GIS 4653/5653: Spatial Programming and GIS

Numeric Processing and Web Programming

Numeric Processing

Lists and range() are inefficient

- Lists are meant for dynamic creation
  - Can hold values of different types
  - Very flexible, but also quite inefficient

```
mylist = [3.14, 2.72]
mylist.append(1.414)

for i in range( len(mylist) ):
    print mylist[i]*mylist[i]
```

- range() creates a list of the specified size
  - Can lead to memory problems

Numpy provides array capability

- Arrays in numpy are all of the same type
  - Element-by-element arithmetic operators are overloaded

```
import numpy as N;
mylist = [3.14, 2.72, 1.414]
arr = N.array(mylist)

print arr*arr
```

Creating and looping

- xrange() is like range() but without memory impact

```
import numpy as N;

arr = N.zeros( (3,2), dtype='f' )

for row in xrange( arr.shape[0] ):
    for col in xrange( arr.shape[1] ):
        arr[row,col] = row*row + col*col
print arr
```

```
>>>
[[ 0.  1.]
 [ 1.  2.]
 [ 4.  5.]]
...
```

Operations on arrays

- Can also carry out Boolean operations

```
large = arr > 3
print large
```

```
[[ 0.  1.]
 [ 1.  2.]
 [ 4.  5.]]
[[False False]
 [False False]
 [ True  True]]
```

```
large = N.logical_and( arr > 1, arr < 3 )
print large
```

Select indexes with “where”

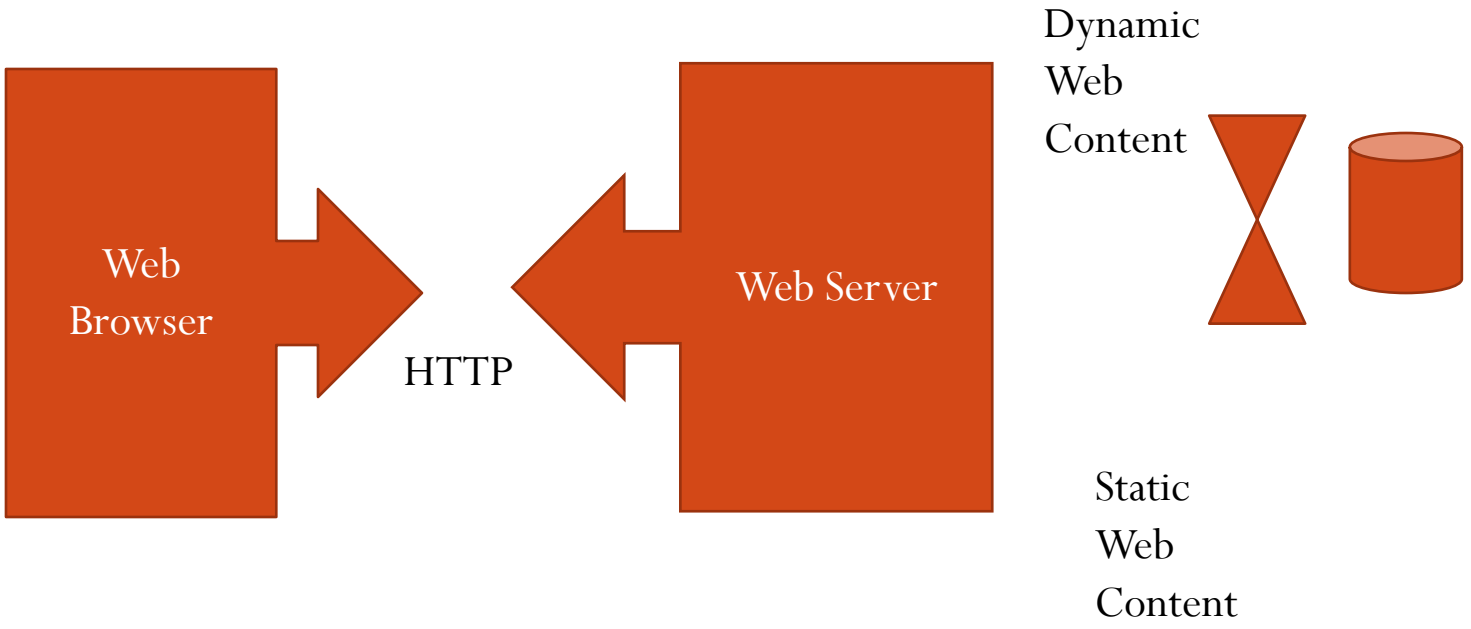
```
condition = N.logical_and( arr > 1, arr < 3 )
print N.where( condition, arr*arr, 0 )
```

```
[[ 0.  1.]
 [ 1.  2.]
 [ 4.  5.]]
[[ 0.  0.]
 [ 0.  4.]
 [ 0.  0.]]
```

```
condition = N.logical_and( arr > 1, arr < 3 )
print arr[ N.where(condition) ]
```

Web display

How web applications work

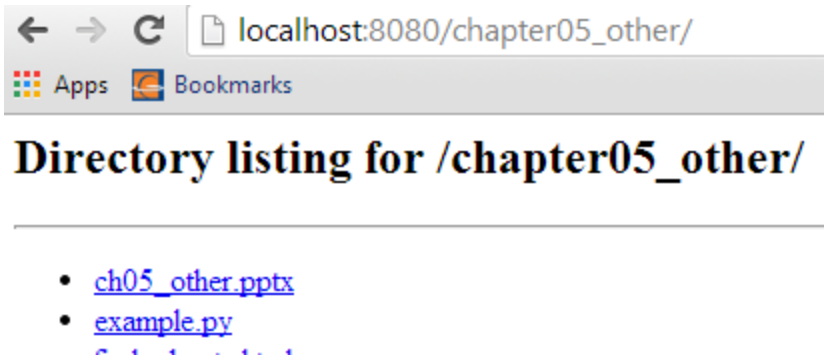


Python web server

- Python comes with a light-weight web server

```
C:\Users\Lak>cd c:\spatialprogramming

c:\spatialprogramming>python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

- Can use it to serve out static web content



Creating GIS web content

- The simplest way to create GIS web content and serve it out:
  - Create GeoJSON data
  - Embed it into JavaScript
  - Serve out the JavaScript
- Can use Google Maps, ArcServer, OpenLayers, etc. for background maps
  - OpenLayers is free and requires no license

http://openlayers.org/dev/examples/

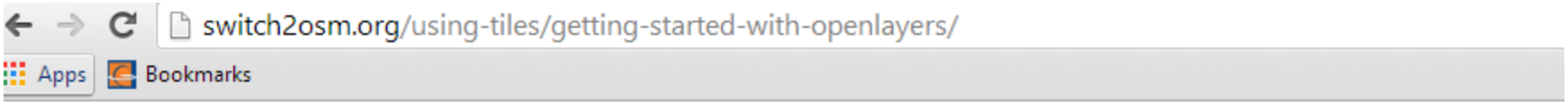
OpenLayers + static web content

- Can use Python to create JSON web content
  - Just text
  - GDAL is capable of exporting to JSON
- Parse the JSON from JavaScript
  - Add as layers

Example

- Let us show the 5 most indispensable fire hydrants

Starting with Open Map



Note use of OpenLayers .js

Street map layer from Open Map

What coordinate system do we need?

```
<!DOCTYPE HTML>
<html>
<head>
    <title>OpenLayers Basic Example</title>

    <script src="http://www.openlayers.org/api/OpenLayers.js"></script>
    <script>
      function init() {
        map = new OpenLayers.Map("mapdiv");
        var mapnik = new OpenLayers.Layer.OSM();
        map.addLayer(mapnik);

        var lonlat = new OpenLayers.LonLat(-1.788, 53.571).transform(
            new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
            new OpenLayers.Projection("EPSG:900913") // to Spherical Mercator
          );

        var zoom = 13;
```

Plotting JSON from Open Layers

view-source:openlayers.org/dev/examples/geojson.html

Bookmarks

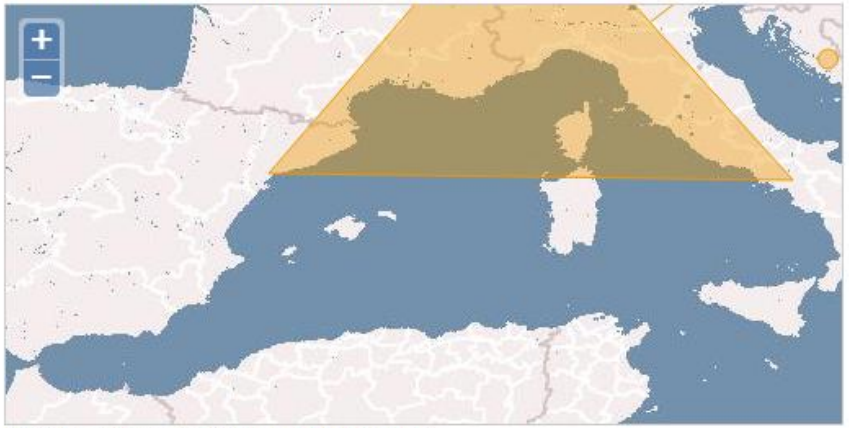
```
var featurecollection = {
  "type": "FeatureCollection",
  "features": [
    {"geometry": {
        "type": "GeometryCollection",
        "geometries": [
            {
                "type": "LineString",
                "coordinates":
                    [[11.0878902207, 45.1602390564],
                    [15.01953125, 48.1298828125]]
            },
            {
                "type": "Polygon",
                "coordinates":
                    [[[11.0878902207, 45.1602390564],
                      [14.931640625, 40.9228515625],
                      [0.8251953125, 41.0986328125],
                      [7.63671875, 48.96484375],
                      [11.0878902207, 45.1602390564]]]
            },
            {
                "type":"Point",
                "coordinates":[15.87646484375, 44.1748046875]
            }

      "type": "Feature",
      "properties": {}}
  ]
};
var geojson_format = new OpenLayers.Format.GeoJSON();
var vector_layer = new OpenLayers.Layer.Vector();
map.addLayer(vector_layer);
vector_layer.addFeatures(geojson_format.read(featurecollection));
```

openlayers.org/dev/examples/geojson.html

Apps Bookmarks

GeoJSON Example

Demonstrate the use of the GeoJSON format.



This example uses the GeoJSON format.

Basic JSON template

```
ofp = open("iow_indispensible.json", "w")
feature_collection = {
                "type": "FeatureCollection",
                "features": [
                    {"geometry": {
                      "type": "GeometryCollection",
                      "geometries": [

                      ]
                    },
                    "type": "Feature",
                    "properties": {}
                  }
                ]
}
```

Points go here

```
for i in range(5):
    print hydrants[i]
    (x,y,z) = ct.TransformPoint( hydrants[i].lon, hydrants[i].lat )
    print "Spherical coordinates are: ",x,y
    h = {"type": "Point",
         "coordinates" : [ x, y ]
         }
    feature_collection["features"][0]["geometry"]["geometries"].append( h )
ofp.write( json.dumps(feature_collection) );
```

Coordinate Transformation

```
osr.UseExceptions()
# Need to set GDAL_DATA to C:\Program Files (x86)\GDAL\gdal-data
fromProj = osr.SpatialReference()
fromProj.ImportFromEPSG(4326);   # Transform from WGS 1984
toProj   = osr.SpatialReference()
toProj.ImportFromEPSG(900913); # to Spherical Mercator Projection
ct = osr.CoordinateTransformation(fromProj,toProj)

for i in range(5):
    print hydrants[i]
    (x,y,z) = ct.TransformPoint( hydrants[i].lon, hydrants[i].lat )
    print "Spherical coordinates are: ",x,y
    h = {"type": "Point",
         "coordinates" : [ x, y ]
         }
```

Reading JSON over Http

-]

```
var request = new XMLHttpRequest();
request.open("GET", "./iow_indispensible.json", false);
request.send(null);
featurecollection = JSON.parse( request.responseText );
alert(request.responseText)
vector_layer.addFeatures(geojson_format.read(featurecollection));
```

```
ofp = open("iow_indispensible.json", "w")
feature_collection = {
              "type": "FeatureCollection",
              "features": [
                  {"geometry": {
                    "type": "GeometryCollection",
                    "geometries": [

                    ]
                  },
                  "type": "Feature",
                  "properties": {}
                }
              ]
}
for i in range(5):
    print hydrants[i]
    (x,y,z) = ct.TransformPoint( hydrants[i].lon, hydrants[i].lat )
    print "Spherical coordinates are: ",x,y
    h = {"type": "Point",
         "coordinates" : [ x, y ]
         }
    feature_collection["features"][0]["geometry"]["geometries"].append( h )
ofp.write( json.dumps(feature_collection) );
ofp.write('\n')
ofp.close()
```

```
    <script type="text/javascript">
        var lon = -76.7;
        var lat =  36.9;
        var zoom =  9;
        var map, layer;

        function init(){
            map = new OpenLayers.Map( 'map' );

var fromProjection = new OpenLayers.Projection("EPSG:4326");   // Transform from WGS 1984
var toProjection   = new OpenLayers.Projection("EPSG:900913"); // to Spherical Mercator Projection

            var mapnik = new OpenLayers.Layer.OSM();
            map.addLayer(mapnik);

            //layer = new OpenLayers.Layer.WMS( "OpenLayers WMS",
            //        "http://vmap0.tiles.osgeo.org/wms/vmap0",
            //        {layers: 'basic'} );
            //map.addLayer(layer);

            map.setCenter(new OpenLayers.LonLat(lon, lat).transform(fromProjection,toProjection), zoom);
           var vector_layer = new OpenLayers.Layer.Vector();
           map.addLayer(vector_layer);

var request = new XMLHttpRequest();
request.open("GET", "./iow_indispensible.json", false);
request.send(null);
featurecollection = JSON.parse( request.responseText );
alert(request.responseText)
vector_layer.addFeatures(geojson_format.read(featurecollection));

        }
    </script>
```

Serving out dynamic web content

- To serve out dynamic web content, we simply throw away the intermediate file
  - Have the request go straight to a script or program
  - The program writes to output, which is then displayed by the browser
  - The running program can be Python ("cgi scripts")
  - Can take user input in the form of URL parameters
- Python (CGI in general) tends to be inefficient for dynamic web pages
  - Java "Servlets" more common

Homework

- Create a web-interactive map that shows railway stations in the UK
  - Data for railway stations:
    - http://alistairphillips.com/nationalrail/stations.sql
    - Just a text file: parse it to pull in lat,lon of stations