Mongoengine Introduction

PyMongo is extremely flexible and fast but sometime writing schema validation can be cumbersome. MongoEngine is a Document-Object Mapper for working with MongoDB from Python. It uses a simple declarative API, similar to the Django ORM.

What does MongoEngine do?

MongoEngine provides several main facilities:

- schema validation and enforcement
- an object-document mapper providing higher-level constructs such as
  - a unit of work pattern
  - relations between documents

This lesson will teach you to install MongoEngine and how to use it.

Installing MongoEngine

Due to some dependency issues, you will need to install `PasteDeploy` and `Paste` before installing MongoEngine: (tutorial-env) $ pip install MongoEngine

Hopefully that proceeded without any problems, and now you should be able to import MongoEngine from Python.

In [1]:
```
import mongoengine
```

Connecting to MongoDB

Connecting to MongoEngine is simple, here we are connecting to the tumblelog database:

In [2]:
```
from mongoengine import connect

conn = connect('tumblelog')
conn
```

Out[2]:
```
Connection('localhost', 27017)
```

Defining a collection

So far, we have shown how you only how to connect to PyMongo without any real benefit. Now, let's actually do something useful: create our first collection. Our schema will be based around a tumblelog application

In [3]:
```
import datetime
import mongoengine as db


class Post(db.Document):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=T
    title = db.StringField(max_length=255, required=True)
    slug = db.StringField(max_length=255, required=True)
    body = db.StringField(required=True)

    def __unicode__(self):
        return self.title or "New Post"

Post.drop_collection()   # Drop collection

Post()
```

```
Out[3]: <Post: New Post>
```

What we've done here is create a Post document class that represents post documents in the `tumblelog.post` collection.

Default Values

By setting a `default` value to the field definitions you can declare default values. Default values can also be callable functions eg: `datetime.datetime.now` will set the current time on creation of a post:

```
In [4]: post = Post()
        post.created_at
```

```
Out[4]: datetime.datetime(2012, 7, 2, 13, 58, 34, 359462)
```

Invalid values

MongoEngine won't throw an exception when you set an invalid attribute but will throw it on `save` or when calling `validate`:

```
In [5]: post.created_at = "This should be a date"
        try:
            post.validate()
        except Exception, e:
            print e.errors['created_at']
```

```
cannot parse date "This should be a date" ("created_at")
```

Saving and querying

Now we have defined our model lets look at how we can add posts and then how we can find documents once in the database.

Saving documents

```
In [6]: post = Post(title="mongoengine post",
                    slug="mongoengine-post",
                    body="Welcome to Europython 2012!")
        post.save()
```

```
Out[6]: <Post: mongoengine post>
```

The save method will then save the data in mongoDB. By default all saves in MongoEngine are synchronous and set the write concern to `safe`.

Note: saves in pymongo will save the *whole* document and in MongoEngine if the document exists then any changes will be converted to updates.

Raw data

MongoEngine adds extra meta data to the documents stored in mongoDB so it can deal with polymorphism **`_types`** and knows what class to use for a document **`_cls`**.

Below we use native pymongo to show the how the data is stored in mongoDB.

```
In [7]: Post._get_collection().find_one()
```

```
Out[7]: {u'_cls': u'Post',
         u'_id': ObjectId('4ff19afabb69331041000000'),
         u'_types': [u'Post'],
         u'body': u'Welcome to Europython 2012!',
         u'created_at': datetime.datetime(2012, 7, 2, 13, 58, 34, 385000),
         u'slug': u'mongoengine-post',
         u'title': u'mongoengine post'}
```

Querying

MongoEngine uses a similar declaritive syntax to Django. To query you need to use the `Queryset Manager` called `objects`. You can explicitly call `filter`:

```
In [8]: Post.objects.filter()
```

```
Out[8]: [<Post: mongoengine post>]
```

Or just apply the filter when calling `objects`:

```
In [9]: Post.objects()
```

```
Out[9]: [<Post: mongoengine post>]
```

To filter you can pass keywords to limit the results returned.

```
In [10]: Post.objects(slug="mongoengine-post")
```

```
Out[10]: [<Post: mongoengine post>]
```

Skips and limits

Sometimes you need to paginate through your result set. There are two ways to achive this with MongoEngine, chaining `skip` and `limit` or by using a slice:

```
In [11]: post = Post(title="second post",
                     slug="second-post",
                     body="Lets skip!").save()
         Post.objects().skip(1).limit(1)
```

```
Out[11]: [<Post: second post>]
```

```
In [12]: Post.objects()[1:2]
```

```
Out[12]: [<Post: second post>]
```

Updating documents

You can use `save` to update a document, but in pymongo that would overwrite the whole document and could create a race condition. As its better to be explicit use the `$set` atomic updates - these are expressed as *operation_name__field_name* eg:

In [13]:
```
Post.objects(slug="mongoengine-post").update(set__title="MongoEngine Post")
Post.objects.first()
```

Out[13]:
```
<Post: MongoEngine Post>
```

Deleting documents

To delete a document you can call the `delete` method on the document or a queryset:

In [14]:
```
# Delete a single document
post = Post.objects.first()
post.delete()

# Delete all matching documents
Post.objects().delete()

# Drop the collection
Post.drop_collection()
```

Exercises

I. Create a Python module containing the tumblelog model.
II. Create a couple of posts in your database
III. Using the `post.delete()` method, remove an instance of a post.
IV. Using the `unset` update operator remove a slug from your first post. What happens when loading that post? What happens if you try to `validate` it?
V. Using the `set` operator what happens if you try to set a value for a field that doesn't exist eg. tags?
VI. Try to insert two posts with the same slug. What happens?