

# **LABORATÓRIOS DIDÁTICOS DE GEOPROCESSAMENTO**

## **Sensoriamento Remoto:**

Introdução ao processamento digital de imagens,  
resoluções, metadados e busca por imagens no GEE

**Docente:** Prof.<sup>a</sup> Dr.<sup>a</sup> Mariana Abrantes Giannotti

**Elaboração do roteiro:** Leonardo Alves Godoy

**Revisão do roteiro:** Giovanna Calil

<b>1 Introdução</b>	<b>2</b>
<b>2 Referências teóricas</b>	<b>2</b>
2.1 O GEE, arquitetura <i>Big Data</i> e padrões de computação paralela	2
2.2 Tipos de dados do GEE	5
<b>3 Roteiro prático</b>	<b>5</b>
3.1 Apresentação do ambiente GEE	5
3.2 Cadastro	5
3.3 Interface do GEE	7
3.4 Dicas de Programação em JS	10
3.5 Criando o primeiro <i>Script</i> no GEE	11
3.6 Explorando os dados do GEE	14
3.6.1 Localização da área de estudo	14
3.6.2 Criando uma feature no GEE	15
3.6.3 Buscando uma coleção de imagens	17
3.7 Trabalhando com imagens de satélite no GEE	18
3.7.1 Adicionando uma imagem ao mapa	18
3.7.2 Recortando uma parte de uma imagem	20
3.7.3 Calculando o valor médio dos <i>pixels</i> de uma banda	21
3.8 Indo além	22
3.8.1 Adicionando outra coleção de imagens	22
3.8.2 Aplicando filtros à coleção de imagens	22
<b>4 Conclusões</b>	<b>22</b>
<b>5 Referências bibliográficas</b>	<b>23</b>

## Objetivos de aprendizagem

- Introdução e cadastro no GEE.
  - Selecionar e visualizar uma imagem *Landsat*.
  - Recortar a imagem a partir de um retângulo envolvente.
  - Explorar a troca da imagem *Landsat* indicada por outra imagem.
  - Trabalhar com Metadados das imagens de uma coleção no GEE.
  - Seleção de imagens em uma coleção utilizando informações dos Metadados.
- Manipulando datas das imagens.
  - Bandas, projeção cartográfica e resoluções no GEE.
  - Criar funções no GEE.
  - Extrair valores com o `reduce`
  - Exemplos de cálculos no GEE.

## 1 Introdução

O Sensoriamento Remoto por si só, provê dados brutos (*raw data*). Esses dados precisam ser processados para ter significado. Nesse ponto alguns passos são necessários, tais como: correção radiométrica, correção geométrica, correção atmosférica. Além disso técnicas apropriadas devem ser aplicadas para tornar esses dados em imagens propriamente ditas, onde interpretação e apresentação visuais podem ser efetuadas. Quando a quantidade de dados utilizada se torna inviável para o trabalho em um computador pessoal, uma alternativa é utilizar uma plataforma de *Cloud Computing* com alta capacidade de armazenamento e processamento. Nesse sentido a Google criou uma plataforma específica para imagens de satélite, chamada *Google Earth Engine*, ou simplesmente GEE. Esse laboratório didático apresentará GEE e mostrará como explorar e manipular coleções de imagens, com foco em suas informações básicas, como as resoluções espacial, radiométrica, temporal e espectral. Atenção especial é dada aos metadados das imagens e como trabalhar com os mesmos. Também será apresentada a forma de se criar e utilizar funções no GEE, além do uso de cálculos básicos com seus objetos.

## 2 Referências teóricas

### 2.1 O GEE, arquitetura *Big Data* e padrões de computação paralela

O GEE (*Google Earth Engine*) é uma plataforma para disponibilização, processamento, análise e visualização de dados de satélites. O volume de dados disponível é enorme, de forma que o GEE provê uma arquitetura apropriada para lidar com essa quantidade de dados, baseada nos modelos de processamento paralelo apropriados para dados do tipo *Big Data*. Essa arquitetura é baseada em *Cloud Computing* (a famosa computação em nuvem), um tipo de sistema que fornece recursos computacionais sem a necessidade da presença física do *hardware* junto ao usuário.

O termo *Big Data* pode se referir tanto aos dados em si, quanto à forma como se trata conjuntos de dados que possuem grande volume e são, muitas vezes, complexos para que possam ser processados pelos meios tradicionais. O conceito de *Big Data* pode ser definido através de três palavras, também conhecidas como os três “v’s”, que são variedade (*variety*), velocidade (*velocity*) e *volume*. Variedade se refere à pouca estrutura definida dos dados, a velocidade se relaciona com a alta taxa de transferência dos dados e volume se refere ao espaço de armazenamento utilizado pelos dados, que é de grande magnitude (MILOSLAVSKAYA; TOLSTOY, 2016). Uma das formas de otimizar o processamento desses dados é através do uso de processamento distribuído.

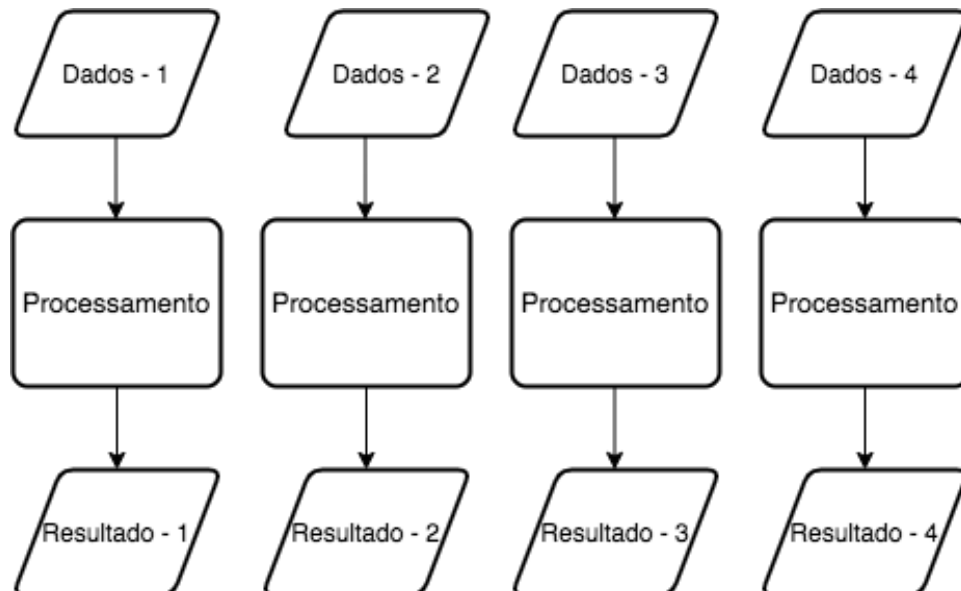
Nesse caso, um modelo de computação paralela pode ser utilizado para processar *Big Data* em *clusters* de computadores. Um *cluster* de computadores, é um conjunto de computadores trabalhando de forma coordenada em trabalhos distribuídos de uma mesma fonte. No caso da *Cloud Computing*, esse *Cluster* de computadores não necessita de contato direto entre o usuário final e as máquinas.

Dois padrões de computação paralela são importantes no GEE e para o *Big Data* em geral, o *Map* e o *Reduce*.

O padrão Map, é um procedimento que aplica uma operação nos dados, em paralelo, resolvendo assim tarefas que podem ter sua execução feita em subtarefas

- independentes e sem comunicação entre si - sobre subconjuntos dos dados originais - Figura 1. Dessa forma, o Map é capaz de executar uma função sobre os diversos elementos de um conjunto paralelamente.

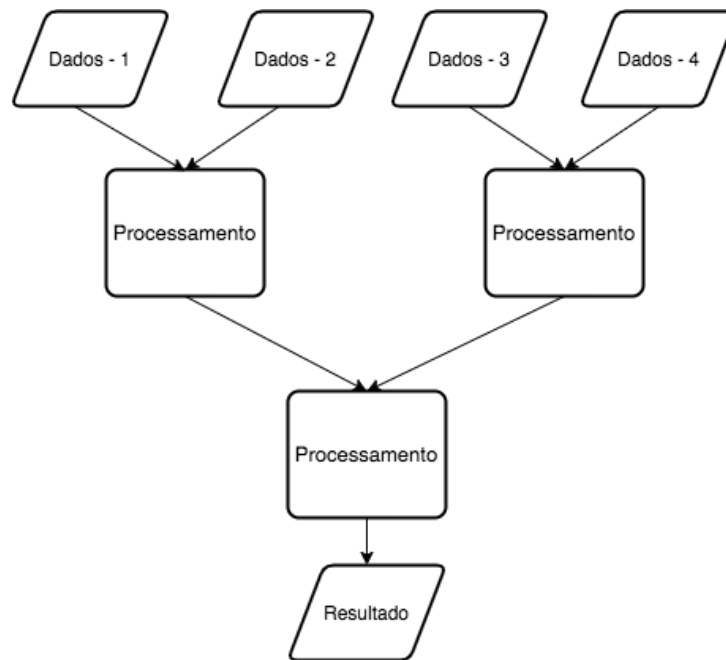
Figura 1 - Diagrama do padrão Map



FONTE: adaptado de GRIEBLER (2011, p. 31)

O padrão *Reduction* - também referido como *Reduce* - é um procedimento que executa operações de sumarização de dados, gerando uma saída única. Conceituando de forma direta, ele reduz o conjunto de dados a um único elemento com valor derivado dos dados de entrada - Figura 2.

Figura 2 - Diagrama do padrão Reduce



FONTE: adaptado de GRIEBLER (2011, p. 32)

Finalmente, o modelo *MapReduce* é uma combinação apropriada dos dois padrões discutidos acima, com o resultado de uma operação em *Map* sendo a entrada para uma em *Reduce*.

## 2.2 Tipos de dados do GEE

Há basicamente quatro tipos de dados no GEE. As *images*, que são os dados do tipo *Raster*, são capazes de armazenar informações numéricas numa forma matricial, basicamente como são as imagens de um satélite. As *image collections*, são *collections* (coleções ou grupos) de objetos do tipo *image*. Também existem os dados do tipo *feature*, que contém geometrias, ou dados do tipo vetorial que utilizam pares de coordenadas como referência, e que também podem carregar atributos associados a eles. Cabe ressaltar que no GEE, uma *feature* pode possuir diversos tipos de geometria dentro dela, como a mistura de pontos e polígonos em um mesmo objeto. O último tipo de dado fundamental do GEE é *feature collection*, que nada mais é do que um grupo de *features*. Mais informações sobre os tipos de dados utilizados no GEE serão vistas no decorrer do curso e também podem ser encontradas em GORELICK et al. (2017).

### 2.3 Dados do tipo *Raster*

Os dados do tipo *raster* são representados computacionalmente como uma *array* de células, onde cada célula armazena valores (atributos) e representam um *pixel* do arquivo *raster* (LONGLEY, 2015, cap. 7). Estes valores podem variar de acordo com a aplicação, podendo ser um binário (0 ou 1) ou um *float-point* (real), por exemplo. Para um computador, cada atributo em uma célula é apenas uma sequência de *bits* de tamanho determinado.

Um arquivo que armazena dados do tipo *raster* pode possuir mais de um atributo armazenado em cada célula (LONGLEY, 2015, cap. 7). Na prática, pense que há uma sobreposição de matrizes, onde os pixels estão distribuídos na forma de uma grade, e cada posição recebe um dos atributos relativos do *array*. Cada uma dessas matrizes é conhecida como uma banda. Alguns dados do tipo *raster* possuem uma única banda, enquanto outros possuem múltiplas bandas. Normalmente, os dados coletados pelos sensores de um satélite, que são distribuídos como um tipo *raster*, possuem mais de uma banda, onde cada banda armazena os dados referentes a um comprimento de onda do espectro eletromagnético. Nesse caso a coincidência das células (*pixels*) tem relação com elas ocuparem o mesmo local na superfície do planeta, mas com o valor medido em uma faixa do espectro diferente. O tipo do valor numérico encontrado em cada *pixel* vai depender de como já foi processado esse dado até a sua distribuição.

### 2.4 Metadados

Os arquivos *raster* podem ser distribuídos com um *header* (cabeçalho) que contém informações sobre o que está armazenado no arquivo, sendo essas informações conhecidas como o *metadata* (metadados ou metainformação) do arquivo - costumeiramente chamadas de “dados sobre dados” (LONGLEY, 2015, cap. 7). Os detalhes armazenados nos metadados, são necessários para a manipulação correta do arquivo. Um exemplo de atributo de metadados de um conjunto de dados de um satélite é a inclinação do sol, que pode ser utilizada para a correção radiométrica dos dados coletados. Outra utilidade dos metadados é facilitar a busca por dados com determinadas características. A descrição dos metadados

pode ser encontrada nos manuais distribuídos juntamente com os dados do satélite, ou mesmo na *internet*.

### 3 Roteiro prático

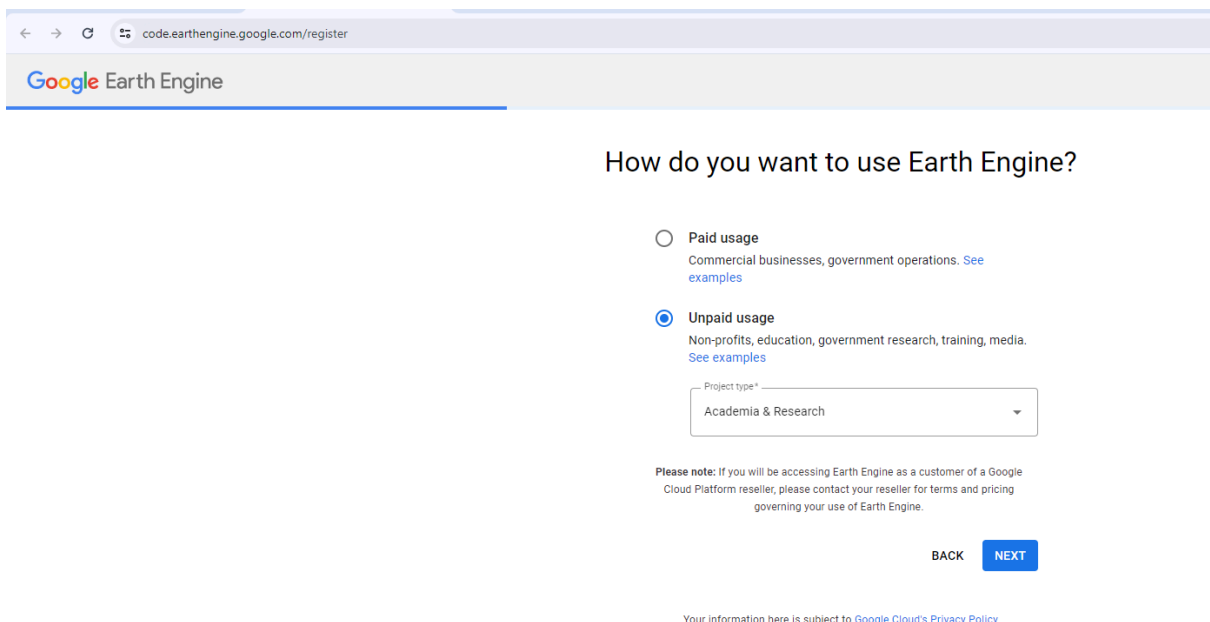
#### 3.1 Apresentação do ambiente GEE

A interface do GEE pode ser acessada no seguinte *link*: <https://code.earthengine.google.com/> (testado em 25 de maio de 2021).

Em caso de dificuldades com o *link*, uma alternativa é buscar pelas palavras-chave “editor google earth engine” no próprio buscador do *Google*.

#### 3.2 Cadastro

Faça o login em conta do gmail que não é da USP, depois acesse a página <https://code.earthengine.google.com/>. Siga as instruções selecionando o uso **Unpaid**, a opção **Academia & Research** (figura a seguir).



← → ↻ code.earthengine.google.com/register

Google Earth Engine

### How do you want to use Earth Engine?

☐ Paid usage  
Commercial businesses, government operations. [See examples](#)

☒ Unpaid usage  
Non-profits, education, government research, training, media. [See examples](#)

Project type\*  
Academia & Research ▼

**Please note:** If you will be accessing Earth Engine as a customer of a Google Cloud Platform reseller, please contact your reseller for terms and pricing governing your use of Earth Engine.

BACK NEXT

Your information here is subject to [Google Cloud's Privacy Policy](#)



Opte por criar um novo projeto, selecione **No organization** e coloque um Projeto ID diferente do Project name. O sistema deve apontar para você aceitar os termos da Cloud Service: clique em **Cloud Terms of Service** para aceitar. E ao final selecione **Continue to Summary** e você deve acessar o Google Earth Engine.

## Create or choose a Cloud Project to register

Create a new project in Google Cloud, or choose one you are authorized to access to enable the API:

☒ Create a new Google Cloud Project

Organization

Project-ID\*

Choose a unique ID. This cannot be changed later.


Project Name (optional)

Choose a name to help you identify the Cloud Project.


☐ Choose an existing Google Cloud Project

BACK

CONTINUE TO SUMMARY

 You must accept the [Cloud Terms of Service](#) before a Cloud Project can be created.

## Confirm your Cloud Project information

<b>Project usage</b> Academia & Research	
<b>Project info</b> ee-geo-luana ee-geo-luana	

[BACK](#)[CONFIRM](#)

Project information cannot be changed later

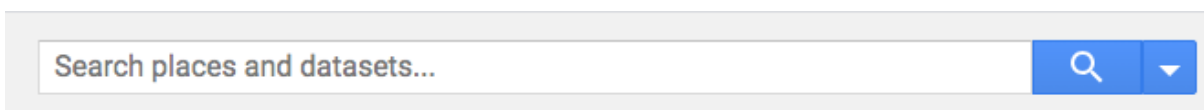
## Welcome to Earth Engine!



### 3.3 Interface do GEE

Na parte superior da interface do GEE há uma caixa para busca de locais (para posicionar o mapa) e conjuntos de dados diversos (Figura 8).

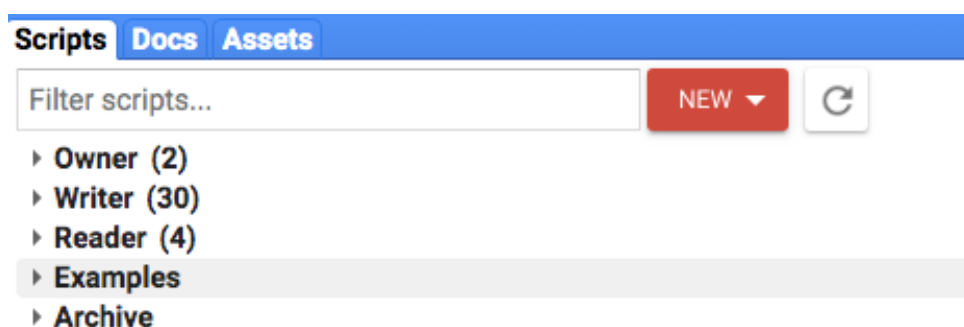
Figura 8 - Caixa de busca do GEE



Do lado esquerdo um painel contendo três abas pode ser vista, como na Figura 9, a aba **“Scripts”** contém os arquivos com *scripts* onde os códigos escritos

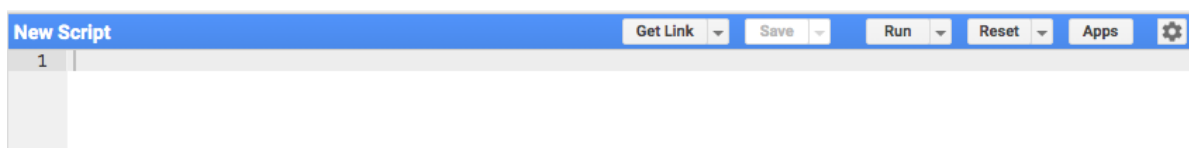
são salvos, além dos repositórios e pastas onde eles podem ser organizados. Já a aba “**Docs**”, contém a documentação, é uma aba muito importante para o usuário, pois ele pode buscar a descrição dos métodos utilizados pelo GEE através de palavras associadas. Finalmente a aba “**Assets**” contém dados que são enviados ou salvos pelo GEE, como arquivos vetoriais, que podem ser utilizados nos *scripts* diretamente.

Figura 09 - Arquivos do usuário e documentação do GEE



Na Figura 10, está o setor central da interface do GEE e onde os códigos dos *scripts* são escritos. Apenas um *script* pode ser editado por vez, quando os dados são carregados (importados no jargão do GEE), e esse setor é dividido em duas partes, sendo a superior referente aos dados carregados com seus respectivos nomes de variáveis, são os *importsets*.

Figura 10 - Editor de *scripts* do GEE

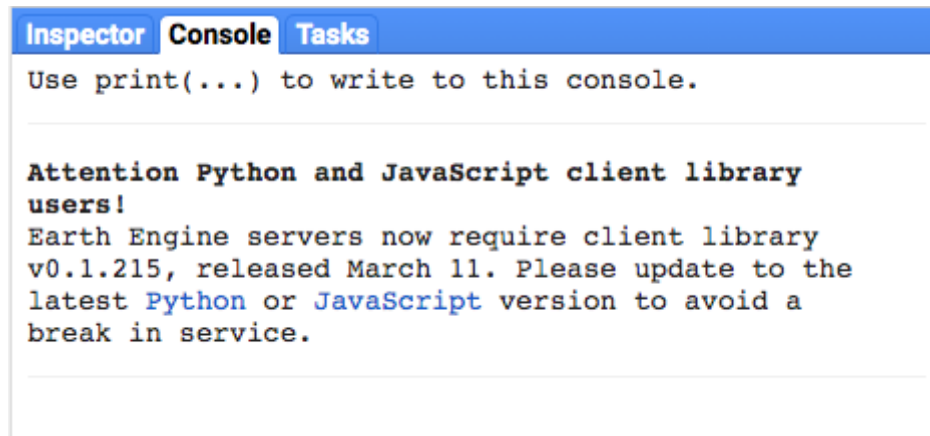


A diferença para uma atribuição de variáveis no *script* é que eles são carregados automaticamente quando se abre o arquivo e não durante a execução do *script*. O botão “**Run**” executa o *script* atual e em “**Get Link**”, é possível criar *links* para compartilhar o seu código com terceiros.

Do lado direito, Figura 11, está um painel com mais três abas. A aba “**Inspector**” exhibe detalhes de variáveis importadas, enquanto a aba Console

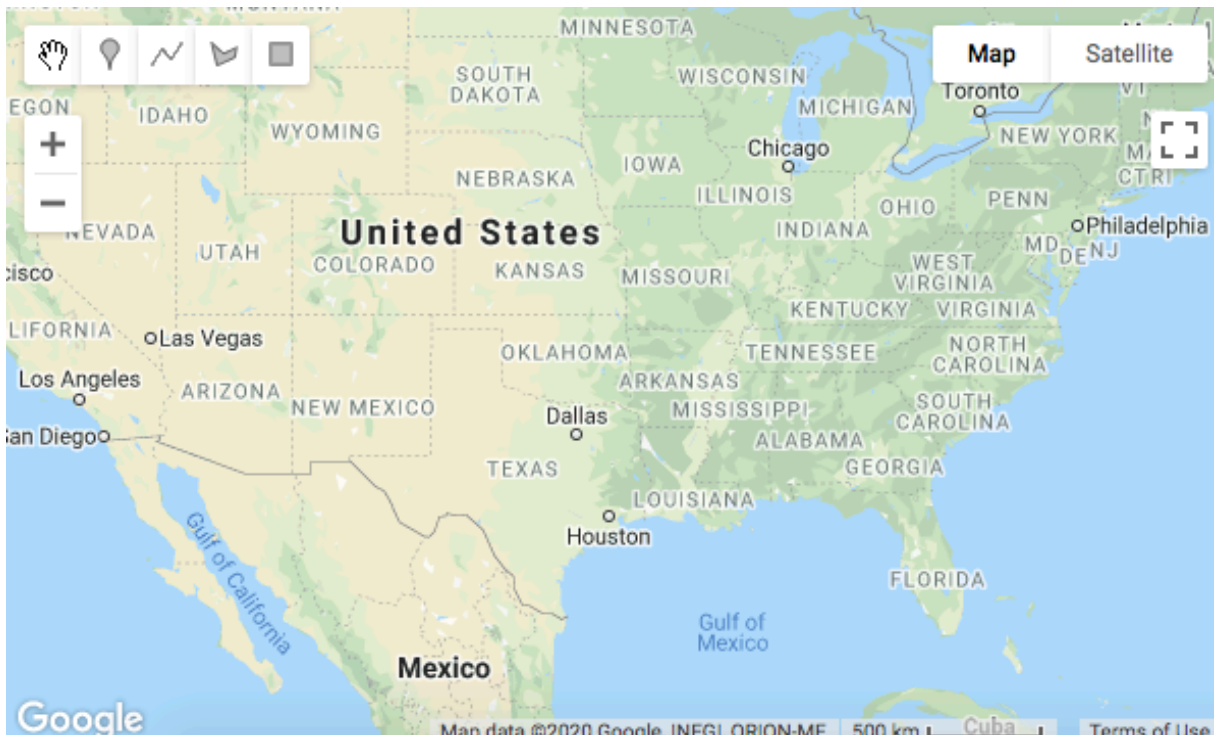
mostra a saída de comandos do *script* que redirecionam algo para a saída padrão (*stdout*) ou de erro (*stderr*). Já a aba “**Tasks**” é onde aparecem procedimentos que são efetuados externamente, como salvar arquivos no *Google Drive* ou carregar dados nos *Assets*.

Figura 11 - Inspector, Console e Tasks



Finalmente, na parte inferior (Figura 12) está o mapa do *Earth Engine*. É onde objetos espaciais podem ser visualizados de acordo com os *layers* criados. Também é possível criar geometrias interativamente nesse mapa, sendo as mesmas carregadas automaticamente como *importsets* no *script* que está sendo editado.

Figura 12 - Mapa na interface do GEE



### 3.4 Dicas de Programação em JS

A linguagem de programação utilizada no GEE é o *JavaScript* (JS). É uma linguagem de dados com tipagem dinâmica. Isso significa que uma variável pode receber qualquer tipo de dado a qualquer momento da execução do *script* e por isso é preciso estar atento às atribuições para se evitar erros de tempo de execução. Ela é também uma linguagem baseada em protótipos, que de certa forma é similar ao paradigma de orientação a objetos, na qual um objeto pode herdar características e métodos de outro objeto já instanciado. É importante atentar para isso, pois isso implica na reutilização de diversos métodos para tipos distintos de objetos. Os métodos são funções associadas a objetos instanciados (dados carregados). Além disso, o GEE possui funções que não estão associadas com objetos e que recebem uma entrada e a processam.

Figura 13 - Uma alegoria para a linguagem de programação *JavaScript*



Uma dica inicial para uso no GEE é a da criação de comentários elucidativos no código, deixando o mesmo intuitivo e mais fácil de se entender. A Figura 13 mostra os dois tipos de comentário existentes no JS, o de bloco, onde as palavras estão envolvidas por um */\** na esquerda e um *\*/* que termina o bloco na direita, e os de linha que são comentários de apenas uma linha precedidos por *//*.

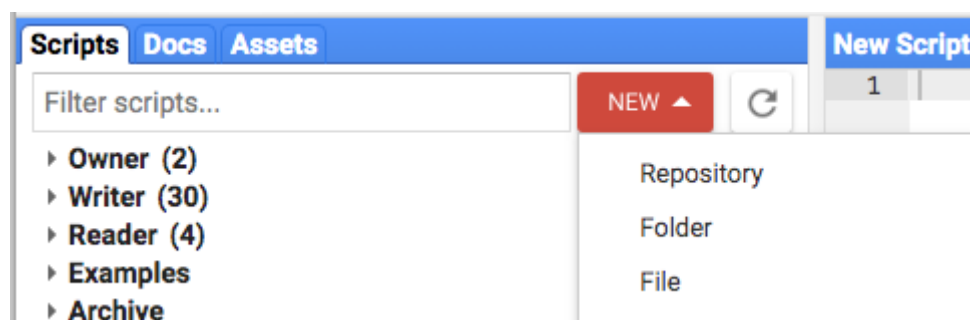
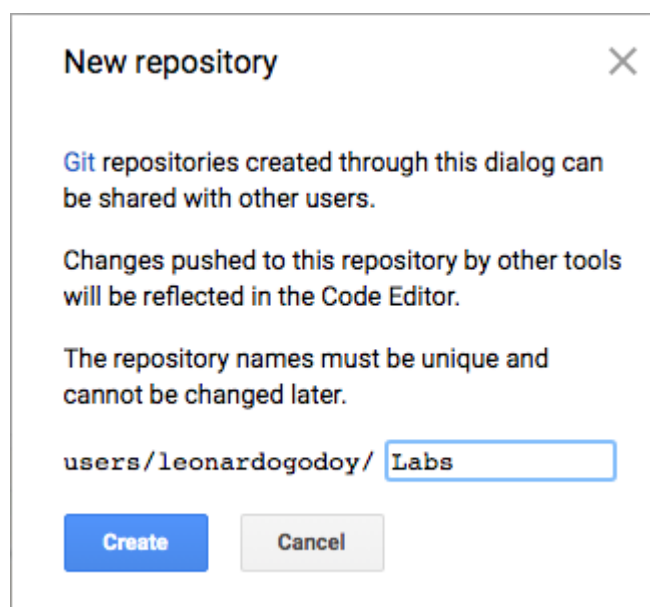
Figura 13 - Comentários em códigos escritos em *JavaScript*

```
1  /* Comentário  
2     em bloco. */  
3  // Comentário em uma linha
```

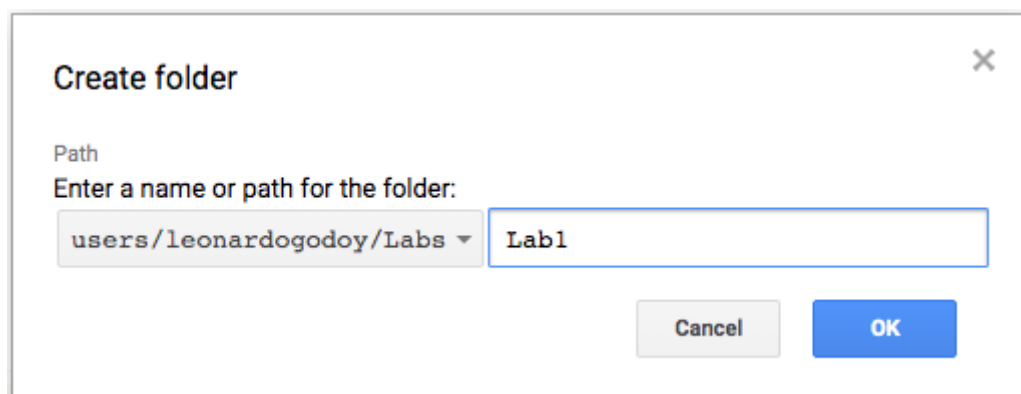
Outra dica, termine cada linha de código com um ponto-e-vírgula (;), embora o código funcione sem o ponto-e-vírgula, é uma boa prática de programação.

### 3.5 Criando o primeiro *Script* no GEE

No box do lado esquerdo, na aba “**Scripts**”, clique no botão “**NEW**” e em seguida em “**Repository**” (Figura 14). Na sequência, na janela que será aberta (Figura 15), dê um nome adequado para o repositório onde serão armazenados os *scripts* com as soluções dos laboratórios práticos do semestre. Esse repositório será criado dentro do link “**Owner**”.

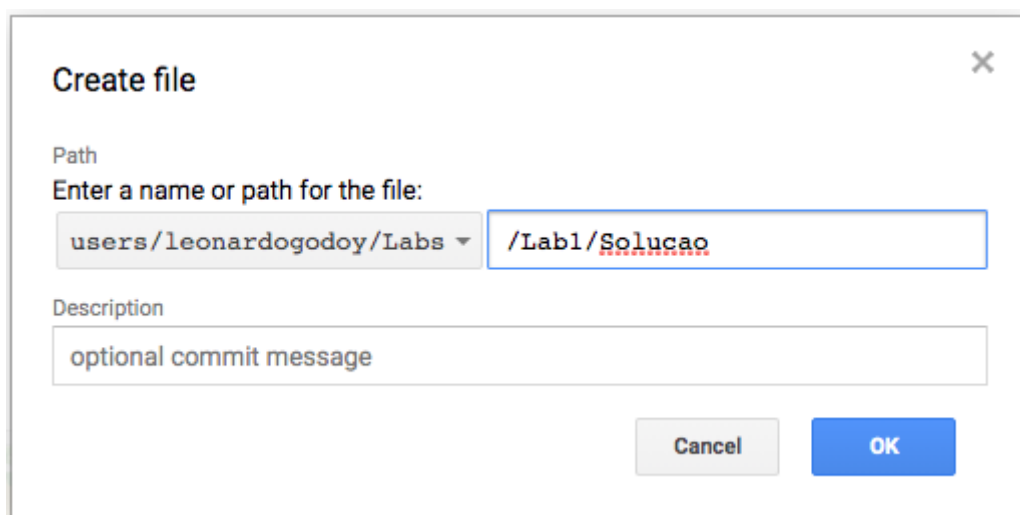
Figura 14 - Menu para criação de repositórios, pastas e *scripts*Figura 15 - Janela para criação de um novo repositório de *scripts*

Dentro do repositório você pode criar pastas para organizar seus arquivos, clicando em **“Folder”** no menu apresentado na Figura 14. A janela de criação de pasta é apresentada na Figura 16. Criar uma pasta é opcional, porém recomendado, especialmente caso queira criar mais de um *script* para cada solução do laboratório.

Figura 16 - Janela para criação de uma pasta para armazenar *scripts*

The dialog box is titled "Create folder" and has a close button (X) in the top right corner. It contains a "Path" label and the instruction "Enter a name or path for the folder:". Below this, there is a dropdown menu showing "users/leonardogodoy/Labs" and a text input field containing "Lab1". At the bottom right, there are "Cancel" and "OK" buttons.

Finalmente, basta criar os arquivos onde serão programados os *scripts*, para isso basta clicar em **“File”** no menu **“NEW”** (Figura 14) e nomear o arquivo do *script* apropriadamente (Figura 17). Note que antes do nome do arquivo, o nome da pasta é inserido envolvido por barras (no exemplo /Lab1/). Caso não coloque o nome da pasta, o arquivo será salvo na raiz do repositório, bastando nesse caso arrastar o arquivo para dentro da pasta no menu expandido do repositório - Figuras 18 e 19.

Figura 17 - Janela para a criação de um novo *script*

The dialog box is titled "Create file" and has a close button (X) in the top right corner. It contains a "Path" label and the instruction "Enter a name or path for the file:". Below this, there is a dropdown menu showing "users/leonardogodoy/Labs" and a text input field containing "/Lab1/Solucao". Below the path field, there is a "Description" label and a text input field containing "optional commit message". At the bottom right, there are "Cancel" and "OK" buttons.



Figura 18 - Arquivos do GEE, caso o arquivo tenha sido criado na raiz

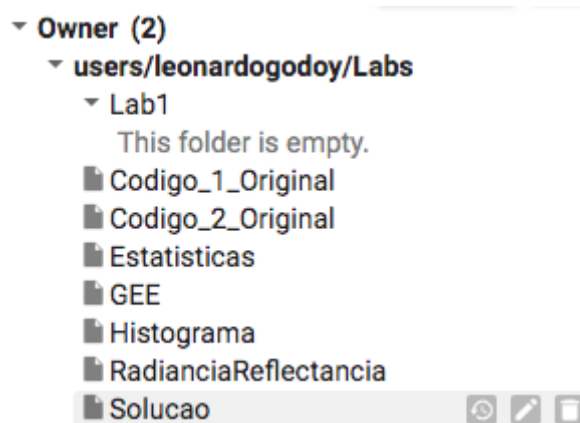
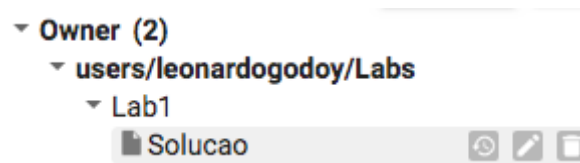


Figura 19 - Caso o arquivo tenha sido criado na raiz basta arrastá-lo para a pasta desejada

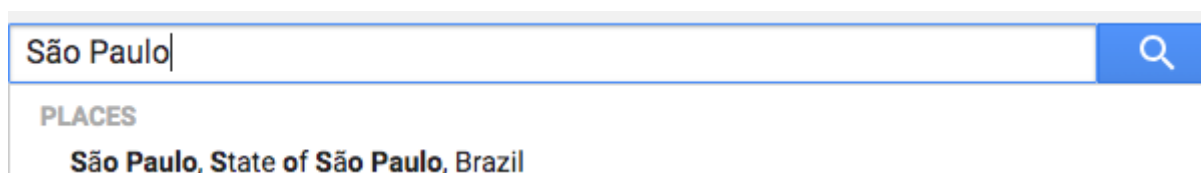


## 3.6 Explorando os dados do GEE

### 3.6.1 Localização da área de estudo

Na caixa de buscas do GEE (Figura 20), digite “**São Paulo**”. Clique na primeira opção que for listada e veja que o mapa será centralizado na cidade de São Paulo. Esta será nossa área de estudo, neste laboratório.

Figura 20 - Buscando por um local pelo nome no GEE



### 3.6.2 Criando uma *feature* no GEE

Para se criar uma *feature* no GEE, basta utilizar o menu que se encontra no lado esquerdo do mapa (Figura 21). Neste menu, pode-se observar mais à esquerda **um ícone com uma “mão” desenhada**, que serve para dar “Pan” (movimentar o mapa) e em seguida estão os ícones para quatro tipos de geometria disponíveis: o ponto, a linha, o polígono e o retângulo, respectivamente. O retângulo é um tipo específico de polígono. A ideia desses itens é criar a geometria de forma interativa no mapa, para isso basta clicar sobre um dos ícones e desenhar a geometria sobre o mapa. É necessário concluir a criação da geometria clicando-se no botão **“Exit”** no painel aberto quanto o modo de criação de geometrias está ativado (Figura 22). A geometria criada estará dentro de um objeto do tipo *feature*, caso se prossiga criando outros pontos para somente depois clicar em **“Exit”**, a *feature* vai conter uma multi-geometria. Caso prossiga criando outro tipo de geometria, selecionando a opção no menu de criação de geometrias (Figura 22), uma coleção de geometrias será criada dentro da *feature* (MultiPolygon). Para se criar uma nova *feature*, deve-se selecionar a opção **“+new layer”**, do menu **“Geometry Imports”** (Figura 24), caso contrário as novas geometrias serão adicionadas a uma já existente (dentro da camada que está selecionada). Esse último passo é desnecessário na criação da primeira *feature* do *script*.

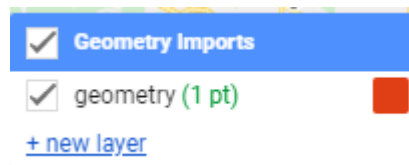
Figura 21 - Menu para criação de geometrias no mapa



Figura 22 - Modo de criação de geometrias no mapa habilitado



Figura 23 - Geometria criada no mapa

Figura 24 - Menu Geometry Imports com “+ *new layer*”

Após a criação da nova *feature*, uma variável se referindo ao objeto que contém a geometria aparecerá no topo do *script* no setor “**Imports**” (Figura 25). Um nome padrão é atribuído e uma maneira de se alterar o mesmo é clicando duas vezes sobre o nome (“**geometry**” na Figura 25) e em seguida inserindo o nome desejado (Figura 26 e 27).

Figura 25 - Objeto armazenando a geometria criada

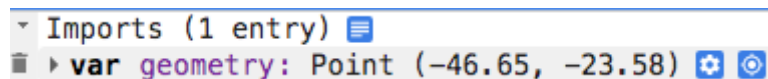


Figura 26 - Objeto com o nome alterado

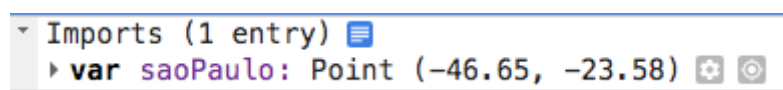
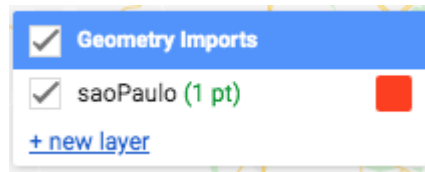


Figura 27 - Objeto com o nome alterado no menu do mapa



### 3.6.3 Buscando uma coleção de imagens

Na caixa de buscas do GEE (Figura 28), digite “*landsat*”. Observe a lista abaixo de “**RASTERS**” e em seguida clique em “**more**”. Uma nova janela será aberta com os resultados da busca (Figura 29). Clique na primeira opção referente ao **Landsat 8 Collection 2 Tier 1 Real Time (na versão anterior era a 7, mas como não está funcionando vamos usar a 8, caso deseje deixar o nome compatível basta tomar o cuidado para trocar em todas as partes do código)** e uma janela descrevendo a coleção será aberta (Figura 30). Ao clicar no botão “**IMPORT**” nesta janela, a coleção será importada para o *script*. Altere o nome da variável, como demonstrado anteriormente, para “**colecacaoLandsat7**” (Figura 31).

Figura 28 - Buscando uma coleção de imagens no GEE

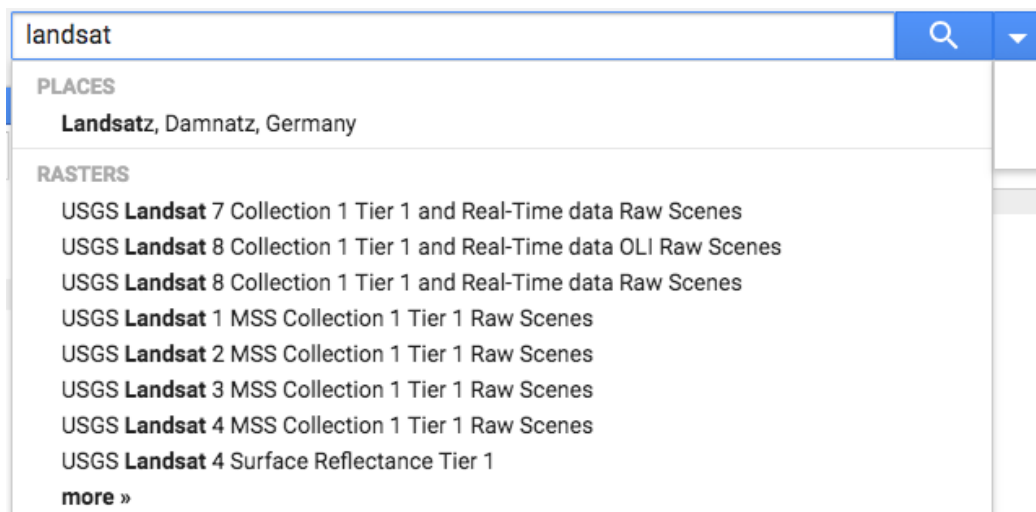


Figura 29 - Lista de coleções de imagens com as palavras buscadas

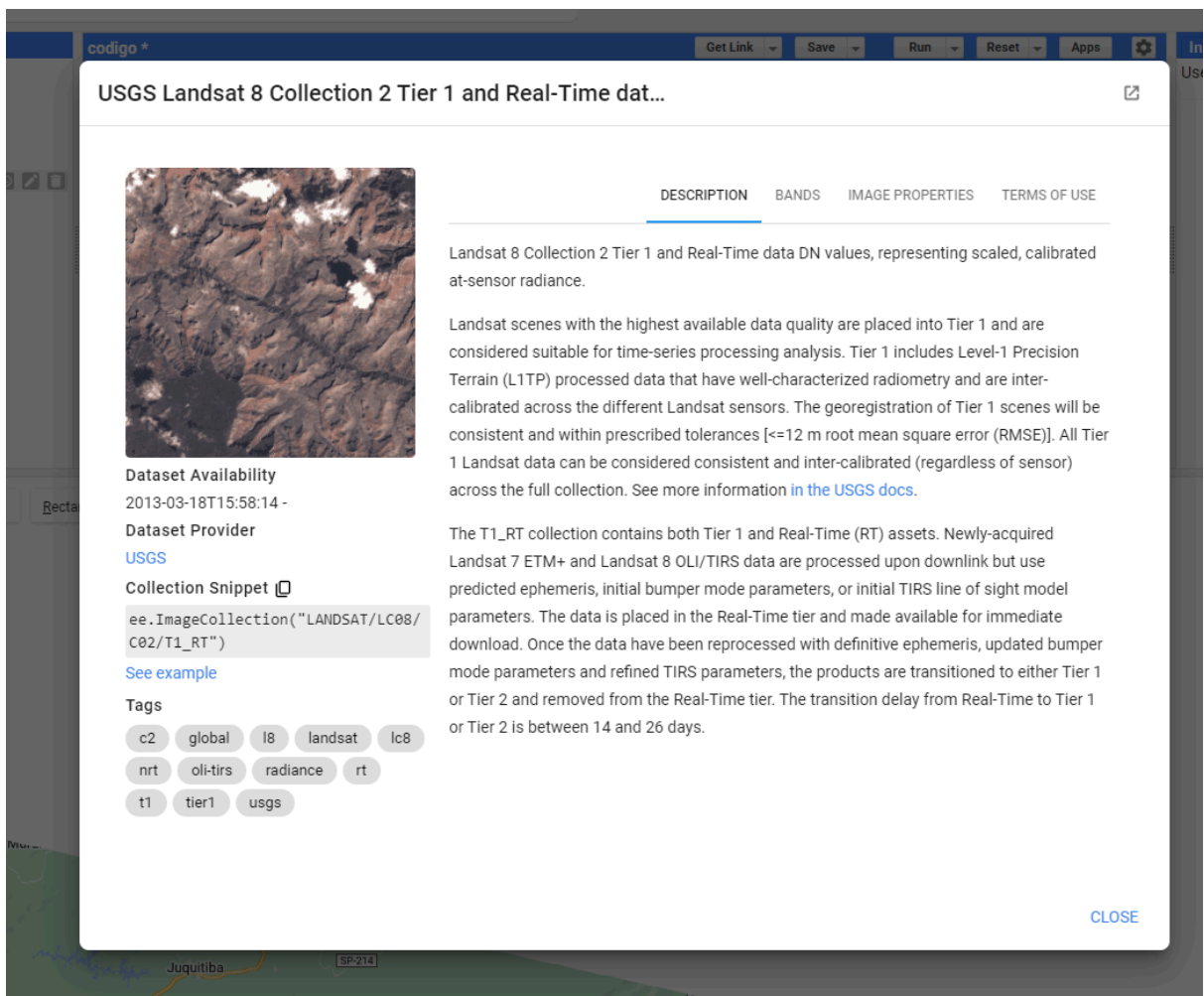


Figura 30- Detalhes de uma coleção de imagens

Figura 31 - Coleção de imagens importada no *script*

```
Imports (2 entries)
var saoPaulo: Point (-46.62, -23.56)
var colecaoLandsat7: ImageCollection "USGS Landsat 7 Collection 1 Tier 1 and Real-Time data Raw Scenes"
```

## 3.7 Trabalhando com imagens de satélite no GEE

### 3.7.1 Adicionando uma imagem ao mapa

Na Figura 32 está o código para adicionar a primeira imagem de uma coleção ao mapa da interface do GEE (a coleção de imagens as armazenam em sequência, como numa pilha de cartas de baralho, a partir da qual puxamos uma para exibir no mapa). Na linha 3 o método ***filterBounds*** do objeto ***colecaoLandsat7*** é responsável por selecionar apenas as imagens da coleção que estão na região da

feature **“saoPaulo”**. O método **“first()”** é aplicado no conjunto resultante gerado pelo método **“filterBounds()”**. O método **“first()”** seleciona apenas a primeira imagem da coleção, com todas as suas bandas, resultando em um objeto que é salvo na variável **“primeiraImagemSpColecao”**. A seguir, na linha 5, o método **“select()”** do objeto **“primeiraImagemSpColecao”**, é responsável por extrair apenas as bandas descritas no padrão passado como parâmetro. Esse parâmetro é uma *string* (cadeia de caracteres), onde o formato **“[1-3]”** funciona como uma forma de listar os valores numéricos de 1 a 3 de forma concatenada com o **“B”**, gerando assim a sequência **“B1”**, **“B2”** e **“B3”**. Essa é a forma que o método interpreta a string, outro formato possível para parametrizar seria **“B1’,B2’,B3”**, com três strings separadas por vírgula. O resultado é então armazenado em **“primeiraImagemSpRgb”**, sendo esse objeto utilizado para ser adicionado no mapa com a função **“Map.addLayer()”**. O segundo parâmetro após a vírgula é um dicionário de dados, uma estrutura declarada entre chaves e com seus elementos separados por vírgula. Cada elemento é da forma chave:valor. Nesse caso o valor da chave não fica entre aspas, já o valor pode ser de diversos tipos de dados. O resultado do código no mapa deve ser semelhante ao da Figura 33. (Importante: clique em save para salvar o código e não perder o trabalho caso tenha algum problema com o computador). No canto superior clicar em *run*, se não houver nenhum erro de digitação a imagem aparecerá como na Figura 34 (caso a imagem não esteja conforme mostrado, vá na opção **“layer”**, no lado direito do mapa, selecione o botão de configurações do *layer* atual e em **“range”** altere a opção **“custom”**, para **“Stretch: 1  $\sigma$ ”**, se o problema persistir, teste as outras opções de *stretch*).

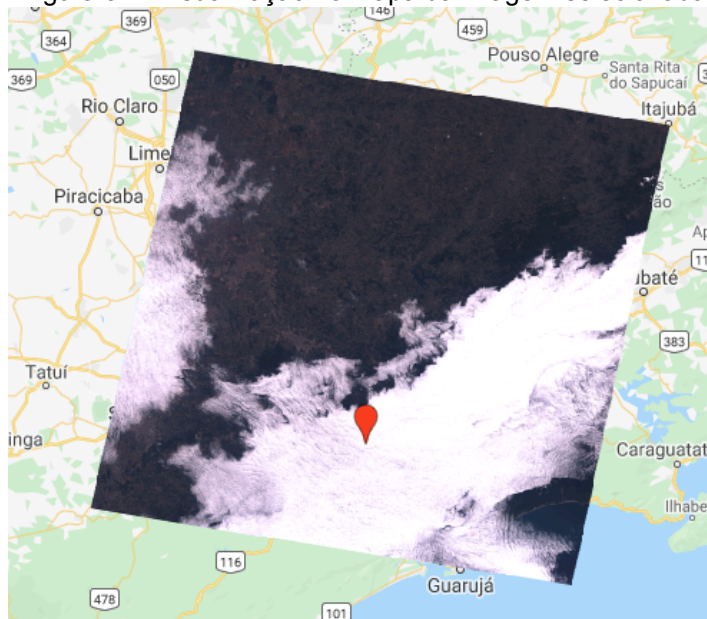
Figura 32 - Código que pega a primeira imagem de uma coleção e adiciona as bandas do visível no mapa

```
1  /* Recebe na variável primeiraImagemSpColecao a primeira imagem da coleção que
2     cobre o ponto criado em São Paulo. */
3  var primeiraImagemSpColecao = colecaoLandsat7.filterBounds(saoPaulo).first();
4  // Seleciona apenas as bandas de 1 a 3 da imagem.
5  var primeiraImagemSpRgb = primeiraImagemSpColecao.select('B[1-3]');
6  // Adiciona ao mapa as bandas de 1 a 3 para visualização.
7  Map.addLayer(primeiraImagemSpRgb, {bands:'B3,B2,B1', min:0, max:255});
```

Figura 33 - Código apresentado na figura 32

```
/* Recebe na variável primeiraImagemSpColecao a primeira imagem da coleção que  
   cobre o ponto criado em São Paulo. */  
var primeiraImagemSpColecao = colecaoLandsat7.filterBounds(saoPaulo).first();  
// Seleciona apenas as bandas de 1 a 3 da imagem.  
var primeiraImagemSpRgb = primeiraImagemSpColecao.select('B[1-3]');  
// Adiciona ao mapa as bandas de 1 a 3 para visualização.  
Map.addLayer(primeiraImagemSpRgb, {bands: 'B3,B2,B1', min: 0, max: 255});
```

Figura 34 - Visualização no mapa da imagem selecionada



### 3.7.2 Recortando uma parte de uma imagem

Para se recortar uma parte de uma imagem, basta criar uma *feature* com a geometria apropriada a partir de “+new layer” no menu “**Geometry Imports**” (Figura 24), como o retângulo na Figura 35 (alterar o nome da variável é importante para facilitar a codificação - Figura 36), e utilizar o método “**clip()**” na imagem desejada, como pode ser visto no código apresentado na Figura 37. O resultado do recorte da imagem pode ser visualizado com a mesma função “**Map.addLayer()**” - utilize o código da Figura 38. Para uma melhor visualização, pode-se ocultar o *layer* com o polígono desenhado, desmarcando o mesmo na lista “**Geometry Imports**” (Figura 39).

Figura 35 - Criação de um retângulo na região coberta pela imagem exibida

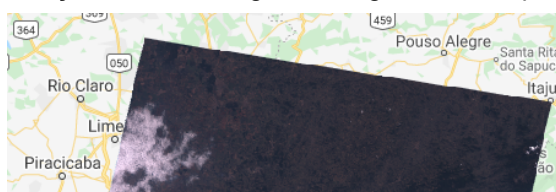




Figura 36 - Variável contendo o retângulo com o nome alterado

```
var retanguloSelecao: Polygon, 4 vertices
```

Figura 37 - Código que recorta as bandas do visível da imagem na região do retângulo desenhado e que em seguida exibe no mapa apenas essa região da imagem

```
8 // Recorta as bandas B1 a B3 da imagem apenas no retângulo desenhado.
9 var primeiraImagemSpRgbRetanguloSelecao = primeiraImagemSpRgb.clip(retanguloSelecao);
10 // Adiciona ao mapa as bandas de 1 a 3 no retângulo desenhado para visualização.
11 Map.addLayer(primeiraImagemSpRgbRetanguloSelecao, {bands: 'B3,B2,B1', min: 0, max: 255});
```

Figura 38 - Código apresentado na figura 37

```
// Recorta as bandas B1 a B3 da imagem apenas no retângulo desenhado.
var primeiraImagemSpRgbRetanguloSelecao =
primeiraImagemSpRgb.clip(retanguloSelecao);
// Adiciona ao mapa as bandas de 1 a 3 no retângulo desenhado para visualização.
Map.addLayer(primeiraImagemSpRgbRetanguloSelecao, {bands: 'B3,B2,B1', min: 0,
max: 255});
```

Figura 39 - Ocultando o retângulo desenhado no mapa



Figura 40 - Recorte da imagem apenas sob o retângulo desenhado



### 3.7.3 Calculando o valor médio dos *pixels* de uma banda

Um exemplo do padrão **reduce**, apresentado anteriormente, pode ser visto ao se calcular a média dos *pixels* de uma imagem. O código da Figura 41 apresenta esse procedimento. O método **“reduceRegion()”** é responsável por disparar o



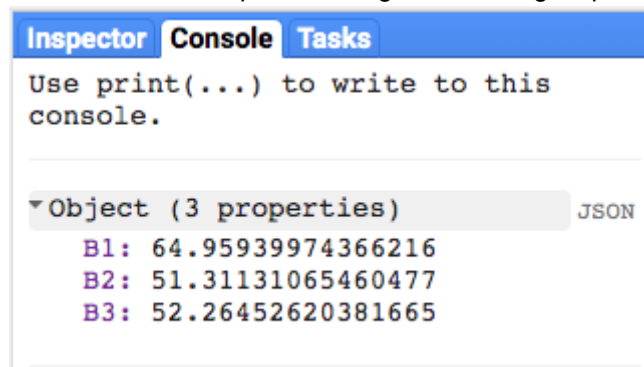
cálculo, com os parâmetros desejados. Novamente, o parâmetro é um dicionário de dados, dessa vez ajustando o valor da chave **“reducer”** para o algoritmo que deve ser utilizado para calcular o valor desejado, no caso a média. Já a chave **“maxPixels”**, diz o máximo número de *pixels* que devem ser processados. Esse parâmetro é importante para não exaurir a capacidade de processamento disponibilizada pelo GEE para o usuário. Após o cálculo efetuado, apenas utilizando a função **“print()”**, visualiza-se o resultado no console como na Figura 42

Figura 41 - Código que calcula o valor médio dos *pixels*

```
12 - /* Utilizando o método reduce, é calculado para a região do retângulo desenhado o
13    valor médio do pixel de cada banda. */
14 - var valorMedioPixelRgb = primeiraImagemSpRgbRetanguloSelecao.reduceRegion({
15     reducer: ee.Reducer.mean(),
16     maxPixels: 1e9
17 });
18 // Exibe no console os dados resultantes da operação.
19 print(valorMedioPixelRgb);

/* Utilizando o método reduce, é calculado para a região do retângulo desenhado
o
    valor médio do pixel de cada banda. */
var valorMedioPixelRgb = primeiraImagemSpRgbRetanguloSelecao.reduceRegion({
    reducer: ee.Reducer.mean(),
    maxPixels: 1e9
});
// Exibe no console os dados resultantes da operação.
print(valorMedioPixelRgb)
```

Figura 42 - Valores médios dos *pixels* da região do retângulo para as bandas do visível



Até o momento o código completo deve estar como na Figura 43.

Figura 43 - Código completo

```

Imports (3 entries)
var saoPaulo: Point (-46.68, -23.11)
var colecaoLandsat7: ImageCollection "USGS Landsat 7 Collection 1 Tier 1 and Real-Time data Raw Scenes"
var retanguloSelecao: Polygon, 4 vertices

1 /* Recebe na variável primeiraImagemSpColecao a primeira imagem da coleção que
2 cobre o ponto criado em São Paulo. */
3 var primeiraImagemSpColecao = colecaoLandsat7.filterBounds(saoPaulo).first();
4 // Seleciona apenas as bandas de 1 a 3 da imagem.
5 var primeiraImagemSpRgb = primeiraImagemSpColecao.select('B[1-3]');
6 // Adiciona ao mapa as bandas de 1 a 3 para visualização.
7 //Map.addLayer(primeiraImagemSpRgb, {bands: 'B3,B2,B1', min: 0, max: 255});
8 // Recorta as bandas B1 a B3 da imagem apenas no retângulo desenhado.
9 var primeiraImagemSpRgbRetanguloSelecao = primeiraImagemSpRgb.clip(retanguloSelecao);
10 // Adiciona ao mapa as bandas de 1 a 3 no retângulo desenhado para visualização.
11 Map.addLayer(primeiraImagemSpRgbRetanguloSelecao, {bands: 'B3,B2,B1', min: 0, max: 255});
12 /* Utilizando o método reduce, é calculado para a região do retângulo desenhado o
13 valor médio do pixel de cada banda. */
14 var valorMedioPixelRgb = primeiraImagemSpRgbRetanguloSelecao.reduceRegion({
15   reducer: ee.Reducer.mean(),
16   maxPixels: 1e9
17 });
18 // Exibe no console os dados resultantes da operação.
19 print(valorMedioPixelRgb)

```

### 3.9 Resolução espacial

Abra novamente a janela com as informações da coleção e em seguida selecione a aba **"BANDS"** (Figura 5). Nesta aba estarão as informações da resolução espacial da coleção. Em alguns casos a informação será listada como uma coluna nomeada **"Resolution"** para cada banda na tabela descrevendo as bandas. De forma diferente, no exemplo da Figura 5, a resolução está na parte de cima da aba **"BANDS"**, por ser a mesma para todas as bandas.

Figura 47 - Aba com as propriedades das bandas  
USGS Landsat 8 Level 2, Collection 2, Tier 1



#### Dataset Availability

2013-03-18T15:58:14 -

#### Dataset Provider

[USGS](#)

#### Collection Snippet

```
ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")
```

DESCRIPTION BANDS IMAGE PROPERTIES TERMS OF USE

#### Resolution

30 meters

#### Bands Table

Name	Description	Min	Max	Units	Wavelength
SR_B1	Band 1 (ultra blue, coastal aerosol) surface reflectance	1	65455		0.435-
SR_B2	Band 2 (blue) surface reflectance	1	65455		0.452-
SR_B3	Band 3 (green) surface reflectance	1	65455		0.533-
SR_B4	Band 4 (red) surface	1	65455		0.636-

### 3.10 Resolução espectral

Ainda na aba “**BANDS**” da janela de informações, na tabela onde estão listadas as características das bandas, é possível observar na coluna “**Wavelength**” a faixa espectral de cada banda. Isso, juntamente com a quantidade de bandas, define a resolução espectral das imagens da coleção. Deve-se atentar para o fato de que algumas bandas na coleção possuem informações diversas das espectrais.

### 3.11 Resolução temporal

A resolução temporal, baseada no tempo de revisita do satélite, pode ser encontrada nas especificações técnicas do satélite, disponíveis em seus manuais. Uma alternativa para saber mais informações sobre a resolução temporal, é através da utilização das informações de datas de coleta das imagens, como será visto mais à frente neste laboratório.

### 3.12 Resolução radiométrica

A resolução radiométrica pode ser encontrada na documentação do satélite que coletou os dados da coleção. Há de se ressaltar que a coleção pode conter imagens derivadas dos dados coletados originalmente (*raw data*), sendo assim esses valores numéricos podem estar em uma faixa diferente dos valores das imagens originais. Uma alternativa é checar o tipo de dado numérico dos pixels das bandas de uma imagem da coleção, conforme será visto adiante.

### 3.13 Trabalhando com metadados via código

Com os dados carregados e selecionadas apenas as imagens na região de São Paulo, pode-se acessar os metadados através do *script*, conforme os códigos na Figura 48 e 49. Aqui, vamos importar a coleção toda, não apenas a primeira imagem, conforme a primeira instrução de código na Figura 49, sem utilizar o método **first()**, como feito anteriormente. Na linha 7 do código, através do método

**`toDictionary()`**”, do objeto onde a coleção foi importada (**`imagensSpColecao`**”), é gerado um dicionário de dados com todas as informações dos metadados da coleção. Exibindo esse dicionário de dados no console - linha 9 - é possível visualizar os seus detalhes. Na linha 11, a função **`print()`** aplicada diretamente no resultado de um método - sem se atribuir à uma variável previamente o valor que deve ser exibido - mostra no console os nomes das propriedades da primeira imagem da coleção. Note a sequência das chamadas dos métodos, da esquerda para a direita. O método **`first()`** extrai a primeira imagem da coleção e, a partir do objeto resultante, a função **`propertyNames()`** é invocada, retornando uma lista com os nomes de todos os atributos dos metadados da imagem (essas propriedades da primeira imagem existem para todas as imagens de uma mesma coleção, quando importada). Uma lista é um tipo de objeto que pode conter um grupo de objetos conectados numa certa ordem, nesse caso os objetos são strings com os nomes das propriedades na imagem. Cabe ressaltar que algumas dessas propriedades não são nativas, mas sim geradas no próprio GEE, essas são as propriedades que iniciam com **`“system:”`**. Isso pode ser notado ao se executar o código da linha 13, onde a função **`toDictionary()`** - aplicada na primeira imagem da coleção - gera um dicionário de dados com os valores atribuídos às propriedades da primeira imagem. Pode-se observar na saída no console, que as propriedades listadas anteriormente com a palavra **`“system:”`** não são exibidas nesta chamada.

Ainda na Figura 48, entre a linha 15 e a 18, o código demonstra como se extrair uma propriedade de uma imagem diretamente pelo nome - no caso **`“CLOUD_COVER”`**, com o método **`get()`**. Repare que o código pode ser quebrado em mais de uma linha para se melhorar a legibilidade.

Figura 48 - Trecho de código extraindo informações dos metadados

```
5 ▾ /* Gera um dicionário de dados com os metadados da coleção e armazena
6   | na variável dictPropriedadesColecao. */
7   var dictPropriedadesColecao = imagensSpColecao.toDictionary();
8   // Imprime no console os metadados da coleção.
9   print("Propriedades da coleção: ", dictPropriedadesColecao);
10  // Imprime no console os nomes dos metadados da primeira imagem da coleção.
11  print('Nomes das propriedades da imagem: ', imagensSpColecao.first().propertyNames());
12  // Imprime no console os valores metadados da primeira imagem da coleção.
13  print("Propriedades da primeira imagem da coleção: ", imagensSpColecao.first().toDictionary());
14  // Exibe a porcentagem da cobertura por nuvens da primeira imagem da coleção.
15  print(
16    "Cobertura por nuvens da primeira imagem da coleção: ",
17    imagensSpColecao.first().get('CLOUD_COVER')
18  );
```

Figura 49 - Código apresentado na figura 48

```
/* Recebe, na variável imagensSpColecao, a coleção de imagens que  
   cobre o ponto criado em São Paulo. */  
var imagensSpColecao = colecaoLandsat7.filterBounds(saoPaulo);  
  
/* Gera um dicionário de dados com os metadados da coleção e armazena  
   na variável dictPropriedadesColecao. */  
var dictPropriedadesColecao = imagensSpColecao.toDictionary();  
// Imprime no console os metadados da coleção.  
print("Propriedades da coleção: ", dictPropriedadesColecao);  
// Imprime no console os nomes dos metadados da primeira imagem da coleção.  
print('Nomes das propriedades da imagem:  
,imagensSpColecao.first().propertyNames());  
// Imprime no console os valores metadados da primeira imagem da coleção.  
print("Propriedades da primeira imagem da coleção: ",  
imagensSpColecao.first().toDictionary());  
// Exibe a porcentagem da cobertura por nuvens da primeira imagem da coleção.  
print(  
    "Cobertura por nuvens da primeira imagem da coleção: ",  
    imagensSpColecao.first().get('CLOUD_COVER')  
);
```

### 3.13.1 Seleção de imagens em uma coleção utilizando informações dos Metadados

No último trecho de código da Figura 48, foi exibido o valor por cobertura de nuvens da primeira imagem da coleção. Sabendo-se que esse valor numérico representa um percentual e analisando o resultado da informação exibida, para a coleção analisada, nota-se que o valor está entre 0 e 100 (outras coleções podem ter esse valor entre 0 e 1). Com essa informação, pode-se fazer uma seleção na coleção através da propriedade **“CLOUD\_COVER”**, das imagens com cobertura por nuvens menor do que 40%. Esse código pode ser observado na linha 22 da Figura 50, onde ocorre a chamada do método **“filterMetadata()”** no objeto **“imagensSpColecao”**. Os parâmetros do método, na ordem, representam o nome da propriedade, seguido por uma palavra chave que representa o operador a ser aplicado (os operadores disponíveis podem ser vistos na documentação do GEE) e finalmente o valor de referência, no caso 40. Após essa seleção ser copiada para o objeto **“imagensSpColecaoSelecaoDataNuvens”**, o tamanho da coleção resultante é impresso no console utilizando-se a função **“size()”** na linha 26.

Na linha 29, da Figura 50, é demonstrado como se filtram os dados de uma coleção dentro de um período de tempo com a função ***“filterDate()”***. As datas fornecidas como parâmetro seguem o formato ***“AAAA-MM-DD”***, onde ***“AAAA”*** representa o ano com quatro algarismo, ***“MM”*** o mês com dois algarismos e ***“DD”*** o dia com dois algarismos. O primeiro parâmetro é a data de início da busca, sendo que valores coincidentes são incluídos na busca, já para o segundo parâmetro, que representa o fim do período, o valor é exclusivo, ou seja ele não é incluído na busca, por esse motivo para se buscar um ano inteiro o valor do final deve ser o primeiro dia do ano subsequente.

Um último detalhe na Figura 50 é que, nas linhas 33 e 34, é executado um código que transforma a coleção em uma lista, através do método ***“toList()”***. O parâmetro fornecido é o máximo de elementos que se deve extrair da coleção, nesse caso todos.

Figura 50 - Seleção de imagens através dos metadados

```
21 // Seleciona apenas as imagens com a cobertura por nuvens menor que o desejado.
22 var imagensSpColecaoSelecaoDataNuvens = imagensSpColecao.filterMetadata('CLOUD_COVER', 'less_than', 40);
23 // Imprime número de imagens com a cobertura por nuvens menor que o desejado.
24 print(
25     "Número total de imagens na coleção com\ncobertura por nuvens menor que o desejado: ",
26     imagensSpColecaoSelecaoDataNuvens.size()
27 );
28 // Seleciona imagens dentro de um período determinado.
29 var imagensSpColecaoSelecaoData = imagensSpColecao.filterDate('2019-01-01', '2020-01-01');
30 // Imprime no console detalhes da coleção no período selecionado.
31 print("Coleção de imagens no ano de 2019: ", imagensSpColecaoSelecaoData);
32 // Transforma a coleção de imagens do período selecionado em uma lista.
33 var lstImagensSpColecaoSelecaoData = imagensSpColecaoSelecaoData
34     .toList(imagensSpColecaoSelecaoData.size());
```

Figura 51 - Código apresentado na figura 50

```
// Seleciona apenas as imagens com a cobertura por nuvens menor que o desejado.
var imagensSpColecaoSelecaoDataNuvens =
    imagensSpColecao.filterMetadata('CLOUD_COVER', 'less_than', 40);
// Imprime número de imagens com a cobertura por nuvens menor que o desejado.
print(
    "Número total de imagens na coleção com\ncobertura por nuvens menor que
o desejado: ",
    imagensSpColecaoSelecaoDataNuvens.size()
);
// Seleciona imagens dentro de um período determinado.
var imagensSpColecaoSelecaoData = imagensSpColecao.filterDate('2019-01-01',
'2020-01-01');
// Imprime no console detalhes da coleção no período selecionado.
print("Coleção de imagens no ano de 2019: ", imagensSpColecaoSelecaoData);
// Transforma a coleção de imagens do período selecionado em uma lista.
```



```
var lstImagensSpColecaoSelecaoData = imagensSpColecaoSelecaoData
    .toList(imagensSpColecaoSelecaoData.size());
```

### 3.13.2 Manipulando datas das imagens

Para se capturar um elemento de uma lista, faz-se uso do método **“get()”** com o índice do elemento como parâmetro. Os índices referem-se às posições dos elementos dentro da lista e começam em 0 (ou seja, o primeiro elemento está na posição 0), como pode ser visto na linha 38 da Figura 52. Nessa mesma linha, pode ser observado um *casting* (conversão) do objeto retornado pelo **“get()”** para o tipo **“ee.Image”**. Isso deve ser feito, pois o tipo retornado pelo **“get()”** é um *Object*, um tipo genérico no GEE, quando a plataforma não sabe qual tipo de dado está retornando.

Uma maneira de se retornar a data de uma imagem é utilizando o método já implementado **“date()”**, esse método se baseia no metadado **“system:time\_start”**, sofrendo um *casting* no tipo de dado para um objeto do tipo *Date*. Isso ocorre pois o valor armazenado em **“system:time\_start”** é um valor numérico do tipo inteiro representando o número de milissegundos desde a 0 horas do dia primeiro de janeiro de 1970. Na Figura 52, na sequência, é possível ver outra maneira de se obter a data de aquisição de uma imagem, através da propriedade que armazena o momento do registro. Nesse caso o valor é retornado em uma *string*.

Figura 52 - Extraindo datas de imagens em uma lista

```
37 // Extrai a primeira imagem da lista.
38 var primeiraImagem = ee.Image(lstImagensSpColecaoSelecaoData.get(0));
39 // Usando o método date(), imprime a data da primeira imagem no console.
40 print("Data de aquisição da primeira imagem: ", primeiraImagem.date());
41 // Extrai a segunda imagem da lista.
42 var segundaImagem = ee.Image(lstImagensSpColecaoSelecaoData.get(1));
43 // Usando diretamente a propriedade da imagem, imprime a data da segunda imagem no console.
44 print("Data de aquisição da segunda imagem: ", segundaImagem.get('SENSING_TIME'));
```

Figura 53 - Código apresentado na figura 50

```
// Extrai a primeira imagem da lista.
var primeiraImagem = ee.Image(lstImagensSpColecaoSelecaoData.get(0));
// Usando o método date(), imprime a data da primeira imagem no console.
print("Data de aquisição da primeira imagem: ", primeiraImagem.date());
// Extrai a segunda imagem da lista.
var segundaImagem = ee.Image(lstImagensSpColecaoSelecaoData.get(1));
// Usando diretamente a propriedade da imagem, imprime a data da segunda imagem
```

```
no console.  
print("Data de aquisição da segunda imagem: ",  
segundaImagem.get('SENSING_TIME'));
```

### 3.13.3 Bandas, projeção cartográfica e resolução espacial

Uma lista com os nomes de todas as bandas de uma imagem pode ser obtida com o uso do método **“bandNames()”**, como na linha 48 da Figura 54.

A projeção cartográfica das bandas de uma imagem devem ser retornadas com o uso do método **“projection()”**, como visto na linha 50 da Figura 9. Nesta linha, o método **“select()”** é utilizado para se selecionar apenas uma banda. Não haveria problema em se aplicar o método em todas as bandas simultaneamente, exceto se houvesse diferença nas projeções das bandas, o que geraria um erro na chamada do método **“projection()”**. Um objeto do tipo *projection* é retornado, em caso de sucesso, com as informações da projeção cartográfica da imagem, ou da banda selecionada, como no exemplo da Figura 54.

Entre as linhas 53 e 56 da Figura 54, é executado o método **“nominalScale()”** no objeto *projection* retornado pelo método **“projection()”**. O resultado da chamada é o valor em metros da escala<sup>1</sup> da projeção. Escala, para uma imagem no GEE, é sinônimo de resolução espacial de seus *pixels*.

Figura 54 - Código para buscar informações das bandas de uma imagem

```
47 // Lista as bandas disponíveis a partir da primeira imagem da coleção.  
48 print("Bandas na imagem: ", imagensSpColecao.first().bandNames());  
49 // Informações da projeção da banda.  
50 var projecaoB1 = imagensSpColecao.first().select('B1').projection();  
51 print('Projeção da banda 1: ', projecaoB1);  
52 // Exibe a resolução, em metros, da banda 1.  
53 print(  
54     'Resolução da banda 1: ',  
55     imagensSpColecao.first().select('B1').projection().nominalScale()  
56 );
```

<sup>1</sup> Informações sobre o conceito de escala no GEE podem ser obtidas no *link* a seguir:  
<https://developers.google.com/earth-engine/guides/scale>



Figura 55 - Código apresentado na figura 54

```
// Lista as bandas disponíveis a partir da primeira imagem da coleção.
print("Bandas na imagem: ", imagensSpColecao.first().bandNames());
// Informações da projeção da banda.
var projecaoB1 = imagensSpColecao.first().select('SR_B1').projection();
// Exibe a resolução, em metros, da banda 1.
print(
  'Resolução da banda 1: ',
  imagensSpColecao.first().select('SR_B1').projection().nominalScale()
);
```

### 3.14 Criando funções no GEE

No GEE, funções podem ser definidas pelo usuário no *script*, desde que antes de serem chamadas no código. A estrutura de uma função pode ser vista na Figura 56, entre as linhas 60 e 62. A função é atribuída a uma variável, pois a mesma fica alocada na memória. Do lado direito da atribuição, a função é definida entre as chaves (`{}`), após a palavra-chave **“function”** e os parâmetros definidos entre parênteses, sendo separados por vírgula. No exemplo, apenas o parâmetro *imagem* é utilizado pela função. Toda função deve ter um valor retornado, esse valor é definido após a palavra-chave **“return”**. Uma linha iniciada com **“return”** termina a função imediatamente após retornar o valor desejado. No exemplo, a função **“dataImagem”**, irá retornar a própria imagem recebida como parâmetro com uma nova banda adicionada pelo método **“addBands()”**. Essa banda é uma imagem gerada com um único valor do tipo numérico extraído de uma propriedade dos metadados, pelo método **“metadata()”**. O primeiro parâmetro do método **“metadata()”** é a propriedade desejada, e o segundo o nome da banda criada na imagem.

Na linha 64, da Figura 56, temos o padrão *map*, mapeando para uma coleção de imagens a função criada anteriormente. A coleção resultante desse processo, é uma coleção onde as imagens da coleção original são copiadas juntamente com uma banda contendo a data em forma numérica.

Embora pareça estranho, esse tipo de informação - carregado dessa forma - pode ser útil no trabalho de imagens em séries temporais, ficando mais fácil de se manipular os dados pelas características do GEE.

Figura 56 - Criando e usando uma função no GEE para uma coleção

```

59 // Adiciona uma banda, na imagem, com o valor numérico da data como constante nos pixels.
60 var dataImagem = function(imagem) {
61   return imagem.addBands(imagem.metadata('system:time_start', 'dataNumerica'));
62 };
63 // Mapeia a função dataImagem para todas as imagens da coleção.
64 var datas = imagensSpColecaoSelecaoData.map(dataImagem);
65 // Coleção de imagens com o valor numérico das datas em uma banda adicionada.
66 print("Coleção com o valor numérico das datas em uma banda: ", datas);

```

Figura 57 - Código apresentado na figura 56

```

// Adiciona uma banda, na imagem, com o valor numérico da data como constante
nos pixels.
var dataImagem = function(imagem) {
  return imagem.addBands(imagem.metadata('system:time_start', 'dataNumerica'));
};
// Mapeia a função dataImagem para todas as imagens da coleção.
var datas = imagensSpColecaoSelecaoData.map(dataImagem);
// Coleção de imagens com o valor numérico das datas em uma banda adicionada.
print("Coleção com o valor numérico das datas em uma banda: ", datas);

```

Na Figura 58, uma função é criada para manipular uma lista. Desta vez, o resultado do “**map()**” é uma lista, e não uma coleção. Essa é uma característica do GEE. O “**map()**” retorna o mesmo tipo de dado do objeto que foi responsável pela sua chamada.

Figura 58 - Criando e usando uma função no GEE para uma lista

```

67 // Extrai a data da imagem.
68 var dataImagemLista = function(imagem) {
69   return ee.Image(imagem).date();
70 };
71 // Mapeia a função dataImagemLista para todas as imagens da lista de imagens.
72 var lstDatas = lstImagensSpColecaoSelecaoData.map(dataImagemLista);
73 // Lista com as datas retiradas das imagens na lista de imagens.
74 print("Lista com as datas das imagens Tier 1 no período: ", lstDatas);

```

Figura 59 - Código apresentado na figura 56

```

// Extrai a data da imagem.
var dataImagemLista = function(imagem) {
  return ee.Image(imagem).date();
};
// Mapeia a função dataImagemLista para todas as imagens da lista de imagens.
var lstDatas = lstImagensSpColecaoSelecaoData.map(dataImagemLista);
// Lista com as datas retiradas das imagens na lista de imagens.
print("Lista com as datas das imagens Tier 1 no período: ", lstDatas);

```

### 3.18 Indo além

#### 3.18.1 Extrair valores com o *reduce*

Essa função será apresentada no próximo laboratório, mas caso queira já entender seu funcionamento, continue o código no segundo arquivo criado. Entre a linha 76 e 78 da figura 60, é utilizado o padrão *reduce* para se obter o máximo valor da banda “*dataNumerica*” nas imagens da coleção. O método “*max()*” extrai o valor máximo de cada posição de pixel de uma mesma banda entre todas as imagens da coleção. Após isso, apenas a banda “*dataNumerica*” é preservada através do “*select()*” (esses métodos poderiam ser chamados de forma inversa). Como o resultado dos procedimentos anteriores é uma imagem, um *reduce* pode ser utilizado para se obter apenas um pixel da imagem (note a diferença no *reduce* na coleção e na imagem). O método “*reduceRegion()*”, aplica um *reduce* em uma região, que pode ser toda a imagem, ou determinada por uma *feature*. O redutor “*Reducer.first()*” retorna o primeiro valor da entrada, no caso o primeiro *pixel* da imagem, e como comentado anteriormente na criação dessa banda, o valor é constante.

Na linha 82 (figura 60), um *casting* para o tipo *Number* ocorre, para se preparar para o código da linha 84, onde se imprime o valor no console, com um *casting* para o tipo *Date*.

Figura 60 - Trabalhando com *reduce* no GEE

```
75 // Extrai a maior data do período.
76 var dataMax = datas.max().select("dataNumerica")
77     .reduceRegion(ee.Reducer.first(), saoPaulo, 30)
78     .get("dataNumerica");
79 // Exibe o valor da maior data no período, no tipo numérico.
80 print("Maior data no período para Tier 1 em formato numérico: ", dataMax);
81 // Converte o valor da data máxima para o tipo Number.
82 var dataMaxNumerica = ee.Number(dataMax);
83 // Exibe a maior data no período.
84 print("Maior data no período para Tier 1: ", ee.Date(dataMax));
```

Figura 61 - Código apresentado na figura 60

```
// Extrai a maior data do período.
var dataMax = datas.max().select("dataNumerica")
    .reduceRegion(ee.Reducer.first(), saoPaulo, 30)
```



### 3.18.3 Exemplos de cálculos no GEE

Na Figura 64, pode ser observado, na linha 106, como se calcula a diferença, em dias, entre duas datas. A chamada é feita a partir da data final do período, sendo a data inicial passada como parâmetro, além de uma string descrevendo qual unidade deve ser utilizada na conta. Ainda, nas linhas 109 e 110, é feita a divisão dessa diferença pelo total de aquisições, menos um, com o intuito de se calcular o número médio de dias entre as aquisições das imagens *tier 1* da *Landsat* (esse número é diferente da resolução temporal, pois exclui-se as imagens que não atingem os níveis de qualidade desejados para o *tier 1*). Os métodos utilizados são “*divide()*” e “*subtract()*”. O importante, aqui, é entender a ordem em que as operações ocorrem e que a aritmética no GEE não ocorre com operadores padrão, como em outras linguagens, por não se trabalhar com tipos primitivos de dados - como um inteiro, e sim com objetos.

Figura 64 - Efetuando cálculos com as datas das imagens

```
104 // Total, em dias, entre a data da última e da primeira aquisição no período.
105 print("Dias entre a primeira e a última aquisições Tier 1:",
106       ee.Date(dataMax).difference(ee.Date(dataMin), 'day'));
107 // Total médio de dias entre aquisições no período.
108 print("Médias de dias entre as aquisições Tier 1 no ano: ",
109       ee.Date(dataMax).difference(ee.Date(dataMin), 'day')
110       .divide(datas.size().subtract(1)));
```

Figura 65 - Código apresentado na figura 64

```
// Total, em dias, entre a data da última e da primeira aquisição no período.
print("Dias entre a primeira e a última aquisições Tier 1: ",
      ee.Date(dataMax).difference(ee.Date(dataMin), 'day'));
// Total médio de dias entre aquisições no período.
print("Médias de dias entre as aquisições Tier 1 no ano: ",
      ee.Date(dataMax).difference(ee.Date(dataMin), 'day')
      .divide(datas.size().subtract(1)));
```

### 3.18.4 Tipos armazenados nas bandas e resolução radiométrica

Na Figura 66, observa-se o método “*bandTypes()*”, na linha 114, que retorna um dicionário de dados que contém o tipo de dado de cada banda de uma imagem, no caso a primeira da coleção, por isso o uso do método “*first()*” antes da chamada de “*bandTypes()*”.

Figura 66 - Extraíndo o tipo de dado de cada banda

```

111 // Tipos de dados nas bandas da imagem.
112 print(
113     "Tipos de dados nas bandas da imagem: ",
114     colecaoLandsat7.filterBounds(saoPaulo).first().bandTypes()
115 );

```

Figura 67 - Código apresentado na figura 66

```

// // Tipos de dados nas bandas da imagem.
print(
    "Tipos de dados nas bandas da imagem: ",
    colecaoLandsat8SR.filterBounds(saoPaulo).first().bandTypes());

```

O dicionário de dados impresso no console (Figura 68) mostra em “**max**” o maior número que pode ser registrado em um *pixel* e, de forma similar, em “**min**” está o valor mínimo. Além disso, ao lado do nome da banda, o tipo de dado que cada *pixel* da imagem armazena é determinado, no exemplo é um tipo inteiro de 16 *bits*. Através dessas informações, pode-se determinar a resolução radiométrica da banda.

Figura 68 - Descrição do tipo de dado da banda B1 no console

```

Tipos de dados nas bandas da imagem:                                JSON
▼ Object (11 properties)                                           JSON
  ▼ B1: signed int16
    type: PixelType
    max: 32767
    min: -32768
    precision: int
  ▶ B2: signed int16

```

### 3.18.5 Adicionando outra coleção de imagens

Anteriormente foi ensinado como adicionar uma coleção de imagens do “*Landsat 7*”. Como exercício adicional, **importe uma coleção de imagens do “*Landsat 8*”**, mude o nome da variável do *import*, se preferir, e altere no código “*colecaoLandsat7*” para o nome da variável dado ao novo *import* (por exemplo “*colecaoLandsat8*”).

### 3.18.6 Aplicando filtros à coleção de imagens

No GEE, também é possível adicionar filtros às coleções de imagens, como por exemplo um período de tempo específico das imagens com a função ***“filterDate()”***.

Especificações deste filtro, assim como outras formas de filtragem e das variáveis utilizadas, podem ser encontradas nos *links* a seguir:

- [https://developers.google.com/earth-engine/guides/ic\\_info](https://developers.google.com/earth-engine/guides/ic_info)
- [https://developers.google.com/earth-engine/guides/ic\\_filtering](https://developers.google.com/earth-engine/guides/ic_filtering)

### 3.18.7 Importando outra coleção de imagens

Com o objetivo de visualizar outra coleção de imagens e reconhecer as diferenças nas características até então identificadas através deste roteiro, pede-se que importe outra coleção de imagens, desta vez do *Sentinel-2*. Assim, propõe-se a alteração o nome de ***“imageCollection”*** para ***“colecacaoSentinel2”*** e portanto, alterar no código (de preferência em um novo script) todas as escritas de ***“colecacaoLandsat8SR”*** para ***“colecacaoSentinel2”***.

## 4 Conclusões

Neste laboratório foi apresentada uma breve introdução ao ambiente do GEE, sua interface e seus tipos de dados. Também foi feita a criação de um simples *script* com algumas operações básicas para familiarização com os códigos de JS. Além disso, foram introduzidos aspectos importantes na busca por informações sobre os dados disponíveis no GEE, através de seus metadados. Foi também demonstrado como se cria e se utiliza uma função dentro de um *script*, além do uso dos padrões *map* e *reduce*. Finalmente, cálculos simples foram introduzidos para a familiarização com a forma que operações de cálculos são efetuadas no GEE.

## 5 Referências bibliográficas

GORELICK, N. et al. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, [s. l.], v. 202, p. 18-27, 1 dez. 2017.

GRIEBLER, D. Padrões e Frameworks de Programação Paralela em Arquiteturas Multi-Core. Porto Alegre: PUCRS, 2011. Disponível em: <http://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr064.pdf>. Acesso em: 17 ago. 2020.

MILOSLAVSKAYA, N.; TOLSTOY, A. Big data, fast data and data lake concepts. Procedia Computer Science, v. 88, p. 300-305, 2016.