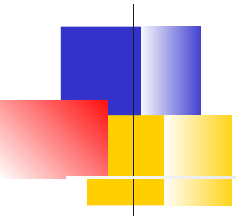


Django Notes

Python lists

Python list syntax looks like array.

```
> fruit = [ "apple", 'banana', "orange" ]
> len(fruit)              # not OO syntax
3
> fruit[1]
'banana'
> fruit[1] = "mango"  # change an element
> fruit.pop()
"orange"
> fruit
['apple', 'mango']
> fruit.append('fig')
```

Python dictionary

A **key-value** collection, like `Map` in Java.

```
> langs = {"python":"easy", "java":"cool",
           "ruby":"weird"}
> langs.keys()    # order is not preserved
dict.keys(['ruby', 'java', 'python'])
> langs['java']
'cool'
> langs['ruby'] = "too much like Perl"
> for lang in langs:  # iterate over all keys
    print("{0} is {1}".format(lang, langs[lang]))
ruby is too much like Perl
java is cool
python is easy
```

More fluent Python

Instead of:

```
for lang in langs:  # iterate over all keys
    print(lang, "is", langs[lang]))
```

Python programmers would write:

```
for (key,value) in langs.items(): # get pairs
    print(key, "is", value)
```

Syntax is similar to *multiple assignment*:

```
(name,id,email) = "Nok,1234,nok@gmail".split(',')
```

Django Page Templates

In a template, you put values inside {{ ... }}

templates/polls/details.html:

```
<p>
Q{{question.id}} is {{question.question_text}}
</p>
```

```
Q1 is What is your favorite food?
```

Django template processing

A template processes an input file and "renders" it:

Create a Django "template" object containing your template text:

```
from django.template import loader
template =
      loader.get_template('polls/details.html')
```

Insert values into template:

```
# insert variables using a dictionary
template.render( {'question': question}, request )
```

Page Templates & Context

The data used in a template is called a context.

Context is *dictionary* (map) of key-value pairs.

```
template =
      loader.get_template('polls/details.html')

# context: key-values to use in template
context = {'question': question, 'user':user, ...}

# render the template using context data
return HttpResponse(
        template.render( context, request ) )
```

Python **kwargs

A Python method may have a parameter like: **`**name`**

`**name` is a *dictionary* of named arguments (key word args) and values. You can use any name for the param.

This enables a method to accept arbitrary parameters.

```
def myfun(x, **kwargs):
    print("x=", x)    # required parameter
    print("Optional arguments are:")
    for key in kwargs:
        print(key, "=", kwargs[key] )

>>> myfun(4)
>>> myfun("hi",id=219245,name="ISP",size=37)
```

In a "view" what is request?

What is `request`? What is `HttpResponse`?

A Django "view" function looks like this:

```
from django.http iport HttpResponse
from django.template import loader

def detail(request, question_id):
    questions = Question.objects.all()[0:10]
    context = {'question_list':questions}
    template = loader.get_template('some_file')

    return HttpResponse(
        template.render(context, request ) )
```

Shortcut for render and return

Rendering a template is common operation.

Django has a "render" shortcut for previous code:

```
from django.shortcuts import render
from django.template import loader

def detail(request, question_id):
    questions = Question.objects.all()[0:10]
    context = {'question_list':questions.
               'username': user.name }

    return render(request, 'some_file', context)
```

URL Dispatching

Each "app" can have a `urls.py` to match request URLs and dispatch them to a "view".

```
from django.urls import path

# app_name is used to define a namespace
# (used for "reverse mapping")
app_name = 'polls'

url_patterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/',
              views.detail, name='detail'),
    path('<int:question_id>/vote',
              views.vote, name='vote'),
    path('<int:question_id>/results',
              views.results, name='results'),
    ]
```

Dispatch these URLs

Which view would handle each of these requests:

1) http://localhost:8000/polls/

2) http://localhost:8000/polls/4/

3) http://localhost:8000/polls/8/vote?username=nok

4) http://localhost:8000/polls/8/vote/summary

```
# URL mapping for /polls/ app
url_patterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/',
              views.detail, name='detail'),
    path('<int:question_id>/vote/',
              views.vote, name='vote'),
    ]
```

Mapping from View to URL

Inside html template, we want to insert a URL of a view.

Example: add a link to the polls index page.

How to "build" this URL inside a template?

```
<!-- question details template -->
<p>Question Id: {{ question.id }} </p>
<p>Text: {{ question.question_text }} </p>

<a href="{% url 'polls:index' %}">
    Back to Polls index
</a>
```

app_name

view name

```
>> Notice that {%...%} is processed even inside "..."
```

Why is this (creating URL for a view) important?

Reverse Dispatch

Sometimes a view controller wants to redirect the user to a different URL.

```
from django.http import HttpResponseRedirect

def vote(request,question_id):
    question = Question.objects.get(id=question_id)
    // TODO save the vote for this question
    ...
    // Show all votes for this question
    _____Redirect to polls/{id}/results_________
```

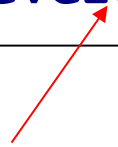
How to redirect the browser to this page?

Reverse Dispatch: reverse()

Redirect uses info from the urls.py files to construct the URL the user should go to.

```
from django.http import HttpResponseRedirect

def vote(request,question_id):
    q = Question.objects.get(id=question_id)
    ## TODO get user's choice and add +1 to votes
    ...
    # Redirect browser to page of vote results
    HttpResponseRedirect(
        reverse('polls:results',args=(q.id)) )
```



Get the URL that matches the named route

Thorough Testing is Needed!

Python code is *interpretted*.

There is no pre-compilation to catch errors (as in Java).
So, you need to **test every path of execution**.

```
NameError at /polls/1/vote/
name 'reverse' is not defined
```

Programmer forgot (in views.py):

```
from django.urls import reverse
```

but error is not detected until `reverse`() is encountered at run-time.

All Frameworks must do this

Most web apps need a way to:

1. Include links to other app urls in HTML page
   - Amazon products page has links to each product
2. Redirect user to another page in our app
   - After add item to cart, redirect to view_cart page.

Issue:

How to *inject* the correct URLs, without hardcoding them?

Django's Solution

Most web apps need a way to:

1. Include links to other app urls in HTML page

```
{% url 'app_name:view_name' args %}
```

2. Redirect user to another page in our app

```
HttpResponseRedirect(
    reverse('app_name:view_name',
    args=(...)))
```

Rationale:

Make "apps" reusable by providing a naming of URL mappings at the app level, e.g. "`polls:results`".

Exploring Models

Use Django to start an interactive Python shell.

This is described in Tutorial part 2.

```
python manage.py shell  [ -i python ]

>>> from polls.models import Question, Choice
>>> q = Question.objects.get(id=1)
>>> q.question_text
"What is your favorite programming language?"
>>> choices = q.choice_set.all( )
>>> for c in choices:
...     print("%-10s %d" % (c.choice_text, c.votes))
Basic      0
C          1
Java       4
Python     2
```

Try out Persistence

Try persistence operations: save(), get(), delete()

```
>>> c = Choice()
>>> c.choice_text = "Lisp"  # or "Racket" ("Scheme")
>>> c.votes = 2
## Foreign Key.  You have to find this separately.
>>> c.question_id = 1
>>> c.save()
>>> for choice in q.choice_set.all():
...     print(choice)
## Now the output includes "Lisp"
>>>
```

Persistence Operations: CRUD

All Persistence Frameworks provide a way to...

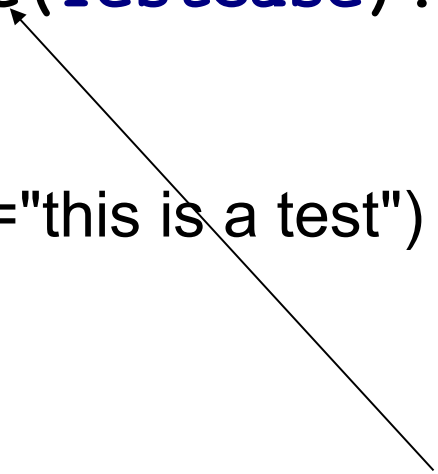
- save (create) an entity to database
- retrieve an object, by id or by field value (query)
- retrieve all objects
- update object data in database
- delete a persisted object from database

How does Django do these?

Testing

Django Unit Tests extend TestCase class.

```
public class QuestionModelTest(TestCase):
    def test_create_question(self):
        question = Question(question_text="this is a test")
        self.assert
```



Wrong Name!

In Tutorial, name is "`QuestionModel`**`Tests`**".

It should be "`xxxTest`" (no "s")!

Don't use plural for your test classes.

What is a django.test.TestCase ?

```
>>> from django.test import TestCase
>>> help(TestCase)
class TestCase(TransactionTestCase)
    ...
    Method resolution order:
        TestCase
        TransactionTestCase
        SimpleTestCase
        unittest.case.TestCase
        builtins.object
```

Running Tests

```
cmd>  python manage.py test polls
```

Criticisms:

- Django test code is in same directory as production code.
- Should have separate "test" files for each test target, don't bundle them into `tests.py`
- `tests.py` is poor name. Test what? Don't use plural!

Design: Low Coupling

Good software design strives for low coupling.

Especially, low or no coupling between unrelated parts.

What features of Django reduce coupling?

1. project divided into independent "apps"
2. views and their templates are not coupled to other views
3. model classes aren't coupled to views or anything else

Design: Portability and Reuse

Good software design enables portability and code reuse.

A framework itself is both portable and reusable (we use it to create our own web app)!

How does Django enable us to move or reuse our own web application code?

Django and Git

When you commit your Django project to Git,

what files should you not import?

```
__pycache__/
*.pyc
sitename/settings.py
db.sqlite3
```

> Add them to .gitignore.

> If you don't know, create a repo on Github and ask Github to create a Python `.gitignore` file for your repo.

> What is `*.pyc` ? What is `*.py[cod]` ? (Github uses this)