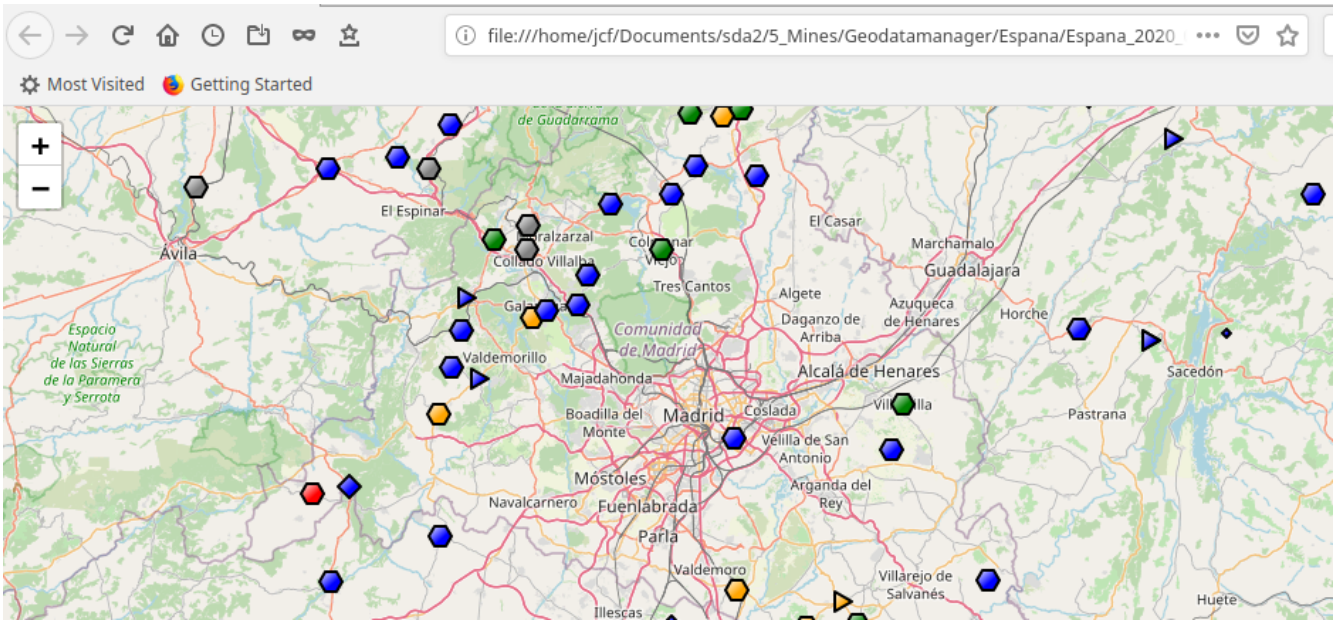
Do I need Folium to create my maps?

By jean-claude.feltes@education.lu

The quick answer, as I see it now is: no. But I must say that, at the beginning, Folium was much easier, as I am a Python, and not a Javascript programmer.

1. The Folium solution

Some time ago I started writing a program that should display the localities that are interesting for a mineral collector on a map. This was quite easy with Folium, and the result looked like this:



The map was displayed in a browser, and it used OSM map tiles. The markers could use different colours, they were polygons with a number of sides that could be decided by the program.
When I clicked on a marker, the accompanying text was displayed.
This text was derived from a “data base” that in reality was a text file containing coordinates, names and free text. Like this:

```
#-----------------------------------------------------------------
# DATA
# Latitude (Y)          Longitude (X)          Locality               Free text
## 37.1636295           -2.829291              San José, Almería, Andalucía, España Position falsch??
38.76068412528926       -3.771889174995498                            Volcán de Columba                                //
38.83278114551823       -3.742916318759674                            Volcán de Cerro Gordo                            //
38.95961998128244       -3.994911293824348                            Cantera Arzollar                                 //
38.85424067572104       -3.667608641892098                            Volcán Yezosa                                    //
```

The map was created like this:

```
mymap = folium.Map(location=[centerLat, centerLong],zoom_start = zoom)
```

To add markers, I did this:

```
m=folium.RegularPolygonMarker(location=(Latitude, Longitude), fill_color=color,
                number_of_sides=nsides, radius=radius, rotation=0,  popup=description[:maxdescriptionlength])
m.add_to(mymap)
```

Then I realized that after creating the map
I could add tiles from other sources, especially from Opentopomaps and ESRI sattelite images:

```
tile_topomap = folium.TileLayer(
    tiles = 'https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png',
    attr = 'Kartendaten: © OpenStreetMap-Mitwirkende, SRTM | Kartendarstellung: © OpenTopoMap (CC-BY-SA)',
    name = 'Opentopomap',
    overlay = False,
    control = True
   ).add_to(mymap)


tile_stamen = folium.TileLayer('Stamen Terrain').add_to(mymap)

tile_satellite = folium.TileLayer(
    tiles = 'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}',
    attr = 'Esri',
    name = 'Esri Satellite',
    overlay = False,
    control = True
   ).add_to(mymap)

ctrl = folium.LayerControl().add_to(mymap)
```

To use Opentopomaps and ESRI images, an attribution attr must be included!
The last line adds a comfortable layer control.

The possibility to use Opentopomaps was really exciting, as I could zoom into the map and eventually discover mine symbols (that are most interesting for a mineral collector). Now I would need the possibility to click on the map and directly get the coordinates to the clipboard in my own format, to put them into my “data base”.
This could be done in Folium, but it was a little bit tricky, as a Javascript template had to be used:

```
el = folium.MacroElement().add_to(mymap)

s1 = """
{% macro script(this, kwargs) %}
// write JS here
var lat_lng_popup = L.popup();
    function latLngPop(e) {

        navigator.clipboard.writeText(e.latlng.lat.toFixed(6) + "\t" + e.latlng.lng.toFixed(6)  + "\t").then(function() {
        }, function() {
            /* clipboard write failed */
        });
        lat_lng_popup
            .setLatLng(e.latlng)
            .setContent(e.latlng.lat.toFixed(6) + ", " + e.latlng.lng.toFixed(6))
            .openOn("""
s2 = map_js
s3 = """);
                }
"""
s4 = map_js
s5 = """.on('click', latLngPop);

{% endmacro %}
"""
s = s1 + s2 + s3 + s4 + s5
el._template = jinja2.Template(s)
```

(The string handling could be done in a more pythonic way, but I did this quick and dirty.)

The biggest part of the green text is Javascript code that handles the click event and puts the coordinates to the clipboard, separated by tabs, which is the format I needed.

This was a big improvement.
But one thing was missing on my map: a search control where I could enter a locality name and have the map moved to this place.
I did some research on the net, but could not find a Folium solution to this problem. Astonishing!
Maybe my search was not deep enough and I missed something. But then another idea came to me: Did I really need Folium?
Folium is essentially a wrapper to the Leaflet library, written in Javascript.

I could analyze the generated code and write my own map library. (This would not be a big job, as I do not need all the possibilities that Folium or Leaflet offer. I could just focus on my own needs.)

2. My map library

This should offer the possibilities I described above, with the possibility to use it in a very simple way, like this:

```
markers = [(50.0312,5.8559,'red', 5, 'Marker 1'),
                    (49.95,5.8559,'blue', 4, 'Blah Marker 2'),
                    ]
mapfile = 'testmap.htm'
center_lat, center_lon = 49.5, 6.2
map.create_map(center_lat, center_lon, mapfile, markers )
```

This piece of code creates a map with a layer control (for OSM, Opentopomap, Stamen Terrain and ESRI images) and a search control. The map is centered at center_lat, center_lon.
The result is written to the file ‘testmap.htm’ and automatically displayed in a browser window.
Markers can be added as an array consisting of tuples
(latitude, longitude, fill colour, number of polygon sides, free text)

Internally this is done by the following functions that can also be used instead of the simple approach:

```
# A map can be created like this (in detail):
'''
_create_header()
_begin_script()
_create_map(49.6, 6.15)          # center lat, long
_add_layers()
_add_clickhandler()
_add_searchcontrol()

create_marker(49.9,5.559,'black', 4, 'Blah')
add_markers(markers)

_end_script()

save_map("test_map.htm")
show_map()
'''
```

The _create_header() function does the HTML part of the job:

```
def _create_header():
    global mapstring
    header="""<!DOCTYPE html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
        <script>L_PREFER_CANVAS=false; L_NO_TOUCH=false; L_DISABLE_3D=false;</script>
        <script src="https://cdn.jsdelivr.net/npm/leaflet@1.3.4/dist/leaflet.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/Leaflet.awesome-markers/2.0.2/leaflet.awesome-markers.js"></script>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/leaflet@1.3.4/dist/leaflet.css"/>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"/>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css"/>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.3/css/font-awesome.min.css"/>
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/Leaflet.awesome-markers/2.0.2/leaflet.awesome-markers.css"/>
        <link rel="stylesheet" href="https://rawcdn.githack.com/python-visualization/folium/master/folium/templates/leaflet.awesome.rotate.css"/>
        <style>html, body {width: 100%;height: 100%;margin: 0;padding: 0;}</style>
        <style>#map {position:absolute;top:0;bottom:0;right:0;left:0;}</style>

        <meta name="viewport" content="width=device-width,
            initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
        <style>#mymap {
            position: relative;
            width: 100.0%;
            height: 100.0%;
            left: 0.0%;
            top: 0.0%;
            }
        </style>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet-dvf/0.3.0/leaflet-dvf.markers.min.js"></script>

         <!-- CSS and JS files for Search Box -->
        <script src="https://cdn-geoweb.s3.amazonaws.com/esri-leaflet/0.0.1-beta.5/esri-leaflet.js"></script>
        <script src="https://cdn-geoweb.s3.amazonaws.com/esri-leaflet-geocoder/0.0.1-beta.5/esri-leaflet-geocoder.js"></script>
        <link rel="stylesheet" type="text/css" href="https://cdn-geoweb.s3.amazonaws.com/esri-leaflet-geocoder/0.0.1-beta.5/esri-leaflet-geocoder.css">


    </head>
    <body>
        <div class="mymap" id="mymap" ></div>
    </body>
    """
    mapstring += header
```

Here the script instructions for the search box are included. I did not find a way to do this in Folium.

The header is put into the global variable mapstring.

After this the script part begins:

```
def _begin_script():
    '''Begin Script (Javascript) part of the HTML '''
    global mapstring
    mapstring += "<script>"
```

The map is created in Javascript:

```
def _create_map(center_latitude, center_longitude):
    global mapstring

    mapdefinition = """
        var bounds = null;


        // MAP
        var mymap = L.map(
            'mymap', {
            center: [$clat, $clong],
            zoom: 10,
            maxBounds: bounds,
            layers: [],
            worldCopyJump: false,
            crs: L.CRS.EPSG3857,
            zoomControl: true,
            });
    """
    lat = str(center_latitude)
    lon = str(center_longitude)

    # create a template of the blueprint and replace variables:
    m_template = Template(mapdefinition)
    mapcode = m_template.safe_substitute(clat = lat, clong = lon)
    mapstring += mapcode
```

To do this I use a template string mapdefinition, where the variables $clat and $clong for center latitude and longitude are substituded by the Python variables center_latitude and center_longitude.
To do this I use the safe_substitute function of the Template part of the string library (that I import at the beginning of the program):

```
from string import Template
```

Then the tile layers are added:

```
def _add_layers():
    ''' Add layers Openstreetmap, Opentopomap, Stamenterrain, Esri satellite images
        and layer control
    '''
    global mapstring

    tile_layers = """
        // TILE LAYERS
        var tile_layer_001 = L.tileLayer(
            'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
            {
            "attribution": null,
            "detectRetina": false,
            "maxNativeZoom": 18,
            "maxZoom": 18,
            "minZoom": 0,
            "noWrap": false,
            "opacity": 1,
            "subdomains": "abc",
            "tms": false
        }).addTo(mymap);
```

….the same for layers Opentopomap, Stamenterrain and Esri.
The attribution must not be forgotten!. For Opentopomap for example:

```
var tile_layer_002 = L.tileLayer(
'https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png',
{
"attribution": "Kartendaten: \u00a9 OpenStreetMap-Mitwirkende, SRTM | Kartendarstellung: \u00a9 OpenTopoMap (CC-BY-SA)",
"detectRetina": false,
```

At the end a layer control is added:

```
        var layer_control = {
            base_layers : { "openstreetmap" : tile_layer_001,"Opentopomap" : tile_layer_002,
                            "stamenterrain" : tile_layer_003,"Esri Satellite" : tile_layer_004, },
            overlays : {  }
            };
        L.control.layers(
            layer_control.base_layers,
            layer_control.overlays,
            {position: 'topright',
             collapsed: true,
             autoZIndex: true
            }).addTo(mymap);

            tile_layer_002.remove();
            tile_layer_003.remove();
            tile_layer_004.remove();
    """
    mapstring += tile_layers
```

The removement of layers 002...004 has the effect that only tiles from layer 001 are loaded. Otherwise all layers would be loaded one above the other, so that only the last one would be visible.

The click handler adds click and copy faculty:

```
def _add_clickhandler():
    global mapstring
    '''This clickhandler puts coordinates in the form latitude \t longitude \t
    to the clipboard after showing them on the map'''

    click_coordinates_handler = """

        // Click and get coordinates
        var lat_lng_popup = L.popup();
            function latLngPop(e) {

                navigator.clipboard.writeText(e.latlng.lat.toFixed(6) + "⟶" + e.latlng.lng.toFixed(6) + "⟶").then(function() {
                 }, function() {
                    /* clipboard write failed */
                });
                lat_lng_popup
                    .setLatLng(e.latlng)
                    .setContent(e.latlng.lat.toFixed(6) + ", " + e.latlng.lng.toFixed(6))
                    .openOn(mymap);
                    }
        mymap.on('click', latLngPop);
    """
    mapstring += click_coordinates_handler
```

I find this code more straightforward than the Folium template.

After the necessary scripts are included in the header part of the HTML, the search control is very easy to do:

```
def _add_searchcontrol():
    global mapstring
    searchcontrol="""
    var searchControl = new L.esri.Controls.Geosearch().addTo(mymap);
    """
    mapstring += searchcontrol
```

One marker can be created with this:

```
def create_marker(latitude, longitude, color, nbsides, description):
    global mapstring
    global index          # unique identifier
    index +=1             # automatically created

    # this is the blueprint for the Javascript code of the marker
    # later, variables $.... will be replaced
    marker_blueprint = """var regular_polygon_marker_$i = new L.RegularPolygonMarker(
                    new L.LatLng($Latitude, $Longitude),
                    {
                        icon : new L.Icon.Default(),
                        color: 'black',
                        opacity: 1,
                        weight: 2,
                        fillColor: '$mcol',
                        fillOpacity: 0.5,
                        numberOfSides: $nsides,
                        rotation: 0,
                        radius: 8
                        }
                    ).addTo(mymap);


                    var popup_$i = L.popup({maxWidth: '300' });


                    var html_$i = $(`<div id="html_$i" style="width: 100.0%; height: 100.0%;"> $text</div>`)[0];
                    popup_$i.setContent(html_$i);


                    regular_polygon_marker_$i.bindPopup(popup_$i)
                ;
    """

    Lat = str(latitude)
    Long = str(longitude)
    ind = str(index)

    # create a template of the blueprint and replace variables:
    m_template = Template(marker_blueprint)
    markercode = m_template.safe_substitute(Latitude = Lat, Longitude = Long, mcol = color, nsides = nbsides, text = description, i = ind)
    mapstring += markercode
```

Here an index variable is included so that every marker has a distinct identity. Maybe this is not necessary, the code seems to work without it, but as Folium takes great care to generate unique identifiers, I thought it would be a good idea to do this also.

Once the function for one marker is defined, it is easy to use an array of markers:

```
#-------------------------------------------------------------
def add_markers(markers):
    '''Add markers to the map
    markers = [marker1, marker2, ....]
    with markerx = (Latitude, Longitude, color, nbsides, text)
    color can be 'red', 'blue' etc.
    '''
    global mapstring
    for marker in markers:
        lat, lon, col, nbsides, text = marker
        create_marker( lat, lon, col, nbsides, text )
```

The last thing to do is to end the script part of the HTML code:

```
def _end_script():
    '''End Script (Javascript) part of the HTML '''
    global mapstring
    mapstring += "</script>"
```

Now the mapstring contains all the HTML and Javascript information that is needed.

It has only to be written to a file:

```
mapfile = ""
def save_map(map_file):
    '''Save map as HTML file
    variable mapfile is used to remember the filename'''
    global mapfile
    print("Writing map to " + map_file)
    with open(map_file, "w") as f:
        f.write(mapstring)
        f.close()

    mapfile = map_file          # Remember filename
```

The filename is remembered in the global variable mapfile, so that this can be used by the next function, the one displaying the resulting map in a browser window:

```
def show_map():
    ''' Use browser to show the current map'''
    import webbrowser
    webbrowser.open_new(mapfile)
```

The result is the same as with Folium, except that I have the desired Search control on my map.
And I have full control of the different steps, so I can adapt my code as I need it.
It should not be too difficult to add new functionalities, as there are many good Leaflet tutorials on the net.

If you find this article helpful, if you have ideas for improvements or if you want to have the code, feel free to contact me.