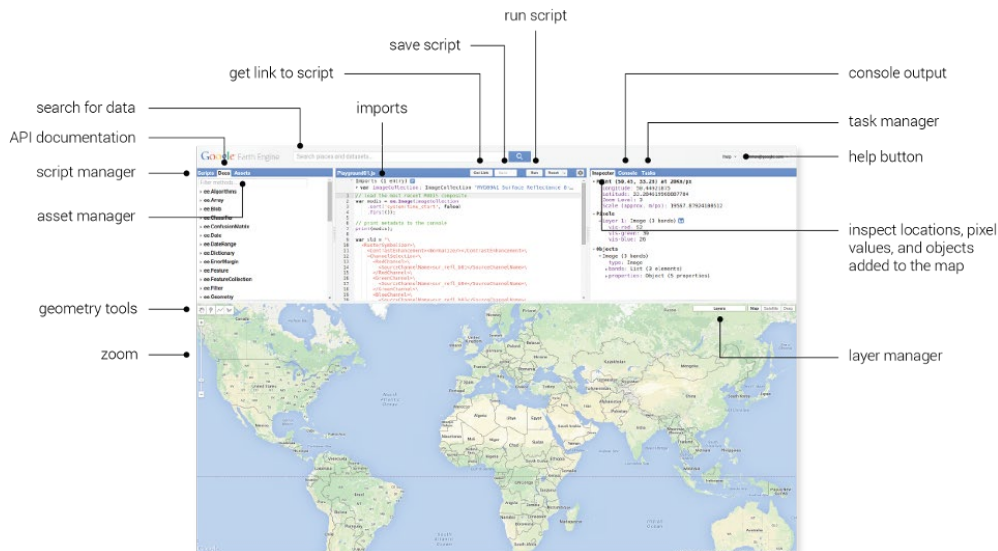


Exercise 1: Introduction to Google Earth Engine Code Editor, Cloud Projects, and Data Archive



Introduction

[Google Earth Engine](#) is a cloud-based geospatial processing platform. It 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 which is what you'll be using for this training. The Code Editor offers access to the full power of Earth Engine. This platform is where users can write and execute scripts to share and repeat geospatial analysis and processing workflows.

In this exercise, you will learn about the Code Editor platform and explore some basic scripting concepts in JavaScript. Some basic coding and JavaScript knowledge is required to use the Earth Engine Code Editor. If you've never written any code before and this tutorial is too advanced, you may want to look at GTAC's *Introduction to Geospatial Scripting* course. This course has some reminder and overview of basic scripting, but that material is not covered in depth.

Objectives

- Log into the Google Earth Engine website
- Explore the Google Earth Engine Explorer (graphical user interface)
- Explore the Google Earth Engine Code Editor



- Load and visualize Landsat imagery in Google Earth Engine
- Learn how to access Google Cloud Projects and create a Google Cloud Project bucket

Prerequisites

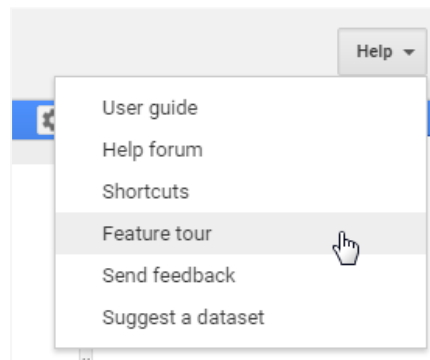
- You have a Google Earth Engine account
 - Register for a Google Earth Engine account using your USDA email. Fill out this [Microsoft Form](#), which will tell GTAC staff to create an account for you. This process may take 1-3 business days.
- Basic coding and Javascript skills
- Optional: GTAC's Introduction to Geospatial Scripting Course



Table of Contents

| | |
|---|----|
| Part 1: Introduction to the Code Editor IDE..... | 4 |
| Part 2: Introduction to repositories..... | 5 |
| Part 3: Run an example script and review the results..... | 7 |
| Part 4: Working with Images..... | 10 |
| Part 5: Explore Google Cloud Projects..... | 18 |
| Part 6: (<i>optional</i>) Explore Data Available in Earth Engine..... | 22 |

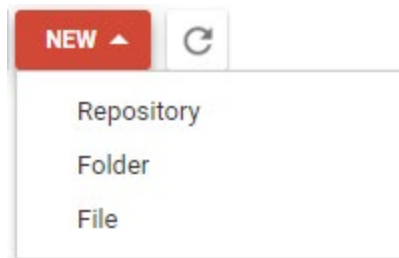
- iii. Briefly explore what kinds of methods and functions are available in GEE for different classes.
 - (a) Select one of interest and **click** on it to see the information window with a description of the methods and associated arguments (required and optional). Any optional arguments are italicized. (The example scripts include examples of many of Earth Engine methods and their arguments. 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. You will use the Inspector (like 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.



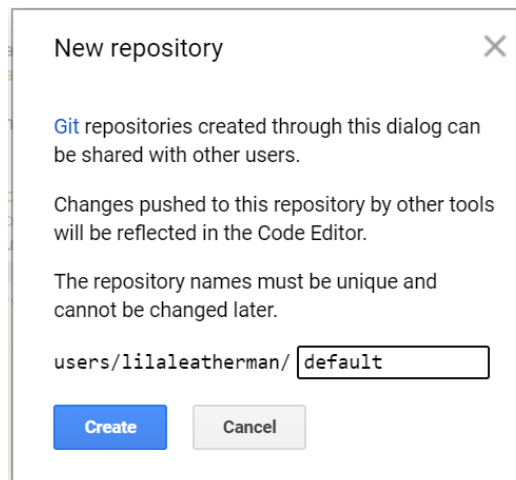
Part 2: Introduction to repositories

A. Create a default repository

1. Google Earth Engine stores scripts in **repositories**. These repositories are like folders—but are enabled with version control and sharing of select project with select groups. You will need to create a repository to save scripts in during this exercise and later exercises in the course.
2. Under the Scripts tab in the upper left window, click the **New** button and select **Repository** to create a new repository.



3. Enter “default” in the dialog box that appears.

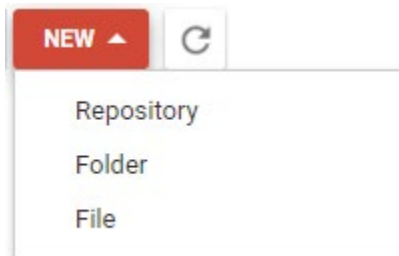


4. You should now see the default repository appear under your **Owner** tab in the scripts pane

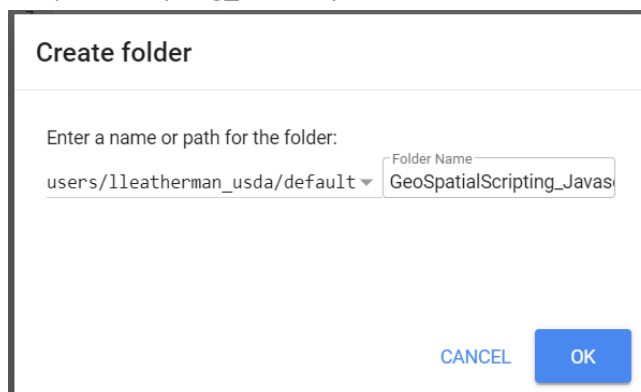
Note: The Earth Engine code editor stores scripts in [Git](#) repositories, or repos, hosted by Google. In your script manager, repositories are arranged by access level and your private scripts are stored in the Owner folder (**users/username/default**). You can create additional repositories, share them, and use the extremely popular distributed version control system, Git, to manage your repos or even sync them with external systems like GitHub.

B. Create a new folder in the code editor

1. Repositories help manage sharing and version control. Folders within those repositories help provide organization. You’ll create a new folder to save the scripts you create during this course.
2. Create a new folder and script in your default repository
 - i. Under the Scripts tab in the upper left window, click the New button and select FolderGTA to create a folder this course.



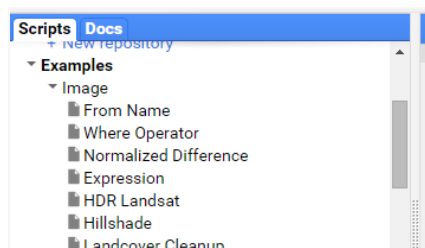
- In the dialog box, enter the 'GeospatialScripting_Javascript' and click OK – This creates a folder called 'GeospatialScripting_Javascript'.



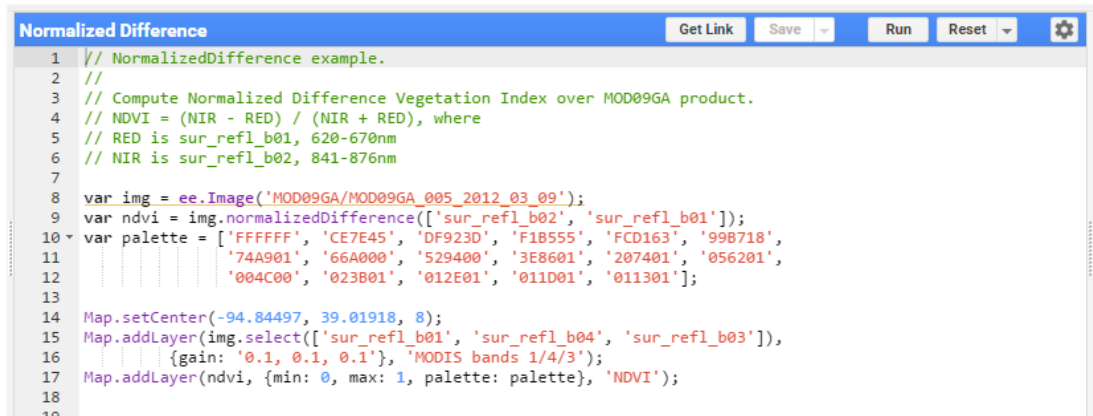
Part 3: Run an example script and review the results

A. The Examples Section

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



- 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.
- The graphic below shows the script that should appear in the Code Editor panel (upper center panel).



```

1 // NormalizedDifference example.
2 //
3 // Compute Normalized Difference Vegetation Index over MOD09GA product.
4 // NDVI = (NIR - RED) / (NIR + RED), where
5 // RED is sur_refl_b01, 620-670nm
6 // NIR is sur_refl_b02, 841-876nm
7
8 var img = ee.Image('MOD09GA/MOD09GA_005_2012_03_09');
9 var ndvi = img.normalizedDifference(['sur_refl_b02', 'sur_refl_b01']);
10 var palette = ['FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718',
11             '74A901', '66A000', '529400', '3E8601', '207401', '056201',
12             '004C00', '023B01', '012E01', '011D01', '011301'];
13
14 Map.setCenter(-94.84497, 39.01918, 8);
15 Map.addLayer(img.select(['sur_refl_b01', 'sur_refl_b04', 'sur_refl_b03']),
16             {gain: '0.1, 0.1, 0.1'}, 'MODIS bands 1/4/3');
17 Map.addLayer(ndvi, {min: 0, max: 1, palette: palette}, 'NDVI');
18
19

```

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 comment lines 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:

$$(\text{sur_refl_b02} - \text{sur_refl_b01}) / (\text{sur_refl_b02} + \text{sur_refl_b01}).$$

The screenshot shows the Earth Engine Code Editor interface. On the left, a list of functions is visible, including `matrixPseudoinverse()`, `matrixSolve(image2)`, `matrixToDiag()`, `matrixTrace()`, `matrixTranspose(axis1, axis2)`, `max(image2)`, `metadata(property, name)`, `min(image2)`, `mod(image2)`, `multiply(image2)`, `neighborhoodToBands(kernel)`, and `neq(image2)`. The main editor area displays the following code:

```

1 // NormalizedDifference example.
2 //
3 // Compute Normalized Difference Vegetation Index over MOD09GA product.
4 // NDVI = (NIR - RED) / (NIR + RED), where
5 // RED is sur_refl_b01, 620-670nm
6 // NIR is sur_refl_b02, 841-876nm
7
8 var img = ee.Image('MOD09GA/MOD09GA_005_2012_03_09');
9 var ndvi = img.normalizedDifference(['sur_refl_b02', 'sur_refl_b01']);
10 var palette = ['FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '998718',
11               '74A901', '66A000', '529400', '3E8601', '207401', '056201',
12               '004C00', '023B01', '012E01', '011D01', '011301'];
13
14 Map.setCenter(-94.84497, 39.01918, 8);
15 Map.addLayer(img.select(['sur_refl_b01', 'sur_refl_b04', 'sur_refl_b03']),
16               {gain: '0.1, 0.1, 0.1', 'MODIS bands 1/4/3'});
17 Map.addLayer(ndvi, {min: 0, max: 1, palette: palette}, 'NDVI');
18
19

```

A tooltip for the `normalizedDifference(bandNames)` method is displayed, providing the following information:

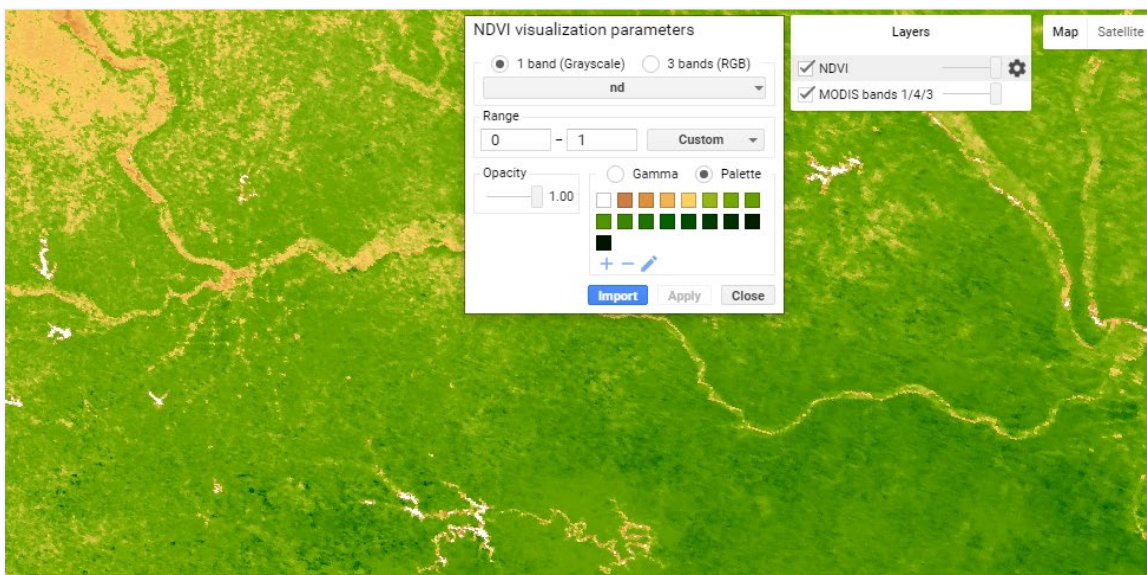
- Arguments:**
 - **this:input (Image):** The input image.
 - **bandNames (List, default: null):** A list of names specifying the bands to use. If not specified, the first and second bands are used.
- Returns: Image**

The background of the screenshot shows a map of the Great Lakes region, with cities like Chicago, Toronto, and New York visible.

- 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).

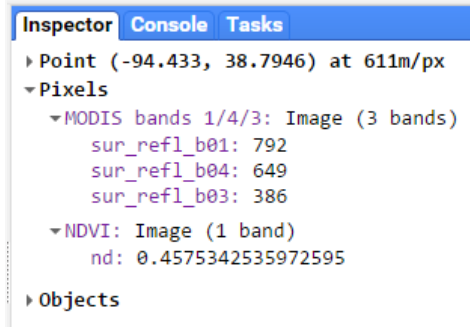


8. Use the **Inspector** Panel to explore the values in the resulting NDVI image.
- i. Click on the Inspector tab in the upper right-hand panel.
 - (a) Hover your cursor over the map. Notice that your cursor has become a cross.



- 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:

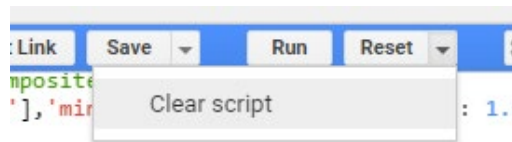
- (i) MODIS band values for the displayed bands appear under the MODIS image name.
- (ii) The computed NDVI values.



Part 4: 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/>
 - i. If you already have the code editor open, but have a script loaded, 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 2021 Landsat 8 image.
 - i. **Copy and paste** the code below into the Code Editor.

```
// Get the image.
var lc8_image =
ee.Image('LANDSAT/LC08/C02/T1_L2/LC08_038032_20210530');
```

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 (remember that if this statement is completely foreign to you, you should consider looking at the Introduction to Geospatial Scripting course for a more detailed explanation of the scripting language). The numbers below give you a quick reminder of some of the important points in the two lines.

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, images, features, etc. Variables are used to store information for use later 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 class designation that tells GEE that you want to load an *ee.Image* object (and in this case, save it as a variable called 'lc8_image'). Earth Engine classes all begin with *ee*. The parentheses at the end let you define which object of the *ee.Image* class type you're loading from the data catalog. 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/LC08/C02/T1_L2/LC08_038032_20210530') 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/LC08/C02/T1_L2/LC08_038032_20210530') 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.

5) JavaScript statements end with a semicolon.

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. 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 Salt Lake City, UT, you won't see anything.
 - i. Use your cursor to **navigate** (left click and drag) in the map view to Utah 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.
2. Click **Run**.

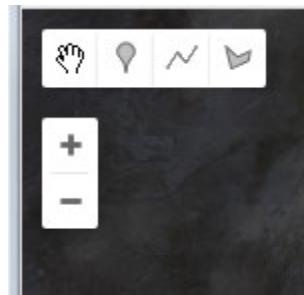
```
// Center the map display on the image.
Map.centerObject(lc8_image, 8);
```

3. 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
4. Click **Run**. What happened?
5. In the panel in the upper left click the **Docs** tab. Type **Map.centerObject()** into the Docs search bar. What is the range of the zoom parameter?

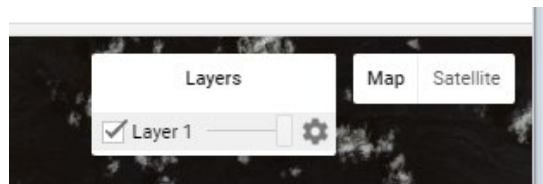
```
// Center map display on the image.
Map.centerObject(lc8_image, 10);
```

E. Explore the map window tools

1. Explore this image 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. Click the **check box** 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/LC08/C02/T1_L2/LC08_038032_20210530* 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. Improve the display by adding scaling factors and changing visualization parameters

Now you can see the image, however the color parameters are not very well suited to this image. This is a result of the values in which the data are stored, as well as the way that we tell GEE to display the image on the map. We first need to apply scaling factors to convert

the values to reflectance which ranges from 0 to 1.

1. Visit the [GEE Data Catalog entry for Landsat 8 Collection 2 imagery](#) by following the link. Scroll down to see the code snippet at the bottom of the page. This code snippet provides the parameters and a function that we will use to apply scaling factors to our image. We will modify it slightly to use it on our single image. Copy the code below to the bottom of your current script:

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

lc8_image = applyScaleFactors(lc8_image);
```

Note: This code is doing several things! This code:

a) Creates a function called **applyScaleFactors** that applies to one image at a time.

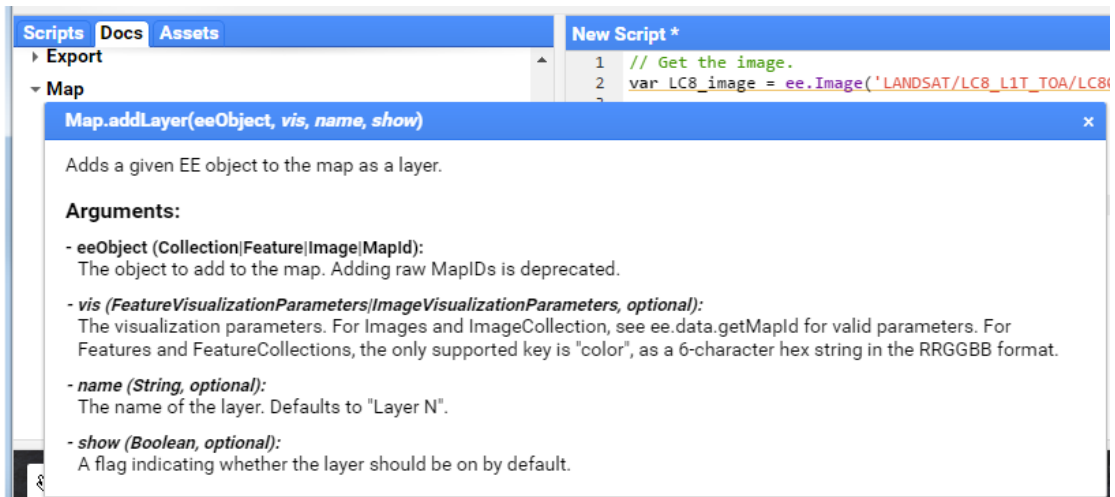
The function also

i) Selects all optical bands in the image and applies scaling factors to each one and

ii) Selects all of the thermal bands in the image and applies scaling factors to each one.

Lastly, the code b) Applies the **applyScaleFactors** function to our **lc8_image** that we loaded in previously. We'll use this function again in a later exercise.

2. Now that we've applied scaling factors to our image, we need to apply visualization parameters. 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.



3. 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']}]}
```

4. **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. “SR” precedes each of the band numbers to indicate that this is Surface Reflectance imagery.

```
// Add the image to map as a false color composite.
Map.addLayer(lc8_image, {bands: 'SR_B6,SR_B5,SR_B4', min: 0.05, max:
0.8, gamma: 1.6}, 'Landsat8falseColor');
```

5. Click **Run** and use the map tools to explore the result. Note that the name under the Layers (legend) is now Landsat8falseColor.

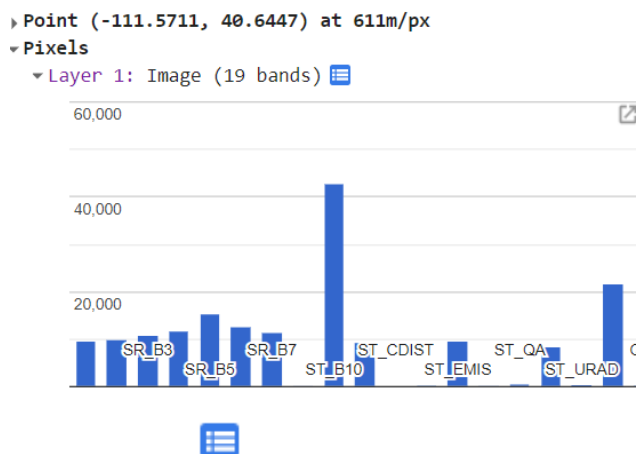
Note: In the statement above, the names of the bands have been inserted for you already. If you wanted to look them up yourself, you can use the print function (or the Inspector) to identify what the bands are named (e.g., SR_B6, SR_B5, SR_B4).

6. 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: ['SR_B6', 'SR_B5', 'SR_B4'], min: 0.05,
max: 0.8, gamma: 1.6}, 'Landsat8falseColor');
```

G. 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.



```

▶ Point (-111.5711, 40.6447) at 611m/px
▼ Pixels
  ▼ Layer 1: Image (19 bands) 
    SR_B1: 9530
    SR_B2: 9958
    SR_B3: 10994
    SR_B4: 11694
    SR_B5: 15279
    SR_B6: 12639
    SR_B7: 11513
    SR_QA_AEROSOL: 96
    ST_B10: 42682
    ST_ATRAN: 9383
    ST_CDIST: 18
    ST_DRAD: 197
    ST_EMIS: 9738
    ST EMSD: 124
    ST_QA: 514
    ST_TRAD: 8496
    ST_URAD: 345
    QA_PIXEL: 21824
    QA_RADSAT: 0
  
```

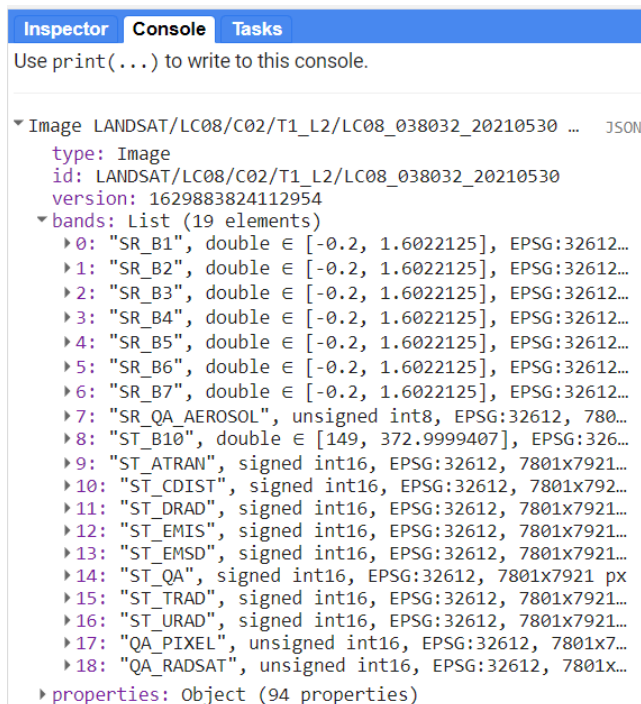
4. Now examine information about the image in the console. You'll need to use a print statement. Copy and paste the following statement in your code editor. Then click Run.

```

// 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 "SR_B1", the second (indexed at 1) is called "SR_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 resource about [common Landsat band combinations for different sensors](#).

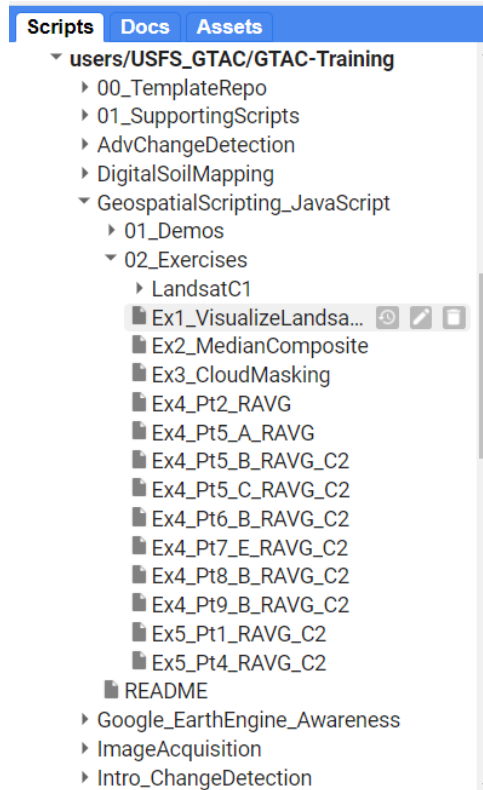
6. Click the **Save** button in the upper right of the Code Editor panel to save your example script to your default/GeoSpatialScripting_Javascript folder.
 - i. Name this script "Ex1_VisualizeLandsat".

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

7. To see an example of the completed code, you can view the script in the course repository. We have drafted and stored all of the scripts for this course—and for other GTAC GEE courses—in a **GTAC-Training** repository. This repository is managed so that anyone can have Read access to the repo—they can read and run the scripts, and they can save new copies that they can modify, but cannot Edit the scripts. GTAC staff are able to edit the scripts.
8. [To add the GTAC-Training repository to your Scripts pane, click this link.](#)



10. Take a moment to browse the contents of the repository, and the folder for this course: **GeospatialScripting_Javascript**.



11. After saving your own script, open the script for exercise 1 in the GTAC-Training repo. This should look pretty similar to what you just produced! For subsequent exercises, we'll share the

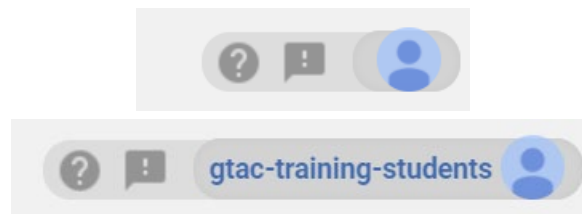
links to the full scripts—or partial scripts, as you’ll see in exercises 4 and 5—so that you can refer to the complete code.

12. You can find the complete script for visualizing Landsat imagery at [Ex1_VisualizeLandsat8Image](#) in the GTAC-Training/GeospatialScripting_JavaScript folder.

Part 5: Explore Google Cloud Projects

A. Selecting a Cloud Project

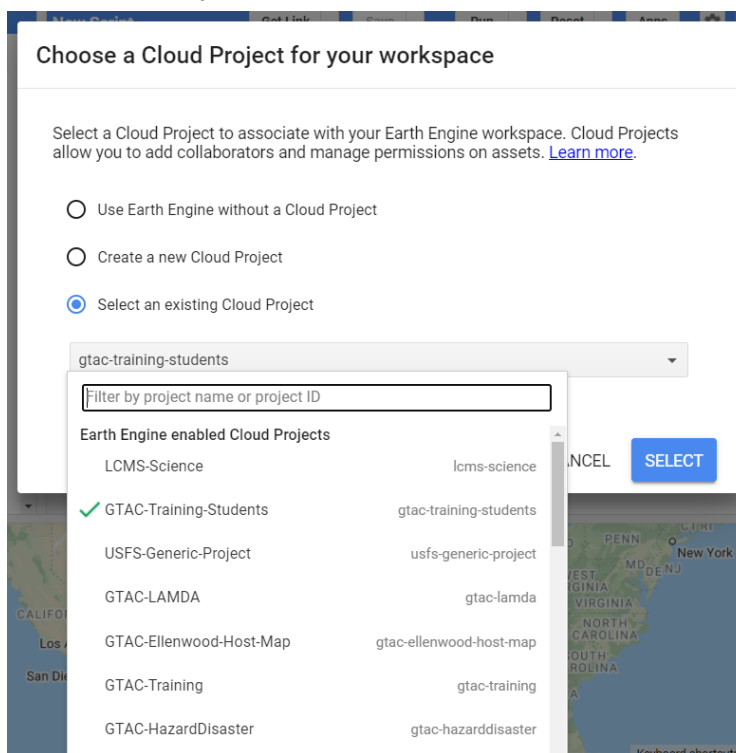
1. First, some background on Google Cloud Projects:
2. The Forest Service uses Google Cloud Projects to manage storage, billing, and privacy of Google Earth Engine projects and products. USFS GEE users need to have access to Cloud Project for computational time to be billed to the appropriate account, and importantly, **to be able to export imagery out of GEE**.
3. The USFS uses Cloud Projects for centralizing work conducted by regions, and specific to operational and ongoing projects. All Forest Service accounts are automatically granted access to regional cloud projects, e.g.: “R01-General-Projects”, “R02-General-Projects”, etc. For as-needed, one-off, and/or “clip, zip, and ship” operations, you should work in the appropriate regional Cloud Project.
4. For trainings, GTAC has created a specific Cloud Project for students to use: “gtac-training-students.” If you were registered for a training course more than 24 hours in advance, you should have been added to this cloud project
5. Notice this icon at the top right of your screen: The text in this box indicates the Google Cloud Project that you are working in. If there is not text in the box, you are not currently working in a Cloud Project.



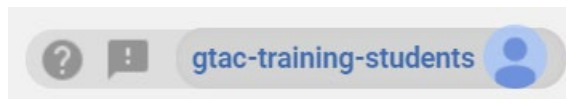
6. Click on the blue person icon. You should get an option to “Choose a Cloud Project”, if no Cloud Project is currently selected, or to “Change Cloud Project” if there is a Cloud Project currently selected.



- Click on “Choose a Cloud Project” or “Change Cloud Project”, whichever is available to you. A window will pop up that says, “Choose a Cloud Project for your workspace.” Make sure “Select an existing Cloud Project” is selected, and click the down arrow to see all Cloud Projects that you have access to. My list will look a lot different than yours, since GTAC has many many projects that have their own Cloud Projects!



- Click on “GTAC-Training-Students” to select this Cloud Project and click select. You should now see “gtac-training-students” in the top right of your screen. If you are following these instructions outside a GTAC training, select your regional General Projects cloud project.



9. Importantly, we wipe the memory in the GTAC-training-students cloud project every month! This Cloud Project is meant to be a place for you to use during your trainings, and for trainers to give specific instructions to their students. For work outside of GTAC trainings, plan to use the regional General Projects cloud project.

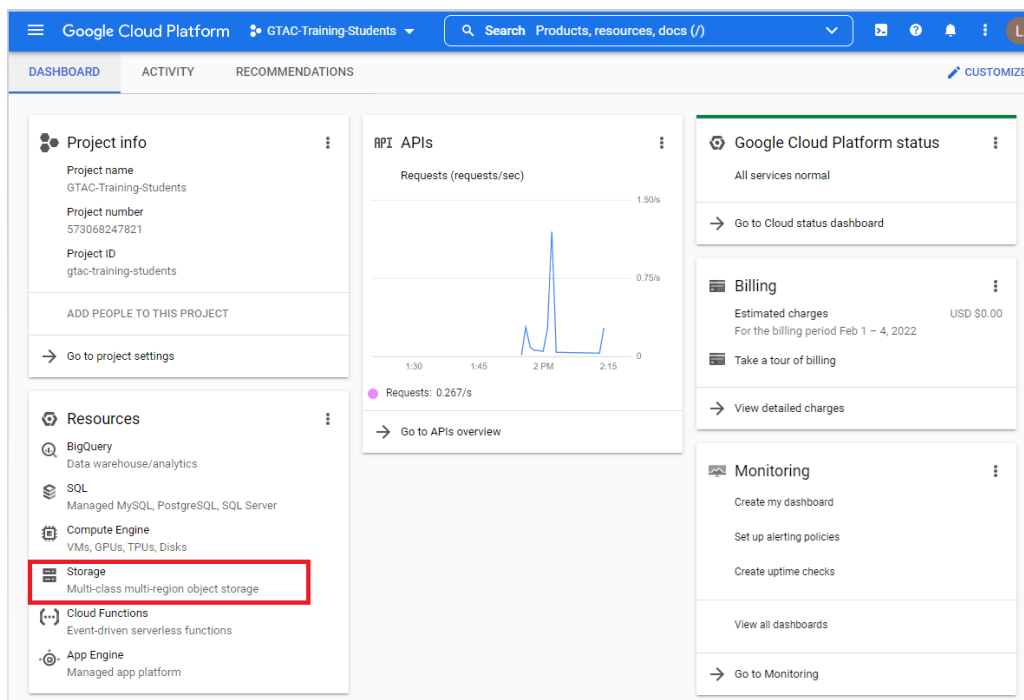
B. Create a bucket

1. Next, we need to create a “bucket” (Google’s name for a folder) that will be a destination for any files you want to export out of GEE.

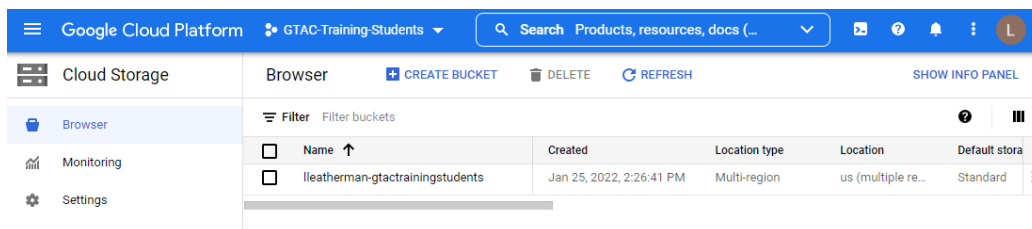
2. First, navigate to this web address:

<https://console.cloud.google.com/home/dashboard?project=gtac-training-students>

- i. This takes us to the Dashboard for the cloud project. This page has a lot of information that is more back-end admin that we don’t need to worry about—about the use of the project, storage space, and billing to the FS.
- ii. Notice that “GTAC-Training-Students” is at the top of the page. If you click the down arrow next to it, you can also see a list of other projects that you have access to and can view their dashboards.
- iii. Look on the right side of the screen, under Resources. Click “Storage.”



3. Here, you will see a list of the buckets in the Cloud Project. To create your own bucket, click “Create Bucket” at the top of the window.



4. Next, name your bucket. Your bucket name must be *globally unique*—like an email address. AT GTAC, we use the naming convention of first initial – last name – project. For example, “lleatherman-gtactrainingstudents”, or “jsmith-gtactrainingstudents” or “jdoe-r02generalprojects”.
 - i. If you have a common name, you may need to add an initial to create a globally unique bucket name. e.g., “jesmith-r02generalprojects”
 - ii. Enter the name of your bucket in the field below “Name your bucket.” We will leave all other settings as the default. Click the “Create” button at the bottom of the window.

The screenshot shows the 'Create a bucket' page in Google Cloud Platform. The main section is titled 'Name your bucket' and includes a text input field with the name 'Isleatherman-gtactrainingstudents'. Below this are sections for 'Choose where to store your data' (Location: us, Multi-region), 'Choose a default storage class for your data' (Standard), 'Choose how to control access to objects' (Uniform), and 'Choose how to protect object data' (None). On the right, a 'Monthly cost estimate' panel shows a cost of \$0.00. A 'CREATE' button is visible at the bottom left of the main configuration area.

5. The next window that appears will show the title of your new bucket, and its contents. It’s empty for now!

The screenshot shows the 'Bucket details' page for the bucket 'Isleatherman-gtactrainingstudents'. The bucket metadata is as follows:

| Location | Storage class | Public access | Protection |
|--|---------------|---------------|------------|
| us (multiple regions in United States) | Standard | Not public | None |

Below the metadata, there are tabs for 'OBJECTS', 'CONFIGURATION', 'PERMISSIONS', 'PROTECTION', and 'LIFECYCLE'. The 'OBJECTS' tab is active, showing a table with columns: Name, Size, Type, Created, Storage class, Last modified, Public access, Version history, and Encryption. The table is currently empty, displaying 'No rows to display'.

6. Click the back arrow <- next to “Bucket details” to navigate back to the list of all buckets in the Google Cloud Project. You should see your new bucket listed.

| Browser | | | | | | + CREATE BUCKET | DELETED | REFRESH | SHOW INFO PANEL |
|--------------------------|-----------------------------------|--------------------------|---------------|--------------------|-----------------------|---------------------------------|-------------------------|-------------------------|---------------------------------|
| Filter Filter buckets | | | | | | | | | |
| <input type="checkbox"/> | Name ↑ | Created | Location type | Location | Default storage class | | | | |
| <input type="checkbox"/> | lleatherman-gtactrainingstudents | Jan 25, 2022, 2:26:41 PM | Multi-region | us (multiple re... | Standard | | | | |
| <input type="checkbox"/> | lsleatherman-gtactrainingstudents | Feb 4, 2022, 2:33:51 PM | Multi-region | us (multiple re... | Standard | | | | |

Part 6: (optional) Explore Data Available in Earth Engine

A. The Earth Engine Data Catalog

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.
3. Towards the top of the page, you can click **View all Datasets** to open a new window where you can search for other data in the catalog that you might be interested in using for your own scripts.

Congratulations! You've completed this exercise and learned some basics about the code editor, Google Cloud Projects, and Earth Engine datasets. In the next exercise you'll learn to use more Earth Engine methods.