Программирование в лингвистике

Изображения, карты

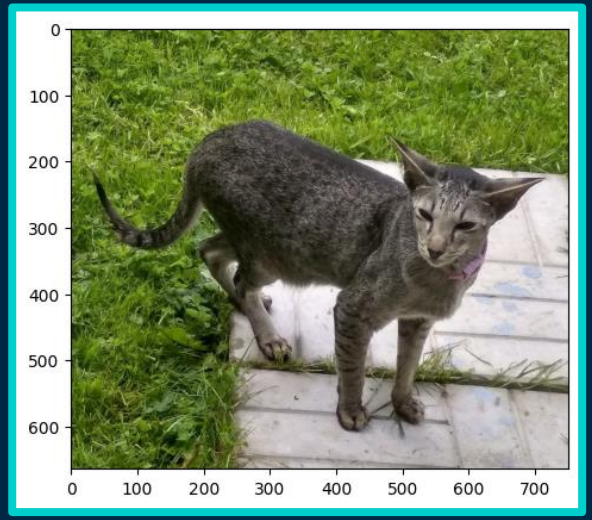
Картинки в Matplotlib

Используя возможности matplotlib, мы можем работать не только с графиками, но и с изображениями. Мы можем заставить matplotlib изобразить любую картинку:

```
import matplotlib.pyplot as plt
import matplotlib.image as img

# загружаем картинку по ее пути
testImage = img.imread('cat.jpg')

# показываем в аутпуте
plt.imshow(testImage)
```



Картинки в Matplotlib

Любая картинка с точки зрения компьютера - это таблица с числами (если картинка монохромная, то это одинокая таблица, в которой в каждой ячейке указана интенсивность цвета в соответствующем пикселе, а если картинка в RGB, то таких таблиц будет три). Matplotlib умеет показывать и эти самые числа, стоящие за картинкой:

```
import matplotlib.pyplot as plt
import matplotlib.image as img

# загружаем картинку по ее адресу
testImage = img.imread('cat.jpg')

# выведем массив чиселок
print(testImage)
```

```
[[[114 140  33]
  [117 143  34]
  [ 78 104   0]
  ...
  [ 97 124  27]
  [150 177  82]
  [141 168  73]]

 [[ 90 116   9]
  [116 142  33]
  [132 158  48]
  ...
```

Каждая строчка является отображением одного пикселя, а числа – значения по RGB-шкале (красный, зеленый, синий).

Для работы с картинками алгоритмы машинного обучения и нейронные сети нуждаются именно в таком представлении.

Каналы RGB

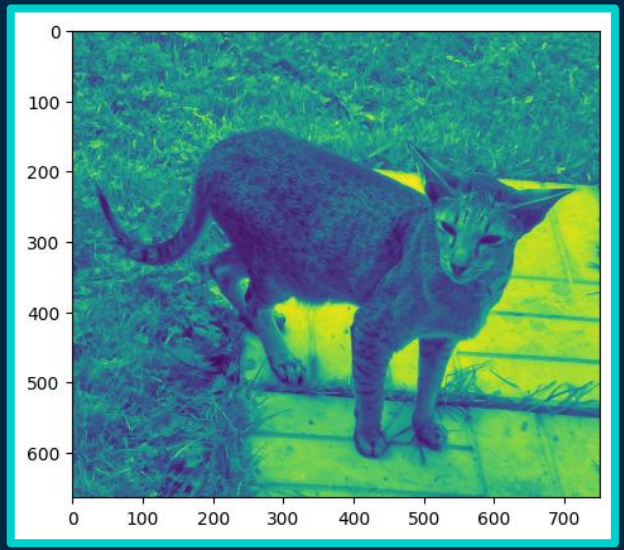
Учитывая сказанное, мы можем попросить matplotlib показать нам только один канал из трех. Здесь мы смотрим второй канал - G (green). Нумерация каналов начинается с 0.

```
'''посмотрим размер картинки:
первые два числа - это высота и ширина,
а третье число - количество каналов'''
print(testImage.shape)
```

```
(664, 750, 3)
```

```
# покажем только второй канал из трех
modifiedImage = testImage[:, :, 1]

# посмотрим
plt.imshow(modifiedImage)
```



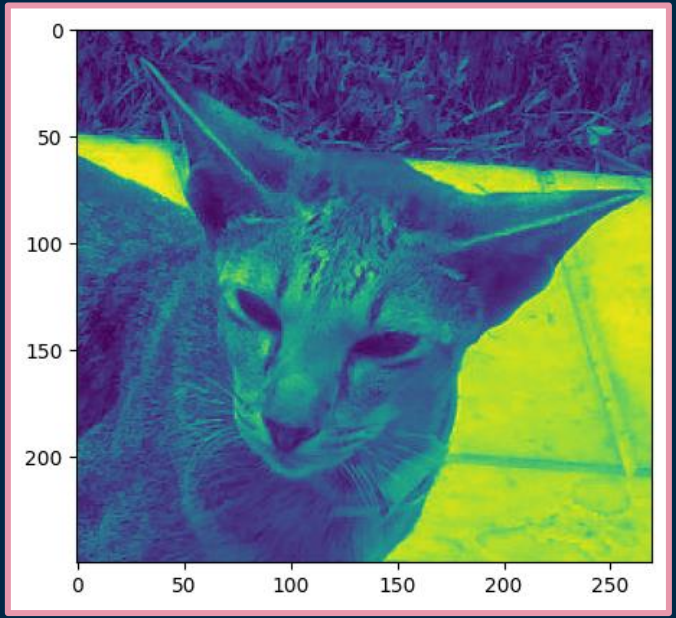
В этой картинке с котиком очень много зеленого и красного цветов! Синий весьма трудно обнаружить.

Каналы RGB

```
# обрежем картинку и посмотрим второй канал
modifiedImage = testImage[150:400, 450:720, 2]

plt.imshow(modifiedImage)
```

Здесь мы выбираем только канал Blue и отображаем часть нашей картинки. Зачастую обрезать изображение приходится методом тыка.

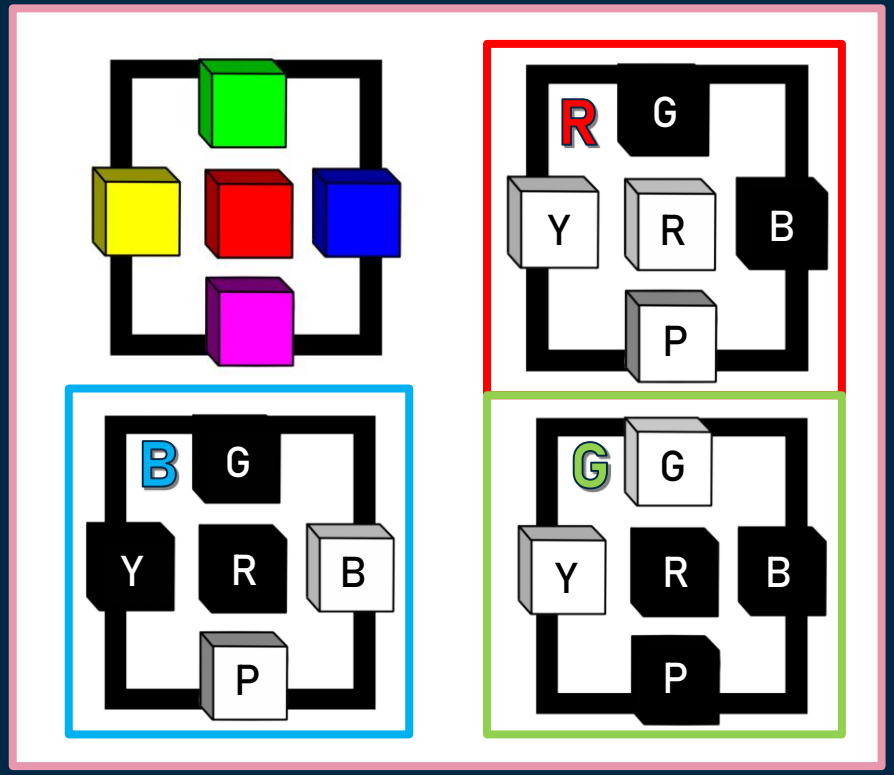


Настройка картинки

```
modifiedImage = testImage[:, :, 0]
# чтобы можно было настраивать, сохраним в переменную
fig = plt.imshow(modifiedImage)
fig.set_cmap('Greys')
'''выставим цветовую схему:
ведь для matplotlib наша картинка -
только набор чиселок, обозначающих яркость цвета'''
# спрячем оси координат
fig.axes.get_xaxis().set_visible(False)
fig.axes.get_yaxis().set_visible(False)
```

Здесь мы можем видеть, как распространен на картинке красный канал. Так как цвета инвертированы, более темные места означают, что там значение R выше.

Каналы RGB



Как мы знаем, белый цвет – это максимальные значения каналов (255, 255, 255). Черный – минимальные (0, 0, 0).

Исходя из этого, когда мы просматриваем каналы отдельно на примере этих ярких кубиков, некоторые кубики становятся черными, а остальные – белыми.

К примеру, красный присутствует в красном (255, 0, 0), желтом (255, 255, 0) и пурпурном (255, 0, 255).

Синий максимально участвует в пурпурном, в зеленый – в желтом (наравне с красным).

Рамка черного цвета не меняется ни в одном отображении, точно так же как и белый фон.

Каналы RGB

Как мы видим, в траве преобладают красный и зеленый цвета.

(цвет травы – 162, 196, 48)

В коте много черного цвета, поэтому на его изображении у каналов будут низкие значения.

(цвет кота – 31, 31, 18)

Плитка – белая, так что в ее области все каналы будут иметь самое высокое значение.

(цвет плитки – 235, 227, 226)

Обработка картинки

Мы можем попытаться использовать matplotlib как самый простой графический редактор и написать специальную функцию, которая принимает картинку и выгружает ее в измененном виде.

```
def make_image(inputname,outputname):
    '''функция для обработки и сохранения картинки'''
    # считываем и сразу оставляем только первый канал из трех
    data = img.imread(inputname)[:,:,0]
    fig = plt.imshow(data) # показываем как картинку
    fig.set_cmap('hot') # выбираем цветовую схему
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    plt.savefig(outputname) # сохраняем по пути результат

make_image('cat.jpg', 'cat2.jpg')
```

Pillow

Для Python есть специальная библиотека для продвинутой работы с картинками.
Pillow может быть предустановлена, но в крайнем случае – pip install Pillow.

У макбуков могут возникнуть с ней проблемы, поэтому в крайнем случае нужно в терминале выполнить пару команд (brew - это программа homebrew, если не установлена, нужно скачать и установить):

```
brew install libtiff libjpeg webp littlecms
sudo pip install Pillow
```

Pillow

Картинка отобразилась в исходном большом разрешении!
Примечание: в тетрадках может не работать метод .show(), поэтому предлагается либо использовать функцию display, либо оставлять показ картинки последним в ячейке.

```
from PIL import Image

myimage = Image.open('cat.jpg')
myimage.load()
'''после того, как выполним эту функцию,
можно будет работать с картинкой
в переменной myimage'''
myimage
```

Перехват исключений

Еще мы можем добавить в наш код конструкцию, которая избавит нас от трейсбеков по поводу File Not Found.
Перехват исключений – большая и важная тема, но мы пока просто воспользуемся этой возможностью самым простым образом.

```
try:
    original = Image.open('qwerty.jpg')
except FileNotFoundError:
    print("Файл не найден")
```

```
Файл не найден
```

Атрибуты Image

```
try:
    cat_picture = Image.open('cat.jpg')
except FileNotFoundError:
    print("Файл не найден")

print("Размер изображения:")
print(cat_picture.format, cat_picture.size,
      cat_picture.mode)
```

```
Размер изображения:
JPEG (750, 664) RGB
```

Pillow может показать нам расширение картинки, размер картинки и режим картинки, если обращаться к атрибутам экземпляра класса Image.

Атрибут size — это кортеж, содержащий ширину и высоту (в пикселях).
Обычные mode: L для изображений с оттенками серого, RGB для изображений с истинным цветным изображением и CMYK для печати изображений.

Фильтры Pillow

```python
from PIL import Image, ImageFilter
from IPython.display import display

try:
    original = Image.open('cat.jpg')
except FileNotFoundError:
    print("Файл не найден")

# размываем изображение
blurred = original.filter(ImageFilter.BLUR)
# открываем оригинал и размытое изображение
display(original) # покажет картинку-оригинал
display(blurred) # покажет, что получилось
# сохраняем изображение
blurred.save("blurred.png")
```

Таким образом мы размыли нашего котика и сохранили результат. Все фильтры тут

Иконки в Pillow

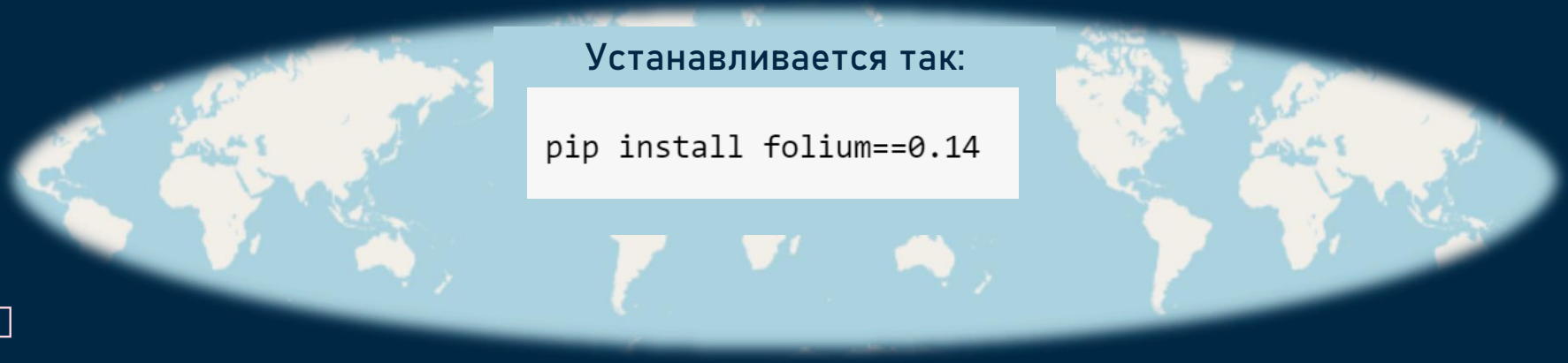
```
size = (128, 128) # размер иконки
img = Image.open('cat.jpg')
img.thumbnail(size)
img.save('cat_thumbnail.jpg')
img
```

Мы можем делать иконки в Pillow. По сути, метод просто подгоняет наше изображение по размеру, чтобы его потом можно было использовать в виде иконки.

Pillow умеет делать и другие вещи: изменять размер изображения, поворачивать его, сжимать, растягивать, обрезать. Но эти вещи подробно рассматривать не будем, их легко нагуглить самостоятельно.

Карты в Python

Иногда бывает нужно создавать интерактивные карты, например, такие карты есть на известном вам сайте WALS Online.
Основная библиотека, которая умеет это делать в питоне, называется folium. Эта библиотека умеет создавать не только сами карты, которые можно масштабировать и всячески перетаскивать мышкой (как в Google Maps), но и размещать маркеры, показывать широту и долготу по наведению мышки и прочее подобное.



Устанавливается так:

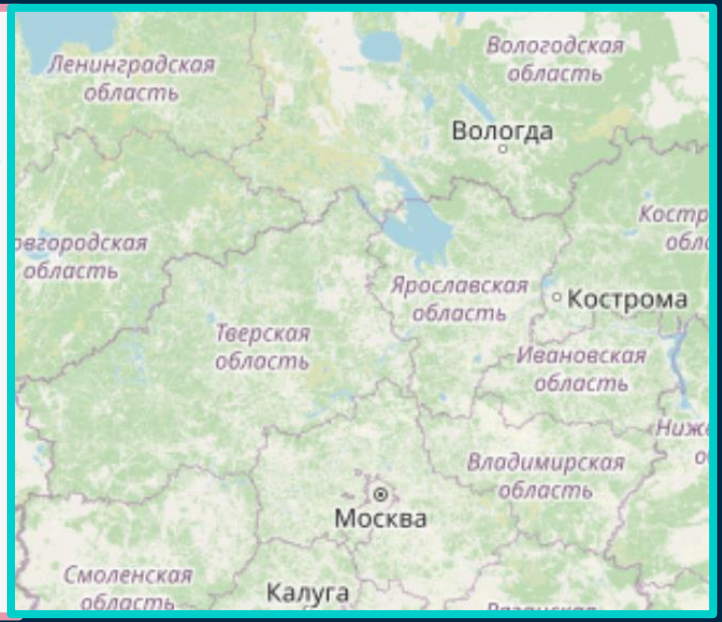
```
pip install folium==0.14
```

folium

По умолчанию при создании экземпляра класса Map() мы получаем карту мира.

```
import folium
from folium import plugins
from folium.plugins import MarkerCluster
from folium.plugins import MousePosition
from folium.features import DivIcon

"""инициализируем объект "карта" -
по умолчанию это карта всего мира"""
world_map = folium.Map()
world_map
```

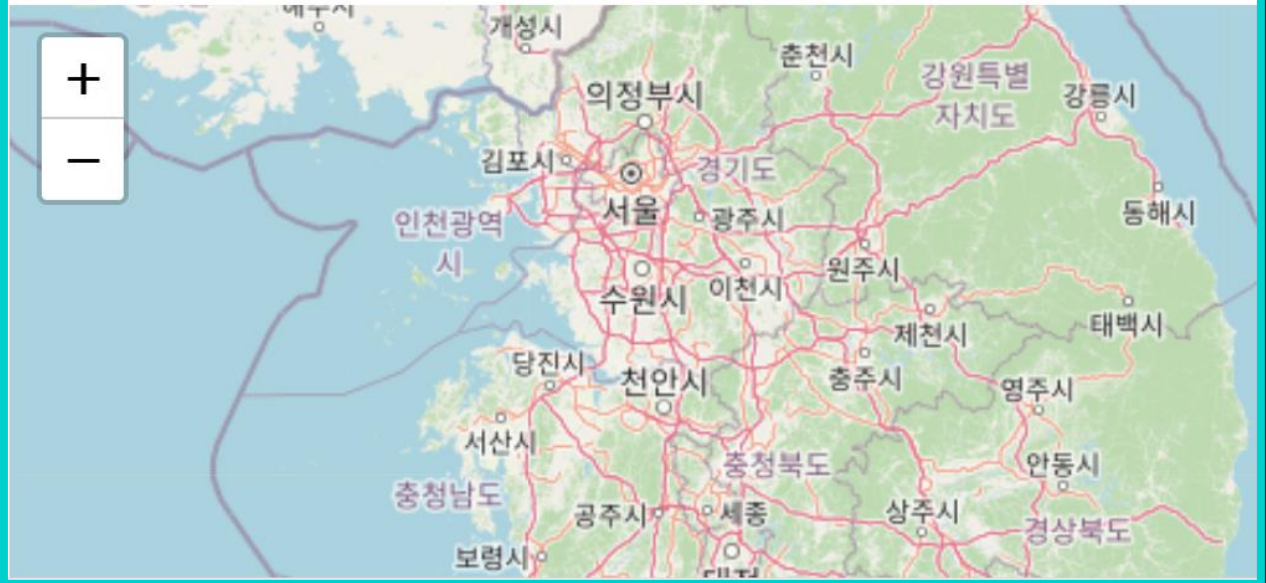


folium: параметры Map

location (tuple or list, default:None)	ширина и долгота карты (например, для России можно указать [64.6863136, 97.7453061]
width & heigth (int, string, default :'100%')	int в пикселях, str в процентах ('100' или '100%')
min_zoom (int, default:0)	минимальный разрешенный зум-уровень
max_zoom (int, default:18)	максимальный разрешенный зум-уровень
zoom_start (int, default:10)	стартовое увеличение
tiles (str)	стиль карты (default=OpenStreetMap) больше про стили тут, но, к сожалению, любые другие стили не работают

folium: карта Кореи

```
korea_map = folium.Map(location=[37, 126], zoom_start=7)
korea_map
```

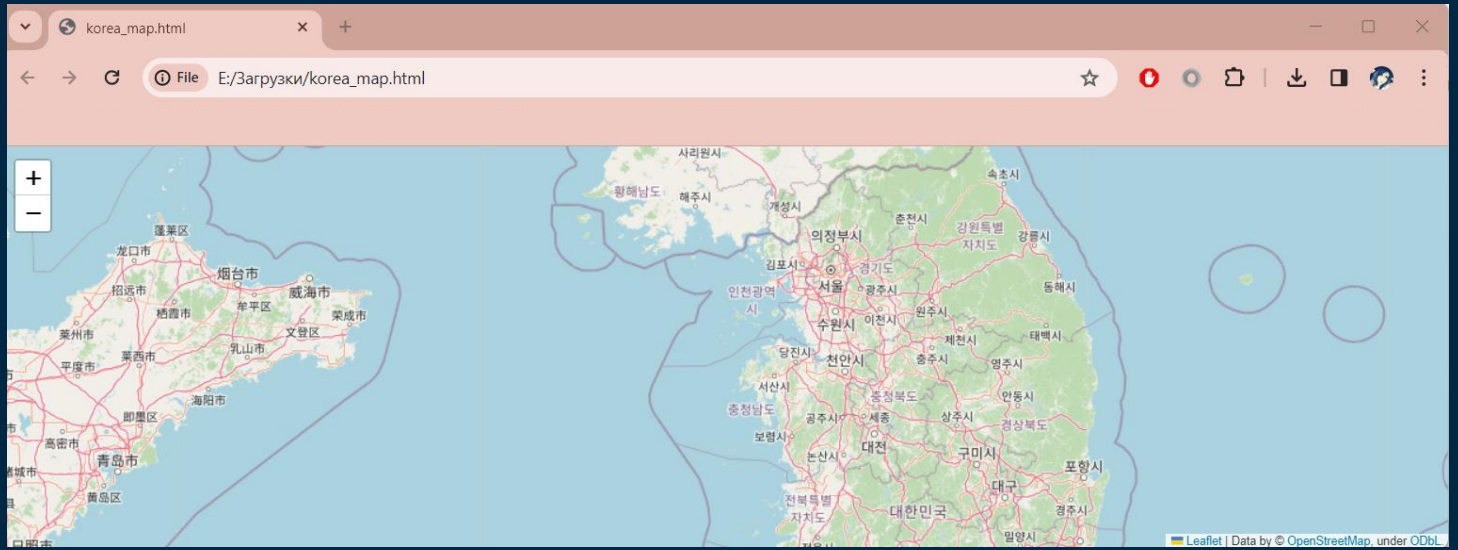


Если мы укажем правильные location и zoom, наша карта отобразит Южную Корею.

folium: сохранение карты

Мы можем сохранить нашу карту в html-формате.

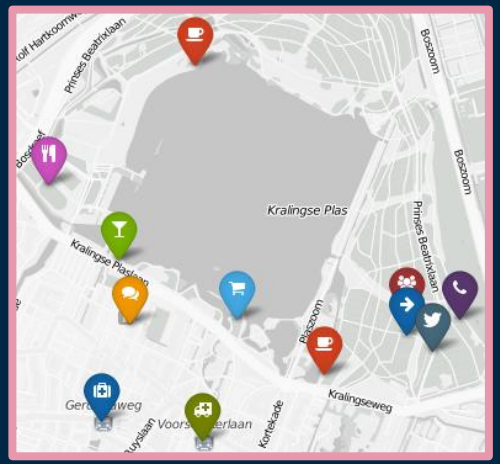
```
korea_map.save('korea_map.html')
```



folium: маркеры

Можно размещать маркеры на нашей карте с помощью объекта folium.Marker. У него тоже есть ряд параметров:

tooltip : str, подсказка, которая будет появляться, когда мышка наводится на маркер.
popup : html-код (str) подсказка (форматированная), которая будет появляться при щелчке на маркер.
icon : внешний вид иконки. В библиотеке есть свой набор иконок, а можно и кастомную задать.



Доступные иконки можно посмотреть тут или тут.
glyphicons не требует префикса, а FontAwesome требует префикс ‘fa’.

folium: установка маркеров

Так как для установки маркеров нам нужно знать широту и долготу, придется имплементировать возможность просматривать их на карте. Об этом дальше.

```
folium.Marker([37.56, 126.97], # приходится знать широту и долготу
              popup='<i>The capital of Korea</i>',
              tooltip='Seoul'
             ).add_to(korea_map) # добавляем на карту

folium.Marker([37.45, 126.70],
              popup='<i>International Airport</i>',
              tooltip='Incheon',
              icon=folium.Icon(icon='plane', color='red')
             ).add_to(korea_map)
korea_map
```

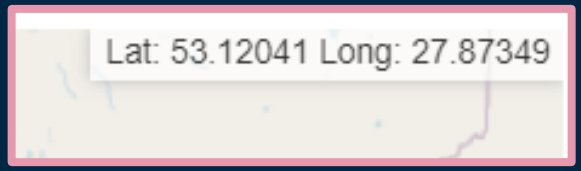
folium: Mouse Position

```
# сделаем карту России
russia_map = folium.Map(
    location = [64.6863136, 97.7453061],
    zoom_start = 4
)

formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright', # где будет находиться окошко с координатами
    separator=' Long: ', # долгота
    empty_string='NaN', # что будет показываться, когда координат нет
    lng_first=False,
    num_digits=20, # сколько цифр влезет
    prefix='Lat:', # широта
    lat_formatter=formatter,
    lng_formatter=formatter,
)

russia_map.add_child(mouse_position)
russia_map
```

Используя объект Mouse Position, мы можем попросить folium показывать широту и долготу по движению курсора.



folium + WALS

Мы можем загрузить карты из WALS в наш скрипт! Делается это так:

1. В уголке карты WALS нажимаем на GeoJson.
2. Открываем один признак (придется повторить со всеми).
3. Копируем код и вставляем его в текстовый документ.
4. Меняем расширение с .txt на .geojson.
5. Загружаем наши файлы в код с помощью модуля json.
6. Далее создаем экземпляры класса GeoJson и добавляем их к нашей карте.


```
import json

simple = json.load(open('12Asimple.geojson'))
mcomplex = json.load(open('12Amcomplex.geojson'))
compl = json.load(open('12Acomplex.geojson'))
```

folium + WALS

```
import json

simple = json.load(open('12Asimple.geojson'))
mcomplex = json.load(open('12Amcomplex.geojson'))
compl = json.load(open('12Acomplex.geojson'))
m = folium.Map(location=[50, 90], zoom_start=2)

folium.GeoJson(
    simple,
    zoom_on_click=True,
    marker=folium.Marker(icon=folium.Icon(color="green",
                                          icon="circle", prefix='fa')),
).add_to(m)

folium.GeoJson(
    mcomplex,
    zoom_on_click=True,
    marker=folium.Marker(icon=folium.Icon(color="pink",
                                          icon="heart", prefix='fa')),
).add_to(m)

folium.GeoJson(
    compl,
    zoom_on_click=True,
    marker=folium.Marker(icon=folium.Icon(color="red",icon="cog",
                                          prefix='fa')),
).add_to(m)

m
```