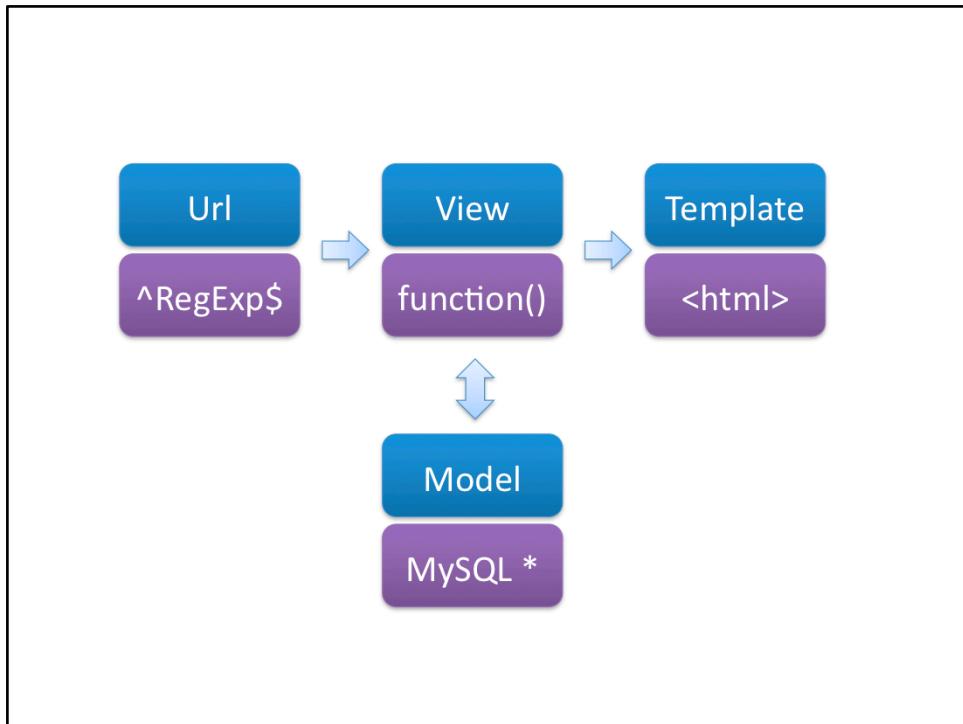


django

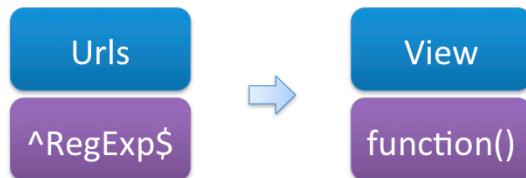
This talk is about Django, my favourite web framework. I'm Emil Stenström, I work for a company called Valtech, and we are looking for developers. Please e-mail emil.stenstrom@valtech.se if you are interested.



I have tried these tools, but none of them really made me love working with them.



This is how Django works, and this is really all you need to know about Django's inner workings. The blue boxes are what things are called, the purple ones is a try to be more concrete in what they mean. Lets go through each box.



```
from django.conf.urls.defaults import *

urlpatterns = patterns("",
    (r'^articles/2003/$', 'news.views.special_2003'),
    (r'^articles/(\d{4})/$', 'news.views.year_archive'),
    (r'^articles/(\d{4})/(\d{2})/$', 'news.views.month_archive'),
    (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'news.views.article_detail'),
)
```

This is a full url configuration file. To the left you have a list of regular expressions, to the right you have a list of functions to run when an URL matches that regexp. Every pair of parenthesis inside the regexps are variables that are sent as arguments to the view.

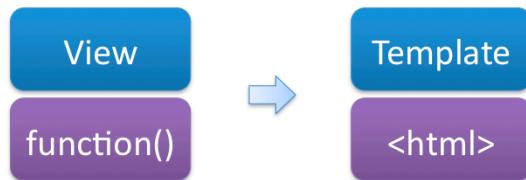
View

function()

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

And this is a view. But you can't have html in the code like that can you?



```
from django.shortcuts import render_to_response
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    return render_to_response("time.html", {
        'now': now
    })
```

Better send your information to a template like this. You can send any number of variables, of any type, to your template.

Template

<html>

```
<h1>{{ section.title }}</h1>
{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"6" }}</p>
{% endfor %}
```

And this is how the template looks. Use dot-syntax to access properties, and for-loops to access things in lists. Pipe chars, |, can be used for filter variables (in this case "upper" is a filter) and django comes with a very good list of default filters.

<html>

<h1>Fräsha nyheter</h1>

<h2>Nya iGlasses släppta</h2>
<p>Riktigt häftiga glasögon släpptes igår av...</p>

<h2>Coolaste hundleksaken</h2>
<p>Har din hund också Apple-feber?...</p>

And this is the rendered HTML. But we have missed one step now, haven't we? The model...



Model

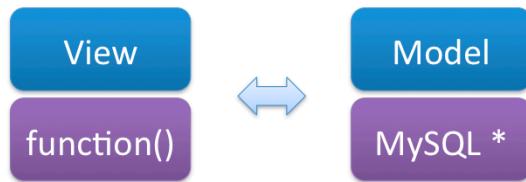
MySQL *

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)

-----
manage.py syncdb
```

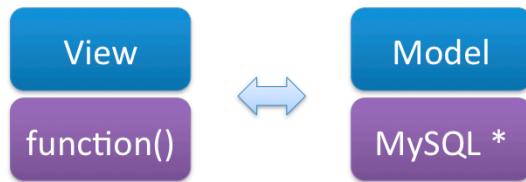
The model! This is what you need to do to store information about a person. Just list the fields you want, there are many field types to choose from. When you're done, go to the console and type the line at the bottom. Voilá, your model is ready to use...



```
from django.shortcuts import render_to_response
from myproject.meeting.models import Meeting

def meeting_list(request):
    meeting_list = Meeting.objects.all()
    return render_to_response("meeting_list.html",
                            {'meeting_list': meeting_list})
)
```

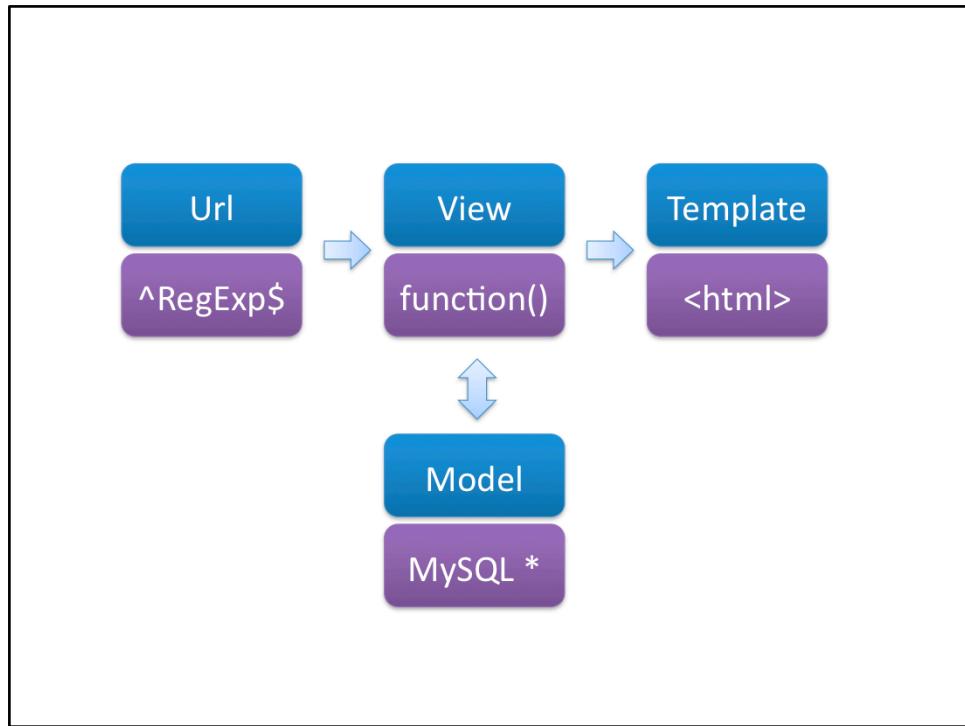
This is how you use your model. The all() function is just one of many ways to access model objects.



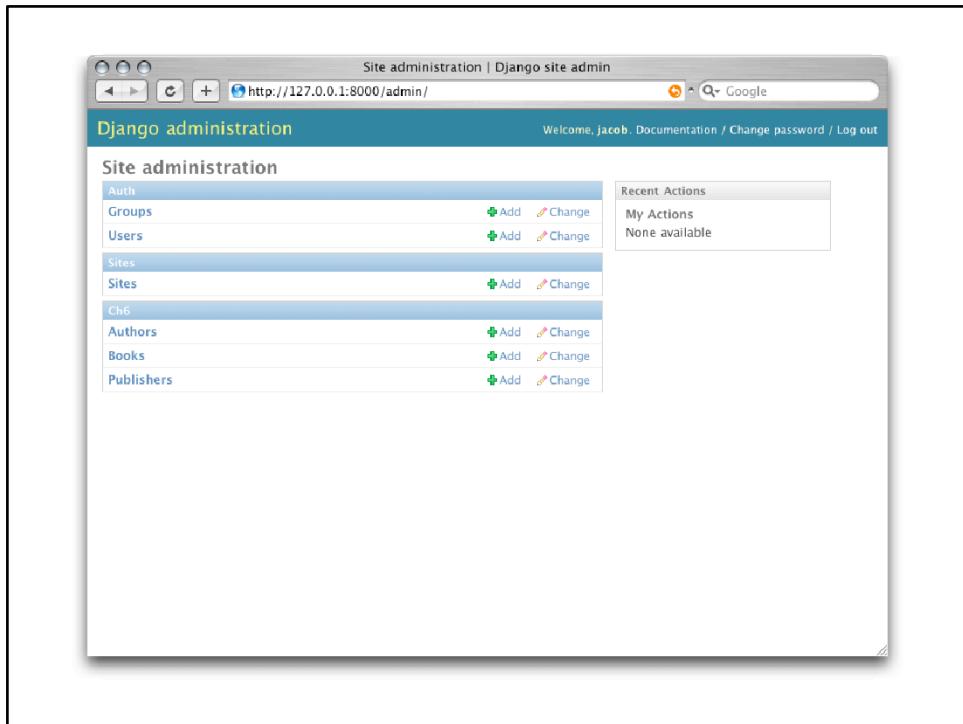
```
from django.shortcuts import render_to_response
from mysite.meeting.models import Meeting

def meeting_detail(request, name):
    meeting = Meeting.objects.get(name=name)
    return render_to_response("meeting_detail.html",
                             {'meeting': meeting})
)
```

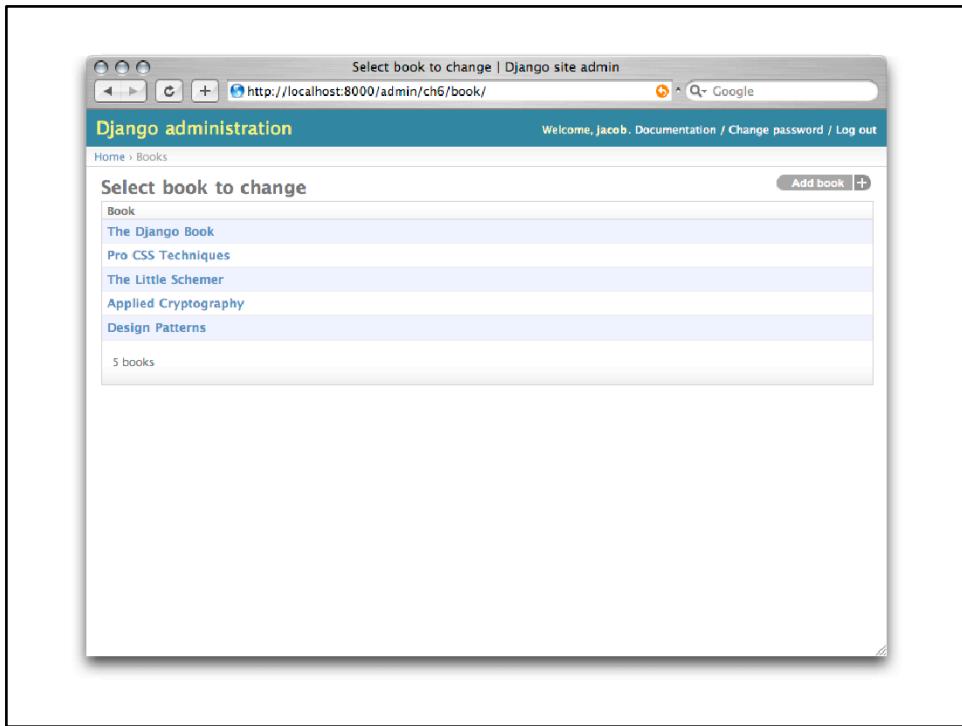
Another way is `get()`, which fetches a specific field for your based on the parameters you send to it. The name parameter comes from the url config. Remember the parenthesis?



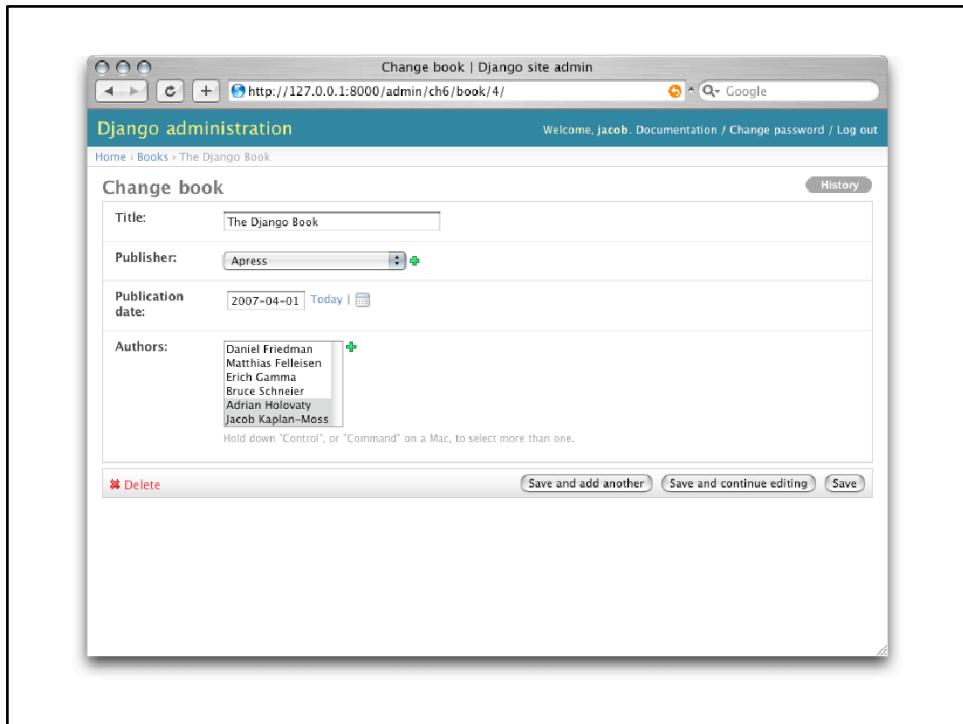
So that's it, now you know everything you need to know about django's functionallity. But now we've only talked about output. How do you insert data into the database then?



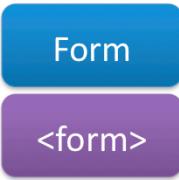
By using Djangos automatic interface. It looks at your model and just renders an interface based on that information. This interface is also highly configurable. Click on Books.



Clicked "the Django Book".



Based on the model the admin also knows how to render the controls for that field. Nice, isn't it? But what if you don't want to use the admin, for instance if you need a comment field?



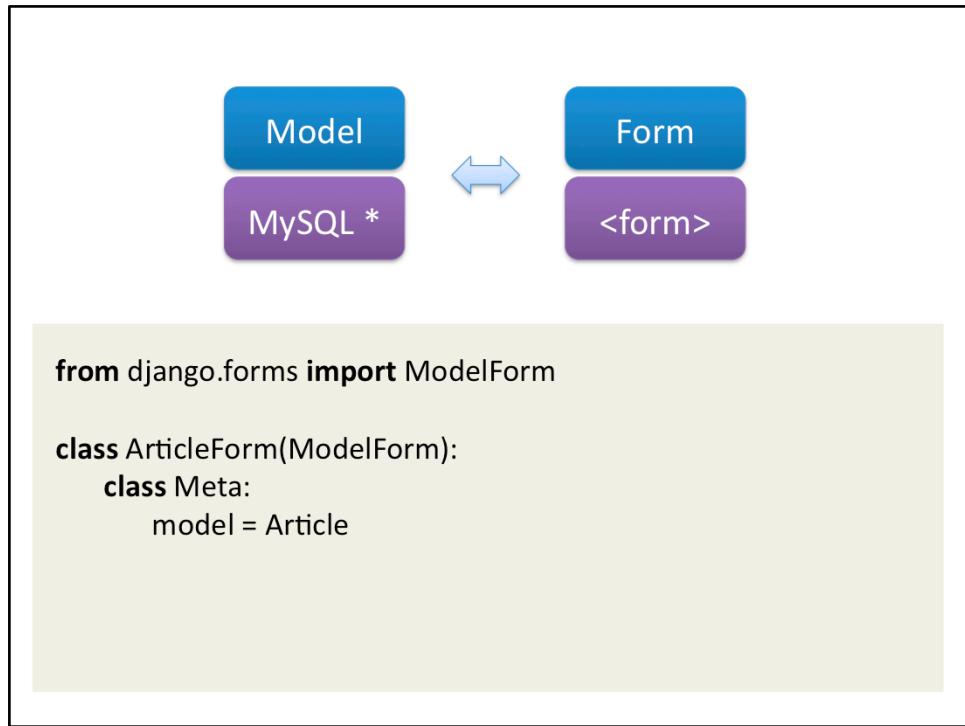
Form

<form>

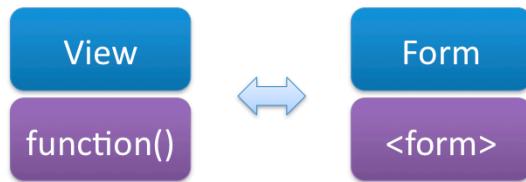
```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

So this is how you make a form in django. Looks pretty similar to how you create a model right?



Well, a little too similar. So this is how you can automatically generate it from a model if you want to.



```
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Process the data in form.cleaned_data
            return HttpResponseRedirect('/thanks/')
    else:
        form = ContactForm()
    return render_to_response('contact.html', { 'form': form })
```

And this is how you process that form in a view. The `#` is a comment, and also the place where you decide what you want to do with your data. Send an e-mail? Save it to the database?

Template

<html>

```
<form action="/contact/" method="POST">
  {{ form.as_p }}
  <input type="submit" value="Submit" />
</form>
```

This is how you render a form. The `as_p` function renders it as `<p>` tags. You can also choose to render it as a unordered list (`as_ul`) or as a table (`as_table`). How bad is the code generated then?

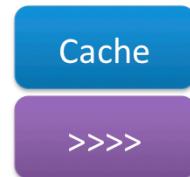
<html>

```
<form action="/contact/" method="POST">
<p><label for="id_subject">Subject:</label>
    <input id="id_subject" type="text" name="subject"
           maxlength="100" /></p>
<p><label for="id_message">Message:</label>
    <input type="text" name="message" id="id_message" /></p>
...
<input type="submit" value="Submit" />
</form>
```

Not bad at all. It has all the little quicks interface developers want it to have. You like it?

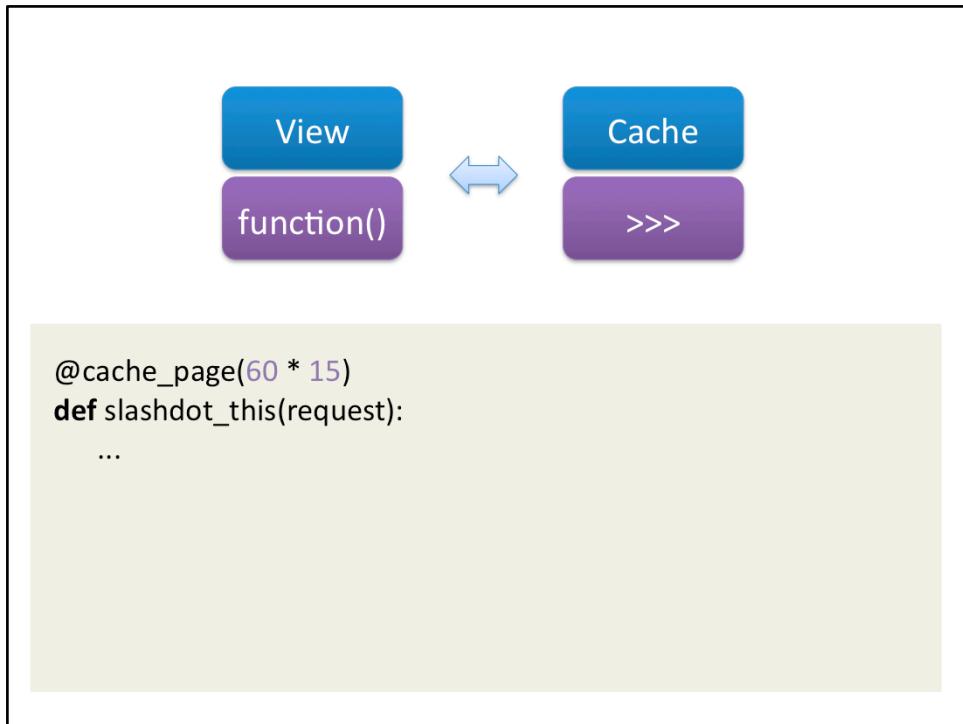


Yes, it does perform. Don't be afraid of python, your database is much slower than python is. So you should instead concentrate in letting the database work as little as possible. How?

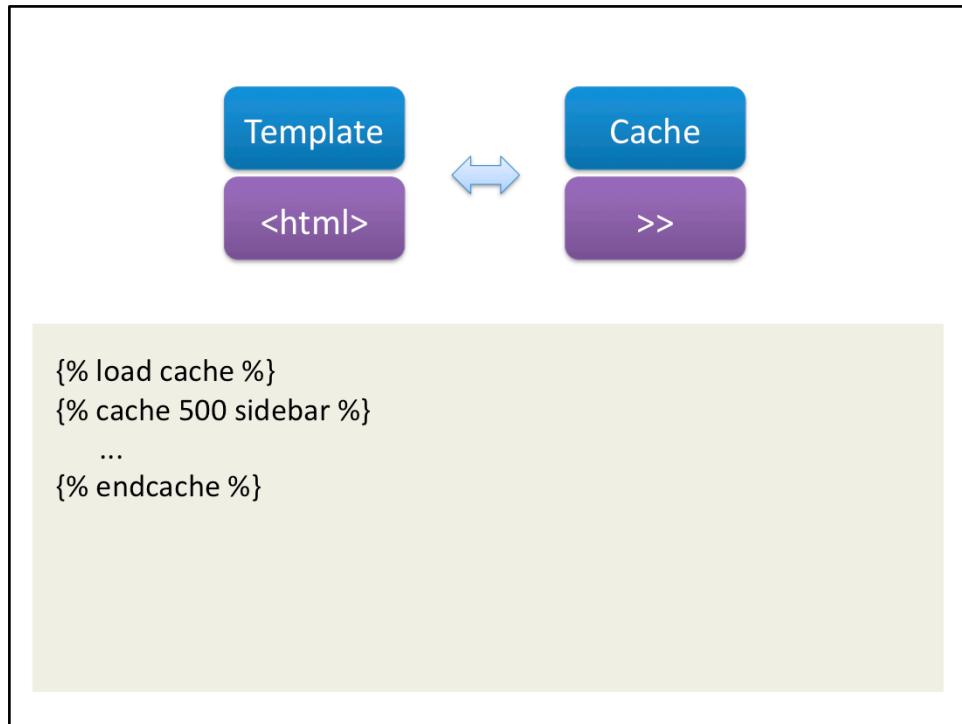


```
MIDDLEWARE_CLASSES = (  
    'django.middleware.cache.UpdateCacheMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.cache.FetchFromCacheMiddleware',  
)
```

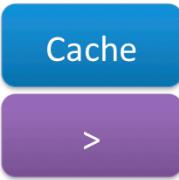
By using caching. Python has four levels of caching, and the first one is on the site level. These two lines in the settings file enables cache on all pages without get and post parameters.



If that's too rough you can also cache on the view level. The `cache_page` thingie takes the number of seconds to cache.



And if that's too rough you can do it on the template level instead. The string after the number of seconds is just a name that identifies this cached block.

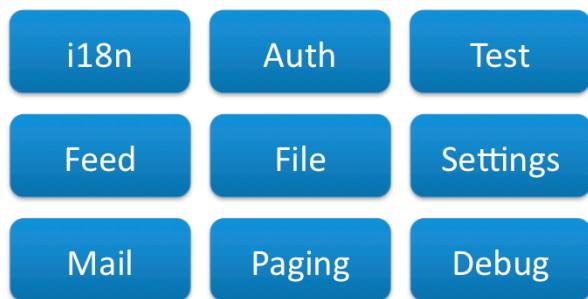


Cache

>

```
from django.core.cache import cache
cache.set('my_key', 'hello, world!', 30)
cache.get('my_key')
```

Or if you really like the gory details, you can go right at the core of things. Django gives you getters and setters directly at the cache that you can use from your views. All of these caching backends goes against the same backend, a setting. Supported backends are Memcached, File system, Database, Memory and a couple of strange ones.



I could talk all night about the different parts of django, but I won't. Just trust me that they are as good as what you've seen this far. So now you probably wonder, what sites use Django?



The Washington Post

Pownce™

Send stuff to your friends.

 **BOKTRAVEN**

 **valtech**

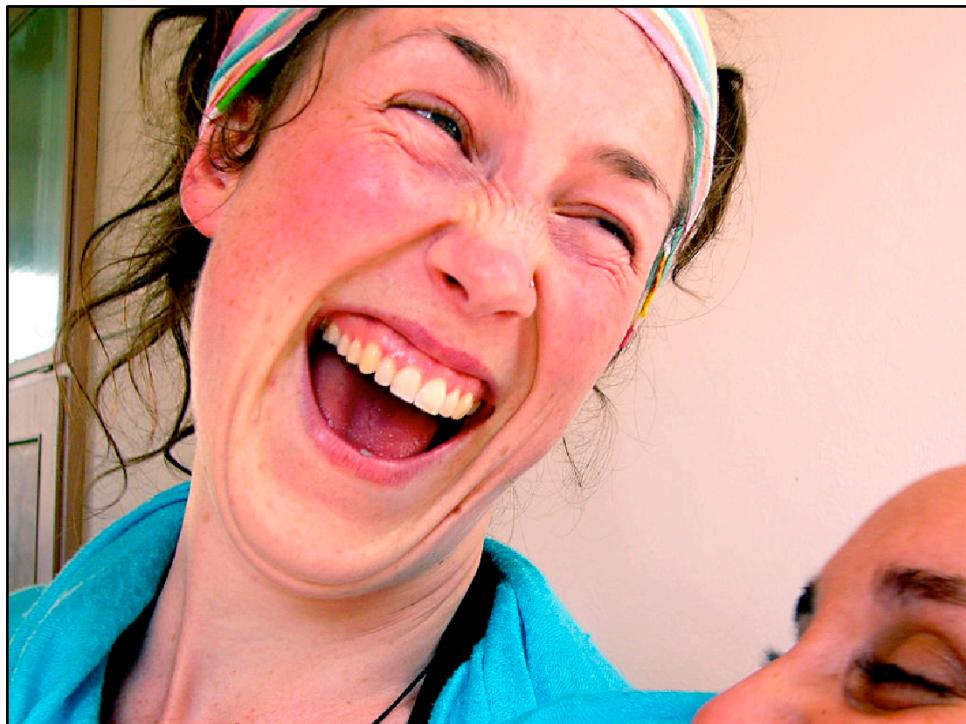
The last two ones are my own projects. Boktraven is not done yet, and I'm building an intranet for Valtech that's obviously not accessible from the outside. You would have liked that wouldn't you? Over all, I would like more big sites to use Django, it would make it easier to convince clients to use it. So what do I think about Django, what are my opinions about it?



Well, it's uncomfortable at first.



But then you realize that it's actually quite easy to use.



But the real reason for using Django is because it's fun. I haven't felt that about any other product or framework I've tried. This is the reason why I'm holding this presentation tonight. I would never have held it about Sharepoint.



One of the best things about Django is its documentation. It rocks. Really.



Also, it's much more hip than Rails.



Hosting is still a problem. Although the few hosts that support it does it well (Compare with the sea view below).



Django is also under active development, and new features are based on real needs, not a need to write a “complete framework”. A framework that concentrates on solving your problems quickly, that’s Django.

Don't clap at me, clap at Django.

I'm Emil Stenström, I blog at <http://friendlybit.com>

Photo credit:

<http://www.flickr.com/photos/ron2/215766370/>
<http://flickr.com/photos/xiaming/50391986/>
<http://flickr.com/photos/dharmaspHERE/32874943/>
<http://www.flickr.com/photos/stephanieasher/2391340330/>
<http://www.flickr.com/photos/orianomada/2244093711/>
<http://www.flickr.com/photos/karoluslinus/2181209112/>
<http://www.flickr.com/photos/timsnell/513350599/>