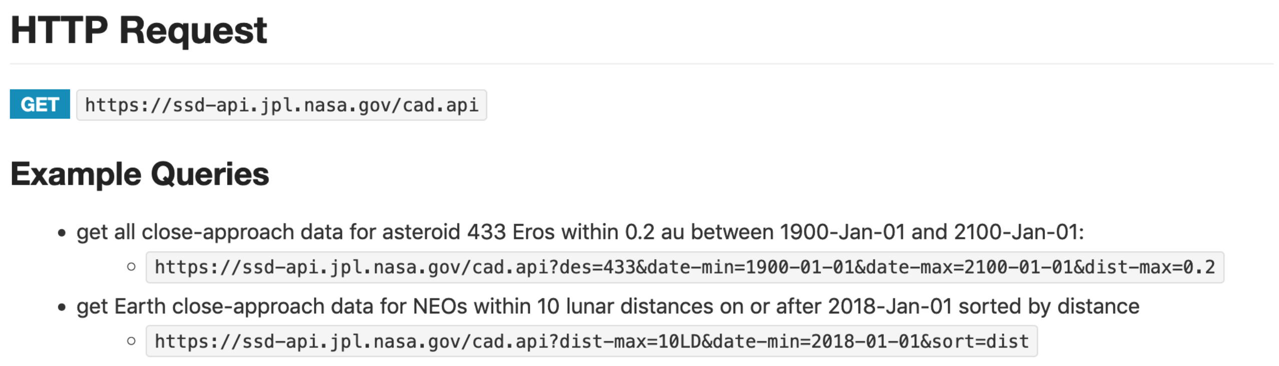
Making your own API client

In this lab, you will make your own API client.

We use the NASA API for comets and asteroids that approach close to Earth (SBDB Close Approach Data API). The API documentation in the link given shows the parameters and acceptable values, and the output data which is in JSON format.

Take a look at the API site above. You will notice the following documentation (Figure 1)



HTTP Request

GET `https://ssd-api.jpl.nasa.gov/cad.api`

Example Queries

- get all close-approach data for asteroid 433 Eros within 0.2 au between 1900-Jan-01 and 2100-Jan-01:
  - `https://ssd-api.jpl.nasa.gov/cad.api?des=433&date-min=1900-01-01&date-max=2100-01-01&dist-max=0.2`
- get Earth close-approach data for NEOs within 10 lunar distances on or after 2018-Jan-01 sorted by distance
  - `https://ssd-api.jpl.nasa.gov/cad.api?dist-max=10LD&date-min=2018-01-01&sort=dist`

Fig 1: Documentation for the NASA Close Approach Data API

This is typical of most APIs:

1. It tells you that the API calls need to be made using a `http-get` (as opposed to a `http-post`).
2. It gives you two sample calls using the API. You should be able to use your browser on either url (`https://ssd-api.jpl.nasa.gov/cad.api?des=433&date-min=1900-01-01&date-max=2100-01-01&dist-max=0.2` and `https://ssd-api.jpl.nasa.gov/cad.api?dist-max=10LD&date-min=2018-01-01&sort=dist`

You should try clicking on each link and reading the documentation before proceeding. The sample links tell us what **URL** to use (`https://ssd-api.jpl.nasa.gov/cad.api`) and what **parameters** are supported in making the API call: `des` , `date-min` and `date-max` , `dist-max` and `sort` . In fact, there are many other supported parameters for this call, listed in the Query section of the site's API documentation.

Making the API Client

There are 4 steps in making an API client:

1. **Prepare the parameters** to make for the API call. Again these should be documented on the API provider (eg NASA) site.
2. **Make the API request** using `http-get` or `http-post` . For the NASA, we know to use `http-get` .
3. **Read the incoming data** into JSON format using `json>` .
4. **Convert** the elements of the JSON document into data types that Smojo understands - hashes, sequences, numbers, dates and strings. This is done using `>array` , `>seq` , `>long` and `>double` .

http-get / http-post

`http-get` and `http-post ( "url" -- "s" | null )` are the Smojo words for making HTTP requests. These words are in the `*utils` module. To use them you need the lines `include utils` and `with *utils` . `close-approach.m` contains an example of how this is used.

In both functions, the URL of the API is given as an input. The API response is then returned **as a string**, or a `null` if the API returns a bad status (status code not equals to 200).

You can find out the correct URL to use from the API's site documentation.

Parameters

In addition to the URL, you need to tell the API what you want. This information is contained in the request's **parameters** or `params` for short.

Each parameter is a key - value pair. A single parameter is declared using the `param ( "key" "value" -- )` word.

Before starting to declare the parameters for an API call, you should clear all previously defined parameters from previous calls using the word `clear-params ( -- )` .

Note that all parameters have to be declared **before** using `http-get` or `http-post` .

You should find out the exact parameters supported from the API site's documentation.

JSON

JSON is a common format used to store and transfer data in many APIs. JSON has **objects** and **arrays** that are similar to **hashes** and **tuples** in Smojo. As such the word `json> ( "s" -- #|tuple|null )` is provided for easy conversion from the JSON string format to a hash or tuple (`null` is returned on error).

JSON Parsing - Data Types

The output of `json>` requires special handling of array and numerical values, due to the different data types used. After retrieving any array or numbers from JSON and before using them, make sure to convert them to a usable type with these words:

1. `>array ( json-array -- tuple )`
2. `>seq ( json-array -- sequence )`
3. `>long ( json-number -- integer )`
4. `>double ( json-number -- real-number )`

These are functions that convert raw JSON data from `json>` into data that you can use in Smojo.

Let's see how all these are put together.

Writing the API Client

First of all, take a look at the NASA API for comets and asteroids that approach close to Earth (SBDB Close Approach Data API) if you haven't already done so.

Step 1: Write the code to make the `http-get` call and conversion using `json>`

```
: api-get ( "url" -- #|seq|null )
    http-get dup null? -> exit |.
    json>
;
```

This word takes in a URL and outputs a JSON object. It assumes that the parameters have already been set prior to it being called. So we need to put in the parameters, and specify the URL to call.

Step 2: There are many parameters supported by this API, so we will just use 2 to illustrate - `fullname` which gives the full name of the object and `diameter` of the object. Both these are booleans that need to be set to `true` or they will not be retrieved.

Here is some code to do this. We'll call this word close-approach-objects

```
: close-approach-objects ( -- seq-json|null)
    clear-params
    "diameter" "true" param
    "fullname" "true" param
    "https://ssd-api.jpl.nasa.gov/cad.api" api-get dup null? -> exit |. { h }
    "data" h #@ >seq
;
```

Note the last two lines - the json data from `api-get` is stored in a hash `h` . Then just the value of `"data"` on `h` is passed to `>seq` for conversion. This result is returned by this word.

The need to use the `>seq` conversion is specific to the NASA API, and is because the "data" section (see listing 1 below), is an array of arrays. For other APIs, this will be different, but you can convert from JSON to Smojo using the conversion words. The NASA format (again, see their documentation) is as below:

```
{
  "signature":{"source" : "NASA/JPL SBDB Close Approach Data API","version" : "1.3"},
  "count":"3",
  "fields":["des","orbit_id","jd","cd","dist","dist_min","dist_max","v_rel","v_inf","t_sigma_f",'
  "data":[
    ["153814","174","2461948.724524223","2028-Jun-26 05:23","0.00166253924938707","0.001662376727
    ["99942","206","2462240.407091595","2029-Apr-13 21:46","0.000254099098170977","0.000254085852
    ["2001 AV43","42","2462452.142037054","2029-Nov-11 15:25","0.00209271674918052","0.002091251
   ]
}
```

Listing 1: The NASA Json document.

The `data` section on this json document contains arrays, one for each close-approach object. We convert the json-array to a Smojo sequence using `>seq` .

Step 3: Each element is a single close-approach object, whose details are contained in a Json array. From Listing 1, you can see that you also need to know the position of the information within the Json array. I've read the NASA docs, and I know that:

1. Index **3** contains the date of close approach to Earth, eg "2028-Jun-26 05:23" in Listing 1.
2. Index **4** contains the distance of closest approach in Astronomical Units (AU),
3. Index **13** contains the official name of the object.

We want to write accessor functions to get these information: `date@` , `distance@` and `name@` . The `...@` is a convention to signify that these are "getters" to read data.

```
: date@ ( tup -- "s" ) 3 @@ ;
: distance@ ( tup -- n ) 4 @@ real "0.000'AU'" format ;
: name@ ( tup -- "s" ) 13 @@ trim ;
```

Note that I have used the Smojo `real ( "s" -- n )` converter function. This is because NASA chose to store distances in strings, not json-doubles.

There is no need to convert date and name as we want to leave these as strings.

Step 4: We need to print out the results from the sequence. For simplicity, I'll only print the first 5 objects from the sequence.

```
: print ( json-array -- ) { xs }
  xs name@ . cr
  xs date@ . xs distance@ . cr
  cr
;

: print-all ( seq-json-array -- )
   5 take [: >array print ;] reduce
;
```

I've put these into a convenient file for you, `closest-approach.m` . Run the program to get the display.

* * *