DANJGO

Exemplo de criação de um Blog

Seu primeiro projeto Django!

- Nós vamos criar um blog simples!
- Na console (em um diretório criado por você):
  - django-admin startproject mysite
- Django-admin é um script que irá criar os diretórios e arquivos que se parece com isso:
- ├── manage.py
- └── mysite
- settings.py
- urls.py
- wsgi.py
- __init__.py

Seu primeiro projeto Django!

- manage.py é um script que ajuda com a gestão do site. Com isso seremos capazes de iniciar um servidor de web no nosso computador sem instalar nada, entre outras coisas.
- O arquivo settings.py contém a configuração do seu site.

Configurando

- Vamos fazer algumas alterações no mysite/ settings.py. Abra o arquivo usando o editor de código que você instalou anteriormente.
- Seria bom ter a hora correta no nosso site.
- Em settings.py, localize a linha que contém TIME_ZONE e modifique para escolher seu próprio fuso horário:
  - TIME_ZONE = ‘America/Bahia'

Configurando

- Nós também precisaramos adicionar um caminho para arquivos estáticos (nós vamos descobrir tudo sobre arquivos estáticos e CSS mais tarde)
- Desça até o *final* do arquivo e logo abaixo da entrada STATIC_URL, adicione um novo um chamado STATIC_ROOT:
  - STATIC_URL = '/static/'
  - STATIC_ROOT = os.path.join(BASE_DIR, 'static')

Instalação de um banco de dados

- Há um monte de software de banco de dados diferente que pode armazenar dados para o seu site.
- Nós vamos usar o padrão, sqlite3.
- Isto já está configurado nesta parte do seu arquivo mysite/settings.py:
  - DATABASES = {
  - 'default': {
  - 'ENGINE': 'django.db.backends.sqlite3',
  - 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
  - }
  - }

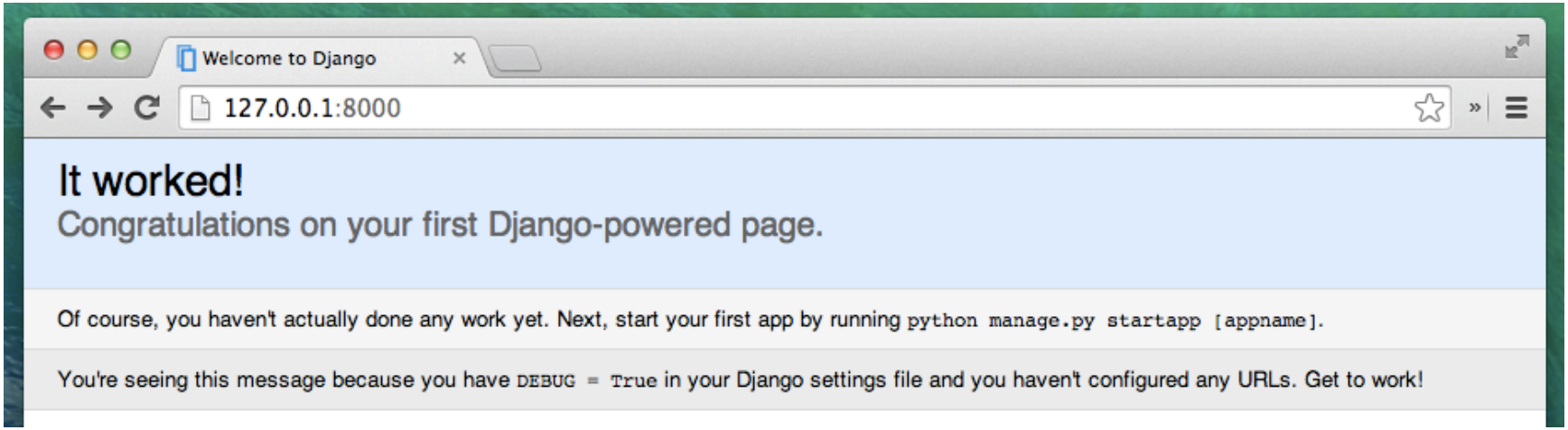
Instalação de um banco de dados

- Para criar um banco de dados para o nosso blog, vamos fazer o seguinte no console. Digite:
  - python manage.py migrate (precisamos estar no diretório que contém o arquivo manage.py)
  - Se isso der certo, você deve ver algo como isto:
    - Operations to perform:
    - Apply all migrations: admin, contenttypes, auth, sessions
    - Running migrations:
    - Applying contenttypes.0001_initial... OK
    - Applying auth.0001_initial... OK
    - Applying admin.0001_initial... OK
    - Applying sessions.0001_initial... OK

E está pronto!

- Hora de iniciar o servidor web e ver se nosso site está funcionando!
- Você precisa estar no diretório que contém o arquivo manage.py
- No console, nós podemos iniciar o servidor web executando o python manage.py runserver
- Abra seu navegador e digite: http://127.0.0.1:8000/

IT WORKED!!!!


Welcome to Django
127.0.0.1:8000
It worked!
Congratulations on your first Django-powered page.
Of course, you haven't actually done any work yet. Next, start your first app by running python manage.py startapp [appname].
You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!

Modelos do Django

- Está na hora de criar algum conteúdo!
- Agora o que nós queremos criar é algo que armazene todos os posts no nosso blog
- Para fazer isso precisamos apenas criar objetos.
- Como nós iremos modelar as postagens do blog então? Queremos construir um blog, certo?

Modelos do Django

- Precisamos responder à pergunta: o que é uma postagem de blog? Que propriedades deve ter?
- Com certeza nosso blog precisa de alguma postagem com o seu conteúdo e um título, certo?
- Também seria bom saber quem a escreveu - então precisamos de um autor.
- Finalmente, queremos saber quando a postagem foi criada e publicada.

Modelos do Django

- Nosso Objeto seria assim:
  - Post
  - --------
    - title
    - text
    - author
    - created_date
    - published_date

Modelos do Django

- Que tipo de coisa pode ser feita com uma postagem?
- Seria legal ter algum método que publique a postagem, não é mesmo?
- Então precisamos de um método chamado publicar.
- Como já sabemos o que queremos alcançar, podemos começar a modelagem em Django!

Modelos do Django

- Um modelo no Django é um tipo especial de objeto - ele é salvo em um banco de dados
- Um banco de dados é uma coleção de dados.
- O banco de dados é um local em que você vai salvar dados sobre usuários, suas postagens, etc.
- Usaremos o SQLite como definido anteriomente.

Criando uma aplicação

- Antes de criar os modelos propriamente, é preciso criar uma aplicação que represente o nosso blog.
- Lembre que até agora nós apenas configuramos um ambiente que poderia ser usado para qualquer aplicação.
- Para criar um aplicativo precisamos executar o seguinte comando no console:
  - python manage.py startapp blog

Criando uma aplicação

- Você vai notar que um novo diretório blog é criado e que ele agora contém um número de arquivos.
- Nossos diretórios e arquivos no nosso projeto devem se parecer com este:
- mysite
- | __init__.py
- | settings.py
- | urls.py
- | wsgi.py
- ├── manage.py
- └── blog
- ├── migrations
- | __init__.py
- ├── __init__.py
- ├── admin.py
- ├── models.py
- ├── tests.py
- └── views.py

Criando uma aplicação

- Depois de criar um aplicativo também precisamos dizer ao Django que deve usá-lo.
- Fazemos isso no arquivo mysite/settings.py.
- Precisamos encontrar o INSTALLED_APPS e adicionar uma linha com 'blog', logo acima do). É assim que o produto final deve ficar assim:

Criando uma aplicação

- INSTALLED_APPS = (
- 'django.contrib.admin',
- 'django.contrib.auth',
- 'django.contrib.contenttypes',
- 'django.contrib.sessions',
- 'django.contrib.messages',
- 'django.contrib.staticfiles',
- 'blog',
-)

Criando o modelo Post do nosso blog

- No arquivo blog/models.py definimos todos os objetos chamados Modelos - este é um lugar em que vamos definir nossa postagem do blog.
- Vamos abrir blog/models.py, remova tudo dele e escreva o código como este:

Criando o modelo Post do nosso blog

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    author = models.ForeignKey('auth.User')
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(
            default=timezone.now)
    published_date = models.DateTimeField(
            blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

    def __str__(self):
        return self.title
```

Criando o modelo Post do nosso blog

- É assustador, não? Mas não se preocupe, vamos explicar o que estas linhas significam!
  - **class Post**(models.Model): - esta linha define o nosso modelo é um objeto do tipo Model.
  - **class** é uma palavra-chave especial que indica que estamos definindo um objeto.
  - **Post** é o nome do nosso modelo, podemos lhe dar um nome diferente (mas é preciso evitar os espaços em branco e caracteres especiais). Sempre comece um nome de classe com uma letra maiúscula.
  - **models.Model** significa que o **Post** é um modelo de Django, então o Django sabe ele que deve ser salvo no banco de dados.

Criando o modelo Post do nosso blog

- Agora podemos definir as propriedades que discutimos: titulo, texto, data_criacao, data_publicacao e autor.
- Para isso precisamos definir um tipo de campo (é um texto? É um número? Uma data? Uma relação com outro objeto, por exemplo, um usuário?).

Criando o modelo Post do nosso blog

- models.CharField - assim é como você define um texto com um número limitado de caracteres.
- models.TextField - este é para textos longos sem um limite. Será ideal para um conteúdo de post de blog, certo?
- models.DateTimeField - este é uma data e hora.
- models.ForeignKey - este é um link para outro modelo.

Criando o modelo Post do nosso blog

- Nós não vamos explicar cada pedaço de código aqui, pois isso levaria muito tempo. Você deve olhar a documentação do DJango se você quiser saber mais sobre campos do Model e como definir coisas além destas descritas anteriormente.
- Que tal **def publish(self):**? exatamente o nosso método de publish que falávamos antes

Criando o modelo Post do nosso blog

- **def**, significa que se trata de um função/ método.
- **publish** é o nome do método.
- Métodos muitas vezes **return** algo
- Há um exemplo disso, o método __str__. Nesse cenário, quando chamamos __str__() teremos um texto (**string**), com um título do Post.

Criando tabelas para nossos modelos no banco de dados

- O último passo é adicionar nosso novo modelo para nosso banco de dados.
- Primeiro temos que fazer o Django saber que nós temos algumas mudanças em nosso modelo (só criamos isso), digite:
  - python manage.py makemigrations blog
- Django prepara um arquivo de migração que temos de aplicar agora para nosso banco de dados, DIGITE:
  - python manage.py migrate blog
- PRONTO O NOSSO MODELO ESTÁ CRIADO!!!

Administração

- Para adicionar, editar e remover postagens (ADMINISTRAR O BLOG) nós criaremos usaremos o Django admin
- Vamos abrir o arquivo blog/admin.py e substituir seu conteúdo por:

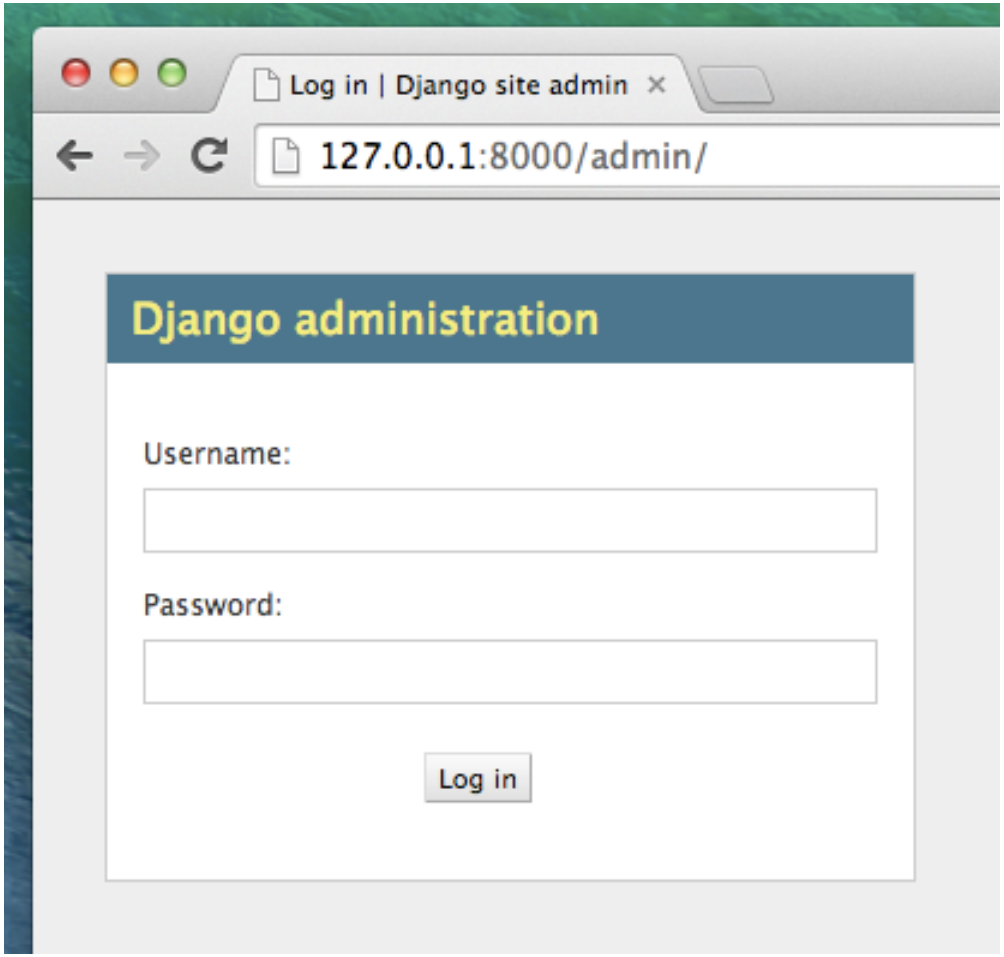
```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Administração

- Como você pode ver, nós importamos (incluímos) o modelo Post definido anteriormente
- Para tornar nosso modelo visível na página de administração, nós precisamos registrá-lo com:
  - admin.site.register(Post)
- OK, hora de olhar para o nosso modelo de Post.
- Lembre-se de executar python manage.py runserver
- Vá para o navegador e digite o endereço http://127.0.0.1:8000/admin/

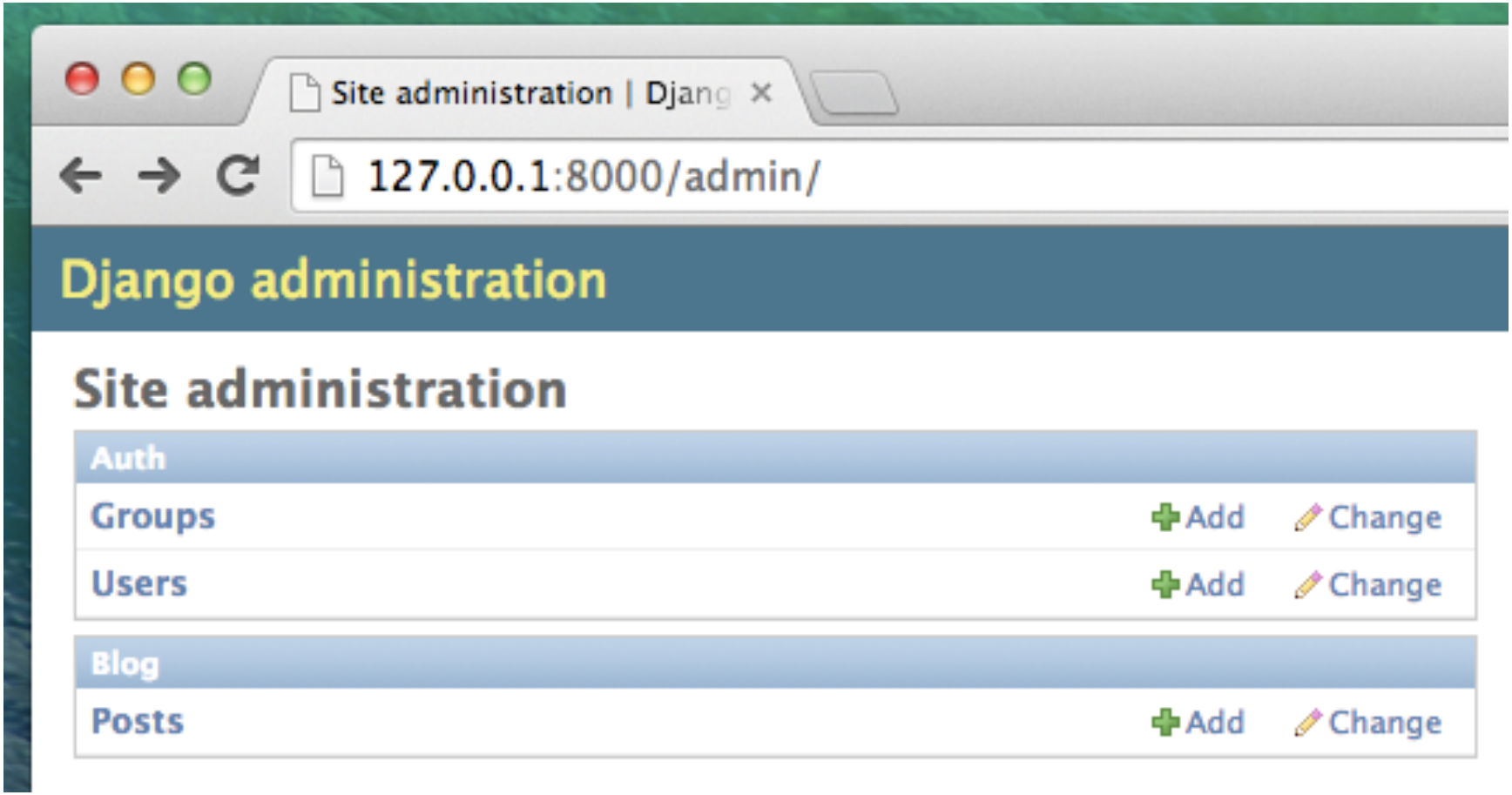
Admistração



Administração

- Para fazer login você precisa criar um *superuser* - um usuário que possui controle sobre tudo do site.
- Volte para o terminal e digite:
  - python manage.py createsuperuser
  - pressione enter e digite seu nome de usuário (caixa baixa, sem espaço), endereço de e-mail e password quando eles forem requisitados.
  - Não se preocupe que você não pode ver a senha que você está digitando - é assim que deve ser
  - Só digitá-la e pressione 'Enter' para continuar.
  - Volte para a o navegador e faça login com as credenciais de superuser que você escolheu

Administração



URLS

- Estamos prestes a construir nossa primeira Web page - uma página inicial para o seu blog!
- Mas primeiro, vamos aprender um pouco mais sobre Django urls.

O que é uma URL?

- Uma URL é simplesmente um endereço da web
- Você pode ver uma URL toda vez que você visita qualquer site
- É visível na barra de endereços do seu navegador
- 127.0.0.1:8000 é uma URL!

O que é uma URL?

- Cada página na Internet precisa de sua própria URL.
- Desta forma seu aplicativo sabe o que deve mostrar a um usuário que abre uma URL.
- Em Django, nós usamos algo chamado **URLconf** (configuração de URL)
- Isso é um conjunto de padrões que Django vai tentar coincidir com a URL recebida para encontrar a visão correta.

Como funcionam as URLs em Django?

- Vamos abrir o arquivo mysite/urls.py e ver com que ele se parece:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
]
```

Como funcionam as URLs em Django?

- Como você pode ver, o Django já colocou alguma coisa lá pra nós.
- As linhas que começam com # são comentários - isso significa que essas linhas não serão executadas pelo Python
- A URL do admin, que você fez no anteriormente já está aqui:
  - url(r'^admin/', include(admin.site.urls)),

Como funcionam as URLs em Django?

- A URL do admin, que você fez no anteriormente já está aqui:
  - url(r'^admin/', include(admin.site.urls)),
- Isso significa que para cada URL que começa com admin / o Django irá encontrar um correspondente *modo de exibição*.

Regex

- Você quer saber como o Django coincide com URLs para views?
- Bem, esta parte é complicada. Django usa o regex -- expressões regulares.
- Regex tem muito (muito!) de normas que formam um padrão de pesquisa.
- Como regexes são um tópico avançado, nós veremos em detalhes como elas funcionam.

Regex

- Se você ainda quiser entender como criamos os padrões, aqui está um exemplo do processo.
- só precisamos um subconjunto limitado de regras para expressar o padrão que procuramos.
  - ^ para o início do texto
  - $ para o final do texto
  - \d para um dígito
  - + para indicar que o item anterior deve ser repetido pelo menos uma vez
  - () para capturar parte do padrão
- Qualquer outra coisa na definição de url será considerada literalmente.

Regex

- Agora imagine que você tem um site com o endereço assim:
  - http://www.mysite.com/post/12345/, onde 12345 é o número do seu post.
- Escrever views separadas para todos os números de post seria muito chato.
- Com expressões regulares podemos criar um padrão que irá coincidir com a url e extrair o número para nós:
  - ^ post/(\d+) / $

Regex

- ^ post/(\d+) / $
  - **^ post /** está dizendo ao Django para pegar tudo que tenha **post /** no início da url (logo após o ^)
  - **(\d+)** significa que haverá um número (um ou mais dígitos) e que queremos o número capturado e extraído
  - **/** diz para o Django que deve seguir outro **/**
  - **$** indica o final da URL significando que apenas sequências terminando com o **/** irão corresponder a esse padrão

Sua primeira url Django!

- É hora de criar nossa primeira URL!
- Queremos http://127.0.0.1:8000 / para ser uma página inicial do nosso blog e exibir uma lista de posts.
- Também queremos manter o arquivo de mysite/urls.py limpo
- aí nós importaremos urls da nossa aplicação blog para o arquivo principal mysite/urls.py

Sua primeira url Django!

- Vá em frente, apague as linhas comentadas
- Depois adicione uma linha que vai importar blog.urls para a url principal ('').
- O seu arquivo mysite/urls.py deve agora se parecer com isto:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'', include('blog.urls')),
]
```

Sua primeira url Django!

- O Django agora irá redirecionar tudo o que entra em 'http://127.0.0.1:8000 /'para blog.urls e procurar por novas instruções lá.
- Ao escrever as expressões regulares em Python é sempre feito com r na frente da sequência
- Isso é só uma dica útil para Python que a seqüência pode conter caracteres especiais que não são destinadas para Python em si, mas em vez disso são parte da expressão regular.

blog.urls

- Crie um novo arquivo vazio blog/urls.py. Tudo bem! Adicione estas duas primeiras linhas:

```
from django.conf.urls import include, url
from . import views
```

- Aqui nós estamos apenas importando métodos do Django e todos os nossos views do aplicativo blog (ainda não temos nenhuma, mas teremos em um minuto!)

blog.urls

- Depois disso nós podemos adicionar nosso primeira URL padrão:

```
urlpatterns = [
    url(r'^$', views.post_list),
]
```

- Como você pode ver, estamos agora atribuindo uma view chamada post_list para ^ $ URL

blog.urls

- Essa expressão regular corresponderá a ^ (um começo) seguido por $ (fim) - então somente uma seqüência vazia irá corresponder.
- E isso é correto, porque em resolvedores de Django url, ' http://127.0.0.1:8000 /' não é uma parte da URL.
- Este padrão irá mostrar o Django que views.post_list é o lugar certo para ir, se alguém entra em seu site no endereço 'http://127.0.0.1:8000 /'.
- Tudo certo? Abra http://127.0.0.1:8000 no seu navegador pra ver o resultado.

blog.urls


AttributeError at /
127.0.0.1:8000
AttributeError at /
'module' object has no attribute 'post_list'
Request Method: GET
Request URL: http://127.0.0.1:8000/
Django Version: 1.6.4
Exception Type: AttributeError
Exception Value: 'module' object has no attribute 'post_list'
Exception Location: /Users/olasitarska/Desktop/Events/mysite/blog/urls.py in <module>, line 5

Funcionou?

- Não tem mais "It Works!' mais hein? Não se preocupe, é só uma página de erro, nada a temer! Elas são na verdade muito úteis:
  - Você pode ler que não há **no attribute 'post_list'**.
  - O *post_list* te lembra alguma coisa? Isto é como chamamos o nosso view!
  - Isso significa que está tudo no lugar, só não criamos nossa *view* ainda.

Views - hora de criar!

- É hora de resolver o bug que criamos anteriormente :)
- Uma *view* é colocada onde nós colocamos a "lógica" da nossa aplicação. Isso mesmo!! Não é MVC…
- Ele irá solicitar informações a partir do model que você criou antes e passá-lo para um template que você vai criar mais adiante.

Views - hora de criar!

- Views, no fundo, não passam de métodos escritos em Python que são um pouco mais complicados do que aquilo que fizemos na **Introdução ao Python**
- As views são postas no arquivo views.py.
- Nós vamos adicionar nossas *views* no arquivo blog/views.py

blog/views.py

- OK, vamos abrir o arquivo e ver o que tem nele:

```
from django.shortcuts import render


# Create your views here
```

- Não tem muita coisa

blog/views.py

- A *view* mais básica se parece com isto:

```
def post_list(request):
    return render(request, 'blog/post_list.html',
{})
```

- Como você pode ver, nós criamos um método (def) chamado **post_list** que aceita o pedido e retornar um método **render** que será processado (para montar) nosso modelo blog/post_list.html.

blog/views.py

- Salve o arquivo, vá para http://127.0.0.1:8000 / e veja o que temos agora.
- Outro erro! Leia o que está acontecendo agora:

blog/views.py

Funcionou?

- Esta é fácil: *TemplateDoesNotExist*. Vamos corrigir este bug e criar um template a seguir !

Seu primeiro template!

- Criar um template significa criar um arquivo de template.
- Os templates são salvos no diretório blog/ templates (crie o seu!!!)
- Em seguida, crie outro diretório chamado blog dentro da diretório templates:

blog

└──templates

  └──blog

Seu primeiro template!

- Você deve estar se perguntando porque nós precisamos de dois diretórios chamados blog
- Essa é uma simples e útil convenção que facilita a vida quando as coisas começarem a ficar mais complicadas.
- E agora nós criamos o arquivo post_list.html (inclua um codigo HTML/text qualquer) dentro do diretório blog/templates/blog.

Seu primeiro template!

```
<html>
    <p>Hi there!</p>
    <p>It works!</p>
</html>
```

Seu primeiro template!

- Agora rode novamente!!!
- Vamos voltar mais adiante a falar de templates.

QuerySets e ORM do Django

- Vamos aprender como Django se conecta ao banco de dados e como ele armazena dados.
- Um **QuerySet** (conjunto de pesquisa), no fundo, é uma lista de objetos de um dado modelo.
- Ele permite que você leia, filtre e ordene os dados do banco.

QuerySets e ORM do Django

- Exemplo, Abra o terminal e digite:
  - python manage.py shell
- Agora você está no console interativo do Django
- Digite:
  - from blog.models import Post
- Digite:
  - Post.objects.all()
- Isso vai exibir a lista dos posts que criamos anteriormente da interface de administração!
  - >>> Post.objects.all()
  - [<Post: Meu Primeiro Post>]

QuerySets e ORM do Django

- Para criar um objeto Post no banco de dados via shell você pode usar:
  - Post.objects.create(author=me, title=’Mais um POST', text=’Este é um texto par o post atual')
- Mas aqui temos um ingrediente que faltava: me
- Importe o modelo User
  - from django.contrib.auth.models import User
- Depois:
  - me = User.objects.get(username=’o nome do seu usuário')
- Ai execute o comando de criação novamente

QuerySets e ORM do Django

- Uma grande parte de QuerySets é a habilidade de filtrá-los
- Digamos que queremos encontrar todos as postagens escritas pelo usuário ‘seu usuario’
- Nós usaremos o **filter** em vez de **all** em **Post.objects.all()**
- Entre parênteses indicamos que as condições precisam ser atendidas por um postagem de blog para acabar em nosso queryset
- Em nosso caso é author que é igual a **me.** Exemplo**:**
  - >>> Post.objects.filter(author=me)
  - [<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>]

QuerySets e ORM do Django

- Outro exemplo. Talvez nós queremos ver todos os posts que contenham a palavra 'title' no campo de title
  - >>> Post.objects.filter(title__contains='title')
  - [<Post: Sample title>, <Post: 4th title of post>]
- Existem dois caracteres de sublinhado (_) entre o title e contains. Django ORM usa esta sintaxe para separar nomes de campo ("title") e operações ou filtros ("contains").
- Se você usar apenas um sublinhado, você obterá um erro como "FieldError: Cannot resolve keyword title_contains".

QuerySets e ORM do Django

- Um QuerySet também nos permite ordenar a lista de objetos
- Vamos tentar ordenar as postagens pelo campo created_date:
  - >>> Post.objects.order_by('created_date')
  - [<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>]
- Você também pode inverter a ordem adicionando (**-**) no início:
  - >>> Post.objects.order_by('-created_date')
  - [<Post: 4th title of post>, <Post: My 3rd post!>, <Post: Post number 2>, <Post: Sample title>]

Django Querysets

- Nós temos diferentes peças aqui: o model **Post** está definido em models.py
- Nós temos **post_list** no views.py e o template adicionado
- Mas como nós faremos de fato para fazer com que as nossas postagens apareçam no nosso template em HTML?
- Nós queremos: pegar algum conteúdo (models salvos no banco de dados) e exibi-lo de uma maneira bacana no nosso template, certo?

Django Querysets

- E isso é exatamente o que as *views* devem fazer: conectar models e templates.
- Na nossa view **post_list** nós vamos precisar pegar os models que queremos exibir e passá-los para o template.
- Então, basicamente, em uma *view* nós decidimos o que (um model) será exibido no template.
- Certo, e como nós faremos isso?

Django Querysets

- Precisamos abrir o nosso blog/views.py.
- Até agora a *view*post_list se parece com isso:

```
from django.shortcuts import render


def post_list(request):
    return render(request, 'blog/post_list.html',
{})
```

Django Querysets

- Agora é o momento em que temos de incluir o model que temos escrito em models.py.
- Vamos adicionar esta linha:
  - from .models import Post
- Para pegar os posts reais do model Post nós precisamos de uma coisa chamada QuerySet.

Django Querysets

- Suponha que estamos interessados em uma lista de posts que são publicados e classificados por **published_date** como fizemos com Querysets
  - Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
- Agora nós colocamos este pedaço de código dentro do arquivo blog/views.py adicionando-o à função def post_list(request):

Django Querysets

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {})
```

Django Querysets

- Note que criamos uma *variável* para nosso o QuerySet: **posts**
- Trate isto como o nome do nosso QuerySet. De agora em diante nós podemos nos referir a ele por este nome
- A última parte que falta é passar o QuerySet posts para o template (Veremos mais adiante)
- Na função **render** já temos o parâmetro **request** (tudo o que recebemos do usuário através da Internet) e um **arquivo de template** 'blog/post_list.html'.

Django Querysets

- O último parâmetro, que se parece com isso: {} é um lugar em que podemos acrescentar algumas coisas para que o template use
- recisamos nomeá-los (ficaremos com 'posts' por enquanto :)).
- Deve ficar assim: {'posts': posts}. Observe que a parte antes de : está entre aspas ''.

Django Querysets

- Então finalmente nosso arquivo blog/views.py deve se parecer com isto:

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

Templates

- Hora de exibir algum dado! Django nos dá **tags de templates** embutidas bastante úteis para isso.
- O que são tags de template?
  - Como pode ver, você não pode colocar código Python no HTML
  - **Tags de template Django** nos permite transformar objetos Python em código HTML
  - Elas permitem que você possa construir sites dinâmicos mais rápido e mais fácil.

Templates

- Para exibir uma variável no Django template, nós usamos colchetes duplos com o nome da variável dentro, exemplo:
  - {{ posts }}
- Tentar fazer isso no seu template blog/templates/blog/post_list.html (substituia o segundo e o terceiro par de tags < div >< / div > pela linha {{ posts }}), salve o arquivo e atualize a página para ver os resultados

Templates


Django Girls blog
127.0.0.1:8000
Django Girls Blog
[<Post: My second post>, <Post: My first post>]

Templates

- O resultutado mostra que o Django entende {{posts}} como uma lista de objetos.
- Podemos usar um loop em python para exibir toda a lista de posts

```
{% for post in posts %}
    {{ post }}
{% endfor %}
```

- Tudo que você põe entre {% for %} e {% endfor %} será repetido para cada objeto na lista

Templates

- Funciona! Mas nós queremos melhorar…

```
<div>
   <h1><a href="/">Django Blog</a></h1>
</div>

{% for post in posts %}
   <div>
      <p>published: {{ post.published_date }}</p>
      <h1><a href="">{{ post.title }}</a></h1>
      <p>{{ post.text|linebreaks }}</p>
   </div>
{% endfor %}
```

Templates

- Funciona bem melhor…
- Você notou que dessa vez nós usamos uma notação um pouco diferente {{ post.title }} ou {{ post.text }}?
- Nós estamos acessando os dados em cada um dos campos que definimos no model do Post