[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx)

**CAPABILITIES**

The data-processing capabilities of the EE API arise from the particular abilities of individual **Operations** and the manner in which these operations are combined.

A large and well-integrated set of operations is available, and these can be invoked through either the JavaScript or Python programming languages. The use of JavaScript, however, is currently more convenient in terms of setup, documentation, and support. In JavaScript, each operation is associated with one of three different types of statement respectively called **Methods**, **Algorithms**, and **Functions**.

Most EE API operations are invoked as JavaScript **Methods**, each of which is uniquely associated with a one type of JavaScript **Object**.

A method is generally invoked by way of a statement that creates a new object from an existing object by indicating

- the name of an existing object to be processed,

- a dot,

- the name of the method to be applied to that existing object, and

- a parenthesized list of comma-separated inputs to that method.\*

NewObject **=** OldObject**.**Method **(** Input, Input, … **)**

For example, **var** newImage **=** oldImage**.**add **(** 100 **);**

Additional operations are invoked as **Algorithms**, none of which is uniquely associated with only one type of object, but each of which may call for specialized inputs.

An algorithm is invoked by way of a statement that creates a new object from one or more existing objects by indicating - the name of the algorithm to be applied, and

- a parenthesized list of comma-separated inputs to that algorithm.\*

NewObject **=** Algorithm **(** Input, Input, … **)**

For example, **var** newImage **=** ee.Algorithms.Terrain **(** oldImage **);**

EE API **Functions** are operations that perform tasks butdo not generate objects as outputs and may or may not have inputs. Theyare generally specified by giving - the name of the algorithm to be applied, and

- a parenthesized list of comma-separated inputs to that function.\*

Function **(** Input, Input, … **)**

For example,  Map.addLayer **(** **);**

\* Since each parenthesized list of comma-separated inputs will be interpreted according to the sequence in which those inputs are specified, any input for which no value is to be specified should nonetheless be explicitly given as “null” in order to maintain the ordinal position of any subsquent input. If there are no subsequent inputs to be specified, however, then this “place holding” is un necessary. For example, an operation that accepts (but doesn’t require) input on day, month, and year might be given as

( 12, 25 1950 ) or ( 12, null, 1950 ) or just ( 12, 25 ) .

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx)

[**CAPABILITIES**](#_top)

The EE API offers hundreds of operations, and one reasonable way to organize them is according to the primary type of object that each operation takes as input and/or generates as output. For this purpose, it is helpful to distinguish among three major types of EE object respectively called “widgets,” “variables,” and “parameters.”

[**Widget**](EE08%20%20%20%20%20%20%20%20%20Widgets.docx) objects represent parts of the EE user interface. These include - [**maps**](EE09%20%20%20%20%20%20%20%20%20%20%20%20Maps.docx),

- [**layers**](EE10%20%20%20%20%20%20%20%20%20%20%20%20Layers.docx),

- [**charts**](EE11%20%20%20%20%20%20%20%20%20%20%20%20Charts.docx), and

- [**sessions**](EE12%20%20%20%20%20%20%20%20%20%20%20%20Sessions.docx).

[**Variable**](EE13%20%20%20%20%20%20%20%20%20Variables.docx) objects represent data to be processed by EE. These include - [**strings**](EE14%20%20%20%20%20%20%20%20%20%20%20%20Strings.docx),

- [**numbers**](EE15%20%20%20%20%20%20%20%20%20%20%20%20Numbers.docx),

- [**arrays**](EE16%20%20%20%20%20%20%20%20%20%20%20%20Arrays.docx),

- [**lists**](EE17%20%20%20%20%20%20%20%20%20%20%20%20Lists.docx),

- [**dictionaries**](EE18%20%20%20%20%20%20%20%20%20%20%20%20Dictionaries.docx),

- [**geometries**](EE19%20%20%20%20%20%20%20%20%20%20%20%20Geometries.docx),

- [**features**](EE20%20%20%20%20%20%20%20%20%20%20%20%20Features.docx),

- [**feature collections**](EE21%20%20%20%20%20%20%20%20%20%20%20%20Feature%20Collections.docx),

- [**images**](EE22%20%20%20%20%20%20%20%20%20%20%20%20Images.docx),

- [**image collections**](EE23%20%20%20%20%20%20%20%20%20%20%20%20Image%20Collections.docx), and

- [**satellite imagery**](EE24%20%20%20%20%20%20%20%20%20%20%20%20Satellite%20Imagery.docx).

[**Parameter**](EE25%20%20%20%20%20%20%20%20%20Parameters.docx) objects represent processing specifications. These include - [**dates**](EE26%20%20%20%20%20%20%20%20%20%20%20%20Dates.docx),

- [**date ranges**](EE27%20%20%20%20%20%20%20%20%20%20%20%20DateRanges.docx),

- [**error margins**](EE28%20%20%20%20%20%20%20%20%20%20%20%20ErrorMargins.docx),

- [**selector sets**](EE29%20%20%20%20%20%20%20%20%20%20%20%20SelectorSets.docx),

- [**pixel types**](EE30%20%20%20%20%20%20%20%20%20%20%20%20PixelTypes.docx),

- [**projections**](EE31%20%20%20%20%20%20%20%20%20%20%20%20Projections.docx),

- [**filters**](EE32%20%20%20%20%20%20%20%20%20%20%20%20Filters.docx),

- [**joins**](EE33%20%20%20%20%20%20%20%20%20%20%20%20Joins.docx),

- [**reducers**](EE34%20%20%20%20%20%20%20%20%20%20%20%20Reducers.docx),

- [**kernels**](EE35%20%20%20%20%20%20%20%20%20%20%20%20Kernels.docx),

- [**classifiers**](EE36%20%20%20%20%20%20%20%20%20%20%20%20Classifiers.docx),

- [**confusion matrices**](EE37%20%20%20%20%20%20%20%20%20%20%20%20ConfusionMatrices.docx), and

- [**algorithms**](EE38%20%20%20%20%20%20%20%20%20%20%20%20Algorithms.docx).

IT IS IMPORTANT to note that these EE objects are different from more general JavaScript or Python objects of the same (or similar) names.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx)

[**CAPABILITIES**](#_top)

Each of the EE objects and its operations are more fully described in one or more separate handouts, and each of these handouts presents its operations in a format like that which is illustrated below. Note here that

- the name of the operation appears in the upperleft corner of a gray-shaded box;

- a brief description of the operation then appears to its right;

- below is what amounts to a statement syntax diagram formatted much like a JavaScript assignment statement

( though without the opening “**var**” or closing “**;**“ ) that includes

- an arbitrary name like “newString” is used to repesent an output variable,

- an arbitrary name like “oldString” is used to repesent an input variable,

- arbitrary names like “startingPosition” and stoppingPosition” repesent ordered input arguments,

- the fact that “*stoppingPosition*” is in italics means that this argument is optional, and

- a brief description of each statement component appears below; and

- beneath the gray box is an example.

string.slice creates a new string by replicating the sub-string of a specified string that starts and stops at specified ordinal positions (starting with 0).

newString = oldString.slice( startingPosition, *stoppingPosition* )

The specified position just before which the substring is to stop, given as an integer starting at 0 with negative indices counted backwards from the list’s end

The specified position at which the substring is to start, given as an integer starting at 0 with negative indices counted backwards from the list’s end

The new string

The specified string

var OldSTRING = ee.String( "What does every fine artist hope for?" );

var NewSTRING = OldSTRING.slice( 11, 24 );

print( NewSTRING );

When reading an example, note that

- lighter text is often used to focus attention on the particular operation being demonstrated;

- the prefixes “Old” and “New” are often used to distinguish between input and output objects;

- UPPERCASE text is often used in object names to indicate object type;

- statements are often vertically aligned with with one another in order to improve to legibility at a glance;

- individual operations tend to be presented in separate statements in order to more clearly articulate the role played by each.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx)

[**CAPABILITIES**](#_top)

When multiple input arguments are specified, the order in which they appear determines the manner in which they are interpreted. For this reason, it is not uncommon to see a a sequence of input arguments given as something like ( 1, 6, null, 5, null, 7 ) where “null” is used as a place-holder in order to repesent tha absence of an argument while assuring that subsequent arguments appear in their proper ordinal positions.

It is possible to avoid this argument-ordering requirement altogether by instead specifying input arguments as illustrated in the example below.

var OldSTRING = ee.String( "What does every fine artist hope for?" );

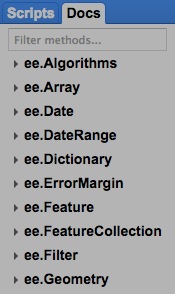
var NewSTRING = OldSTRING.slice( {end:24, start:11} );

print( NewSTRING );

Note here that was earlier specified as ( 11, 24 ) is now given as ( {end:24, start:11} ) .

What had been an prdered list of argument values is now a dictionary of *keyword:value* pairs, where the keywords called for are *start* and *end*

and the order they are specified no longer matters.

To employ this alternative input argument format, you must know the required keywords. Though they are not generally noted in this documentation (where we have instead opted for more descriptive terms in referring to input arguments), they are noted in the online documentation available through the **Docs** tab

in the Reference Panel of the EE API. Here, for example, the entry for the **slice** operation explicity identifies **start** and **end**

as the keywords associated with

that operation’s two input arguments.

