).



G. Change visualization parameters to improve the display

Now you can see the image, however the color parameters are not very well suited to this image. You will change the visualization parameters next.

1. Open the documentation for addLayer function in the Map group by clicking on the Docs tab in the left panel in the Code Editor.
   i. Expand the Map group and select Map.addLayer from the list (as illustrated below); or
   ii. Search for Map.addLayer in the Filter methods… search bar.



2. Review the documentation that appears (shown above). This provides information about the use and arguments for this function.
   i. Notice that some input options (such as *vis*) are italicized in the documentation. This means that these are optional parameters that can be specified or left out of the Map.addLayer statement. If you want to skip an optional parameter use "undefined" as a place holder. See the statement below for an example.

```
// Add the image to the map and name the layer in the map window.
Map.addLayer(lc8_image, undefined, 'Landsat8scene');
```

There are a number of options available to adjust how images are displayed. The inputs that are most commonly used to modify display settings include the following:

Bands: *allows the user to specify which bands to render as red, green, and blue.*
Min and max: *sets the stretch range of the colors. The range is dependent on the data type. For example unsigned 16-bit imagery has a total range of 0 to 65,536. This option lets you set the display to a subset of that range.*
Palette: *specifies the color palette used to display information. You will see how to use this option later in the tutorial.*
Naming convention (in the Layers Legend): *you can specify the name that appears in the layers legend here as well. You named this layer* ***'Landsat8scene'*** *in the code above.*

Syntax: *Most of these optional parameters are entered as a key-value pair in a dictionary object. The syntax is:*

{vis_param1: number, number, number

vis_param2: 'string, string, string',

// or an array of strings like this:

*vis_param2: **['string', 'string', 'string']}***

1. Modify the Map.addLayer() function to display the image as a false color composite and apply a stretch to improve the display. Modify the Map.addLayer() statement from the previous steps to look like the code below. The statement below includes the optional parameters for which bands to display (bands 6, 5, and 4), specifies a stretch to improve the visualization, and finally gives the image a display name.
2. Click Run and use the map tools to explore the result. Note that the name under the Layers (legend) is now Landsat8scene.

```
// Add the image to map as a false color composite.
Map.addLayer(lc8_image,
  {bands: 'B6, B5, B4', min: 0.05, max: 0.8, gamma: 1.6},
  'Landsat8scene');
```

Note: In the statement above, the names of the bands have been inserted for you already. If you wanted to look them up yourself, you can use the print function (or the Inspector) to identify what the bands are named (e.g., B6, B5, B4).

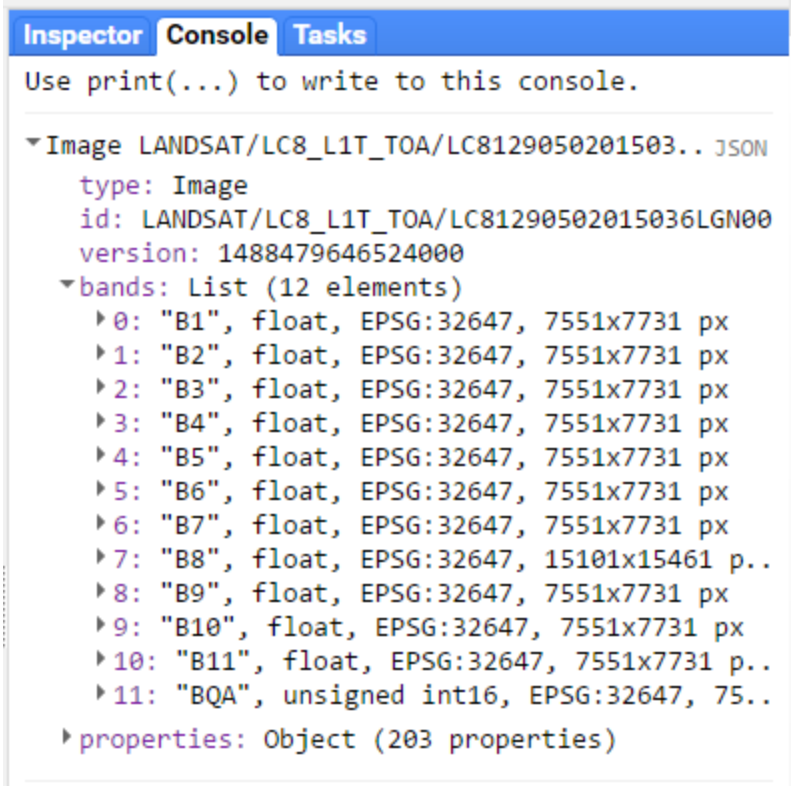
3. You can also specify the bands as strings in a list. Look at the statement below, it will do the same thing as the statement above. Do you notice the difference in syntax?

```
// Add the image to map as a false color composite.
Map.addLayer(lc8_image,
  {bands: ['B6', 'B5', 'B4'], min: 0.05, max: 0.8, gamma: 1.6},
  'Landsat8scene');
```

4. Copy and paste the following statement in your code editor. Then click Run.

```
// Print the image information.
print(lc8_image);
```

5. Now in the Console tab, click on the arrow next to Image LANDSAT/… to display the image properties. Then click on the arrow next to bands: to display the band properties. This will reveal that the first band (indexed at 0) is called “B1”, the second (indexed at 1) is called “B2”, etc. Refer to following graphic for an example.



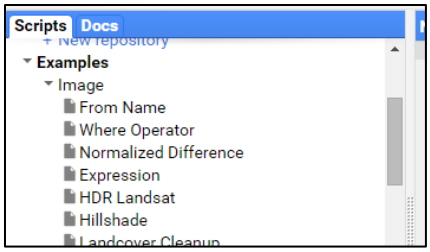
Note: *To learn about band combinations for viewing Landsat 8 and Landsat 5 or 7, check out this comparison of Landsat bands: http://landsat.usgs.gov/L8_band_combos.php.*

6. Click the Save button in the upper right of the Code Editor panel to save your example script for future reference.
    i. Name this script Visualize a Landsat 8 image.

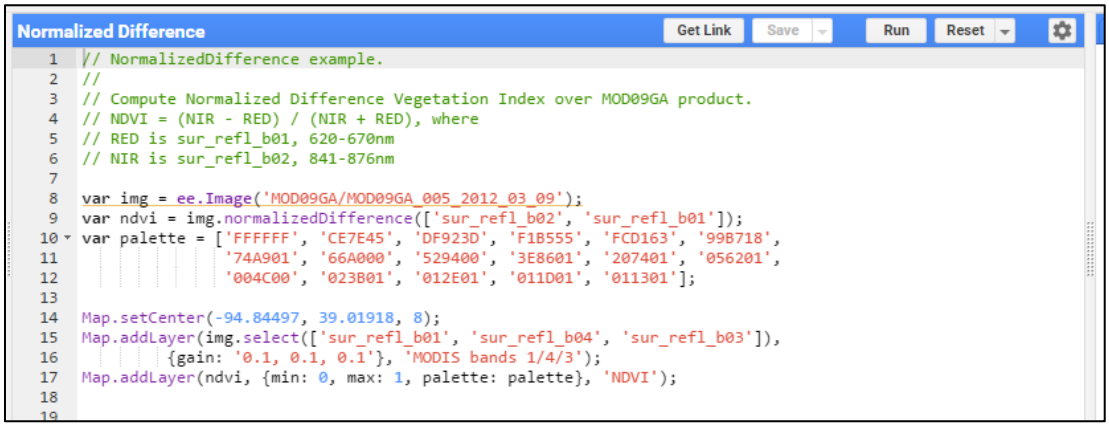
Note: *there are many more options for visualizing data in the map window, such as setting a mask or mosaicking two data sets together.*

Part 3: Run an example script and review the results

1. Click on the Scripts tab in the left-hand panel and expand the Examples group.
2. Scroll down until you see the Image group. Click on the triangle to expand this group if necessary.



3. Select the Normalized Difference script from the list of example scripts (stored within the Image group). It will copy the script into your Code Editor panel.
4. The graphic below shows the script that should appear in the Code Editor panel (upper center panel).



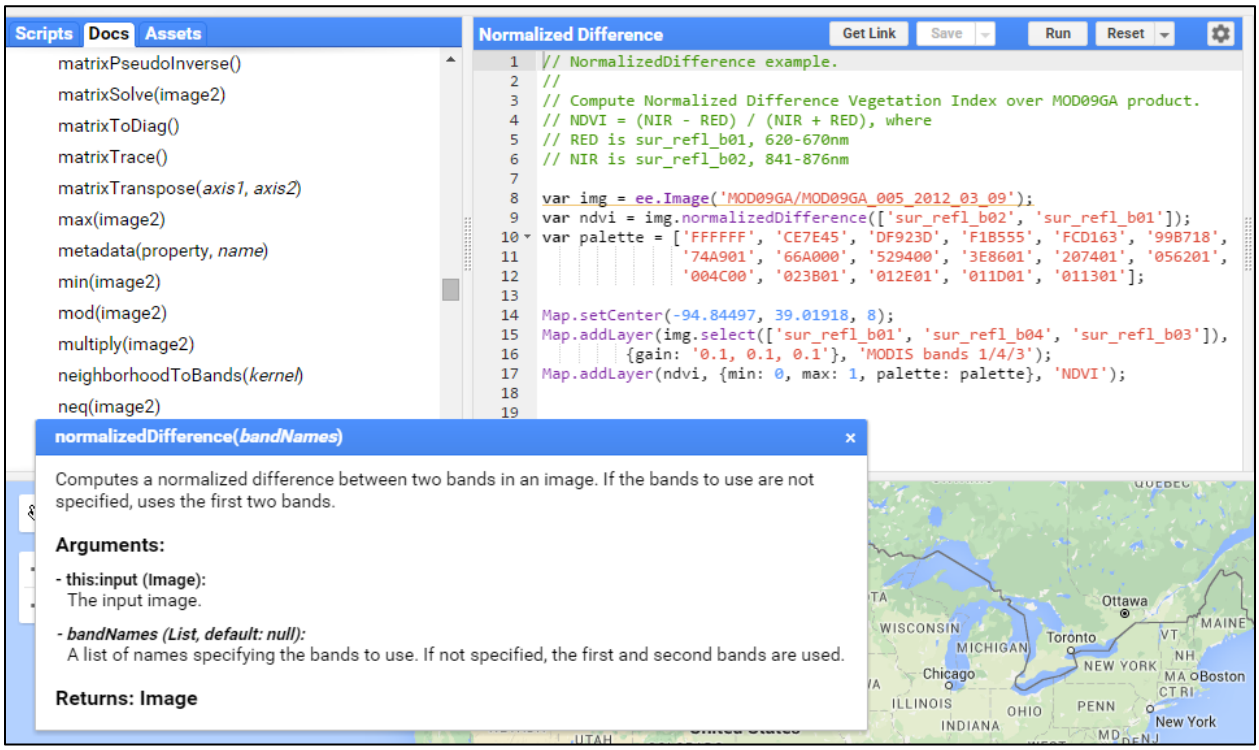
What is a script?

A script is a series of instructions that a computer follows to execute some process. It is like a recipe – a set of instructions that a cook follows one by one to create a dish. In the Code Editor platform, the instructions are written as JavaScript statements. The computer uses these statements to tell Earth Engine what information you are requesting.

5. Read the Normalized Difference script, line by line (or statement by statement), to see what it is doing:
   i. Lines 1 to 6 are notes, or comments, the developer included to describe the script. Line comments are designated with the //, the double slashes at the beginning of the line. Comments are ignored by the Code Editor when the script executes.
   ii. Line 8 accomplishes two things. It declares a variable, called *img*. It then assigns a value to this variable. The value is a MODIS image ee.Image('MOD09GA/MOD09GA_005_2012_03_09').
   iii. Line 9 does several things. It declares and assigns a value to a variable, called *ndvi*. It also calls the Earth Engine NormalizedDifference method and applies it to the variable "img" defined in the previous line. The bands "sur_refl_b02" and "sur_refl_b01" are specified as inputs to that calculation (arguments to the method). These two bands are the NIR and

Red MODIS bands, so the result of the calculation is a Normalized Difference Vegetation Index (NDVI) image. This calculated NDVI image is what is being assigned to the *ndvi* variable. Specifically, it's an image that represents the computation

(sur_refl_b02 - sur_refl_b01) / (sur_refl_b02 + sur_refl_b01).



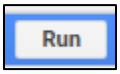
iv. Lines 10-12 declare a variable, palette, and assign to it an array that specifies a palette of hexadecimal color codes for displaying the resulting NDVI image. The hexadecimal colors range from white (FFFFFF) to browns (e.g., CE7E45) to yellows (e.g., FCD163) to greens (e.g., 529400) to very dark (011301).

Note: *You can read more about hexadecimal color codes here http://www.colorhexa.com/.*

v. Line 14 centers the map to the area of interest. The arguments, the values inside the brackets, are the longitude and latitude values for Kansas City, USA; the third value sets the zoom level.

vi. Lines 15-17 add data to the map output window (lower panel). Two images are displayed - the *img* variable, which points to the original MODIS image, and the *ndvi* variable, which points to the normalized difference image created in line 9.

6. Click the Run button in the upper-right of the code editor to run the Normalized Difference script.



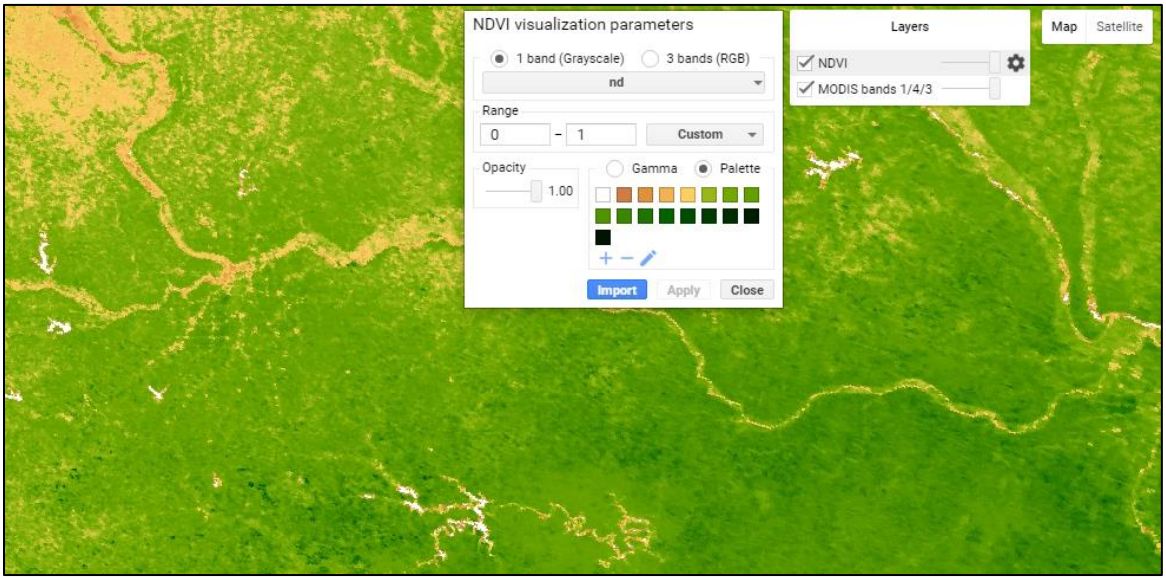
i. You should see a MODIS image and the resulting NDVI image appear in the map output window at the bottom of your screen.

7. Visually examine the results in the map output window using the map viewer tools.

   i. Click or mouse-hover on the Layers button in the upper right hand corner of the Map output panel at the bottom of your screen (shown in the following graphic).

   ii. Toggle the NDVI layer on and off by unchecking and checking the box next to the NDVI Layer.

   iii. Click and drag the slider-bar back and forth to adjust the transparency of the NDVI layer and view the MODIS image beneath the NDVI image (see following image, with visualization parameter box).



8. Use the Inspector Panel to explore the values in the resulting NDVI image.

   i. Click on the Inspector tab in the upper right-hand panel.

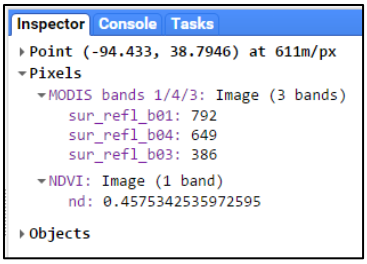
      (a) Hover your cursor over the map. Notice that your cursor has become a cross.

   ii. Click anywhere on the map and observe the values that appear in the window under the Inspector tab.

      (a) These are the pixel values at this location for:

         (i) MODIS band values for the displayed bands appear under the MODIS image name.

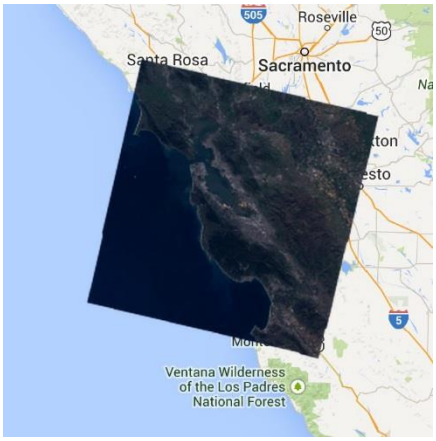
         (ii) The computed NDVI values.



Part 4: Explore Data Available in Earth Engine

1. In a web browser, such as Google Chrome, open the Google Earth Engine homepage: https://earthengine.google.com/.
2. Click on Datasets in the upper right corner. This will give you a quick overview of some of the data that is available in Earth Engine. Take a moment to read through the information on imagery, geophysical data, climate and weather, and demographic data.

Exercise 2: Earth Engine Image Objects and Methods



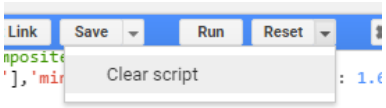
Introduction

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.

Part 1: Set up your workspace

A. 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.



B. 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.
   i. 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 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);
```

Part 2: 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.

A. 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");
```

B. Mask clouds

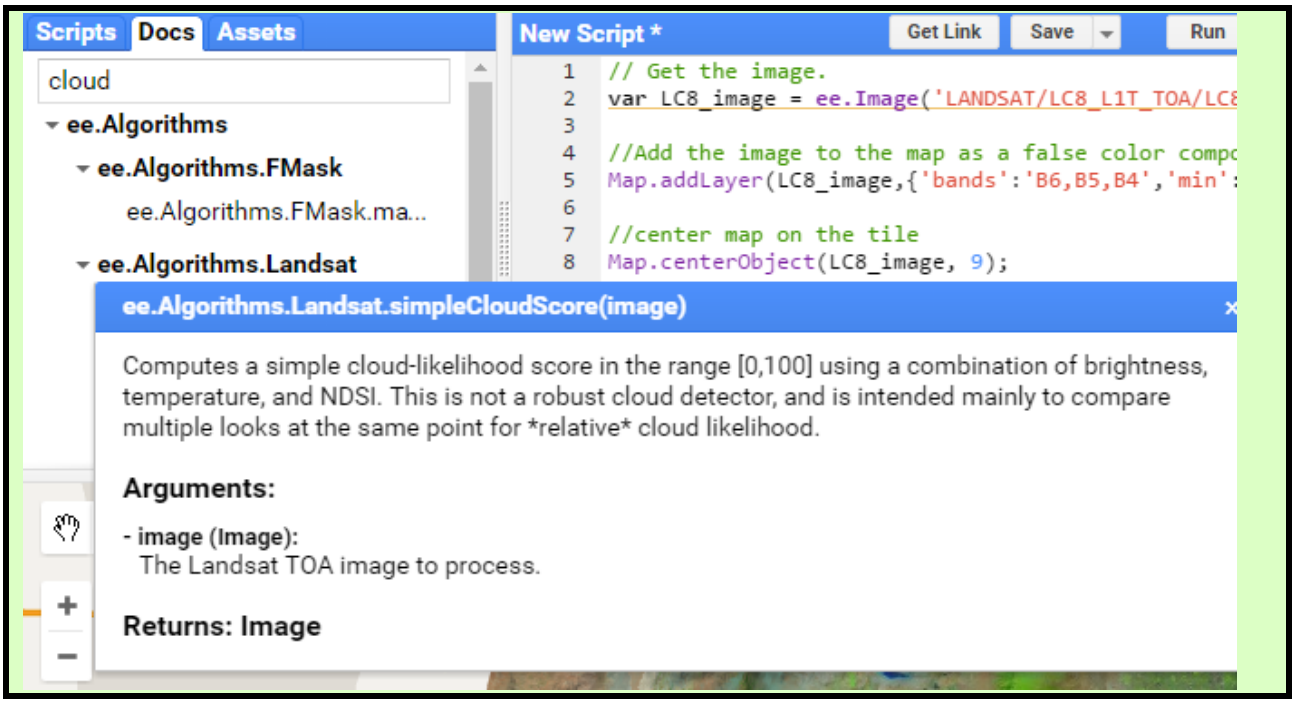
1. Clear your script from Part 2 A, except for the lines below.

```
// Get the 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);
```

2. 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.
3. 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).



4. 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);
```

5. *(Optional)* Add the cloud layer to the map by copying the lines below and adding them to the bottom of your script. Click Run.
   i. What values do the cloudy areas get assigned? (hint: use the inspector to look up values at different locations in the map)
   ii. 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)
   iii. 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');
```

6. 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:
    i. One (1) if the cloud likelihood, quality variable, is less than the cloud threshold value;
    ii. 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');
```

2. Run the code and inspect the values at different locations (hint: use the Inspector tab).
3. Copy the lines below and paste them at the bottom of your script.
    i. 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.
    ii. 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');
```

C. 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.
    i. 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

ii. 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.*

Part 3: *(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);
```

Part 4: *(Optional Read)* Some Background on Objects

A. 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)

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.

B. 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.

C. 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

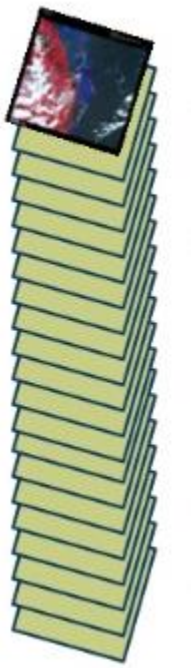
Image collection: a set of images. For example the Landsat 8 TOA Reflectance collection (LANDSAT/LC8_L1T_TOA) includes all of the imagery Landsat 8 has collected since April 11, 2013, orthorectified and corrected to Top of Atmosphere reflectance. Collections are useful for temporal analysis, or creating cloud free composites that include imagery from several acquisitions.

D. 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, then accessed in Earth Engine as a FeatureCollection.

Exercise 3: Writing Custom Functions and Mapping across Image Collections



Introduction

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

Objectives

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

Required Materials

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

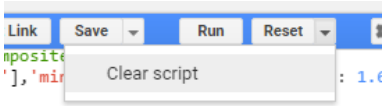
Part 5: Working with Image Collections

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

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

A. 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.



B. Create a Landsat Image Collection object

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

```
// Get an image collection.
var landsat8_collection = ee.ImageCollection('LANDSAT/LC8_L1T_TOA');
```

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

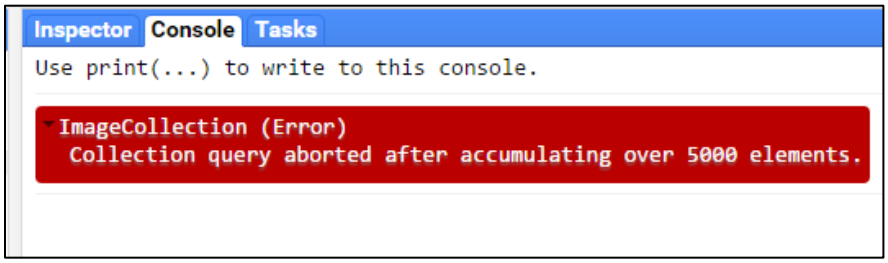
```
// Get an image collection and center the display.
Map.addLayer(landsat8_collection, {min: 0.05, max: 0.8, bands: 'B6,
  B5, B4' }, 'Landsat Collection');
Map.setCenter(100.56, 13.94, 7);
```

   i. What is mapped here? The image collection has a number of images, but not all are mapped. When you use the Map.addLayer function to add an image collection to the map window, by default the most recent pixel is shown.

3. *(Optional)* If you add a print statement, you can determine how many images are in the collection. Copy and paste the following lines into the bottom of your script.

```
// Print the information about the image collection.
print(landsat8_collection);
```

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



ii. Comment out the print statement. You will return to this later.

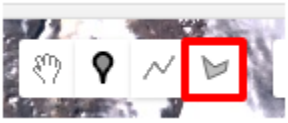
```
// print(landsat8_collection);
```

Part 6: Filter Image Collection with a Spatial Boundary

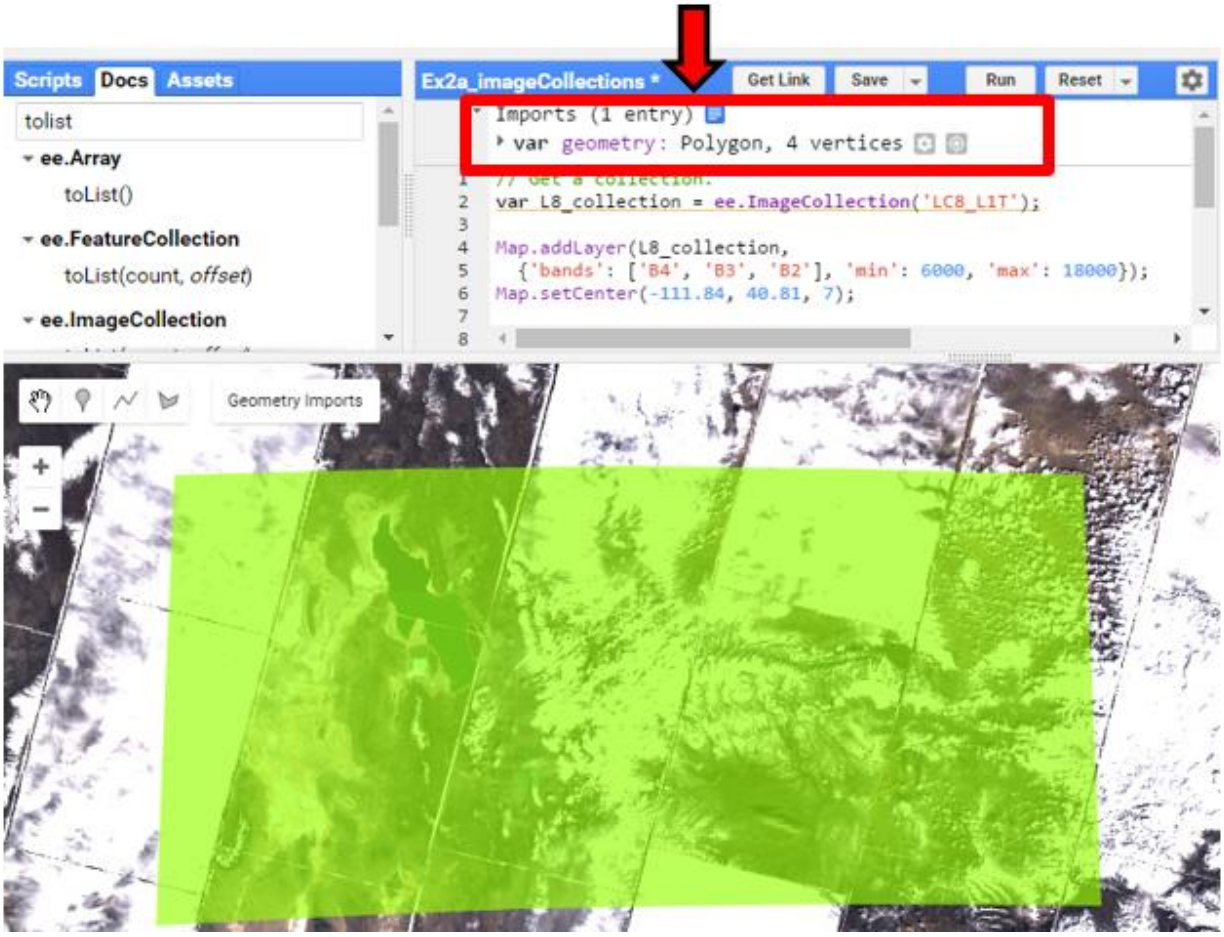
A. Draw a geometry to use later as a geographic filter (spatial boundary)

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

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

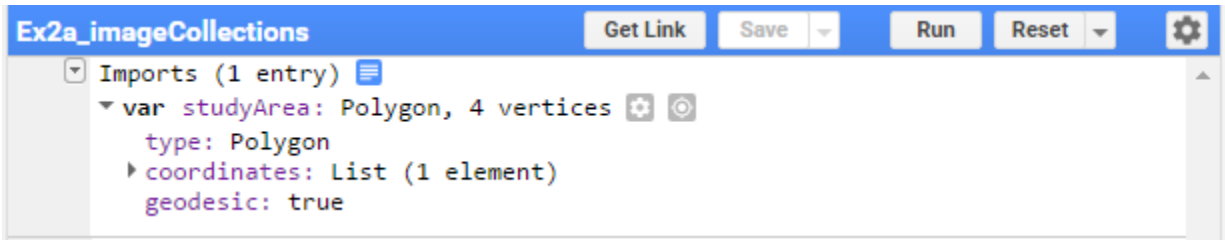


2. This will allow you to draw a geometry that represents your study area of interest. Click in the map display window to create a polygon around an area of interest to you (e.g., around Bangkok or your home town). Remember you can toggle the Landsat image on and off to see the base layer below. Double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



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

4. Click on the word geometry. Change the name from *geometry* to *studyArea*.



5. Click on the arrow next to the `var studyArea` line to see the details of the geometry you created (see preceding image for an example).

i. You can show generated code by clicking on the blue box next to the Import line. This code can be copied and pasted into your script below, or any script.

ii. If you hover over the `var studyArea` line, a trashcan icon appears to the left. This can be used to delete your imported geometry.

B. Filter image collection by geometry

1. Now you are ready to filter the image collection, landsat8_collection. Back in the code panel, copy the lines below and paste them into the code editor. Make sure you copy them below the statement that initially creates the image collection, landsat8_collection.

```
// Filter to the scenes that intersect your study region.
var landsat8_studyArea = landsat8_collection.filterBounds(studyArea);
```

1. Change the variable in the Map.addlayer statement to landsat8_studyArea (adapted statement is included below). Move it below the statement that creates the landsat8_studyArea variable.

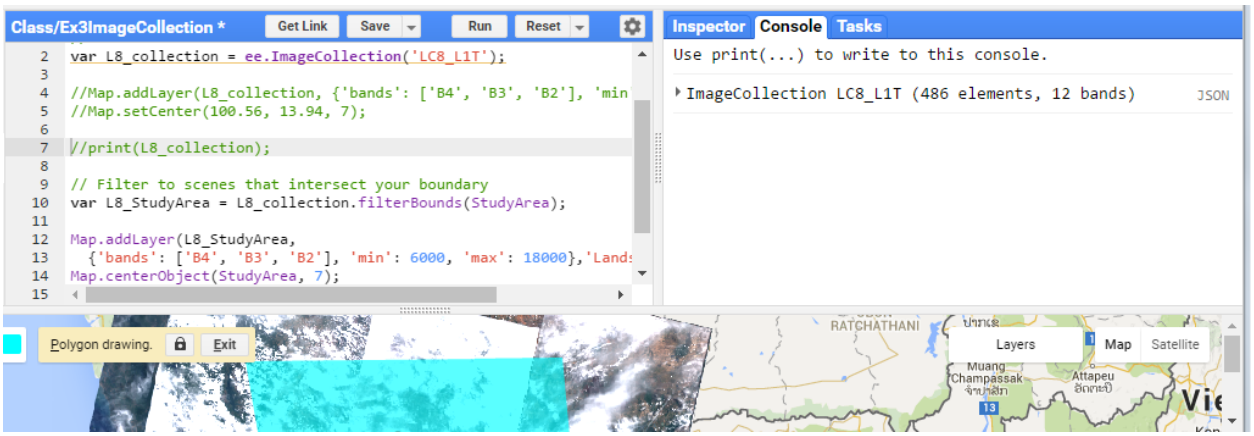
2. Change the Map.setCenter statement to map.centerObject and update the input parameters. This will center the map window on the geometry you created (example included below). Move it below the statement that creates the landsat8_studyArea variable.

```
// Display the image.
Map.addLayer(landsat8_studyArea,
  {min: 0.05, max: 0.8, bands: 'B6, B5, B4'},
  'Landsat 8 in study region');

// Center the display on the study region.
Map.centerObject(studyArea, 7);
```

3. Click Run again. Now the image collection will be filtered to only include images that intersect the polygon that you drew.

4. *(Optional)* modify the print statement to print the details of your new image collection, landsat8_studyArea (make sure to move this statement below the creation of the landsat8_studyArea variable). If your image collection has fewer than 5000 images, information about the collection is printed to the console. The size of your image collection varies depending on the size of the geometry you've digitized.

```
// Print information about the filtered image collection.
print(landsat8_studyArea);
```



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

```
// Count and print the number of images.
var count = landsat8_studyArea.size();
print('Count of landsat8_studyArea: ', count);
```

Part 7: Temporally Filter Image Collection

A. Create a date-limited image collection

1. Next add a statement that filters your image collection by a length of time, using the filterDate() method (see example script below). filterDate() allows us to specify a start and end date as parameters to reduce the size of the collection to meet your project goals. The new lines add a filter to the data collection, landsat8_studyArea, based on the date the images were taken.
2. Next, add a count and print statement to see how many images are in the new image collection.
3. The complete script is copied below for your reference. Modify your script to match and Run the code.

```
// Get an image collection.
var landsat8_collection = ee.ImageCollection('LANDSAT/LC8_L1T_TOA');

// Filter the collection to scenes that intersect your study region.
var landsat8_studyArea = landsat8_collection.filterBounds(studyArea);

// Filter the collection to a time period of interest.
var landsat8_SA_2015 = landsat8_studyArea.filterDate('2015-01-01',
  '2015-12-31');
```

```
// Display the image collection and
// center the map on the study region.
Map.addLayer(landsat8_SA_2015,
  {min: 0.05, max: 0.8, bands: 'B6, B5, B4'});
Map.centerObject(studyArea, 7);

// Count the number of images.
var count = landsat8_studyArea.size();
print('Count of landsat8_studyArea: ', count);

// Count the number of images.
var count15 = landsat8_SA_2015.size();
print('Count of landsat8_SA-2015: ', count15);
```

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

```
// Get statistics for a property of the images in the collection.
var sunStats = landsat8_studyArea.aggregate_stats('SUN_ELEVATION');
print('Sun elevation statistics: ', sunStats);

// Sort by a cloud cover property, get the least cloudy image.
var LoCloudimage =
  ee.Image(landsat8_studyArea.sort('CLOUD_COVER').first());
print('Least cloudy image: ', LoCloudimage);

// Limit the collection to the 10 most recent images.
var recent = landsat8_studyArea.sort('system:time_start',
  false).limit(10);
print('Recent images: ', recent);
```

Part 8: Introduction to Reducers

A. Image Collection Reducers, median pixel value

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

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

```
// Get a collection.
var landsat8_collection = ee.ImageCollection('LANDSAT/LC8_L1T_TOA');

// Filter to scenes that intersect your boundary.
var landsat8_studyArea = landsat8_collection.filterBounds(studyArea);

// Filter to scenes for time period of interest.
var landsat8_SA_2015 = landsat8_studyArea.filterDate('2015-01-01',
  '2015-12-31');
print(landsat8_SA_2015, 'landsat8_SA_2015');
```

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

```
// Reduce the ImageCollection to get the median in each pixel.
var median_landsat8_2015 = landsat8_SA_2015.median();
print(median_landsat8_2015, 'median_landsat8_2015');

// Display the result and center the map on the study region.
Map.addLayer(median_landsat8_2015,
  {min: 0.05, max: 0.8, bands: 'B6, B5, B4'});
Map.centerObject(studyArea, 7);
```

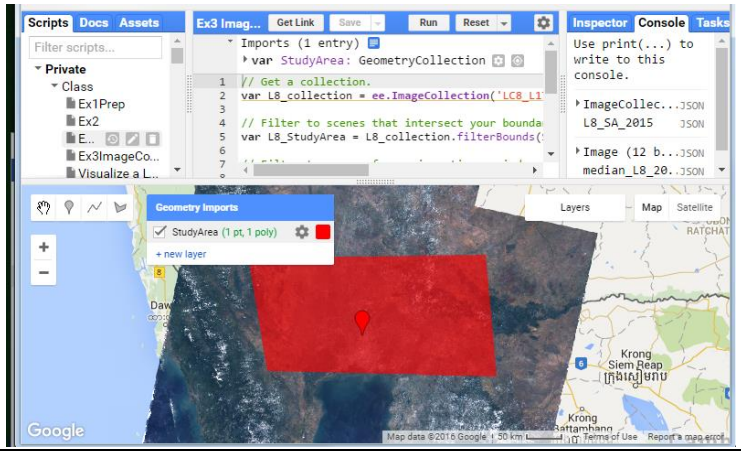
3. Review the result and note the following:

i. Look at the output from the print statements. Do you see the difference between landsat8_SA_2015 and median_landsat8_2015? Hint: one is an image collection object, the other is an image object (single raster).

ii. You can turn on the display of your study region by toggling the layer off in the Geometry Imports box in the upper left hand portion of the map screen (refer to following image).

iii. The composite image contains only the Landsat path rows that intersect your study area (zoom out to see the full extent).

iv. The image is also relatively cloud-free since you are viewing median pixel value for all the images in the collection. Are there still clouds present?



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

4. Save your script, name it 'Ex 3 Image Collection Part 8'. You will return to it later in the exercise.

Part 9: Functions Primer

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

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

A. Structure

Here's how functions are built:

I. **var**: Declare a variable to store the function in.

II. **functionName:** The word function is followed by the name you would like to call the function. You can name the function anything you like, the same rules and style suggestions of how to name variables apply to naming functions (can't start with a number, should be descriptive of what it does, etc). Follow this with an equals sign.

III. ***= function*:** Indicate the new variable is a function object. Write the JavaScript word **function**, all in lower case.

IV. **Input parameters:** After the word function, add an open and closed parenthesis. These will be filled with the set of input parameters that the user passes to the function, or left empty for functions that don't have parameters. An input parameter is a piece of information (it can be stored in a variable) that is passed into the function when it is called. If you have many parameters that you would like to include, separate them with commas. To create a function to calculate the NDVI of an image, set the image on which you are computing NDVI as an input parameter.

V. **Curly brackets:** After the parentheses that contain the input parameters, add an open and closed curly bracket.

VI. **Code:** Write the code you want to execute within your function (one statement or hundreds of lines) in between the open and closed curly brackets.

VII. **Call the function:** Once you've created a function, it doesn't execute unless you call it. To use the function in your code, write the name of the function, followed by parenthesis, with the required input parameters inside the parenthesis, and end the statement with a semicolon.

```
var functionName = function(parameter_1, parameter_2){
  // code to execute
  // more code to execute
  // …
}

// call the function
var storeOutput = functionName(input_1, input_2);
```

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

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

B. Create a function to calculate the sum of numbers

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

```
var calculate_sum = function(in_value1, in_value2) {

  // Calculate the sum.
  var sum = ee.Number(in_value1).add(ee.Number(in_value2));

  // Return the sum.
  return sum;
}

// Now declare a variable and
// set it to the value the function returns.
// Include two numbers to sum as the input parameters.
var sum_test = calculate_sum(75, 82);
print(sum_test);
```

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

```
var sum_test2 = calculate_sum(790, 1.555);
print(sum_test2);


var sum_test3 = calculate_sum(133, 765);
print(sum_test3);
```

Part 10: Create a cloud mask function

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

A. Add in cloud masking statements from Exercise 2

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

```
// Get an image.
var lc8_image = ee.Image('LANDSAT/LC8_L1T_TOA/LC81290502013110LGN01');

// Specify the cloud likelihood threshold.
var cloud_thresh = 40;

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

  // Add the cloud likelihood band to the image.
  var cloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);

  // Isolate the cloud likelihood band.
  var cloudLikelihood = cloudScore.select('cloud');

  // Compute a mask in which pixels below the threshold are 1.
  var cloudPixels = cloudLikelihood.lt(cloud_thresh);

  // Mask these pixels from the input image.
  // Return the masked input image.
  return image.updateMask(cloudPixels);
```

```
}

// Run the function on the lc8_image and save the result.
var lc8_imageNoClouds = maskClouds(lc8_image);

// Review the masked image and assess the output.
Map.addLayer(lc8_imageNoClouds,
  {bands: ['B6', 'B5', 'B4'], min: 0.1, max: 0.5},
  'Landsat8scene_cloudmasked');
```

2. Save your script as “Ex3_CloudMaskFunction”.

Part 11: Mapping Functions Across Image Collections

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

1. Replace the *lc8_image* with the filtered image collection into the script. See full example below.

```
// Store the Landsat 8 image collection in a variable.
var landsat8_collection = ee.ImageCollection('LANDSAT/LC8_L1T_TOA');

// Filter to scenes that intersect your study region.
var landsat8_studyArea = landsat8_collection.filterBounds(studyArea);

// Filter to scenes for your time period of interest.
var landsat8_SA_2015 = landsat8_studyArea.filterDate('2015-01-01',
  '2015-12-31');

// Specify the cloud likelihood threshold.
var cloud_thresh = 40;

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

  // Add the cloud likelihood band to the image.
  var cloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);

  // Isolate the cloud likelihood band.
  var cloudLikelihood = cloudScore.select('cloud');

  // Compute a mask in which pixels below the threshold are 1.
  var cloudPixels = cloudLikelihood.lt(cloud_thresh);

  // Mask these pixels from the input image.
  // Return the masked input image.
  return image.updateMask(cloudPixels);
};
```

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

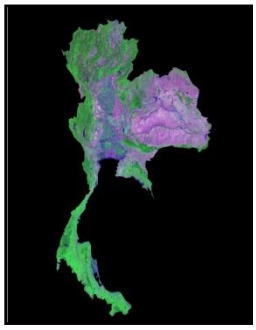
```
// Mask the clouds from all images in the image collection
// with the map function.
var landsat8_SA_2015NoClouds = landsat8_SA_2015.map(maskClouds);

// Add the first masked image in the collection to the map window.
Map.addLayer(ee.Image(landsat8_SA_2015NoClouds.first()),
  {min:0.05, max: 0.8, bands: 'B6, B5, B4'},
  'first image with clouds masked');

// Center your map.
Map.centerObject(studyArea, 7);
```

3. Use a median reducer on the `landsat8_SA_2015NoClouds` image collection that you just created to aggregate the information from all the images in the collection. See example code below.

```
// Reduce the collection to the median value per pixel.
var median_L8_2015 = landsat8_SA_2015NoClouds.median();

// Print the information of the reduced image.
print(median_L8_2015, 'median_L8_2015');

// Display reduced image in the map window.
Map.addLayer(median_L8_2015,
  {min: 0.05, max: 0.8, bands: 'B6, B5, B4'},
  'median composite of cloud free images');
```

Exercise 4: Exporting Data

Introduction



In the previous exercises, you worked with data and generated new output entirely through the GEE Code Editor. 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.

Objectives

- Learn how to export results and images,
- Export data from a complete image processing script to get useful data.

Part 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);
```

2. 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.

Part 2: (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);
```

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.

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

C. 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.

D. 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 (10^8).*

region – *By default, the viewport of the Code Editor is exported but you can also specify a geometry to change the export extent.*

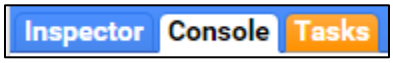
crs – *The coordinate reference system for the output image. This is specified using the EPSG code. You can look up the* ***EPSG*** *code for your desired spatial projections at http://spatialreference.org.*

scale – *The resolution in meters per pixel. The native resolution for the canopy height data set is 30 arc-seconds or approximately one kilometer.*

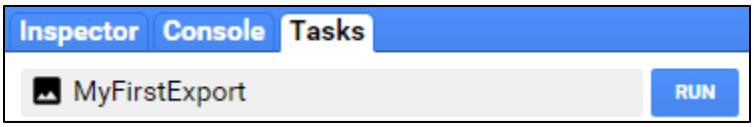
E. Run the Task to export an image to Google Drive

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.



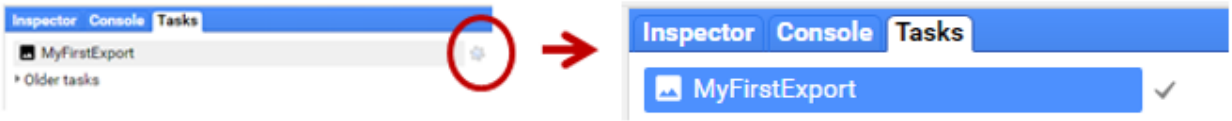
2. Click on the Tasks tab. Then click the blue Run button (shown below) to export the data to your Google Drive.



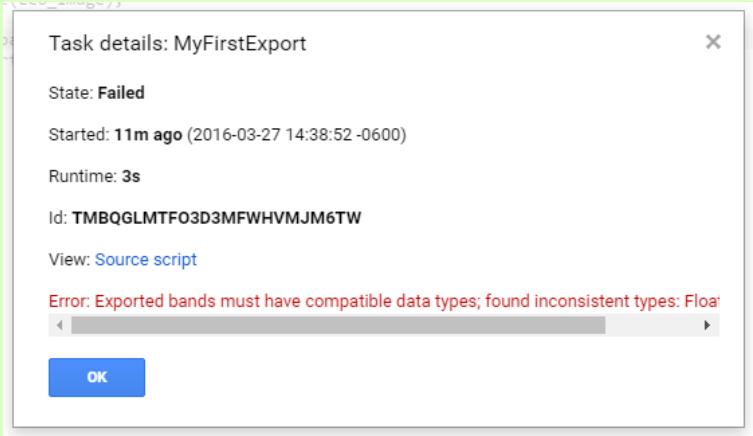
3. Review the information in the Initiate export window that appears (below). Then click Run to begin.

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****.*

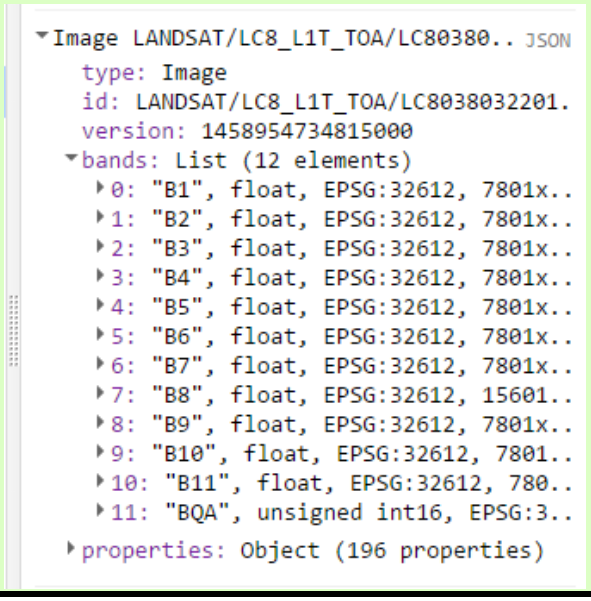
4. After you click Run at the bottom of the Initiate export window to start the export, your screen will show the export is processing. The task under the Code Editor Tasks tab should now have a spinning GEE icon next to it. It can take some time to export the task. When it is completed, this icon will 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).



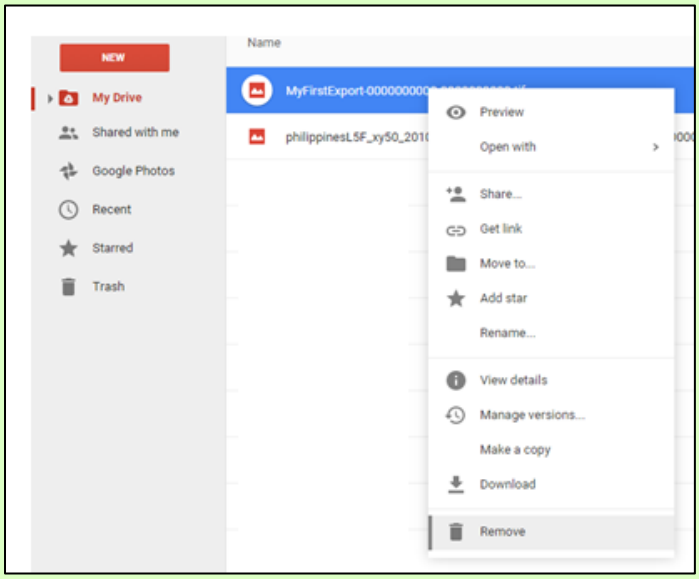
F. Review the result

1. When the export has completed, navigate to your Google Drive using the link below: https://drive.google.com/drive/my-drive

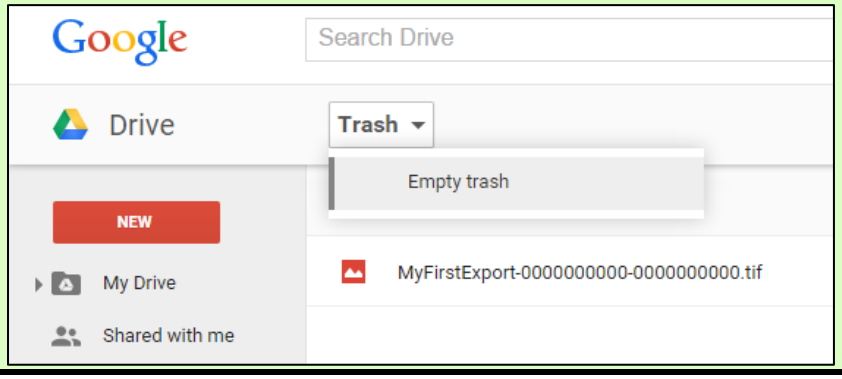
2. Locate the new images, called something similar to MyFirstExport-0000000000-0000000000.tif, in your Google Drive.
3. *Optional* – Download one of the images and view it in your preferred GIS software (e.g., ArcMap or QGIS).

Note - *Once your data 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.

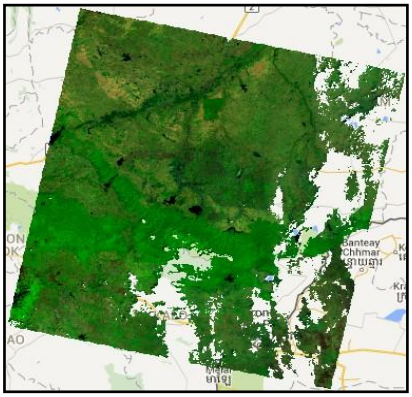


G. Save the complete script to your GEE account

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.*

Exercise 5: Preparing and exporting a cloud-free image composite in Google Earth Engine



Introduction

In the previous exercises you briefly explored the generic NDVI function and ran a relatively simple script to generate a cloud free composite. The following exercise will walk you through a customized script that will create a cloud free composite that utilizes imagery from three Landsat sensors (5, 7, and 8) and incorporates a sophisticated cloud and shadow masking algorithm. The provided script is relatively complex, preventing an in depth exploration of the script given the limited time. The script enables you to create a composite that can be used in subsequent sections of this change detection training. You can refer to this script to download data specific to your project study region(s).

You can read more about the cloud and shadow masking process we use here:
http://www.leafasia.org/sites/default/files/public/resources/USAIDLEAF-RSAC-ForestLossReport-November2015-Revised.pdf

Objectives

- Create cloud-free Landsat images for two time periods within a specific study area (Krabi, Thailand).
- Download those study area composites for use in later exercises.
- Create a cloud free composite for an entire country (in this case the Philippines).

Part 1: Investigate the "Cloud Free Composite Script"

A. Open the Script link

1. Use the link below to run a prepared script

https://code.earthengine.google.com/98fce3174f3d3169c8b3d9d8b3d25fca

2. Expand the code and review the script
   i. Note the "User Editable Variables" section, lines 11 – 74. This section is divided into four categories; area, time period, parameters, and export.

B. Explore the following:

1. Composite area - this section enables the user to specify the area for which composites are made.
2. Composite time period - this section enables the user to specify the years for which composites are made.
3. Composite parameters - this section enables the user to modify various parameters to customize the composites.
4. Export parameters - this section enables the user to specify output parameters.

Note: the cloud and shadow masking process is described here: http://www.leafasia.org/sites/default/files/public/resources/USAIDLEAF-RSAC-ForestLossReport-November2015-Revised.pdf

Part 2: Prepare a cloud free image composite for a study area

In this section you will create a cloud free composite using an uploaded fusion table. You will use a fusion table that delineates an area near Krabi, Thailand. However, you could use these same instructions for any study area that you are interested in. If you were using your own fusion table you would insert that fusion table ID into the script for the variable fusion_table_id. Review Appendix 1 for more information on uploading data to create your own custom fusion tables.

A. Run the script with a user selected country

1. Set up the user editable variables according to the code below, note the provided code should be identical to the one below. So no editing is necessary this time.

```
/////////////////////////////////////////////////////
// User Editable Variables
/////////////////////////////////////////////////////
// Composite area
// Create a Feature Collection for the area of interest.
var country_name = ['Thailand'];
```

```
var countries =
  ee.FeatureCollection('ft:1tdSwUL7MVpOauSgRzqVTOwdfy17KDbw-1d9omPw');
var country = countries.filter(ee.Filter.inList('Country',
  country_name));

// Set the variable, studyArea, to point to your FeatureCollection,
// country. Note, you can also load your own Fusion Table as a
// FeatureCollection. Then set the studyArea, line 24, to the
// FeatureCollection you created.
var studyArea = country;

//////////////////////////////////////
// Composite time period
// startYear: First year to include imagery from
var startYear = 2000;

// endYear: Last year to include imagery from
var endYear = 2002;

// startJulian: Starting Julian Date- Supports wrapping for tropics and
// southern hemisphere
var startJulian = 305;

// endJulian: Ending Julian date- Supports wrapping for tropics and
// southern hemisphere
var endJulian = 90;

//////////////////////////////////////
// Composite parameters
// cloudThresh: If using the cloudScoreTDOMShift method-Threshold for cloud
// masking (lower number masks more clouds. Between 10 and 30 generally
// works best)
var cloudThresh = 20;

// dilatePixels: Number of pixels to buffer clouds and cloud
// shadows by (1 or 2 generally is sufficient)
var dilatePixels = 2;

// cloudHeights: Height of clouds to use to project cloud shadows
var cloudHeights = ee.List.sequence(200,5000,500);

// zScoreThresh: Threshold for cloud shadow masking- lower number masks out
// less. Between -0.8 and -1.2 generally works well
var zScoreThresh = -0.8;

// shadowSumThresh: Sum of IR bands to include as shadows within TDOM and the
// shadow shift method (lower number masks out less)
var shadowSumThresh = 0.35;

//////////////////////////////////////
// Export parameters
```

```
// Give the study area a descriptive name. This name is used for output
// composites file names.
var exportName = 'Thailand';

// EPSG number for output projection. 32647 = WGS84/UTM Zone 47N.
// For more info- http://spatialreference.org/ref/epsg/
var crs = 'EPSG:32647';

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// End of user editable parameters. Do not edit anything below this
line.
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

2. Run the script and review the results.
    i. Click the **Run** button.
    ii. Click on the **Tasks** tab and review the composite available for export, Thailand_2000_2002_305_90_Composite. Don't download this yet, it is a large file.

Part 3: Assess composites, adjust parameters

A. Examine imagery

1. Use the **Viewer** tools to explore the composites created by the script.
    i. If the results are not visible, use the Layer buttons to toggle off and on the composites from each year.
    ii. Is there complete coverage for both composites? Are there any clouds that were not be masked?

B. Adjust cloud and shadow thresholds and compare changes

1. Adjust cloud threshold, lowering the cloud threshold will reduce the presence of cloudy and/or hazy pixels in the image
    i. Adjust the **cloudThresh** variable on *line 45* to obtain an optimum level of cloud masking.
    ii. You might not be able to eliminate all clouds. Some areas have such persistent cloud cover that there may not be clear imagery within your composite time period.

C. Examine pixel values

1. Click on the Inspector tab.
2. Use the Inspector to examine pixel values in the composites, just click on the pixel of interest while the Inspector Tab is highlighted.
    i. Note that the composite has the standard Landsat bands (blue – swir2) and four additional bands:
        (a) p10NDVI – the tenth percentile NDVI value (scaled by 10,000)

(b) p50NDVI – the fiftieth percentile NDVI value (scaled by 10,000)

(c) p90NDVI – the ninetieth percentile NDVI value (scaled by 10,000)

(d) pxlcount – count of the number of scenes that were used to make that composite per pixel

ii. These additional values can be used to evaluate data quality and assist in the classification process

Part 4: Review user editable variables

Now that you have had a chance to run the script, review the input parameters.

A. Composite Area – controls for the spatial extent

1. country_name (*line 17*): specifies the country in the GEE Fusion Table dataset.
2. studyArea (*line 24*): can be changed to point to a Fusion Table you provide and load as a Feature Collection.

B. Composite time period – controls temporal extent

1. startYear (*line 29*): specify the start year to use to create the composite.
2. endYear (*line 32*): specify the end year to use to create the composite.
3. startJulian (*line 36*): Julian day for starting composite.
4. endJulian (*line 40*): Julian day of year to end composite. The startJulian and endJulian dates allow you to specify a subset of each year to include in the composite. For example, if your startJulian day = 305 (November 1) and your endJulian day = 90 (March 31) your composite will only include imagery collected from November 1 to March 31. This allows you to constrain the composite to a particular season (e.g., winter) for a more uniform image.

C. Composite parameters

1. cloudThresh (*line 47*): controls the sensitivity of the cloud masking algorithm. Lower numbers increase masking, higher numbers decrease masking.
2. dilatePixels (*line 51*): controls the number of pixels used to buffer clouds and cloud shadows. Default value should work well for most areas.
3. cloudHeights (*line 54*): the height of clouds to use to project cloud shadows. Default value should work well for most areas.
4. zScoreThresh (*line 58*): the threshold for cloud shadow masking, a lower number will mask out fewer pixels. Default value should work well for most areas.
5. cloudHeights (*line 62*): the sum of infrared bands to include as shadows within the dark object method, TDOM, and the shadow shift method. A lower number will mask out fewer pixels. Default value should work well for most areas.

D. Export parameters

1. exportName (*line 68*): specifies the prefix both for exported imagery and labeling in the viewer

2. crs (*line 72*): specifies the coordinate reference system as an EPSG number for the exported imagery. Refer to this online resource for more information on how to look up the EPSG code for your project. http://spatialreference.org/ref/epsg/

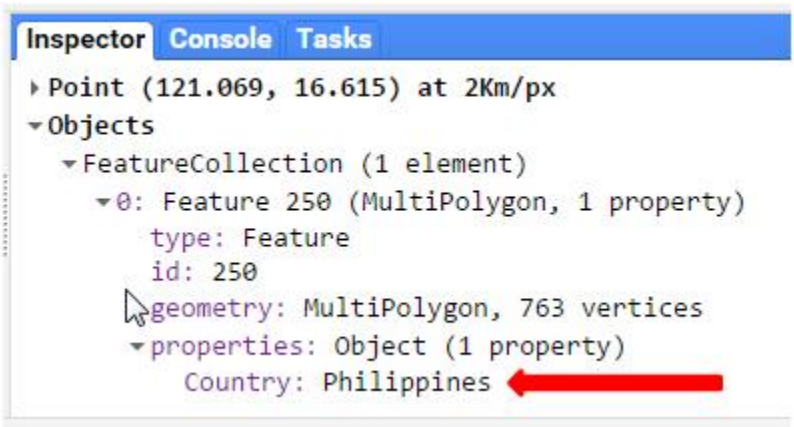
Part 5: Create a cloud free composite by country

A. Determine country name

1. Go to https://code.earthengine.google.com/ to get an empty code editor panel.
2. Paste in the short code below and click **Run** to load the countries dataset.

```
// Create a FeatureCollection from a public Fusion Table of the
// countries around the world.
var countries =
  ee.FeatureCollection('ft:1tdSwUL7MVpOauSgRzqVTOwdfy17KDbw-1d9omPw');

// Add the countries feature collection to the map.
Map.addLayer(countries);
```

3. In the **Viewer**, pan/zoom to the Philippines.
4. Activate the **Inspector** tab and click on the Philippines.
5. On the **Inspector** tab, expand the Feature and properties to see the country name (see figure on next page). This is the value you should enter for the country_name (line 24) and exportName (line 68) variables in the composite script.
6. Close this script. There is no need to save it.



B. Determine the appropriate ESPG Spatial Reference Code

1. In the script you will notice on line 71 in the comments there is a link for more information on the EPSG number for output spatial reference. Since you are now going to work in the Philippines you need a different code to represent the coordinate reference system, the one you have been working with is appropriate for Thailand. Visit the website and find the new coordinate reference system code. Click on the link: http://spatialreference.org/ref/epsg/

2. The Philippines is in UTM Zone 51N. Enter "UTM Zone 51N" in the Search References box and click search. Choose the WGS 84 /UTM zone 51N option (second one from the bottom), which is EPSG number **32651**. You will need to enter that into your code on line 72, refer to the graphic below. Note if you didn't want to use a projected coordinate system you could just use the WGS84 geographic system (EPSG: 4326).

C. Run the script with a user selected country

1. Set up the user editable variables.

   i. Change line 17 to filter by the `Philippines`.

```
// Composite area
// Create a Feature Collection for the area of interest.
var country_name = ['Philippines'];
```

   ii. Keep the time period dates and the composite parameters the same.

   iii. Change the exportName, line 68, to `Philippines`.

```
////////////////////////////////////////
// Export parameters
// Give the study area a descriptive name. This name is used for output
// composites file names.
var exportName = 'Philippines';
```

   iv. On line 72, change the crs to `32651`.

```
// EPSG number for output projection. 32647 = WGS84/UTM Zone 47N.
// For more info- http://spatialreference.org/ref/epsg/
var crs = 'EPSG:32651';
```

2. Run the script and review the results.

   i. Click the **Run** button.

   ii. Use the viewer to toggle the composites off and on.

   iii. Note the name of the composites in the viewer.

      (a) The composite nomenclature consists of the exportName, years included in the composite, and the Julian dates for each year.

   iv. Click on the **Tasks** tab and review the two composites available for export.

      (a) Philippines_2000_2002_305_90_Composite

   v. The output for this full country composite is very large (~10 GB) so you will not be exporting this image.

D. (optional) use a Fusion Table to set a study area

1. Set up the user editable variable on line 24 to point to a Fusion Table that you have created. For this exercise, we will refer to a Fusion Table that outlines an area in southern Thailand, https://fusiontables.google.com/DataSource?docid=15QnxFwFyvvsXqA0b6XIS1DDfZFtBX0DWn4en88Sw#map:id=3.

   i. Change line 24 to point to this fusion table, set up as a feature collection object.

```
// Composite area
```

```
// Set the variable, studyArea, to point to your FeatureCollection,
// country. Note, you can also load your own Fusion Table as a
// FeatureCollection. Then set the studyArea, line 24, to the
// FeatureCollection you created.
var studyArea =
 ee.FeatureCollection('ft:15QnxFwFyvvsXqA0b6XIS1DDfZFtBX0DWn4en88Sw');
```

ii. Adjust the time period parameters if you like.

iii. Then update line 68 to a more appropriate name, such as `S_Thailand`.

iv. Make sure the crs on line 72 is set to `32647`.

```
////////////////////////////////////////
// Export parameters
// Give the study area a descriptive name. This name is used for output
// composites file names.
var exportName = 'S_Thailand';

// EPSG number for output projection. 32647 = WGS84/UTM Zone 47N.
// For more info- http://spatialreference.org/ref/epsg/
var crs = 'EPSG:32647';
```

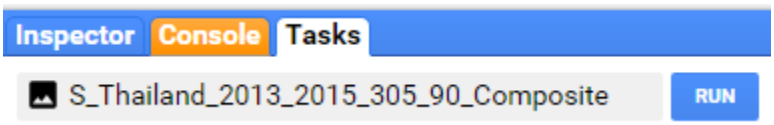
Part 6: Export imagery

A. Run the Tasks to export the images to Google Drive

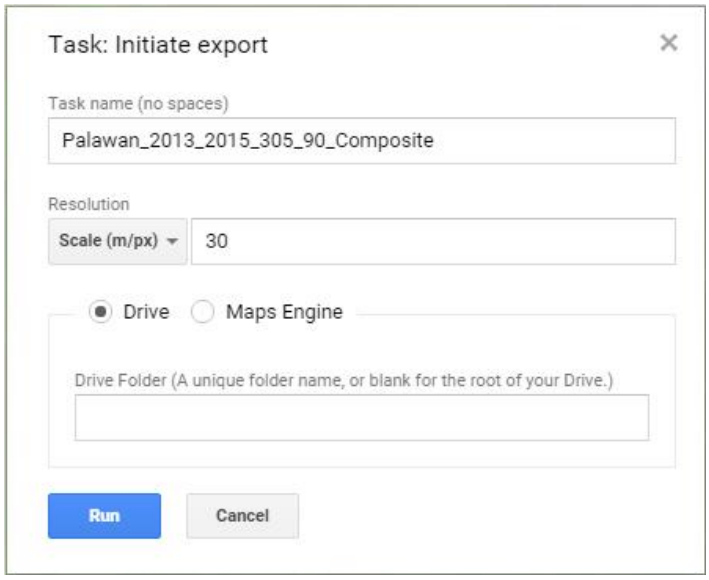
1. Run the script to be sure all your changes have been incorporated.

   i. After the script runs, select the **Tasks** tab.

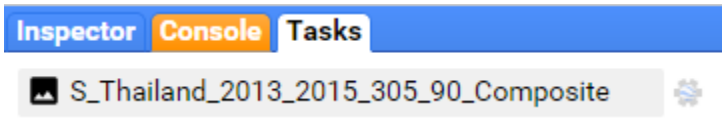
   ii. Click **RUN** to initiate an export of the composite you just created.



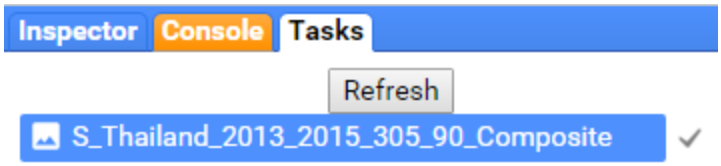
2. Review the information in the **Initiate export** window that appears (below) for each composite (note, the name below is for an island in the Philippines).



3. After you click run, the export will begin and the Run button will be replaced by spinning icons indicating the export is progressing (see image on following page).

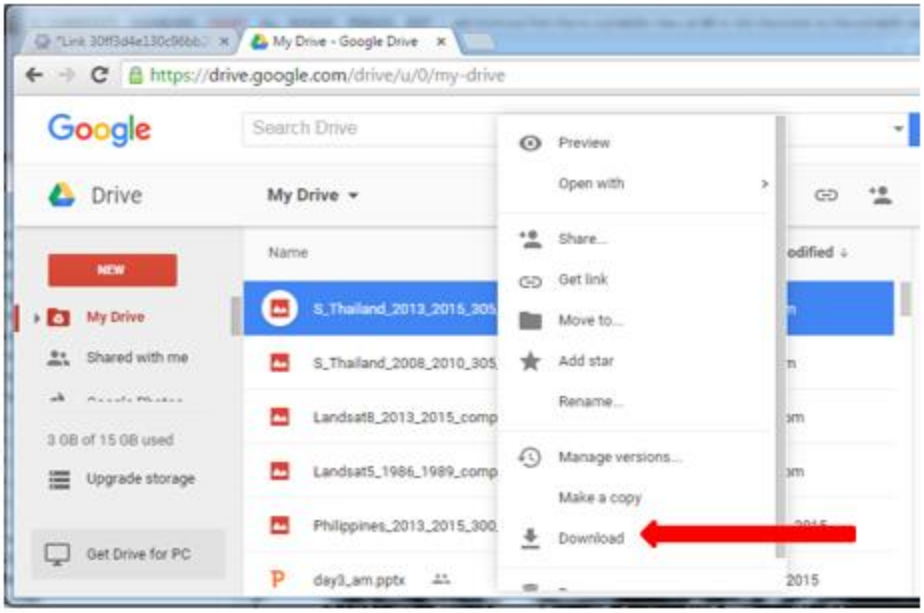


4. Once the export is complete (~10 minutes) the spinning icons will be replaced with a check mark indicating the export is complete. This indicates that your composites have been exported to your Google Drive.



B. Download composites from Google Drive

1. Go to your Google Drive and locate the exported composites.



2. Select the composite, right click and select download.
3. Follow the prompts to specify a download location.
4. Repeat this process for the second composite.

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. The composite for the whole country of Thailand is approximately 1728 MB, this subset was 933 MB. Before you export a lot of data, make sure you have room on your Google Drive! If necessary delete and empty the trash to make more room.*