Table of Contents

1 Basic Map Parameters
2 Basic Map
3 Create a Split Map
4 Reading the data as provided by the gadget (there are always two types of files - tracks and ratings of spots)
5 Frequency of categories 1 - 5
6 Adding tree- and rubish bin locations to the map
7 Grouping of markers (not really useful in this context)
8 The resulting HTML file should be visible in any browser (however, icons will be missing - fixible)
9 Some cleaning up if needed
10 Integrating a different basemap with more details
11 For later: Experimenting with coloring regions or neighborhoods

In [1]:

```
# not all of these libraries are needed but for now let's load them to be prepared :-)
from ipyleaflet import (Map, GeoData, WidgetControl, GeoJSON, basemaps, LocalTileLayer,
 LayersControl, Icon, Marker,basemap_to_tiles, Choropleth, AntPath,
 MarkerCluster, Heatmap, SearchControl, FullScreenControl, AwesomeIcon,
 ScaleControl, MeasureControl, SplitMapControl, WMSLayer, Polygon, Choropleth)
```

In [2]:

```
# ipywidgets add user interactions to our notebook cells
# read the docs https://ipywidgets.readthedocs.io/en/latest/
from ipywidgets import Text, HTML, IntSlider, ColorPicker, jslink, Layout
from branca.colormap import linear
```

In [3]:

```
# something to look into when wanting to color areas of a city or a region (Choropleth Map
# https://blog.datawrapper.de/choroplethmaps/
#import geopandas as gpd
#import json
```

In [4]:

```
# Pandas = derived from "Python and data analysis"
import pandas as pd
```

1 Basic Map Parameters

In [5]:

```
# the centre of your map should be about the starting point of your mapping tour
center = [48.231139, 16.374955]
zoom =16

# you can adjust the map size via ipwidgets *Layout* attribute
basemap = basemaps.Esri.WorldStreetMap
layout = Layout(width='100%', height='600px')

"""
alternative options for basemaps are
basemap = basemaps.Stamen.Watercolor
basemap = basemaps.Stamen.Toner
basemap = basemaps.Stamen.Terrain
etc

"""
```

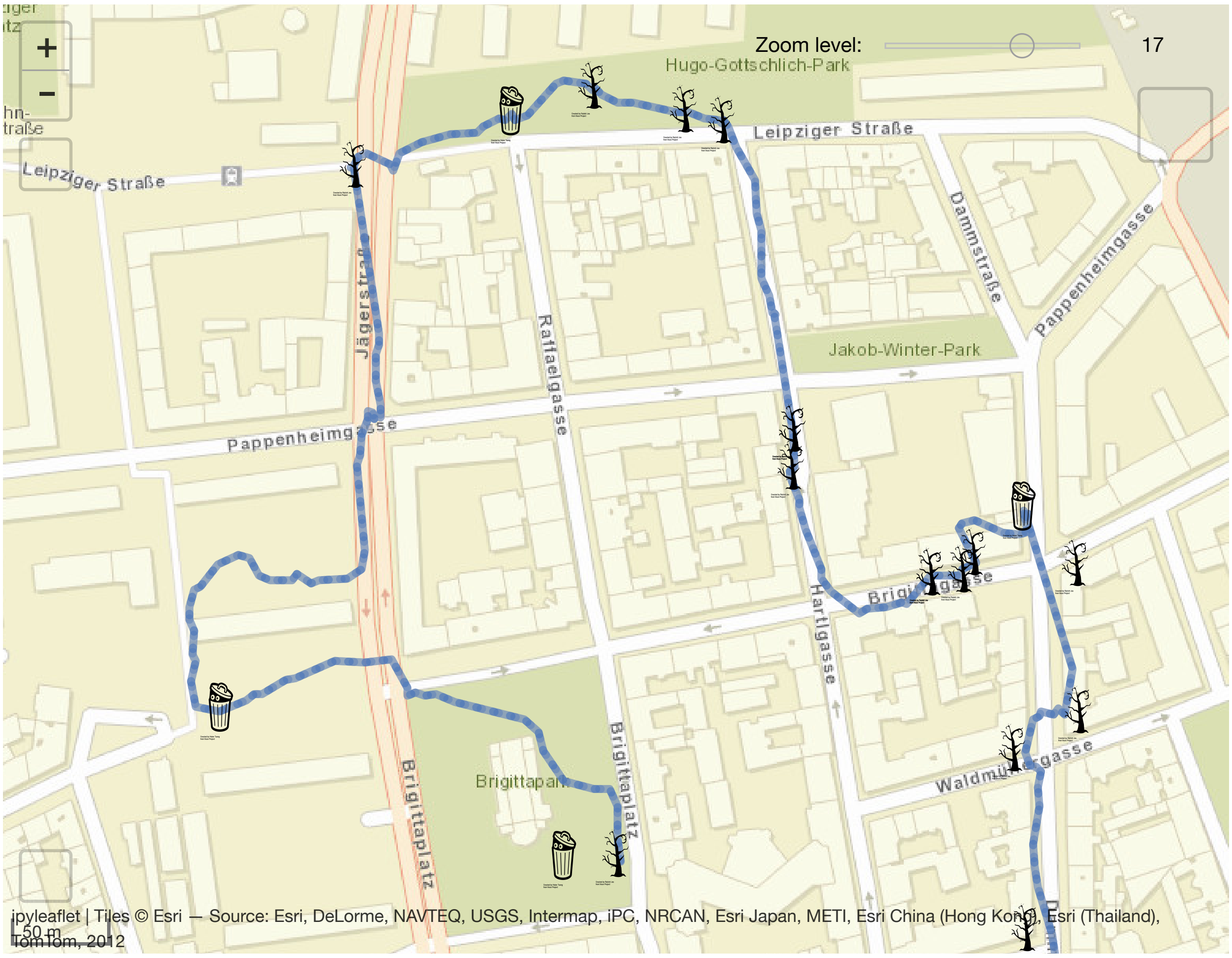
Out[5]:

```
'\nalternative options for basemaps are \nbasemap = basemaps.Stamen.Watercolor\nbasemap = ba
semaps.Stamen.Toner\nbasemap = basemaps.Stamen.Terrain\netc \n\n'
```

2 Basic Map

In [6]:

```
m = Map(center=center, zoom=zoom, basemap = basemap, layout=layout)

# add user interaction / user information such as the scale of a map
zoom_slider = IntSlider(description='Zoom level:', min=10, max=20, value=16)
jslink((zoom_slider, 'value'), (m, 'zoom'))

widget_control1 = WidgetControl(widget=zoom_slider, position='topright')
m.add_control(widget_control1)
m.add_control(FullScreenControl())
m.add_control(ScaleControl(position='bottomleft', imperial = False))
m.add_control(LayersControl(position='topright'))

# this adds a nice feature to meassure the length of a path or a polygone area in square m
measure = MeasureControl(
    position='bottomleft',
    active_color = 'orange',
    primary_length_unit = 'meters',
    primary_area_unit = 'sqmeters',
    completed_color = 'blue'
)
m.add_control(measure)
display (m)
```

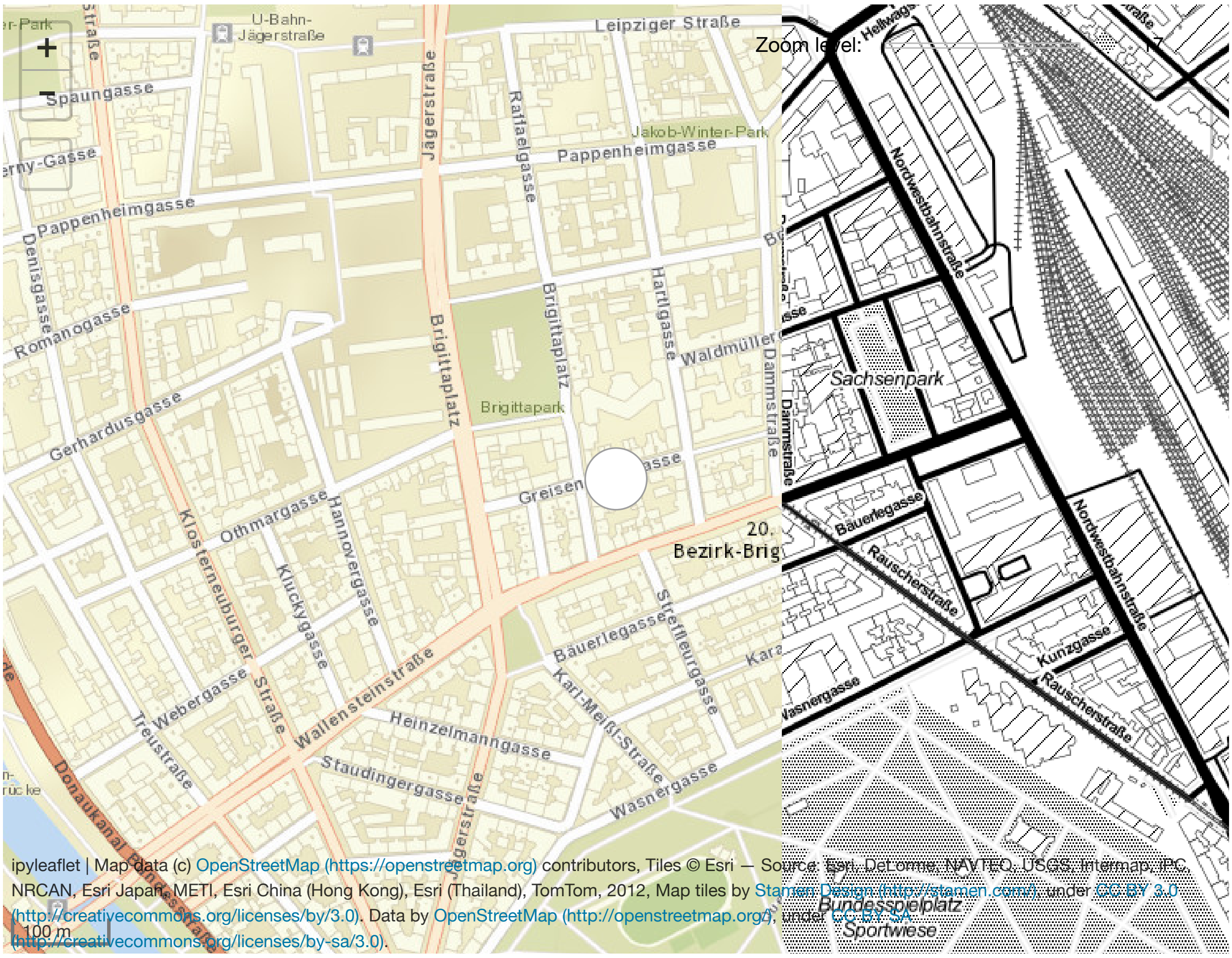


ipyleaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

3 Create a Split Map

In [7]:

```
basemap = basemaps.Esri.WorldStreetMap
split_map = Map(center=center, zoom=zoom, layout=layout)

# create right and left layers
left_layer  = basemap_to_tiles(basemap=basemap)
right_layer = basemap_to_tiles(basemap=basemaps.Stamen.Toner)

# create split control
control = SplitMapControl(left_layer=left_layer, right_layer=right_layer)

#add control to map
split_map.add_control(control)

# display map

zoom_slider = IntSlider(description='Zoom level:', min=10, max=20, value=16)
jslink((zoom_slider, 'value'), (m, 'zoom'))

widget_control1 = WidgetControl(widget=zoom_slider, position='topright')
split_map.add_control(widget_control1)
split_map.add_control(FullScreenControl())
split_map.add_control(ScaleControl(position='bottomleft', imperial = False))
split_map.add_control(LayersControl(position='topright'))

display(split_map)
```



ipyleaflet | Map data (c) OpenStreetMap (https://openstreetmap.org) contributors, Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012, Map tiles by Stamen Design (http://stamen.com/), under CC BY 3.0 (http://creativecommons.org/licenses/by/3.0). Data by OpenStreetMap (http://openstreetmap.org/), under CC BY SA (http://creativecommons.org/licenses/by-sa/3.0).

4 Reading the data as provided by the gadget (there are always two types of files - tracks and ratings of spots)

In [8]:

```
ratings=pd.read_csv('data/2020-10-12_rate_waste.csv', sep=';')
tracks=pd.read_csv('data/2020-10-12_track_waste.csv', sep=';')
print('number of track points: ', len (tracks))
tracks.head(3)
```

```
number of track points:  500
```

Out[8]:

	Date	Time	Raw_Time	Latitude	Longitude	Altitude	Sats	Sat_Speed	Precision
0	121020	07:12:47	6124700	48.2311	16.375006	160.2	4	0.5926	6.51
1	121020	07:12:47	6124700	48.2311	16.375006	160.2	4	0.5926	6.51
2	121020	07:12:47	6124700	48.2311	16.375006	160.2	4	0.5926	6.51

5 Frequency of categories 1 - 5

In [9]:

```
ratings.groupby(['Category']).size()
```

Out[9]:

```
Category
1    13
2     4
4     5
dtype: int64
```

In [10]:

```
# at this point we add some meaning to the categories
tree_waste = ratings[ (ratings['Category'] == 1) | (ratings['Category'] == 2) ]
bin_waste = ratings[ ratings['Category'] == 4]
print ('number of polutted trees: ', len(tree_waste))
print ('number of overflowing bins: ', len(bin_waste))
```

```
number of polutted trees:  17
number of overflowing bins:  5
```

In [11]:

```
# a collection of points (GPS coordinates) needs to be provided as a list of lists, this i
def location_converter (df):
    markers = df.loc[:,{'Latitude','Longitude'}] #ouput dataframe
    markers = markers.reindex(columns = ['Latitude','Longitude']) #
    markers = markers.to_records(index=False) #output array
    markers = list (markers) #output list of tuples
    markers = [list(i) for i in markers] # list of lists
    return markers

```

In [12]:

```
# calling the funnnction and checking outpu
tree_pos = location_converter (tree_waste)
bin_pos = location_converter (bin_waste)
track_pos = location_converter (tracks)
tree_pos [0:2]
```

Out[12]:

```
[[48.231136, 16.374966], [48.231619, 16.374845]]
```

In [17]:

```
import os
os.getcwd()
```

Out[17]:

```
'/Users/me/code/notebooks/ipyleaflet'
```

6 Adding tree- and rubish bin locations to the map

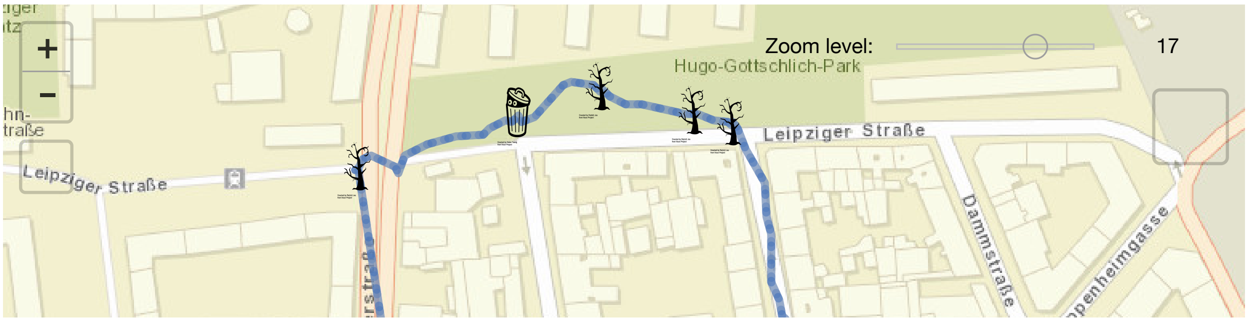
In [18]:

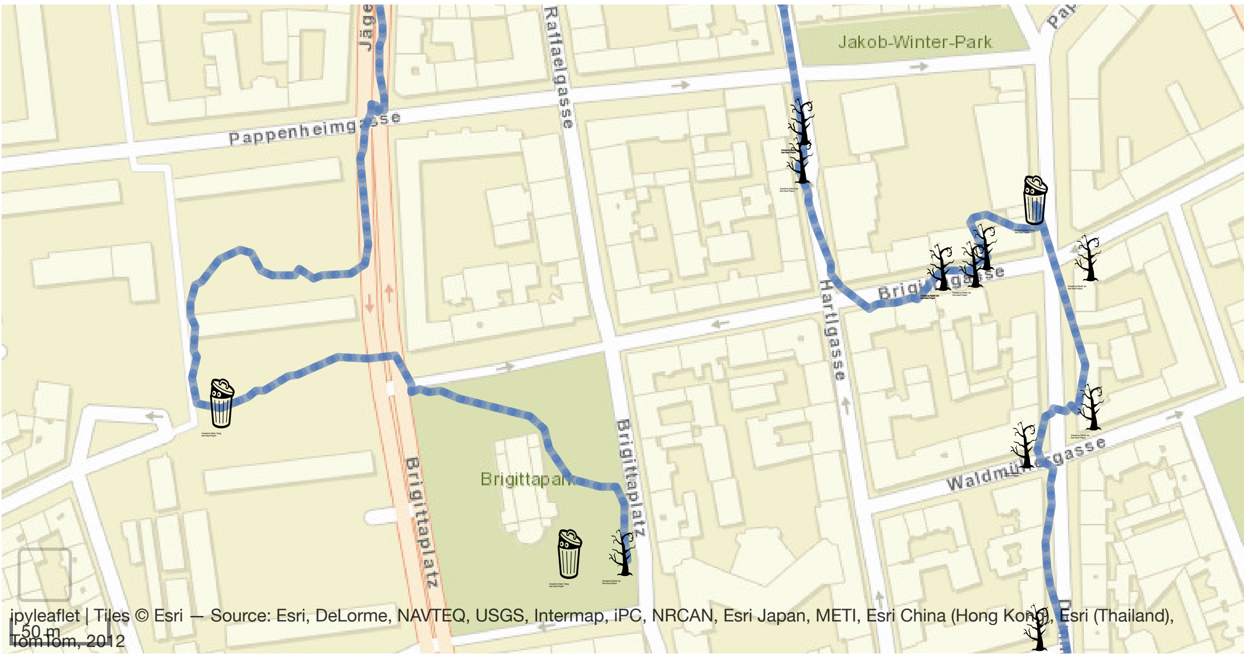
```
print (get_ipython().__class__.__module__)
```

```
ipykernel.zmqshell
```

In [19]:

```
#trash_icon = AwesomeIcon (name='trash', marker_color='white', icon_color='black', spin=Tr
#trash_icon = Icon(icon_url='https://leafletjs.com/examples/custom-icons/leaf-red.png', ic

bin_markers = []
tree_markers = []


#for jupyter notebook
trash_icon = Icon(icon_url= 'icons/trashbin.png', icon_size=[30, 40])
tree_icon =  Icon(icon_url= 'icons/tree.png', icon_size=[30, 40])

'''
#for jupyter labs
trash_icon = Icon(icon_url='files/'+os.getcwd().split('/')[-1]+'/icons/trashbin.png', icon
tree_icon = Icon(icon_url='files/'+os.getcwd().split('/')[-1]+'/icons/tree.png', icon_size
'''
#external icons
#icon = Icon(icon_url='https://leafletjs.com/examples/custom-icons/leaf-green.png', icon_s


# Just to see what os.path returns?
'''
try:
    print("File exist: ", os.path.isfile(foot_icon))
except:
    print(foot_icon)
'''


for i in range(len(bin_pos)):
    markertrash  = Marker(location=bin_pos[i], icon = trash_icon)
    m.add_layer(markertrash);
    #bin_markers = bin_markers + marker


for i in range(len(tree_pos)):
    markertree  = Marker(location=tree_pos[i], icon = tree_icon)
    m.add_layer(markertree);
    #tree_markers = tree_markers + marker

'''
for i in range(len(track_pos)):
    marker  = [Marker(location=track_pos[i], icon = foot_icon)]
    foot_markers = foot_markers + marker


bin_markers = tuple (bin_markers)
tree_markers = tuple (tree_markers)
#foot_markers = tuple (foot_markers)
'''
 # creating the path of the mapping exercise

ant_path = AntPath (
    locations=track_pos,
    dash_array=[1, 10],
    delay=2000,
    color='#7590ba',
    pulse_color='#3f6fba',
    name='Trail')

m.add_layer(ant_path)

display (m)
```

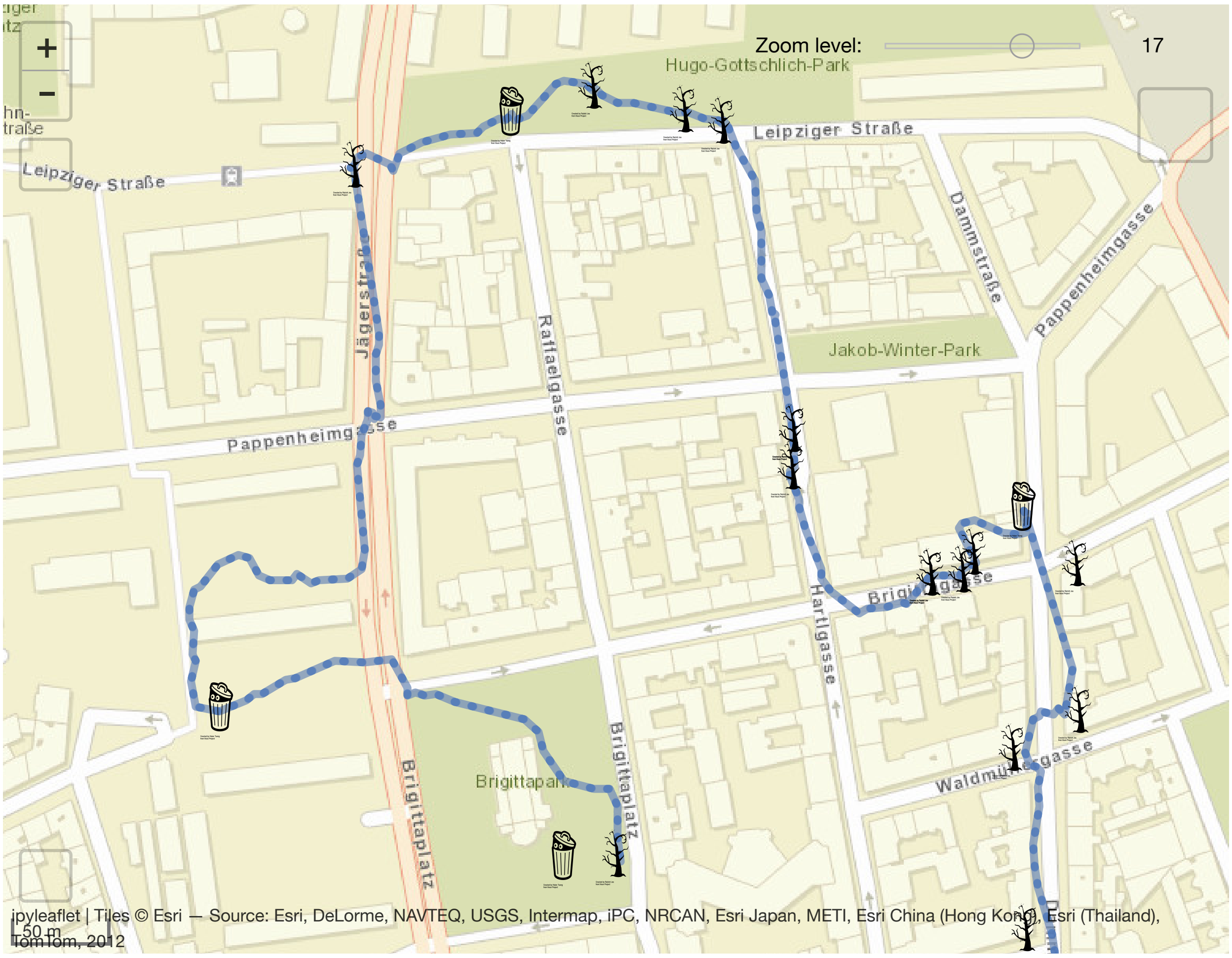




ipyleaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

In [16]:

```
# creating the path of the mapping exercise

ant_path = AntPath (
    locations=track_pos,
    dash_array=[1, 10],
    delay=2000,
    color='#7590ba',
    pulse_color='#3f6fba',
    name='Trail')

m.add_layer(ant_path)

display (m)
```



ipyleaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

7 Grouping of markers (not really useful in this context)

In [20]:

```
"""
bin_layer = MarkerCluster(markers = bin_markers, name='Trash bins')
tree_layer = MarkerCluster(markers = tree_markers, name='Tree w. trash')

#foot_layer = MarkerCluster(markers = foot_markers, name='Trail')
# print(bin_layer)

m.add_layer(bin_layer)
m.add_layer(tree_layer)
m.add_layer(ant_path)
m
"""

```

Out[20]:

```
"\nbin_layer = MarkerCluster(markers = bin_markers, name='Trash bins')\ntree_layer = MarkerC
luster(markers = tree_markers, name='Tree w. trash')\n\n#foot_layer = MarkerCluster(markers
= foot_markers, name='Trail')\n# print(bin_layer)\n\nm.add_layer(bin_layer) \nm.add_layer(tr
ee_layer) \nm.add_layer(ant_path)\nm\n"
```

8 The resulting HTML file should be visible in any browser (however, icons will be missing - fixible)

In [21]:

```
m.save('my_map.html', title='My Map')
```

9 Some cleaning up if needed

In [ ]:

```
m.clear_layers()
```

In [ ]:

```
m.remove_layer(bin_layer)
m.remove_layer(tree_layer)
m.remove_layer (ant_path)
```

10 Integrating a different basemap with more details

In [ ]:

```
from ipyleaflet import Map, WMSLayer, basemaps
#wmts = "http://maps.wien.gv.at/basemap/geolandbasemap/normal/google3857/{z}/{y}/{x}.png"

wms = WMSLayer(
    url='http://maps.wien.gv.at/basemap/geolandbasemap/normal/google3857/{z}/{y}/{x}.png',
    format='image/png',
    transparent=True,
    attribution='wait'
)

m.add_layer(wms)

m
```

11 For later: Experimenting with coloring regions or neighborhoods

In [ ]:

```
import csv
from collections import defaultdict
```

In [ ]:

```
#the syntax is: mydict[key] = "value"
#mydict ["iphone 5S"] = 2013

def parse_csv_by_field(filename, fieldnames):
    print(fieldnames)
    d = defaultdict(list)
    with open(filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile, fieldnames)
        next(reader)  # remove header
        for row in reader:
            d[row ['bundesland']] = int (row ['measurement'])
    return dict(d)


area_data = parse_csv_by_field('data/area_data.csv', ['bundesland','measurement'])
area_data


```

In [ ]:

```
m.clear_layers()
```

```
geo_json_borders

{'type': 'FeatureCollection',
 'name': 'gemeinden_999_geo',
 'crs': {'type': 'name',
  'properties': {'name': 'urn:ogc:def:crs:OGC:1.3:CRS84'}},
 'features': [{'type': 'Feature',
   'properties': {'name': 'Pöttsching', 'iso': '10609'},
   'geometry': {'type': 'MultiPolygon',
    'coordinates': [[[[16.404354111718263, 47.79918128500937],
       [16.400857594414486, 47.79178318259396],
       [16.370098559225617, 47.75647909430695],
       [16.36178609891293, 47.750404442983026],
       [16.337313248332276, 47.775956948979676],
```

In [ ]:

```
import geopandas as gpd
import json
states = gpd.read_file('geojson/laender.json')
print(states.head())
```

```
borders1 = 'geojson/bezirke_vienna.json'
borders2 = 'geojson/gemeinden_999_geo.json'
borders3 = 'geojson/laender.json'

with open(borders3) as f:
    geo_json_borders = json.load(f)

wms = WMSLayer(
url='http://maps.wien.gv.at/basemap/geolandbasemap/normal/google3857/{z}/{y}/{x}.png',
    format='image/png',
    transparent=True,
    attribution='wait'
)


m = Map(center=center, zoom=12, layout=Layout(width='100%', height='600px'))

'''
border_layer = GeoJSON(data=geo_json_borders,
                                 style = {'color': 'red',
                                          'opacity': 1.0,
                                          'weight': 2.9,
                                          'fill': 'blue',
                                          'fillOpacity': 0.2})
'''

layer = Choropleth(
    geo_data=geo_json_borders,
    choro_data=area_data,
    key_on= 'iso',
    colormap=linear.YlOrRd_04,
    border_color='black',
    style={'fillOpacity': 0.8, 'dashArray': '5, 5'})


m.add_layer(wms)
m.add_layer(border_layer)

m
```

In [ ]:

```
geo_json_borders ['features'] [0] ['properties'] ['name']
```

In [ ]:

```

```