# Inheritance

MongoEngine allows you to define documents that inherit from each other. In the raw data MongoEngine creates a **_types** list that lists all the inherited document classes for a document.

Currently, inheritance is on by default but in the next major release it will be off by default, so to future proof we set `allow_inheritance` to True in the meta.

## Initial setup

Lets get our schema setup as before with clean data.

In [1]:
```
import datetime
import mongoengine as db

conn = db.connect('tumblelog')
conn.drop_database('tumblelog')


class Post(db.Document):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=
    title = db.StringField(max_length=255, required=True)
    slug = db.StringField(max_length=255, required=True)
    body = db.StringField(required=True)
    author = db.ReferenceField('User', required=True)
    comments = db.ListField(db.EmbeddedDocumentField('Comment'))

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


class Comment(db.EmbeddedDocument):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=
    body = db.StringField(verbose_name="Comment", required=True)
    author = db.StringField(verbose_name="Name", max_length=255, required=

    def __unicode__(self):
        return (u"comment by %s" % self.author) if self.author else "New C


class User(db.Document):
    email = db.StringField(required=True)
    first_name = db.StringField(max_length=50)
    last_name = db.StringField(max_length=50)

ross = User(email="ross@10gen.com",
            first_name="Ross",
            last_name="Lawley").save()

Post(title="mongoengine post",
     slug="mongoengine-post",
     body="Welcome to Europython 2012!",
     author=ross).save()

comment_1 = Comment(author="Bob",
                    body="Nice post thanks")
comment_2 = Comment(author="Ross",
                    body="Florence rocks!")
```

```
Post.objects.update(push_all__comments=[comment_1, comment_2])

post = Post.objects.first()
post
```

Out[1]: <Post: mongoengine post>

## Coverting from a Blog to a Tumblelog

Currently we just have `Posts` but we've named our collection as `tumblelog` we can use inheritance to extend the `Post` schema. Lets add the following types of posts: BlogPost, Video, Image, Quote.

**Note:** We redefine `Post` to remove the `body` field as this is only needed for `BlogPosts`.

In [2]:
```
class Post(db.Document):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=
    title = db.StringField(max_length=255, required=True)
    slug = db.StringField(max_length=255, required=True)
    author = db.ReferenceField('User', required=True)
    comments = db.ListField(db.EmbeddedDocumentField('Comment'))

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

    meta = {'allow_inheritance': True}


class BlogPost(Post):
    body = db.StringField(required=True)


class Video(Post):
    embed_code = db.StringField(required=True)


class Image(Post):
    image_url = db.StringField(required=True, max_length=255)


class Quote(Post):
    body = db.StringField(required=True)

# Migrate the old Post to a BlogPost
BlogPost(**post.to_mongo()).save()
BlogPost.objects()
```

Out[2]: [<BlogPost: mongoengine post>]

## Querying Inherited Documents

Because of the way MongoEngine stores inherited documents, we can query using the lowest Document class and MongoEngine will return the correct instance for each Type of Document.

In [3]:
```
Post.objects()
```

Out[3]:
```
[<BlogPost: mongoengine post>]
```

In [4]:
```
tim = User(email="tim.peters@example.com",
           first_name="Tim",
           last_name="Peters").save()

Quote(title="Zen of Python",
     slug="zen-of-python",
     body="Beautiful is better than ugly. Explicit is better than implicit
     author=tim).save()

Post.objects.all()
```

Out[4]:
```
[<Quote: Zen of Python>, <BlogPost: mongoengine post>]
```

# Advanced features - Document Meta

The main way we control the schema, inheritance and indexes is via the `meta` attribute. As mentioned previously inheritance is controlled by setting the `allow_inheritance` option in the meta.

## Indexes

By defining `indexes` to the `meta` of a document you can control which indexes are created. Below is an example of how you can create indexes:

In [5]:
```
class SomeDoc(db.Document):
    date = db.DateTimeField(db_field='addDate', default=datetime.datetime.
    category = db.StringField()
    tags = db.ListField(db.StringField())

    meta = {
        'indexes': [
            '-date',
            'tags',
            ('category', '-date')
        ],
        'allow_inheritance': True
    }
```

In the example above we have created an Ascending index on `date`, an index on `tags` and a compound index on `category` and `date`

## Exercises

I. What indexes would you create for the `Post` document?
II. What happens if you dont include `allow_inheritance` setting for a document.
III. What happens if you turn inheritance off and try to inherit from the document.