Outline

• Internationalization
• Localization
• Middleware
• Signals (Yeah, sure, we’ll get to that.)
Internationalization & Localization

Je suis un livre.
Internationalization & Localization

- Goal: to allow a single Web application to offer its content in languages and formats tailored to the audience.

- Django does two things:
  - It allows developers and template authors to specify which parts of their apps should be **translated** or **formatted** for local languages and cultures.
  - It uses these hooks to localize Web apps for particular users according to their preferences.

What is the difference between “translation” and “formatting”?
Internationalization & Localization

- HTTP Headers: Accept-Language vs. Content-Language

```
Accept-Language: fr, en-gb;q=0.8, en;q=0.7
Content-Language: da
```

Request
Response
Definitions

• **Internationalization**
  Preparing the software for localization. Usually done by developers.

• **Localization**
  Writing the translations and local formats. Usually done by translators.

• Lazy people are hard at work.
Other Definitions

- **Locale Name**
  \( ll \text{ or } ll\_CC \rightarrow \text{fa\_IR, en\_US, en\_GB, fr} \)

- **Language Code**
  it, fr–ch, fr–fr, pt–br

- **Message File**

- **Translation String**
Translation in **django**

- Need to add a minimum amount of **hooks**, which are called **translation strings**.
“This text should be translated into the end user’s language, if a translation for this text is available in that language.”

–A translation string whispered in django’s ear.
Translation in **Django**

- It’s developer’s responsibility to mark translatable strings.

- Django provides utilities to extract all these marked strings into a message file.

- Translators fill the message file for different languages.

- This relies on GNU gettext toolset.
Internationalization: in Python Code

• Specify a translation string by using the function `ugettext()`.

```python
from django.utils.translation import ugettext as _
from django.http import HttpResponse

def my_view(request):
    output = _("Welcome to my site.")
    return HttpResponse(output)
```

It’s convention to import this as a shorter alias, `_`, to save typing.
Internationalization: in Python Code

```python
def my_view(request):
    words = ['Welcome', 'to', 'my', 'site.]
    output = _(' '.join(words))
    return HttpResponse(output)

def my_view(request):
    sentence = 'Welcome to my site.'
    output = _(sentence)
    return HttpResponse(output)
```

Django `makemessages` utility won't find these strings.
Internationalization: in Python Code

- English: “Today is **December 22.**”
- Spanish: “Hoy es **22 de Diciembre.**”
- Chinese: “今天是**12月22日。**”
- Russian: “Сегодня **22 декабря.**”
- Persian: “**22 دسامبر امروز.**”

```python
def my_view(request, m, d):
    output = _('Today is %(month)s %(day)s.') %
              {'month': m, 'day': d}
    return HttpResponse(output)
```

Don’t forget to use named-string interpolation syntax.
Pluralization

- English: “I have one apple.” and “I have five apples.”
- French: “J'ai une pomme.” and “J'ai cinq pommes.”
- Korean: “나는 1 개의 사과가 있다.” and “난 5 사과가 있다.”
- Persian: “من پنج سیب دارم.” and “من یک سیب دارم.”

```python
from django.utils.translation import ungettext
from django.http import HttpResponse

def hello_world(request, count):
    page = ungettext('I have %(count)d apple.', 'I have %(count)d apples.', count) % {
        'count': count,
    }
    return HttpResponse(page)
```

There is an extra n!
Lazy Translation

from django.utils.translation import ugettext_lazy as _
Internationalization: in Template Code

- `{% trans %}` template tag.

```
<title>{% trans "This is the title." %}</title>

{% trans "This is the title" as the_title %}

<title>{{ the_title }}</title>
<meta name="description" content="{{ the_title }}">
```
Internationalization: in Template Code

• `{% blocktrans %}` template tag.

```latex
{% blocktrans %}This string will have {{ value }} inside.{% endblocktrans %}

{% blocktrans with amount=article.price %}
That will cost $ {{ amount }}.
{% endblocktrans %}

{% blocktrans with myvar=value|filter %}
This will have {{ myvar }} inside.
{% endblocktrans %}

{% blocktrans with book_t=book\title 
author_t=author\title %}
This is {{ book_t }} by {{ author_t }}
{% endblocktrans %}
```
Internationalization: in Template Code

{% load i18n %}

{% get_current_language as LANGUAGE_CODE %}
{% get_available_languages as LANGUAGES %}
{% get_current_language_bidi as LANGUAGE_BIDI %}

{% load i18n %}

{% get_current_language as LANGUAGE_CODE %}
<!-- Current language: {{ LANGUAGE_CODE }} -->
<p>{% trans "Welcome to our page" %}</p>

{% language 'en' %}
  {% get_current_language as LANGUAGE_CODE %}
  <!-- Current language: {{ LANGUAGE_CODE }} -->
  <p>{% trans "Welcome to our page" %}</p>
{% endlanguage %}
Internationalization: in URL Patterns

- Django provides *two* mechanisms to internationalize URL patterns:
  - Adding the language prefix to the root of the URL patterns.
  - Making URL patterns themselves translatable.

- `django.middleware.locale.LocaleMiddleware` must be activated.
from django.conf.urls import patterns, include, url
from django.conf.urls.i18n import i18n_patterns

urlpatterns = patterns('',
    url(r'^sitemap\.xml$', 'sitemap.view', name='sitemap_xml'),
)

news_patterns = patterns('',
    url(r'^$', 'news.views.index', name='index'),
    url(r'^category/(?P<slug>[\w-]+)/$', 'news.views.category', name='category'),
    url(r'^(?P<slug>[\w-]+)/$', 'news.views.details', name='detail'),
)

urlpatterns += i18n_patterns('',
    url(r'^about/$', 'about.view', name='about'),
    url(r'^news/', include(news_patterns, namespace='news')),
)

Only allowed in root URLconf.
Internationalization: in URL Patterns

```python
from django.conf.urls import patterns, include, url
from django.conf.urls.i18n import i18n_patterns
from django.utils.translation import ugettext_lazy as _

urlpatterns = patterns(''
    url(r'^sitemap\.xml$', 'sitemap.view', name='sitemap_xml'),
)

news_patterns = patterns(''
    url(r'^$', 'news.views.index', name='index'),
    url(r'^(?P<slug>[\w-]+)/$', 'news.views.details', name='detail'),
)

urlpatterns += i18n_patterns(''
    url(_(r'^about/$'), 'about.view', name='about'),
    url(_(r'^news/'), include(news_patterns, namespace='news')),
)```

How **django** Discovers Language Preferences

```python
# Internationalization
# https://docs.djangoproject.com/en/1.6/topics/i18n/

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```
How **django** Discovers Language Preferences

```python
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.locale.LocaleMiddleware',  
    'django.middleware.common.CommonMiddleware',  
)
```
How **django** Discovers Language Preferences

1. Looks for language in the URL.

2. Looks for **django_language** key in current session.

3. Looks for a cookie.

4. Looks at HTTP header, Accept-Language.

5. Uses global LANGUAGE_CODE setting.
Localization

• Once the string literals have been tagged for translation, the translation themselves need to be written.

• The first step is to create a **message file** for a new language.

• A message file is a plain-text file, representing a single language, that contains all available translation strings and how they should be represented in the given language.

• They have a `.po` file extension.
Message File

django-admin.py makemessages -l fa

#: path/to/python/module.py:23
msgid "Welcome to my site."
msgstr ""

#: path/to/python/module.py:29
msgid "I have a book."
msgstr ""
Compiling Message Files

- After creating message files or making changes to it, you need to *compile* it into a more efficient form.

```bash
django-admin.py compilemessages
```

- It runs over all of `.po` files and creates `.mo` files.
Internationalization and Localization

- Showtime!
Any Questions?
References

- https://docs.djangoproject.com/en/1.6/topics/i18n/
- https://docs.djangoproject.com/en/1.6/topics/i18n/translation/
- http://www.python.org