Developing Geospatial software with Python, Part 1

Alessandro Pasotti (apasotti@gmail.com), Paolo Corti (pcorti@gmail.com)

Summary

Part 1 - Paolo Corti

- **GDAL/OGR (Python bindings, GeoDjango)**
- **GEOS (GeoDjango, Shapely)**
- **PROJ.4 (Python bindings, GeoDjango)**
- **GeoAlchemy**

GFOSS Day, Foligno - 18/19 November 2010

Part 2 - Alessandro Pasotti

- OWS (OWSLib, pyWPS)
- WebServices (GeoPy, Mapnik, Mapscript)
- Desktop (QGIS, GRASS)

Building blocks

- **GDAL Geospatial Data Abstraction Library**
    - **OGR** Simple Feature Library
- **GEOS** Geometry Engine, Open Source
- **PROJ.4** Cartographic Projections Library

Building blocks: GDAL/OGR

GDAL Geospatial Data Abstraction Library

OGR Simple Feature Library

- basic library for almost all FOSS4G projects (and often proprietary projects)
- library and utilities for reading/writing a plethora of GIS formats
- raster/cover (GDAL) and vectorial (OGR)
- OGR follows the OGC Simple feature access specifications
- written mostly in C++ by Frank Warmerdam
- license: **X/MIT**

GDAL/OGR: large diffusion

Large list of sofware using GDAL, here only most important (full list @ GDAL website)

- **FOSS4G**: GRASS, GeoServer, gvSIG, MapServer, MapGuide, Orfeo Toolbox, OSSIM, QGIS, R
- **proprietary**: ArcGis, ERDAS, FME, Google Earth

GDAL/OGR: vector formats

Long list too, here only most important (full list @ gdal website)

OGR (Vector) (get the full list with **ogrinfo --formats**):

- **FOSS4G** (RW): PostGis, Spatialite, MySQL, csv, GeoJSON, GeoRSS, GML, GPSBabel, GPX, GRASS, KML, WFS
- **proprietary**: Shapefile, ArcInfo Coverage (R), ArcInfo .E00 (R), AutoCAD DXF, Esri PGDB (R), ArcSde (R), FMEObjects Gateway (R), MapInfo, Microstation DGN, Oracle Spatial, Microsoft MS Spatial

GDAL/OGR: raster formats

GDAL (Raster) (get the full list with **gdalinfo --formats**):

- **FOSS4G** (RW): GRASS Rasters, WKTRaster, Rasterlite
- **proprietary**: ArcInfo ASCII Grid, ArcSde Raster (R), ERDAS (R), Oracle Spatial GeoRaster, Intergraph, TIFF/GeoTIFF (Adobe)

GDAL/OGR: bindings

bindings (based on SWIG) provide the GDAL power to developers using other languages than C/C++

- **Python**
- Perl
- VB6 (COM) - No SWIG
- Ruby
- Java
- .Net (VB, C#, ...)
- R

GDAL/OGR: utilities (raster)

The power of GDAL/OGR at your fingertips (mostly in c, cpp but some written in python)!

- gdalinfo: info about a file
- gdal_translate: copy a raster with control on output
- gdal_rasterize: rasterize vectors

- gdalwarp: warp an image into a new coordiante system
- gdaltransform: transform coordinates
- gdal_retile.py: build tiled pyramid levels
- gdal_grid: create raster from scattered data
- gdal_polygonize.py: generate polygons from raster

GDAL/OGR: utilities (vector)

- ogrinfo: lists information about an OGR supported data source
- ogr2ogr: converts simple features data between file formats
- ogrtindex: creates a tileindex

Building blocks: GEOS

- it is a C++ port of **JTS** (Java Topology Suite from Vivid Solutions)
- originally started from Refractions for PostGIS

- provides all the **OGC Simple Feature Access** implementations for SQL spatial predicate functions and spatial operators
- license: **LGPL**

GEOS: Geometry Engine, Open Source

GEOS Geometry Engine, Open Source

- **model for geometric objects** (Point, Linestring, Polygon, Multipoint, Multipolygon, GeomCollection)
- **geometric attributes and methods** (empty, geom_type, num_coords, centroid, area, distance, length, srs, transform...)
- **representation and interoperation** (ewkt, hex, hexewkb, json, geojson, kml, ogr, wkb, ewkb, wkt)
- **unary predicates** (has_z, simple, valid...)
- **binary predicates** (contains, crosses, equals, intersects, touches, within, ...)
- **constructive spatial analysis methods** (buffer, difference, intersection, simplify, union, envelope, ...)

GEOS: huge diffusion

Large list of sofware using GEOS, here only most important (full list @ GEOS website)

- **FOSS4G**: PostGIS, Spatialite, MapServer, QGIS, OGR, Shapely, GeoDjango
- **proprietary**: FME, Autodesk MapGuide Enterprise

GEOS: bindings

bindings provide the GEOS power to developers using other languages than C/C++

- **Python** (not maintained anymore --> **GeoDjango, Shapely**)
- Ruby
- PHP

Java developers of course must use the JTS!

.NET developers can use the .NET JTS port (NetTopologySuite)

Building blocks: PROJ.4

PROJ.4 Cartographic Projections Library

- PROJ.4 Cartographic Projections library originally written by Gerald Evenden then of the USGS
- written in C/C++
- both a **command line** and an **API**
- used from almost any FOSS4G project
- ported to javascript (**Proj4js**) and Java (**Proj4J**)
- license: **MIT**

PROJ.4: API

Just 2 methods available:

- create a projPJ coordinate system object from the string definition
- transform the x/y/z points from the source coordinate system to the destination coordinate system:

```
projPJ pj_init_plus(const char *definition);

int pj_transform( projPJ srcdefn, projPJ dstdefn, long point_count, int point_offset, double *x, double *y, double *z );
```

Implementations

- **GDAL/OGR bindings**: Python API to GDAL/OGR, PROJ.4 and GEOS (parts of)
- **GeoDjango**: Python API to GDAL/OGR, PROJ.4 and GEOS plus other goodness
- **Shapely**: Python API to GEOS
- **GeoAlchemy**: Python API integrating SQLAlchemy for spatial database support

GDAL/OGR bindings

- GDAL/OGR library offers Python bindings generated by **SWIG**
- GDAL is for raster, OGR for vector
- GDAL Python bindings is the **only solution for raster**
- documentation to be improved

• license: **X/MIT**

GDAL/OGR bindings: GDAL example (1)

accessing the raster, getting the projection and reading general properties

```
>>> from osgeo import gdal
>>> ds = gdal.Open('aster.img', gdal.GA_ReadOnly)
>>> ds.GetProjection()
'PROJCS["UTM Zone 12, Northern Hemisphere",...AUTHORITY["EPSG","32612"]]'
>>> print 'Raster has %s cols, %s rows' % (ds.RasterXSize, ds.RasterYSize)
Raster has 5665 cols, 5033 rows
>>> print 'Raster has %s bands' % ds.RasterCount
Raster has 3 bands
```

GDAL/OGR bindings: GDAL example (2)

accessing the raster geotrasform parameters - aka the georeferencing information

```
>>> geotransform = ds.GetGeoTransform()
>>> print geotransform
(419976.5, 15.0, 0.0, 4662422.5, 0.0, -15.0)
>>> print 'top left x is %s' % geotransform[0]
top left x is 419976.5
>>> print 'top left y is %s' % geotransform[3]
top left y is 4662422.5
>>> print 'pixel width is %s' % geotransform[1]
pixel width is 15.0
>>> print 'pixel height is %s' % geotransform[5]
pixel height is -15.0
>>> print 'raster rotation is %s' % geotransform[2]
raster rotation is 0.0
```

GDAL/OGR bindings: GDAL example (3)

reading the value of a cell for a given band (optimization issues, this is just a sample)

```
>>> cols = ds.RasterXSize
>>> rows = ds.RasterYSize
>>> band1 = ds.GetRasterBand(1)
>>> data = band1.ReadAsArray(0,0, cols, rows) // 0,0 is the offset
>>> value = data[2000,2000]
>>> value
61
```

GDAL/OGR bindings: OGR example (1)

reading a shapefile

```
>>> from osgeo import ogr
>>> driver = ogr.GetDriverByName('ESRI Shapefile')
>>> datasource = driver.Open('regioni.shp', 0)
>>> print datasource.GetLayerCount()
1
>>> layer = datasource.GetLayer()
>>> print layer.GetFeatureCount()
20
```

GDAL/OGR bindings: OGR example (2)

accessing shapefile metadata

```
>>> srs = layer.GetSpatialRef()
>>> print srs.ExportToWkt()
PROJCS["UTM_Zone_32_Northern_Hemisphere",GEOGCS["GCS_International 1909 (Hayford)",....
>>> print layer.GetExtent()
(313352.32445650722, 1312130.1391031265, 3933804.0026830882, 5220607.6164360112)
>>> layerDefn = layer.GetLayerDefn()
>>> layerDefn.GetFieldCount()
9
>>> layerDefn.GetGeomType()
3
>>> fieldDefn = layerDefn.GetFieldDefn(2)
>>> fieldDefn.GetName()
'REGIONE'
```

```
>>> fieldDefn.GetTypeName()
'String'
```

GDAL/OGR bindings: OGR example (3)

accessing shapefile features and geometries

```
>>> feature = layer.GetFeature(0)
>>> feature.GetFID()
0
>>> feature.GetField('REGIONE')
'PIEMONTE'
>>> geometry = feature.GetGeometryRef()
>>> geometry.GetEnvelope()
(313352.32445650722, 517043.7912779671, 4879624.4439933635, 5146102.0567664672)
>>> geometry.GetGeometryName()
'MULTIPOLYGON'
>>> geometry.IsValid()
True
```

```
>>> geometry.GetDimension()
2
```

GDAL/OGR bindings: OGR example (4)

accessing shapefile features and geometries

```
>>> geometry.ExportToWkt() # GML, KML, Wkb, Json
'MULTIPOLYGON (((456956.454114792693872 5146065.056706172414124,...
>>> geometry.GetArea()
25390743681.717426
>>> poly0 = geometry.GetGeometryRef(0)
>>> poly0.GetArea()
25390649513.408951
>>> poly0.GetGeometryName()
'POLYGON'
>>> mybuffer = poly0.Buffer(10000)
>>> mybuffer.GetArea()
```

```
35462220275.922073
```

GDAL/OGR bindings: resources

- samples on svn: http://svn.osgeo.org/gdal/trunk/gdal/swig/python/samples/
- some GDAL command line utilities
- many GDAL regression tests are written in Python: http://svn.osgeo.org/gdal/trunk/autotest/
- Geoprocessing with Python using OpenSource GIS: http://www.gis.usu.edu/~chrisg/python/2009/
- GDAL mailing list: http://lists.osgeo.org/mailman/listinfo/gdal-dev/

GeoDjango

- **Django**: The Web framework for perfectionists with deadlines. A **DRY** framework with an **ORM** (object relational mapper), a router, a **MVC** implementation and a great backend application
- **GeoDjango**: The Geographic Web Framework for perfectionists with deadlines

- since Django 1.0 is a **core package**
- it is a framework including a set of API, utility and tool for developing GIS application with Django
- as Django, you may use GeoDajngo both in **web** and **desktop** context
- excellent documentation
- license: **BSD**

GeoDjango: Index

- GeoDjango Model API
- GEOS API
- GDAL/OGR API
- Measurement Units API
- GeoDjango Admin site
- Utilities (LayerMapping, OgrInspect)

GeoDjango: Architecture

- **Spatial Database**
    - PostGis
    - Spatialite
    - MySql (not OGC-compliant, limited functionality)
    - Oracle

- **GIS Libraries (Python API via ctypes)**
  - GEOS (Geometry Engine Open Source)
  - GDAL/OGR (Geospatial Data Abstraction Library)
  - PROJ.4 (Cartographic Projections Library)
  - GeoIP

GeoDjango: Model API (1)

Geometry Field (django.contrib.gis.db extends django.db)

- PointField, LineStringField, PolygonField
- MultiPointField, MultiLineStringField, MultiPolygonField
- GeometryCollectionField
- GeometryField

Geometry Field options

- **srid** (default 4326 = WGS84 dd)

- **dim** (default 2, 3 will support z)
- **spatial_index** (default True, spatial index is built)

GeoDjango: Model API (2)

In Django models we get **Geometry Field** and **GeoManager**

```
from django.contrib.gis.db import models

class Site(models.Model):
    """Spatial model for site"""
    code = models.IntegerField()
    name = models.CharField(max_length=50)
    geometry = models.MultiPolygonField(srid=4326)
    objects = models.GeoManager()
```

GeoDjango: Model API (3)

```
$ ./manage.py sqlall myapp
```

```
BEGIN;
CREATE TABLE "myapp_site" (
    "id" serial NOT NULL PRIMARY KEY,
    "code" integer NOT NULL,
    "name" varchar(50) NOT NULL
)
;
SELECT AddGeometryColumn('myapp_site', 'geometry', 4326, 'MULTIPOLYGON', 2);
ALTER TABLE "myapp_site" ALTER "geometry" SET NOT NULL;
CREATE INDEX "myapp_site_geometry_id"
    ON "myapp_site" USING GIST ( "geometry" GIST_GEOMETRY_OPS );
COMMIT;
```

GeoDjango: Model API (4)

CRUD methods: Create, Update

```
>>> from myapp.models import *
>>> new_point = SandboxLayer(nome='punto 1', geometry='POINT(13.8 42.5)')
>>> new_point.save()
>>> print(connection.queries[-1])
{'time': '0.061', 'sql': 'INSERT INTO "fauna_sandboxlayer" ("nome", "geometry")
VALUES (E\'punto 1\', ST_GeomFromEWKB(E\'\\\\001\\\\...'))'}
```

```
>>> new_point = SandboxLayer.objects.get(nome__contains='pun')
>>> new_point.nome = 'punto 2'
>>> new_point.save()
>>> print(connection.queries[-1])
{'time': '0.002', 'sql': 'UPDATE "fauna_sandboxlayer" SET "nome" = E\'punto 2\',
    "geometry" = ST_GeomFromEWKB(E\'\\\\001\\\\...')
    WHERE "fauna_sandboxlayer"."id" = 1 '}
```

GeoDjango: Model API (5)

CRUD methods: Read, Delete

```
>>> avvistamento = Avvistamento.objects.get(id=1)
>>> regione = Regione.objects.filter(geometry__intersects=avvistamento.geometry)
>>> regione
[<Regione: ABRUZZO>]
>>> print(connection.queries[-1])
{'time': '0.187', 'sql': 'SELECT "fauna_regione"."id", "fauna_regione"."codice",
    "fauna_regione"."nome", "fauna_regione"."geometry"
    FROM "fauna_regione" WHERE ST_Intersects("fauna_regione"."geometry",
    ST_GeomFromEWKB(E\'\\\\001\...')) LIMIT 21'}
```

```
>>> sandfeat = SandboxLayer.objects.get(id=1)
>>> sandfeat.delete()
>>> print(connection.queries[-1])
{'time': '0.002', 'sql': 'DELETE FROM "fauna_sandboxlayer" WHERE "id" IN (1)'}
>>> SandboxLayer.objects.all().delete()
>>> print(connection.queries[-2])
{'time': '0.002', 'sql': 'DELETE FROM "fauna_sandboxlayer" WHERE "id" IN (3, 2)'}
```

GeoDjango: GEOS API (1)

a model for geometric objects (Simple Feature Access)

- Point
- LineString, LinearRing
- Polygon
- Geometry Collections (MultiPoint, MultiLineString, MultiPolygon, GeometryCollection)

GeoDjango: GEOS API (2)

- **geometric attributes and methods** (empty, geom_type, num_coords, centroid, area, distance, length, srs, transform...)
- **representation and interoperation** (ewkt, hex, hexewkb, json, geojson, kml, ogr, wkb, ewkb, wkt)
- **unary predicates** (has_z, simple, valid...)
- **binary predicates** (contains, crosses, equals, intersects, touches, within, ...)

- **constructive spatial analysis methods** (buffer, difference, intersection, simplify, union, envelope, ...)

GeoDjango: GEOS API, Example 1

geometric objects (point), geometric properties (hasz, geom_type) and representation and serialization

```
>>> from myapp.models import Place
>>> place = Place.objects.get(id=1)
>>> point = place.geometry
>>> point.x, point.y
(13.798828125, 42.5390625)
>>> point.hasz
False
>>> point.geom_type
'Point'
>>> point.json
'{ "type": "Point", "coordinates": [ 13.798828, 42.539062 ] }'
```

```
>>> point.ewkt # extended wkt
'SRID=4326;POINT (13.7988281250000000 42.5390625000000000)'
```

GeoDjango: GEOS API, Example 2

predicates and relationships, transformations (requires GDAL), spatial analysis methods

```
>>> from myapp.models import *
>>> abruzzo = Regione.objects.get(nome='ABRUZZO')
>>> avvistamento = Avvistamento.objects.get(id=1)
>>> abruzzo.geometry.contains(avvistamento.geometry)
True
>>> avvistamento.geometry.ewkt
'SRID=4326;POINT (13.7988281250000000 42.5390625000000000)'
>>> transformed_point = avvistamento.geometry.transform(3395,clone=True)
>>> transformed_point.ewkt
'SRID=3395;POINT (1536078.5204189007636160 5213176.4834084874019027)'
>>> buffer = SandboxLayer(nome='buffer',geometry=transformed_point.buffer(20000))
>>> buffer.save()
```

GeoDjango: GDAL/OGR API

excellent alternative to GDAL/OGR Python bindings

- not **required** for GeoDjango (required only for srs trasformations and for LayerMapping)
- via the **DataSource** class get the access to any **OGR** format, (R/W in many cases)
- get access to the GEOS API via geos method on **OGRGeometry** class
- get access to other API via interoperation and representation properties (wkt, wkb, json, ...)

GeoDjango: GDAL/OGR API, Example

```
>>> from django.contrib.gis.gdal import *
>>> ds = DataSource('data/shapefile/myshape.shp')
>>> print(ds)
data/shapefile/myshape.shp (ESRI Shapefile)
>>> print(len(ds))
1
>>> lyr = ds[0]
>>> print(lyr)
myshape
>>> print(lyr.num_feat)
20
>>> print(lyr.geom_type)
Polygon
>>> print(lyr.srs.srid)
4326
```

GeoDjango: GDAL/OGR API, Example

```
>>> print(lyr.fields)
['gid', 'objectid', 'code', 'name', 'shape_area', 'shape_len']
>>> for feat in lyr:
   ....:        print(feat.get('name'), feat.geom.num_points)
   ....:
first_feature 14811
second_feature 3598
...
last_feature 19131
>>> feat = lyr[1]
>>> print(feat.get('name'))
first_feature
>>> geom = feat.geom # OGRGeometry, non GEOSGeometry
>>> print(geom.srid)
4326
>>> print(feat.geom.wkt[:100])
MULTIPOLYGON (((8.439415832216145 46.465900481500874,8.439484266241374 46.465576832714113,8.43950386...
```

GeoDjango: Measurement Units API

API for measurement units conversion and management

```
>>> from django.contrib.gis.measure import Distance
>>> d1 = Distance(km=5)
>>>  print d1
5.0 km
>>>  print d1.mi
3.10685596119
>>>  d2 = Distance(mi=5)
>>>  print d1 + d2
13.04672 km
>>>  a = d1 * d2
print a
40.2336 sq_km
```

GeoDjango: resources

- excellent documentation: http://docs.djangoproject.com/en/dev/ref/contrib/gis/
- official tutorial: http://docs.djangoproject.com/en/dev/ref/contrib/gis/tutorial/
- GeoDjango Basic Apps: http://code.google.com/p/geodjango-basic-apps/

- Python Geospatial Development, a book from Packt: https://www.packtpub.com/python-geo-spatial-development/book
- Justin Bronn at DjangoCon 2008: http://www.youtube.com/watch?v=zOaimbSe6n8
- mailing list: http://groups.google.com/group/geodjango
- an overview by Dane Springmeyer: http://www.geowebguru.com/articles/99-overview-geodjango
- workshop @ FOSS4G-IT 2010 (Lugano): https://github.com/capooti/geodjango-tutorial (in Italian)
- a day with GeoDjango: http://www.paolocorti.net/2009/04/01/a-day-with-geodjango/

Shapely

- it is a Python binding library to GEOS via ctypes (like the GeoDjango GEOS API)
- aims to be **general purpose**, not only for GIS stuff (even if it is a loyal OGC SFA implementation)
- excellent documentation (very nice manual)
- **integration**: via serialization/deserialization with standard well known formats (wkt, wkb)
- **projections are not supported**, so geometries must be in a unique projected srs

- license: **BSD**

Shapely features: OGC SFA (1)

a model for geometric objects (Simple Feature Access)

- Point
- LineString, LinearRing
- Polygon
- Geometry Collections (MultiPoint, MultiLineString, MultiPolygon, GeometryCollection)
- **Empty features, Linear Referencing**

Shapely: OGC SFA (2)

- **general attributes and methods** (area, bounds, length, geom_type, distance, centroid, representative_point, coords, exterior, interiors)
- **representation and interoperation** (ewkt, hex, hexewkb, json, geojson, kml, ogr, wkb, ewkb, wkt)

- **unary predicates** (has_z, is_empty, is_ring, is_simple, is_valid)
- **binary predicates** (contains, crosses, equals, intersects, touches, within, ...)
- **constructive spatial analysis methods** (buffer, difference, intersection, simplify, union, polygonize, linemerge, ...)
- **diagnostics** (explain_validity)

Shapely: Example 1

geometric objects (point), general attributes and methods and representation and interoperation

```
>>> from shapely.geometry import Point
>>> point = Point(0.0, 0.0)
>>> point.area
0.0
>>> point.bounds
(0.0, 0.0, 0.0, 0.0)
>>> point.x, point.y
(0.0, 0.0)
```

```
>>> point.area
0.0
>>> point.length
0.0
>>> point.geom_type
'Point'
>>> point.wkt
'POINT (0.0000000000000000 0.0000000000000000)'
```

Shapely: Example 2

geometric objects (polygon), general attributes and methods and representation and interoperation

```
>>> from shapely.geometry import Polygon
>>> polygon = Polygon([(-1,-1), (-1,1), (0,1), (0,-1)])
>>> polygon.area
2.0
>>> polygon.length
```

```
6.0
>>> polygon.bounds
(-1.0, -1.0, 0.0, 1.0)
>>> polygon.geom_type
'Polygon'
>>> polygon.wkt
'POLYGON ((-1.0000000000000000 -1.0000000000000000, ...
>>> list(polygon.exterior.coords)
[(-1.0, -1.0), (-1.0, 1.0), (0.0, 1.0), (0.0, -1.0), (-1.0, -1.0)]
>>> list(polygon.interiors)
[]
```

Shapely: Example 3

unary predicates, binary predicates, spatial analysis methods

```
>>> polygon.has_z
False
```

```
>>> polygon.is_empty
False
>>> polygon.is_valid
True
>>> polygon.contains(point)
False
>>> buffer = polygon.buffer(1)
>>> buffer.contains(point)
True
```

Shapely: Example 4

diagnostics

```
>>> coords = [(0, 0), (0, 2), (1, 1), (2, 2), (2, 0), (1, 1), (0, 0)]
>>> p = Polygon(coords)
>>> from shapely.validation import explain_validity
>>> explain_validity(p)
```

```
'Ring Self-intersection[1 1]'
```

Shapely: resources

- excellent documentation: http://gispython.org/shapely/docs/1.2/
- GIS Python Lab: http://trac.gispython.org/lab
- Sean Gillies Blog: http://sgillies.net/blog/
- Python Workshop at FOSS4G 2010: http://www.mapfish.org/doc/tutorials/python-workshop/geoalchemy.html
- mailing list @gispython.org: http://lists.gispython.org/mailman/listinfo/community

GeoAlchemy

GeoAlchemy

- it is a spatial extension to **SQLAlchemy**

- it provides support for Geospatial data types at the ORM layer using SQLAlchemy
- it aims to support spatial operations and relations specified by the Open Geospatial Consortium (OGC). The project started under Google Summer of Code Program
- differently from other libraries, it does **NOT DEPEND** on other GIS building blocks (GDAL, GEOS...)
- still not so mature like GeoDjango
- license: **MIT**

GeoAlchemy: SQLAlchemy notes

SQLAlchemy

- SQLAlchemy is the most powerful Python SQL Toolkit and ORM
- compared to the Django ORM, it has a most powerfull abstraction
- supports not only tables (like Django) but also **joins, subqueries, and unions**
- higly **scalable** (ie: configuration of how many SELECT to emit while loading whole graphs of objects: lazy loading and eager load)
- greater set of DB supported if compared to Django

- support for **transactions** (i.e. nice rollback mechanism)
- excellent documentations

GeoAlchemy: Spatial database

Supported spatial database

- PostGis
- Spatialite
- MySQL (not OGC-compliant, limited functionality)
- Oracle
- MS SQL Server 2008 (Django does not support it)

GeoAlchemy: Use cases

- like Django, you may use it in **desktop and web** application

- web frameworks that integrates well with SQLAlchemy: TurboGears and Pylons (highly configurable for models, templates and helpers)
- Django is not the best SQLAlchemy friend, but you may still use it :D

GeoAlchemy: features

- **NO** model for geometric objects (Simple Feature Access) :(
- **geometric attributes and methods** (dimension, srid, geometry_type, num_points, length, area, centroid, transform, coords)
- **representation and interoperation** (wkt, wkb, svg, gml, kml, geojson)
- **unary predicates** (has_z, is_valid, is_empty, is_simple, is_closed, is_ring)
- **binary predicates** (contains, crosses, equals, intersects, touches, within, ...)
- **constructive spatial analysis methods** [limited] (buffer, boundary, convex_hull)

GeoAlchemy: Example 1

GeoAlchemy: Example 2

the model

```
engine = create_engine('postgresql://postgres:postgres@localhost/gis_test', echo=True)
Session = sessionmaker(bind=engine)
session = Session()
metadata = MetaData(engine)
Base = declarative_base(metadata=metadata)

class Spot(Base):
    __tablename__ = 'spots'
    id = Column(Integer, primary_key=True)
    name = Column(Unicode, nullable=False)
    height = Column(Integer)
    created = Column(DateTime, default=datetime.now())
    geom = GeometryColumn(Point(2))

metadata.drop_all()   # comment this on first occassion
metadata.create_all()
```

data creation

```
>>> wkt_spot1 = "POINT(-81.40 38.08)"
>>> spot1 = Spot(name="Gas Station", height=240.8, geom=WKTSpatialElement(wkt_spot1))
>>> geom_spot2 = WKTSpatialElement('POINT(30250865 -610981)', 2249)
>>> spot2 = Spot(name="Park", height=53.2, geom=geom_spot2
>>> session.add_all([spot1, spot2])
>>> session.commit()
```

GeoAlchemy: Example 3

data reading and representation

```
>>> s = session.query(Spot).get(1)
>>> session.scalar(s.geom.wkt)
'POINT(-81.42 37.65)'
>>> session.scalar(s.geom.gml)
'<gml:Point srsName="EPSG:4326"><gml:coordinates>-81.42,37.65</gml:coordinates></gml:Point>'
>>> session.scalar(s.geom.kml)
'<Point><coordinates>-81.42,37.65</coordinates></Point>'
```

GeoAlchemy: Example 4

geometric properties

```
>>> s = session.query(Spot).filter(Spot.height > 240).first()
>>> session.scalar(s.geom.geometry_type)
'ST_Point'
>>> session.scalar(s.geom.x)
-81.420000000000002
>>> session.scalar(s.geom.y)
37.649999999999999
>>> s.geom.coords(session)
[-81.420000000000002, 37.649999999999999]
```

GeoAlchemy: Example 5

constructive spatial analysis methods and binary predicates

```
>>> r = session.query(Road).first()
>>> l = session.query(Lake).first()
>>> buffer_geom = DBSpatialElement(session.scalar(r.geom.buffer(10.0)))
>>> session.scalar(buffer_geom.wkt)
'POLYGON((-77.4495270615657 28.6622373442108,....
>>> session.query(Road).filter(Road.geom.intersects(r.geom)).count()
1L
>>> session.query(Lake).filter(Lake.geom.touches(r.geom)).count()
0L
```

GeoAlchemy: resources

- documentation: http://www.geoalchemy.org/
- official tutorial: http://www.geoalchemy.org/tutorial.html
- Python Workshop at FOSS4G 2010: http://www.mapfish.org/doc/tutorials/python-workshop/geoalchemy.html
- mailing list: http://groups.google.com/group/geoalchemy?pli=1

Notes on implementations

- **pure Python** (GeoAlchemy, GeoPy, OWSLib, pyWPS)
- **Python and C/C++ libraries**
    - with **SWIG** (GDAL/OGR bindings, Mapscript, GRASS, QGIS)
    - with **ctypes** (GeoDjango, Shapely)
    - with **Boost.Python** (Mapnik)

Notes on implementations: SWIG

- a software development tool that connects programs written in C and C++ with a variety of high-level programming languages
- scripting languages: Perl, PHP, **Python**, Tcl and Ruby
- non-scripting languages: C#, Common Lisp, Go language, Java, Lua, Modula-3, OCAML, Octave and R

- used to parse C/C++ interfaces and generate the 'glue code' required for the above target languages to call into the C/C++ code
- nice tutorial: http://www.swig.org/tutorial.html
- basically you write an interface library to the C/C++ code and then you can build the Python module with the swig command

Notes on implementations: ctypes

- as SWIG it aims to give connection features to programs written in C, but it is a **Python** specific library

```
>>> from ctypes import *
>>> libc = CDLL('libc.so.6')
>>> print libc.time(None)
1289407624
```