Earth Engine 101 - Introduction to the API

(edited May 2023)

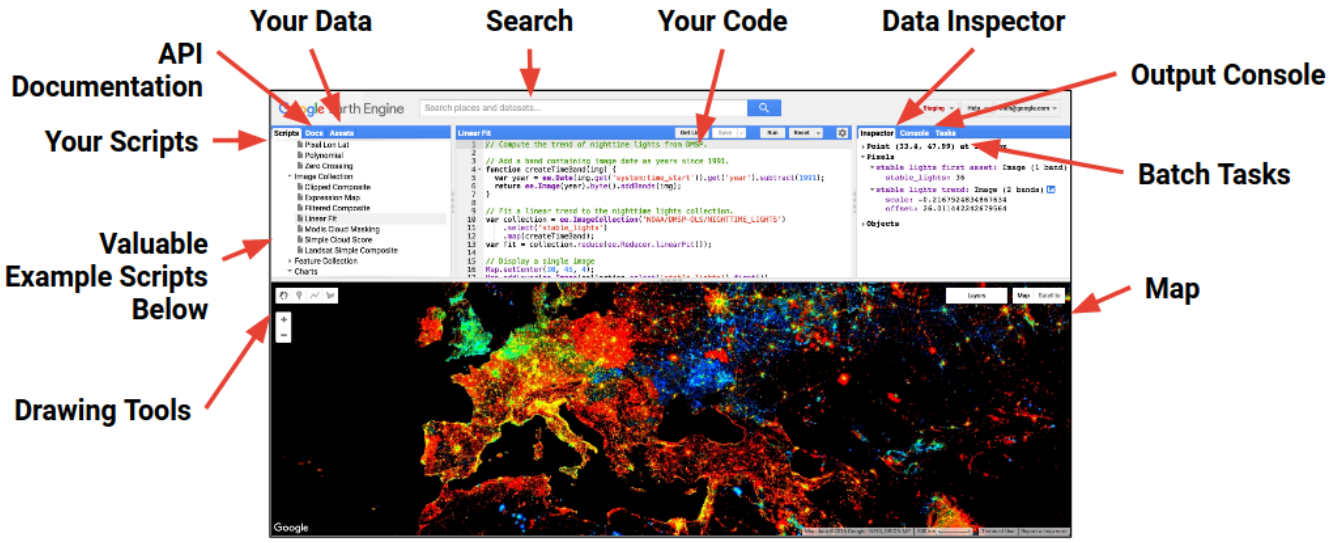
Section list

Section 00 - Go to Earth Engine site

Go to https://code.earthengine.google.com/

Some browsers work better than others for this. We use Google Chrome and Safari for Earth Engine access.

Section 1a - Code Editor Orientation



Gives participants a brief overview of the various panels of the Earth Engine code editor.

1. Editor Panel
    a. The Editor Panel is where you write and edit your Javascript code
2. Right Panel
    a. Console tab for printing output.
    b. Inspector tab for querying map results.
    c. Tasks tab for managing long-running tasks.
3. Left Panel
    a. Scripts tab for managing your programming scripts.
    b. Docs tab for accessing documentation of Earth Engine objects and methods, as well as a few specific to the Code Editor application
    c. Assets tab for managing assets that you upload.
4. Interactive Map

    a. For visualizing map layer output
5. Search Bar
    a. For finding datasets and places of interest
6. Help Menu
    a. User guide - reference documentation
    b. Help forum - Google group for discussing Earth Engine
    c. Shortcuts - Keyboard shortcuts for the Code Editor
    d. Feature Tour - overview of the Code Editor
    e. Feedback - for sending feedback on the Code Editor
    f. Suggest a dataset - Google's intention is to continue to collect datasets in our public archive and make them more accessible, so they appreciate suggestions on which new datasets to ingest into the Earth Engine public archive.

Section 1b - Javascript Syntax

Open up the Earth Engine JavaScript code editor, and type or paste the following text.

```
print('Hello, world!');
```

Now look for the **Run** button, push it, the text will appear in the **Console**. That is the output of your first piece of JavaScript code. Congratulations!

Variables are used to store objects, and are defined using the keyword var.

```
var the_answer = 42;
```

String objects start and end with a single quote.

```
var my_variable = 'I am a string';
```

String objects can also start and end with double quotes.

```
// But don't mix and match them.
var my_other_variable = "I am also a string";
```

Square brackets are used for items in a list. The zero index refers to the first item in the list.

```
var my_list = ['eggplant', 'apple', 'wheat'];
print(my_list[0]);
```

Curly brackets (or braces) can be used to define dictionaries (key:value pairs).

```
var my_dictionary = {'foodComponent': 'bread', 'colorComponent': 'red',
'numberComponent': the_answer};
```

Square brackets can be used to access dictionary items by key.

```
print(my_dictionary['colorComponent']);
```

Or you can use the dot notation to get the same result.

```
print(my_dictionary.colorComponent);
```

Can you get the editor to access the item that has the value ‘bread’?

```
// print(what do I print here? Put it between the parentheses below);
print()
```

Statements should end in a semi-colon, or the editor complains.

```
var test = 'Look for the little letter i to the left for this warning...'
```

Parentheses are used to pass parameters to functions.

```
print('This string will print in the Console tab.');
```

Functions can be defined as a way to reuse code and make it easier to read.

```
function my_hello_function(str) {
  return 'Hello ' + str + '!';
};
print(my_hello_function('world'));

// Line comments start with two forward slashes. Like this line.

/* Multi line comments start with a forward slash and a star,
and end with a star and a forward slash. */
```

Section 1c - Hello, World

1. We will now generalize the ‘Hello world’ call from earlier by refactoring (i.e. rewriting without changing the results) the script to first store the message in a string object, and then print the results of the object:

```
var msg = 'Hello, world!';
print(msg);
```

2. Here is a link to the resulting script that was obtained by clicking on the "Get Link" button.

   Ending script: https://code.earthengine.google.com/8ebfe243992ea918fd362eeeba1c759b

Section 2 - Hello, Images

1. Clear the script by selecting "Clear script" from the Reset button dropdown menu.
2. Search for “elevation” and click on the **SRTM Digital Elevation Data Version 4** result to show the dataset description.
3. Click on Import, which moves the variable to the Imports section at the top of your script. Rename the default variable name "image" to be "srtm".
4. Print the image object with the script:

```
print(srtm);
```

5. Browse through the information that was printed. Open the “bands” section to show the one band named “elevation”. Note that all this same information is automatically available for all variables in the Imports section.
6. Use the Map.addLayer() method to add the image to the interactive map. We will start simple, without using any of the optional parameters:

```
Map.addLayer(srtm);
```

The displayed map will look pretty flat grey, because the default visualization parameters map the full 16-bit range of the data onto the black–white range, but the elevation range is much smaller than that in any particular location. We’ll fix it in a moment.

7. Select the Inspector tab. Then click on a few points on the map to get a feel for the elevation in this area. Finally, set visualization parameters:

```
Map.addLayer(srtm, {min: 0, max: 3000});
```

   Ending script: https://code.earthengine.google.com/346d11d7fdbfd462a32994deefbd1e85

Section 3 - Apply a Computation to an Image

1. Pan over to the Grand Canyon, where there are some dramatic elevation differences.
2. Next add a simple computation, for example a threshold on elevation.

```
var high = srtm.gt(2000);
Map.addLayer(high, {}, 'above 2000m');
```

3. Do another computation to compute slope from the elevation data and display it on the map as a separate layer. Also add a third parameter to the addLayer() method, which names the layer.

```
var slope = ee.Terrain.slope(srtm);
Map.addLayer(slope, {min: 0, max: 60}, 'slope');
```

4. Layers added to the map will have default names like "Layer 1", "Layer 2", etc. To improve the readability, give each layer a human-readable name.

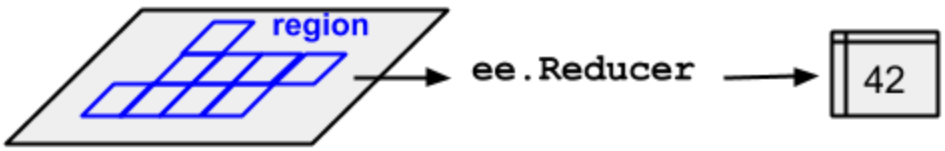
```
var slope = ee.Terrain.slope(srtm);
Map.addLayer(srtm, {min: 0, max: 3000}, 'DEM');
Map.addLayer(slope, {min: 0, max: 60}, 'slope');
```

Ending script: https://code.earthengine.google.com/cb0385ce64d5bf1669f012e689aa91ae

Extra things to try:

- Search for a specific location
- Use the inspector to estimate the value range
- Adjust the visualization parameters
- Look at the documentation tab (i.e. Map.addLayer)
- Try the context-specific help (Ctrl-space)

Section 4 - Apply a Spatial Reducer



1. Select the polygon geometry tool and draw a triangle (or more complex polygon) on the map.
2. Print the mean value for the region.

```
Map.addLayer(srtm, {min: 0, max: 3000}, 'DEM');
```

```
Map.addLayer(geometry);
var dict = srtm.reduceRegion({
  reducer: 'mean',
  geometry: geometry,
  scale: 90
});
print('Mean elevation', dict);
```

3. Clear the workspace by clicking
   Reset -> Clear script

   Ending script: https://code.earthengine.google.com/a0dc428620e5bd73086eeb5fbbed7121

Extra things to do:

- Demonstrate autocomplete for method signature.
- Discuss named parameters.
- Discuss the return type of reduceRegion (Dictionary).
- Try out different reducers: min, max, etc.
- Try out different geometry types: point, line
- Discuss pros and cons of `maxPixels` and `bestEffort`

Section 5 - Load and Filter an Image Collection

1. Delete the **srtm** object from the Import section, by rolling your cursor over the object, then clicking on the trash can icon.
2. Search for "Landsat 8 toa" and add the result to the imports section. Rename the collection "L8".
3. Apply a date filter to the image collection, using the filterDate() method.

```
var filtered = L8.filterDate('2013-04-15', '2013-04-20');
Map.addLayer(filtered);
```

   Ending script: https://code.earthengine.google.com/f5677fa228ae2944a23d4277f5ed41dc

Section 6 - Play with Image Bands

1. With the default visualization parameters, the data looks dark and the colors look wrong. Pick better visualization parameters.

```
var filtered = L8.filterDate('2013-04-15', '2013-04-20');
Map.addLayer(filtered, {min: 0, max: 0.3, bands:['B4', 'B3', 'B2']});
```

2. Copy and paste the second line and modify the bands to create the classic false-color look, with vegetation highlighted in red, and demonstrate giving each layer a human-readable name.

```
var filtered = L8.filterDate('2013-04-15', '2013-04-20');
Map.addLayer(filtered, {min: 0, max :0.3, bands:['B4', 'B3', 'B2']},
'RGB');
Map.addLayer(filtered, {min: 0, max: 0.3, bands:['B5', 'B4', 'B3']}, 'False
Color');
```

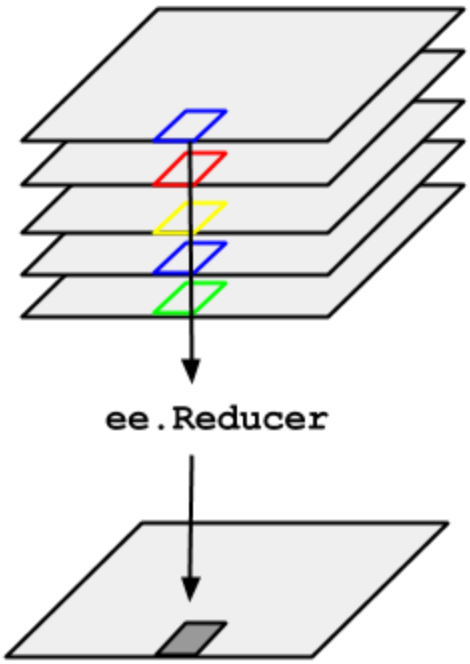
3. We're going to use these visualization parameters a lot, so pull them out into a variable.

```
var rgb_vis = {min: 0, max: 0.3, bands:['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2013-04-15', '2013-04-20');
Map.addLayer(filtered, rgb_vis, 'RGB');
```

4. You can also use the visualization dialog to select appropriate visualization parameters.
    a. Zoom over an area of open ocean (for example, the channel islands off the coast of Los Angeles)
    b. Click on the layer picker, then on the setting icon to the right of the layer opacity slider.
    c. Select a Range Stretch method, and then apply it to the image.

Ending script: https://code.earthengine.google.com/f31a1607692568c097458426a4dbfe43

Section 7 - Reducing Image Collections



1. Expand out the date range.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2014-01-01', '2015-01-01');
Map.addLayer(filtered, rgb_vis, 'RGB');
```

2. Show that you get the same result if you mosaic() the filtered collection

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2014-01-01', '2015-01-01');
Map.addLayer(filtered.mosaic(), rgb_vis, 'RGB');
```

3. Show the results of using the median() reducer

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2014-01-01', '2015-01-01');
Map.addLayer(filtered, rgb_vis, 'RGB');
Map.addLayer(filtered.median(), rgb_vis, 'RGB - median reducer');
```

Ending script: https://code.earthengine.google.com/68b5d9cf1da2cd3dcab268a1cda1b5d7

Section 8 - Isolate an Image

1. Narrow the time range to a 7-10 day window.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2015-09-01', '2015-10-30');
Map.addLayer(filtered, rgb_vis, 'RGB');
```

2. Add a Region of Interest (ROI) to the map, by using the point geometry tool. In the Imports section, rename the object to **"roi"**.
3. Add an additional filter to limit it to images covering the ROI. Note that the resulting filtered object is an image collection which may contain one or more images.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
Map.addLayer(filtered, rgb_vis, 'RGB');
```

4. Extract a sample image from the image collection, and add it as a layer.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
var image = ee.Image(filtered.first());
Map.addLayer(image, rgb_vis, 'RGB');
```

Ending script: https://code.earthengine.google.com/cc07cd2148b16dccd525705048a8a8bf

5. Extra things to do: Sort the filtered collection by `'CLOUD_COVER'`

Section 9 - Compute NDVI

1. Using the .select() method, pick out the NIR and red bands, and do some math the “hard” way by hand.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
var image = ee.Image(filtered.first());
```

```
var red = image.select('B4');
var nir = image.select('B5');
var ndvi = nir.subtract(red).divide(nir.add(red));
Map.addLayer(image, rgb_vis, 'RGB');
Map.addLayer(ndvi, {min: 0, max: 1}, 'NDVI');
```

2. Find the ee.Image.normalizedDifference() method in the docs. Use it to simplify the script.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
var image = ee.Image(filtered.first());
var ndvi = image.normalizedDifference(['B5', 'B4']);
Map.addLayer(image, rgb_vis, 'RGB');
Map.addLayer(ndvi, {min: 0, max: 1}, 'NDVI');
```

Ending script: https://code.earthengine.google.com/b4f6cbe2362beca66a6dd906f92ec551

Section 10 - Write a Function

Our next goal is to calculate NDVI for a collection of images. To do so, we first need to refactor (rewrite) our code to use a function, which can then be applied to all images in a collection.

1. Start by writing a function that adds a band with NDVI data to an image:

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
var image = ee.Image(filtered.first());
var ndvi = addNDVI(image);
Map.addLayer(image, rgb_vis, 'RGB');
Map.addLayer(ndvi, {min: 0, max: 1}, 'NDVI');
```

2. Click on the Inspector tab, then on the image. Look in the Inspector tab results to see that the code has added a band called “nd”.

3. Note that the image has changed, because the first band (B1) is being displayed by default, instead of the "nd" band. Add a visualization parameter to correct this:

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
var image = ee.Image(filtered.first());
var ndvi = addNDVI(image);
Map.addLayer(image, rgb_vis, 'RGB');
Map.addLayer(ndvi, {bands: 'nd', min: 0, max: 1}, 'NDVI');
```

Ending script: https://code.earthengine.google.com/6a560cb635de3fe260b3772f2f02e7f4

Section 11 - Map a Function over a Collection

1. Now that we have a function, we will 'map' the function across the filtered Landsat 8 collection.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-09-01', '2015-10-30')
  .filterBounds(roi);
var with_ndvi = filtered.map(addNDVI);
Map.addLayer(filtered, rgb_vis, 'RGB');
Map.addLayer(with_ndvi, {bands: 'nd', min: 0, max: 1}, 'NDVI');
```

2. Using the inspector tab, click on the map to show that each image now has an ‘nd’ band containing the NDVI.
3. Get rid of the bounds filter, so that all areas in the viewport are used.
4. To get rid of some clouds, expand out the time range and switch to using a median reducer.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
```

```
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-01-01', '2015-10-30');
var with_ndvi = filtered.map(addNDVI);
Map.addLayer(filtered.median(), rgb_vis, 'RGB');
Map.addLayer(with_ndvi.median(), {bands: 'nd', min: 0, max: 1}, 'NDVI');
```

Ending script: https://code.earthengine.google.com/58d5c94fb654846fcf9eac0295695a83

Section 12 - Build a Greenest-Pixel Composite

1. We will now work on an even better way to produce cloud-free mosaics. The ee.ImageCollection.qualityMosaic() method can be used to order image bands, on a pixel-by-pixel basis, by a specified metric (we will use NDVI).

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-01-01', '2015-10-30');
var with_ndvi = filtered.map(addNDVI);
Map.addLayer(filtered.median(), rgb_vis, 'RGB (median)');
var greenest = with_ndvi.qualityMosaic('nd');
Map.addLayer(greenest, rgb_vis, 'RGB (greenest pixel)');
```

2. If your chosen location doesn't make for a great max-NDVI composite, at this point you can fly somewhere like the rainforest belt where that particular technique works well; try Guyana.

   Ending script: https://code.earthengine.google.com/30003261c641d6bf0fa9a917f7143ade

Section 13 - Chart NDVI over Time

1. Click on the ROI point that was added earlier, and then drag it to an agricultural field. Add the following line to make a chart of NDVI over time for your ROI.

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-01-01', '2015-10-30');
var with_ndvi = filtered.map(addNDVI);
var greenest = with_ndvi.qualityMosaic('nd');
Map.addLayer(filtered.median(), rgb_vis, 'RGB (median)');
Map.addLayer(greenest, rgb_vis, 'RGB (greenest pixel)');
print(Chart.image.series(with_ndvi.select('nd'), roi));
```

2. Try out the interactivity of the chart by hovering, expand it to full screen, and testing out the SVG/PNG/CSV download buttons.

   Ending script: https://code.earthengine.google.com/dce9f1f9a0b4f8ce309b8337d1ab6286

Section 14 - Export an RGB Image

The source data that you work with can have many different characteristics (single-band, multispectral, high-bit-depth, etc.), and the visualization tools in Earth Engine allow you to display the data in a variety of ways. You can also export the data in a variety of ways, such as a multi-band image. This section will demonstrate how to export a 3-band (RGB) 8-bit image that can be easily displayed in other tools outside of Earth Engine.

1. Start by creating an 8-bit RGB image object, using the ee.Image.visualize() command, and add it to the interactive map

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-01-01', '2015-10-30');
var with_ndvi = filtered.map(addNDVI);
var greenest = with_ndvi.qualityMosaic('nd');
var rgb = greenest.visualize(rgb_vis);
Map.addLayer(rgb,{}, 'RGB');
```

2. Export the image (by default to your Google Drive account).

```
var rgb_vis = {min: 0, max: 0.3, bands: ['B4', 'B3', 'B2']};
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']);
  return image.addBands(ndvi);
}
var filtered = L8.filterDate('2015-01-01', '2015-10-30');
var with_ndvi = filtered.map(addNDVI);
var greenest = with_ndvi.qualityMosaic('nd');
var rgb = greenest.visualize(rgb_vis);
Map.addLayer(rgb, {}, 'RGB (greenest pixel)');
Export.image(rgb, 'GreenestPixel');
```

3. Look at the Docs tab for the Export object, and check out the other things that you can export. Tables! Videos!

   Ending script: https://code.earthengine.google.com/f6e9d5f20ed15cdf191f8a2907fced59

Section 15 - Some Future Ideas…

1. Read through the Earth Engine API docs. Read through the Introduction and Get Started sections, work your way through the API tutorials, and then dive deeper into any concepts that you are interested in. From the Javascript Playground, the docs can be accessed via the menu **Help -> User guide**.
2. Join and participate in the Google Earth Engine Developers forum, where Earth Engine users post and answer questions about the platform.

Section 16 - Rapid Land Cover Classification

Part 1: Creating a land-cover map for a specific place, and learning the steps. Following the steps below we will create a land-cover map using pre-defined land-cover classes and the classification and regression trees (CART) model.

1. Select country of interest
   We will use a Feature Collection available in GEE, LSIB_SIMPLE that contains country boundary data

```
var countries = ee.FeatureCollection("USDOS/LSIB_SIMPLE/2017");
var country = countries.filter(ee.Filter.eq('country_na', 'Mexico'));
```

Center map to country selected and add boundary layer to the map viewer

```
Map.centerObject(country, 4);
Map.addLayer(country, {color: '001155'}); // Color:  bluish
```

2. Create a L8 annual composite
   Define time frame of interest:

```
var year = 2019; // Year
var startDate = (year)+'-01-01'; // beginning of date filter | month-day
var endDate = (year)+'-12-31'; // end of date filter | month-day
```

Identify source of satellite data to use, in this case Landsat 8 Surface Reflectance Tier 1 product:

```
var l8 = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR');
```

Set up function to mask clouds:

```
function maskL8srClouds(image) {
  // Bits 3 and 5 are cloud shadow and cloud, respectively.
  var cloudShadowBitMask = (1 << 3);
  var cloudsBitMask = (1 << 5);
  // Get the pixel QA band.
  var qa = image.select('pixel_qa');
  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
                 .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  return image.updateMask(mask);
}
```

Build basic composite:

```
var l8compositeMasked = l8.filterBounds(country)
                .filterDate(startDate,endDate)
                .filterMetadata('CLOUD_COVER','less_than',50)
                .map(maskL8srClouds)
                .median()
                .clip(country);
```

Create visualization dictionary for band combination 7-5-4

```
var L8_754_viz = {bands:['B7', 'B5', 'B4'], min:[500,500,500],
max:[2000,4500,1500]}
```

Add annual composite to the map (this may take some time)
-Extract country name to name layer in map viewer

```
var country_name = country.aggregate_first('country_na').getInfo()

Map.addLayer(l8compositeMasked,L8_754_viz,'Landsat 8 composite masked ' +
(country_name));
```

Try to visualize annual composite using other band combination, such as natural color

```
var L8_432_viz = {bands:['B4', 'B3', 'B2'], min:[200,200,200],
max:[2000,2500,2500]};
Map.addLayer(l8compositeMasked, L8_432_viz,'Landsat 8 composite masked Nat
Color');
```

3. Create Training data

We will use the geometry options available in the Map section of GEE

- The classes to use for this exercise are: **1 forest, 2 non-forest, 3 water**. Let's start the process:
    - Zoom in your annual mosaic
    - Select the “Draw rectangle” icon on the top left of the map window
    - Hold cursor on the 'geometry' panel that just appeared and select the gear icon 'Edit layer properties'
    - Change Name field and enter class of interest (Forest)
    - In "Import as', change from 'Geometry' to 'FeatureCollection'

    - Click on "+Property', In Property write 'label', in value write '1'
    - Change color by clicking on a color that better represents that type of land cover, green for Forest
    - Click OK to save changes
    - Navigate through your map and using visual interpretation, your expert knowledge, activating and deactivating layers, draw as many polygons as possible that represent forest
    - On Geometry Imports, select '+new layer' and repeat steps these steps until you have a Feature Collection set up for each land cover type of interest
    - Click on 'exit' in the Rectangle drawing
- Tip, if you miss classified an area, after clicking on exit, you can select polygon of interest and delete it
- Check: At the top of your script you should see the FeatureCollections created

Merging training data:

```
var training = Forest.merge(NoForest)
                     .merge(Water);
```

4. Prepare to train CART model (Classification and regression tree (CART) approach)

   Select bands to use to train the model. We will use the visible, near infrared, and shortwave infrared bands

```
var bands_to_use = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7'];
```

   Sample Landsat pixels using the geometries we created (spatial overlay)

```
var samples = l8compositeMasked.select(bands_to_use).sampleRegions({
collection: training, // Set of geometries created
properties: ['label'], // Label from each geometry
scale: 30 // Make each sample the same size as Landsat pixel
}).randomColumn('random'); // creates a column with random numbers
```

   Inspect size of samples,

```
print('Samples n =', samples.aggregate_count('.all'));
```

   The "samples" feature collection contains attributes about both the land cover class and the spectral values.

5. Train classifier CART

```
var landcover_labels = 'label'

var trained_CART = ee.Classifier.smileCart()
  .train(samples, landcover_labels, bands_to_use);
```

The trained_CART object is a classifier. Its properties can be viewed as:

```
print('CART properties: ' , trained_CART);
```

6. Apply classifier to annual composite image

```
var classified_CART =
l8compositeMasked.select(bands_to_use).classify(trained_CART);
```

Add classified product to map

```
Map.addLayer(classified_CART, {min:1, max:3,
  palette:['25CF1C', // forest
  'FE9D02', // nonforest
  '2E3FAC']}, // water
  'CART Classification'
)
```

Final script: https://code.earthengine.google.com/690c0695ef8cdcb0ab558bcf101b6a51

Part 2: Exercise to create a land cover map for a given year using Landsat 8 data in one ISFL country (Colombia, Ethiopia, Mexico, Indonesia and Zambia) or your country of interest.

Tip: the only things you need to change from the script are 1) country of interest and 2) selection of new training data

Working in a small group repeat the steps above to create your own land cover map and take some screenshots of your results and put them in these slides. Discuss your results and process.

Additional Optional Steps to go further

7. Split training data to have: 1) data to train model (70%) and 2) data to validate model (30%)
8. Assess accuracy of model fit

9. Resources:

a. https://code.earthengine.google.com/a853489c298b44356dac38310a4c9270
b. Lab 5: Supervised classification and regression - Google Docs
c. Rapid Classification of Croplands | Google Earth Engine