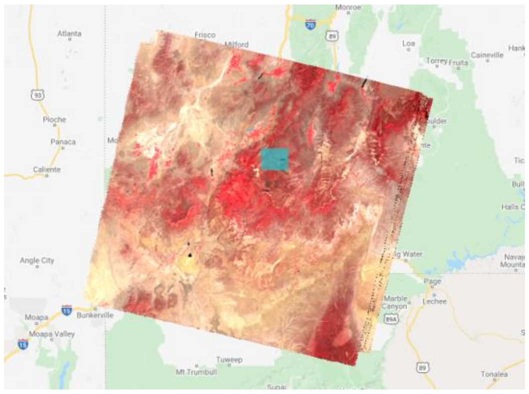
Last Updated: March 2022

EXERCISE 1

Process & explore Landsat time series



Introduction

Advanced change detection analysis techniques typically leverage much more image data than traditional two-date methods. The selection and processing of satellite imagery for change analyses is among the most crucial steps for change detection. Earth Engine is a platform developed by Google that provides access and tools for analyzing large collections of satellite data. As such, Earth Engine vastly improves the speed, ease and efficiency of acquiring, filtering and processing image data for change analysis.

In this exercise, you will first review functions, data types and algorithms in Earth Engine. You will then use a custom library of functions for assembling a cloud and shadow free seasonal Landsat time series (LTS). This is a critical prerequisite for the change detection methods we will be exploring in this course. This exercise will prepare you to assemble the analysis-ready Landsat time series (LTS) containing seasonal image composites.

This exercise introduces the Earth Engine interface and tools. If you're already familiar with Earth Engine, you may prefer to move ahead to Exercise 2.

Objectives

- Prepare a Landsat time series of seasonal composites with clouds and shadows removed
- Use ready-made Earth Engine algorithms as well as shared modules built by other users
- Execute a basic linear trend analysis on a Landsat time series and plot results

Required Data

- None – Everything we need is in Earth Engine!
- You can find a copy of the script created in this exercise in the course repository.

Prerequisites

- **Google Chrome installed on your machine (available in the FS Software Center)**
- **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 knowledge of and experience with Earth Engine as taught in: Geospatial Scripting in JavaScript and Earth Engine**

Table of Contents

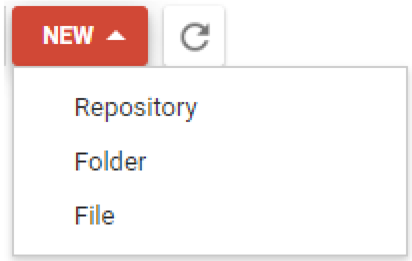
Part 1: Review methods and objects for working with Landsat imagery 4
Part 2: Access a custom module from a shared repo 11
Part 3: Create a cloud and shadow masked LTS 11
Part 4: Review the final script 15

Part 1: Review methods and objects for working with Landsat imagery

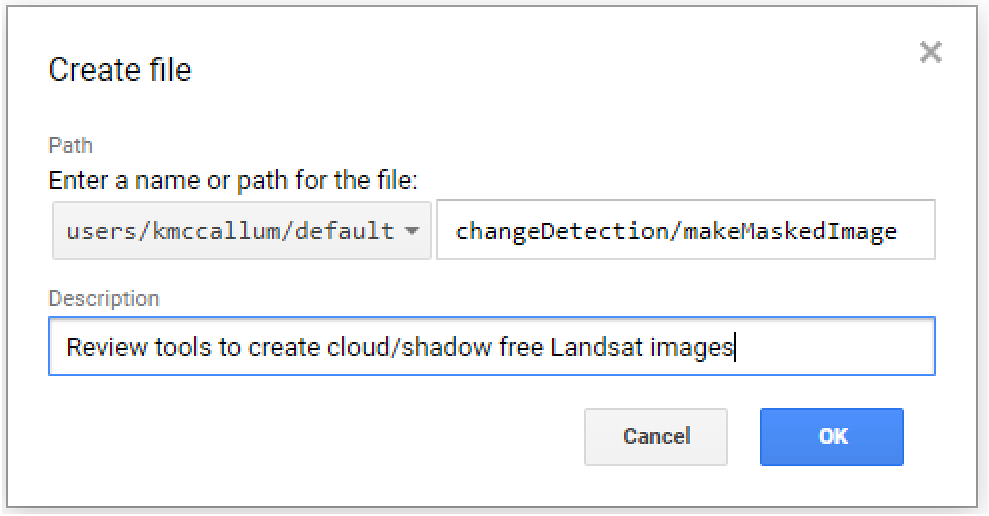
In this section, you will use basic Earth Engine methods to create a collection of Landsat images and use basic cloud masking and compositing methods. If this material is all new to you, you might want to refer back to Geospatial Scripting in JavaScript and Earth Engine.

A. Create a new folder script in the code editor

1. In your browser, navigate to https://code.earthengine.google.com/ and login to your Google account if prompted
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 File to create a folder this course and script for this exercise.



3. In the dialog box, enter the path 'changeDetection/makeMaskedImage' and click OK – This creates a folder called changeDetection and within it, a script called 'makeMaskedImage'



Create file

Path

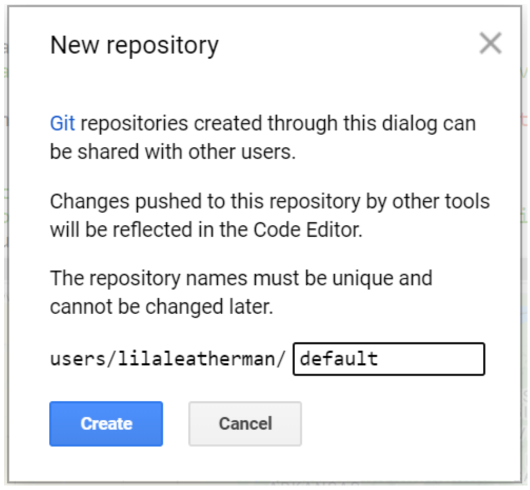
Enter a name or path for the file:

users/kmccallum/default changeDetection/makeMaskedImage

Description

Review tools to create cloud/shadow free Landsat images

   i. After the script has been created, click on the script in your repository to bring it into the code editor.
   ii. If you don't have a "default" repository available, you can create one before completing Step 3. Click New > Repository and enter "default" in the dialog box that appears.


New repository

Git repositories created through this dialog can be shared with other users.

Changes pushed to this repository by other tools will be reflected in the Code Editor.

The repository names must be unique and cannot be changed later.

users/lilaleatherman/ default

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. Import a collection of Landsat 8 images

1. An Image Collection is the Earth Engine name for a suite of imagery data—literally, a collection of images. An Image Collection can be any geography, imagery type, and duration that you specify.
2. Use the data search box at the top of the code editor to search for the Landsat 8 SR (surface reflectance) collection. Note that you can import the imagery here to add it to your script.
3. You can also import imagery using a line of code. Copy the code below and paste it on the first line of your new script.

```
var landsat8Collection = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2");
```

4. This line of code a) creates a new variable named “landsat8Collection” b) calls the function **ee.ImageCollection** and c) specifies the Landsat 8 Level 2 Collection 2 Tier 1 as the source imagery.
   i. Now that you have an object representing an image collection in the global scope of your script, you can start manipulating it to suit your needs.

C. Improve the display by adding scaling factors and changing visualization parameters

1. Now, before we can display the image properly, we need to first apply scaling factors to convert the values to reflectance which ranges from 0 to 1.
2. 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:

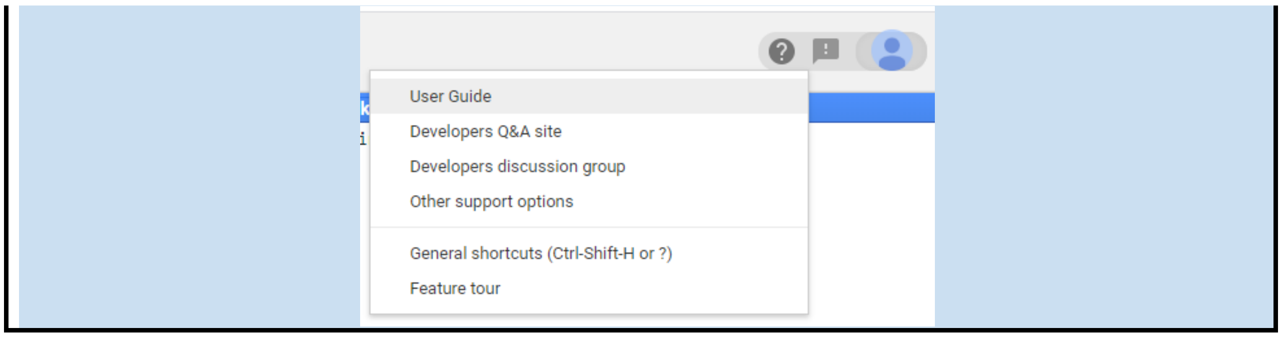
```
function applyScaleFactors(image) {
  var opticalBands = image.select('SR_B.').multiply(0.0000275).add(-0.2);
  var thermalBands = image.select('ST_B.*').multiply(0.00341802).add(149.0);
  return image.addBands(opticalBands, null, true)
              .addBands(thermalBands, null, true);
}

landsat8Collection = landsat8Collection.map(applyScaleFactors);
```

Note: *Every programming language has a set of style conventions that help to improve clarity and readability. It is a good idea to start building clean coding habits from the start. In JavaScript we use camelCase for variable and function names. Simple statements end with a semicolon and spaces are used to organize complex statements and code blocks. Refer to this guide from w3schools for JavaScript conventions.*

D. Review the Earth Engine documentation (docs)

1. In the upper left-hand panel, select the Docs tab. You will see a list of all the Earth Engine methods listed.
   i. This should be your go-to reference point for all the JavaScript methods specific to Earth Engine. This is where you find information on what you can do with each type of object, what arguments are required, which arguments are optional, and what the output will be.
2. Quickly review a few Earth Engine methods
   i. See if you can locate the following methods from the list. What arguments do they take and what do they return?
      (a) **ee.ImageCollection.filterDate()**
      (b) **ee.ImageCollection.filterBounds()**
      (c) **ee.Reducer.median()**
      (d) **ee.Algorithms.Landsat.simpleCloudScore()**

Don't worry about understanding everything in here! This is a complete list of what is possible in Earth Engine and is structured by data type. You should refer to these docs for quick reference, showing you what is possible or as a reminder if you have forgotten how to use a function. For learning, typically you will want to refer to the User Guide for example code snippets with explanations. Use the menu in the upper-right to access the user guide.


User Guide
Developers Q&A site
Developers discussion group
Other support options
General shortcuts (Ctrl-Shift-H or ?)
Feature tour

E. Filter your image collection to a single season

1. Add a date filter to reduce the number of images in your collection.
   i. In your script (below the Imports section), type the code below on line 1 or 2 (depending if you moved the first statement into the imports). Feel free to change the dates to represent any single season you are interested, keeping in mind that Landsat 8 was launched February 11, 2013. To limit the size of your collection, use dates that will allow you to capture just a single growing season.

```
landsat8Collection = landsat8Collection.filterDate('2019-05-01', '2019-10-01');
```

   ii. This **landsat8Collection** variable now references all the Landsat 8 images around the world that fall in this date range. Since we are interested in getting imagery for a study area, let us add another filter to reference only those images in this time period that intersect a point of interest.

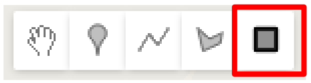
   iii. Add the code below to the script to print the total number of images in this collection to the Console, to the right of the code editor.

```
print('number of images in collection:', landsat8Collection.size());
```

   iv. Click the Run button to run the script and wait patiently for this total to appear in the console. You should see a large number printed to the console. That is a lot of image data!

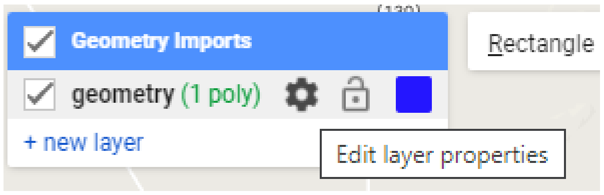
F. Filter the image collection by location

1. Use the map navigation controls (zoom, pan, etc.) to navigate to a location of your choice. Anywhere is fine!
2. Click the 'draw a rectangle' button (highlighted in the graphic below) to open controls to interactively create an EE geometry object. Keep your area of interest to a moderate size to keep processing time low—e.g., your local National Forest boundary or geographic sub-region of interest; not your entire FS region.



3. Draw a rectangle around an area of interest on the map and notice that a new variable is added to the imports section of your script.
   i. By default, this variable is given the name 'geometry' and a default color. Feel free to change the color by clicking on the gear icon in the imports section or by hovering over

the layer in the map and clicking the gear icon to access properties. Don't change the name.



4. To filter **landsat8Collection** by this geometry, add another filter method to your image collection object. The **filterBounds()** method takes a geometry object such as the one you have just created.
   i. Add **.filterBounds(geometry)** to the end of line 1. Line 1 should now look the line below:

```
landsat8Collection = landsat8Collection.filterDate("2019-05-01", "2019-10-
01").filterBounds(geometry);
```

5. With the print statement still in place on line 2, click the Run button again to run the script and note the total in that appears the console. There are far fewer images to work with now!

G. Add the filtered collection to the map and explore

1. First, create a variable containing the visualization parameters for your image collection as an object in brackets **{}** as shown in the code below.
2. Using **Map.addLayer()**, add the entire image collection to the map. Use the code below to pass in the following parameters: an image collection object to render on the map, the visualization parameters variable, and a name for your image collection that will appear in the map pane.
   i. If the syntax below is confusing to you, check out the documentation on visualization parameters.

```
var visualization = {
  bands: ['SR_B4', 'SR_B3', 'SR_B2'],
  min: 0.05,
  max: 0.7,
  gamma: 1.6
};

Map.addLayer(
  landsat8Collection,
  visualization,
  "Landsat8 Collection"
);
```

3. Save your script and click Run.
   i. You should see a stack of images added to the map as the entire image collection has been added to the map. Depending on where you placed your study area, you might see multiple Landsat path rows

H. Use a Landsat algorithm to remove clouds

1. Copy the function below and paste it into line 1 of your script. We paste this into line 1 so that we create a function to have access to for the remainder of our script.

```
function maskClouds(image) {
  var cloudsBitMask = (1 << 3);
  var cloudShadowBitMask = (1 << 4);
  var qa = image.select('QA_PIXEL');
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
                 .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  return image.updateMask(mask);
```

***Note**: The content above is one method to mask clouds. There are many cloud masking methods available. This one will work well for this task. It wraps the cloud masking routine in a function that creates a mask of selected the cloud and cloud shadow bit information from the Landsat 8 QA_pixel band and applies it to the input image. Best practice is to include the functions in your script at the top.*

2. Use the map method for image collections to apply this function to every image in the collection. At the bottom of your script, create a new variable called **maskedL8Collection** for the cloud masked collection.

```
var maskedL8Collection = landsat8Collection.map(maskClouds);
```

3. Adapt the code provided above for **Map.addLayer()** to add this new collection to the map and compare it to the unmasked imagery.
   i. Hint, you can copy/paste the **Map.addLayer** code block and just change the 'landsat8Collection' to 'maskedL8Collection' and change the last string parameter in the same way.

I. Reduce the collection to a single seasonal composite image

1. For the final step of our review, use the **ee.ImageCollection.median()** reducer to create a single composite image from your cloud masked collection. Add the following line to the bottom of your script:

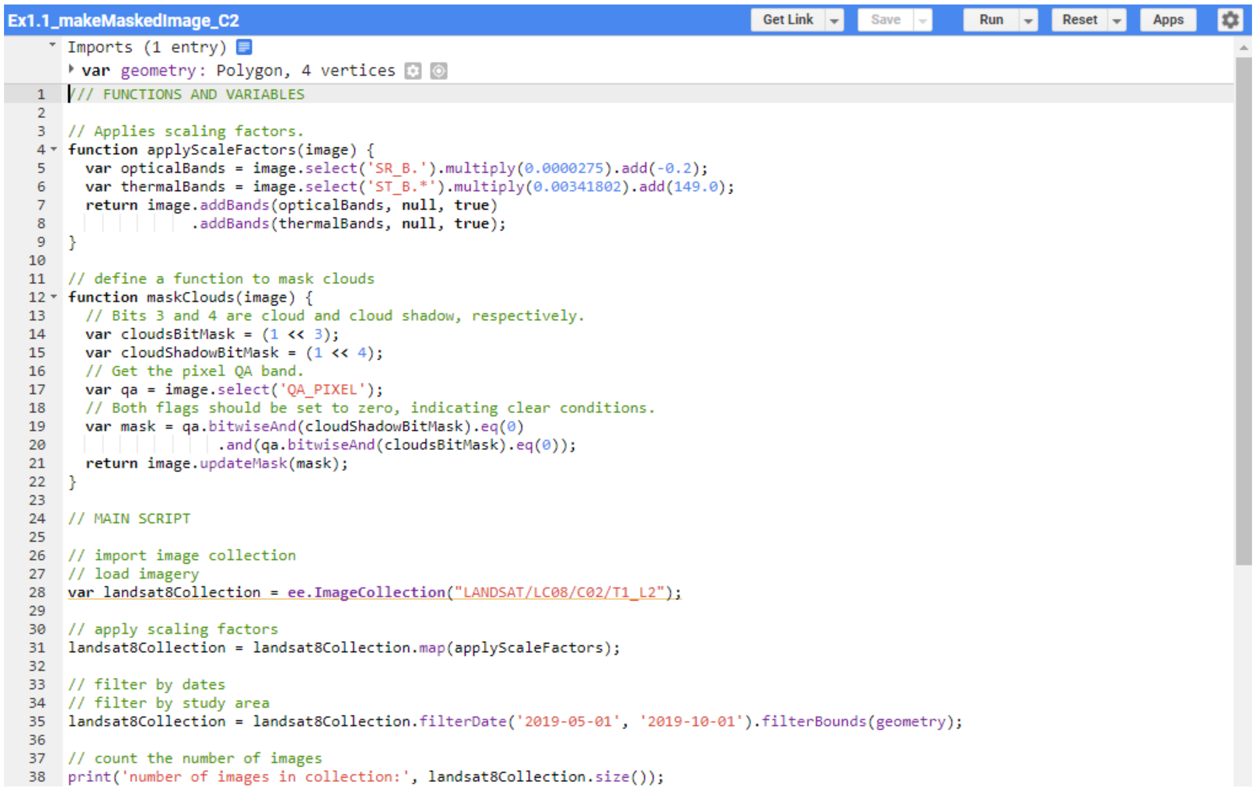
```
var l8Median = maskedL8Collection.median();
```

2. Add the new image object to the map in the same way and compare results to the other items on the map.
   i. You should see a clear, median composite image. Depending on the cloudiness of your chosen area, you might still be able to find some cloud or shadow artifacts if you inspect closely.

***Note:** Obtaining really good quality seasonal image composites takes some expertise though it is easily achievable in EE. We just created a single season median composite from Landsat 8 imagery in just a few lines of code. This was accomplished using 'out-of-the-box' Earth Engine functionality. Scaling this method up to create an entire annual Landsat time series instead of a single seasonal composite would simply require a bit more JavaScript and logic. Additionally, we would likely want to improve our cloud and shadow masking techniques and use imagery from additional sensors where available. This can get* *complex… Luckily, we don't need to be experts in image processing, and we can leverage the hard work of other gracious Earth Engine developers who have shared custom libraries for this kind of thing.*

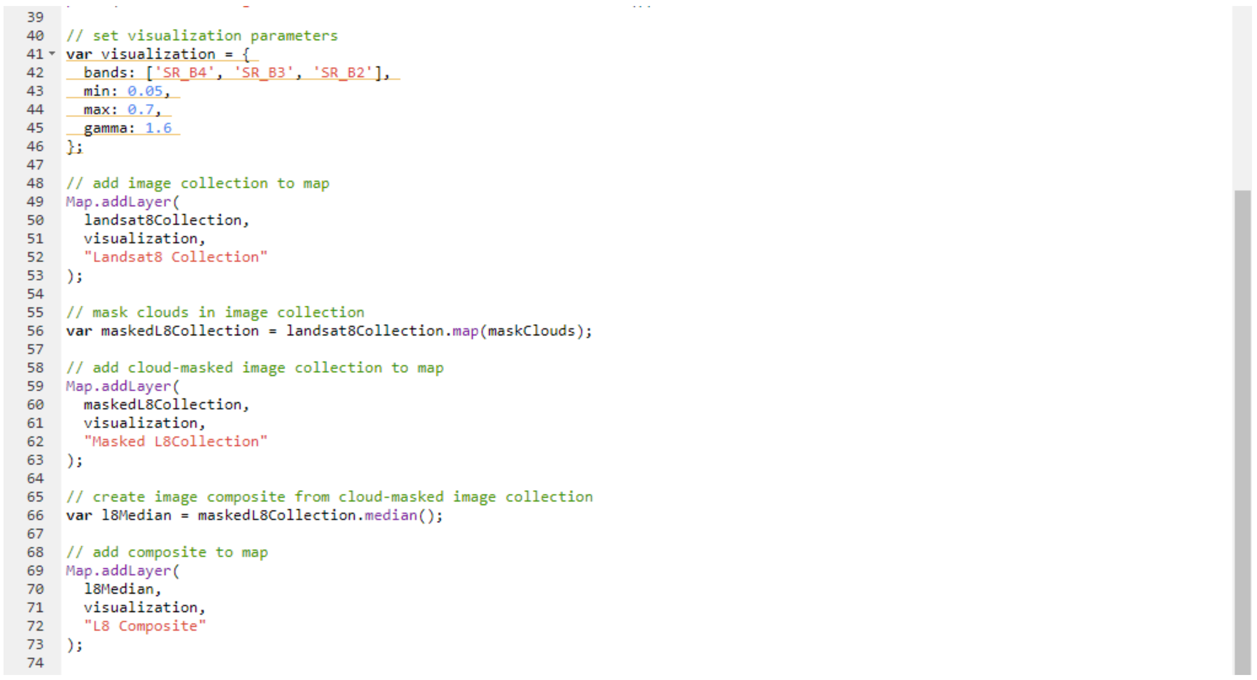
J. Review, tidy and save your script

1. Review the script you have just created, remove redundant or unnecessary code and make sure you understand everything.
2. Go through line-by-line and add comments to anything that is unclear or was new to you.
3. Make sure that you understand what each line of code is doing and see if you can describe each object in the script.
   i. What is the datatype for each variable or object?
   ii. When functions and methods are called, what arguments do they each take? What objects do they return?
4. Your final script may resemble the image below. Comments appear in green below and begin with (at least) two forward slashes **//** .



```
Imports (1 entry)
  var geometry: Polygon, 4 vertices
```

```javascript
/// FUNCTIONS AND VARIABLES

// 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);
}

// define a function to mask clouds
function maskClouds(image) {
  // Bits 3 and 4 are cloud and cloud shadow, respectively.
  var cloudsBitMask = (1 << 3);
  var cloudShadowBitMask = (1 << 4);
  // Get the pixel QA band.
  var qa = image.select('QA_PIXEL');
  // 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);
}

// MAIN SCRIPT

// import image collection
// load imagery
var landsat8Collection = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2");

// apply scaling factors
landsat8Collection = landsat8Collection.map(applyScaleFactors);

// filter by dates
// filter by study area
landsat8Collection = landsat8Collection.filterDate('2019-05-01', '2019-10-01').filterBounds(geometry);

// count the number of images
print('number of images in collection:', landsat8Collection.size());
```



```
// set visualization parameters
var visualization = {
  bands: ['SR_B4', 'SR_B3', 'SR_B2'],
  min: 0.05,
  max: 0.7,
  gamma: 1.6
};

// add image collection to map
Map.addLayer(
  landsat8Collection,
  visualization,
  "Landsat8 Collection"
);

// mask clouds in image collection
var maskedL8Collection = landsat8Collection.map(maskClouds);

// add cloud-masked image collection to map
Map.addLayer(
  maskedL8Collection,
  visualization,
  "Masked L8Collection"
);

// create image composite from cloud-masked image collection
var l8Median = maskedL8Collection.median();

// add composite to map
Map.addLayer(
  l8Median,
  visualization,
  "L8 Composite"
);
```

Part 2: Access a custom module from a shared repo

A. Look at EE docs on creating script modules

1. Open the Earth Engine developers' website by clicking on the question mark icon in the upper right-hand corner of the code editor and selecting User Guide.
2. On the top menu click on Guides then find the **Development Environments** section and open the article titled **Earth Engine Code Editor**.
3. Scroll down to locate the section title **Script modules** and review the rationale and syntax for creating modules. Note how a module can be loaded from repositories that you have access to using the **require** keyword.

B. Access a public repository built by the research group behind LandTrendr

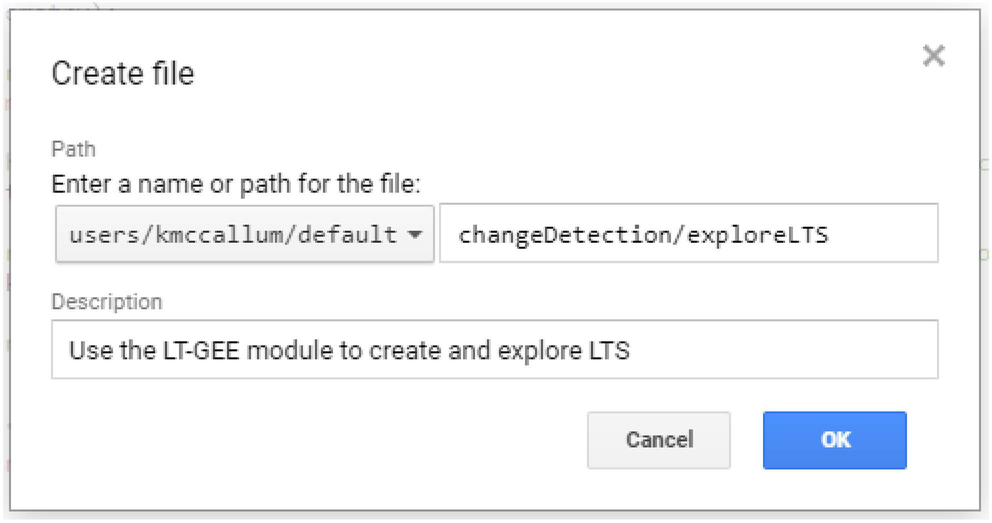
1. To get read access to the eMapR repository containing the module we want (see below), visit the link below and accept the repository. This should add the *users/emaprlab/public* repository to your EE account.
   i. https://code.earthengine.google.com/?accept_repo=users/emaprlab/public

Note: *The Environmental Monitoring, Analysis and Process Recognition (eMapR) lab at Oregon State University shares remote sensing tools and training materials through their public LandTrendr Google Earth Engine (LT-GEE) GitHub repository. We will be using this module and EMAPR tools extensively in this course, and we recommend additional exploration of the eMapR LT-GEE guide.*

Part 3: Create a cloud and shadow masked LTS

A. Create a new script

1. Click the New button and select 'File' to create a new file.
2. In the **Create file** dialogue, specify the path and the file name to create a new script in your changeDetection course folder (see below).
3. Name this file '**exploreLTS**'



Create file

Path

Enter a name or path for the file: users/kmccallum/default changeDetection/exploreLTS

Description: Use the LT-GEE module to create and explore LTS

B. Create a geometry object representing a study area of your choice

1. Just as you did in Part 1, step E above, use the polygon drawing tool to create and import a geometry object with the default name 'geometry'
2. Feel free to create a polygon anywhere you like but this is a great opportunity to start exploring data and landscape trends in an area of personal interest to you.

C. Require the Landtrendr-GEE module

1. At the top of your script (after the EE imports) add the code to require this module.

```
var ltgee = require("users/emaprlab/public:Modules/LandTrendr.js");
```

D. Review documentation for the LTS building function

1. Visit the link below and review the description and parameters of the function named buildSRcollection
   i. https://emapr.github.io/LT-GEE/api.html#buildSRcollection
2. Note the datatypes for each parameter and which parameters have default values.

E. Create a Landsat time series using the buildSRcollection function

1. Create a new cloud, shadow, and snow masked image collection using this function. Pass the parameter values in directly using the correct types (integer, string, geometry and array).
   i. Add the code below to your script just below your **require** statement. Feel free to change the dates if you like.

```
var annualSRcollection = ltgee.buildSRcollection(
  1985,
  2019,
  "06-20",
  "09-20",
  geometry,
  ["cloud", "shadow", "snow"]
);
```

Note the following in the function call above:

*1. Just about everything in JavaScript is an object and we use dot notation to access methods or properties of objects. In this case, we are using this notation to access the **buildSRcollection** method defined in the **ltgee** object.*

*2. As you saw in the documentation, the parameter **maskThese** is optional and if omitted has the default value of the array containing strings: 'cloud', 'shadow', 'snow'. We have explicitly written this out for clarity, but for fun you could try leaving it out or removing one or more of the string values from the array and seeing what you get later when you add this to the map.*

F. Explore the masked LTS

1. Add a print statement to explore the contents of the new collection. Use the code below.

```
print(annualSRcollection);
```

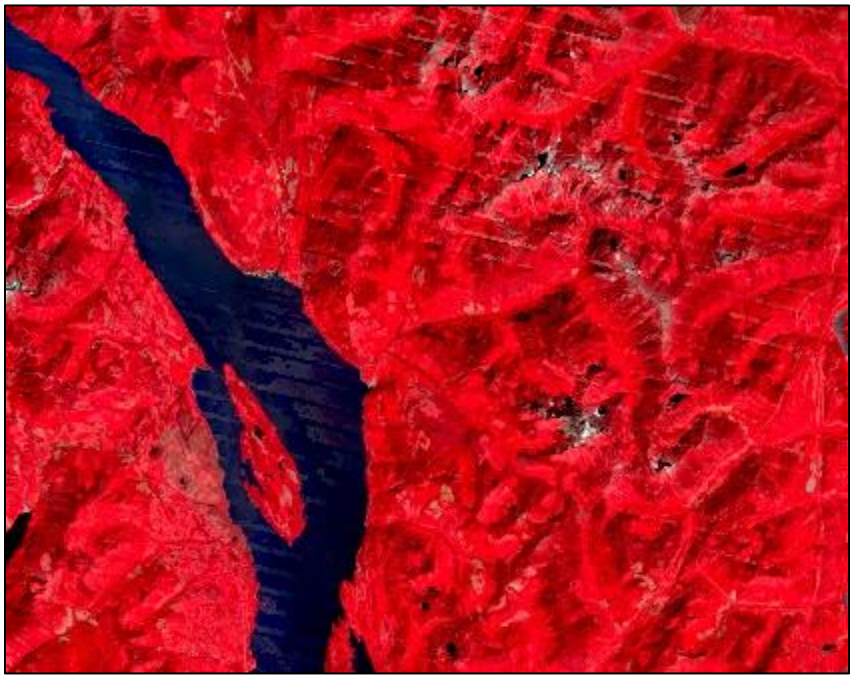
2. Click the run button to run the script.
3. Explore the ImageCollection object that is printed to the console.
   i. Note the number of images (35 elements) corresponding to the specified years in the time series
   ii. Expand the ImageCollection by clicking on the small black triangle and observe the list of image features. Note that they all have 6 bands.
   iii. Open up the object to look at a single band in a single image. Do this by expanding the feature list, then the 0th (first) image in the list, then the band list and finally the **data_type** property. You will see that the values are now signed 16bit and the surface reflectance values are now represented by large integer values.

Note that in order to assemble annual composites for the date range provided, the function combines imagery from Landsat 5, 7 and 8. Click here for more information about the history of Landsat missions. Also, this function does some additional image processing and reordering of bands in order to combine images from these three Landsat missions. If you are curious and want more information than is shared in the documentation, you can always review the code inside the LandTrendr.js module!

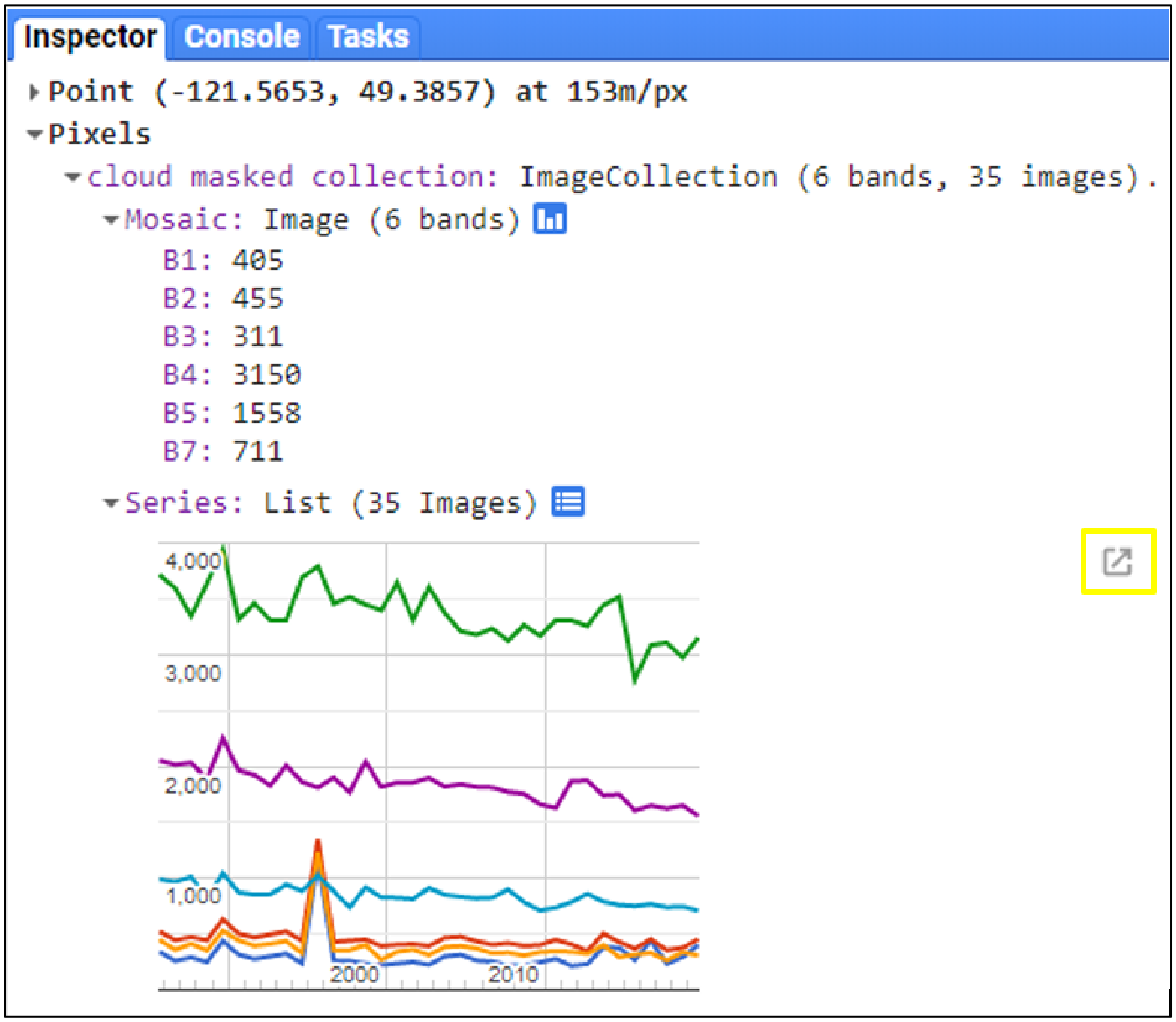
4. Use the **Map.addLayer** method to add the *entire* collection to the map.
   i. Add the code below and note the new min and max value provided in the visualization parameters object.
   ii. Recall that by default, adding the whole collection will display the last image on top.

```
Map.addLayer(
  annualSRcollection,
  { bands: ["B4", "B3", "B2"], min: 250, max: 3000, gamma: 1.6 },
  "cloud/shadow/snow masked SR collection"
);
```

5. Use the map controls and to visually inspect the image.
   i. Pick an area of interest and zoom in.
   ii. Pan around the image and see if you can find any artifacts from the image processing. If you can't find any artifacts, that is excellent! Depending on what area you chose, it can sometimes be difficult to get enough cloud free observations and you can end up with some unavoidable artifacts in the composite. The example below shows Landsat 7 scanline error artifacts in a particularly cloudy area in the north Cascades mountains near Vancouver, British Columbia.



6. Use the inspector to view pixel values through time
   i. Click to activate the **Inspector** tab in the upper right.
   ii. Click anywhere on the map and observe the data that is printed to the console (resembling the output below).



```
▸Point (-121.5653, 49.3857) at 153m/px
▾Pixels
  ▾cloud masked collection: ImageCollection (6 bands, 35 images).
    ▾Mosaic: Image (6 bands)
       B1: 405
       B2: 455
       B3: 311
       B4: 3150
       B5: 1558
       B7: 711
    ▾Series: List (35 Images)
```

7. Click on the small gray icon (highlighted in yellow in the screenshot above) to open this graph in another tab.
   i. This graph shows the time series values for the selected pixel across the entire image collection. This graph shows the selected pixel's spectral trajectory through time which is the basis for all the disturbance mapping algorithms that we will explore in this course.

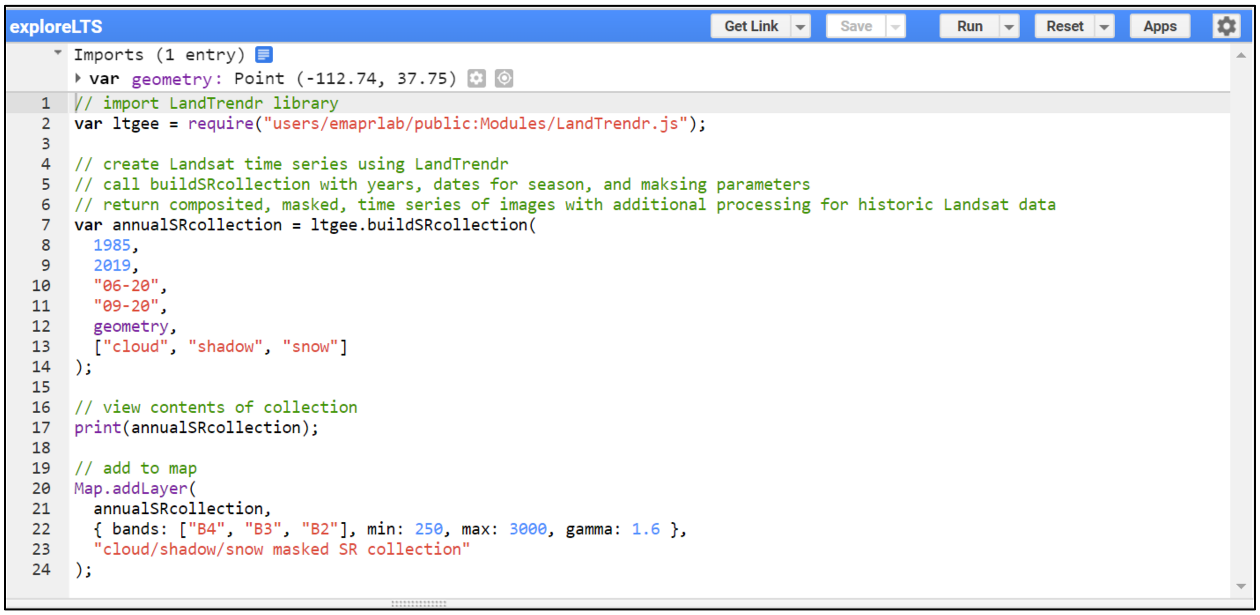
Part 4: Review the final script

Before we move on to the next exercise, take a moment to read through the script that you have written.

A. Review script and add comments

1. Go through line-by-line and add comments to anything that is unclear or was new to you.

2. Make sure that you understand what each line of code is doing and see if you can describe each object in the script.
   i. What is the datatype for each variable or object?
   ii. When functions and methods are called, what do they each take and return?
3. Your final script may resemble the screenshot below.
   i. You can compare your script to the final script for this exercise in the course repository.



```
Imports (1 entry)
  var geometry: Point (-112.74, 37.75)
// import LandTrendr library
var ltgee = require("users/emaprlab/public:Modules/LandTrendr.js");

// create Landsat time series using LandTrendr
// call buildSRcollection with years, dates for season, and maksing parameters
// return composited, masked, time series of images with additional processing for historic Landsat data
var annualSRcollection = ltgee.buildSRcollection(
  1985,
  2019,
  "06-20",
  "09-20",
  geometry,
  ["cloud", "shadow", "snow"]
);

// view contents of collection
print(annualSRcollection);

// add to map
Map.addLayer(
  annualSRcollection,
  { bands: ["B4", "B3", "B2"], min: 250, max: 3000, gamma: 1.6 },
  "cloud/shadow/snow masked SR collection"
);
```

Congratulations, you have successfully completed the exercise reviewing working with image collections in Earth Engine and learned how to use custom modules to generate a cloud, shadow, and snow masked collection of seasonal composites. In subsequent exercises, you will learn how to use, analyze and map change in Landsat Time Series collections like the one you have just created.