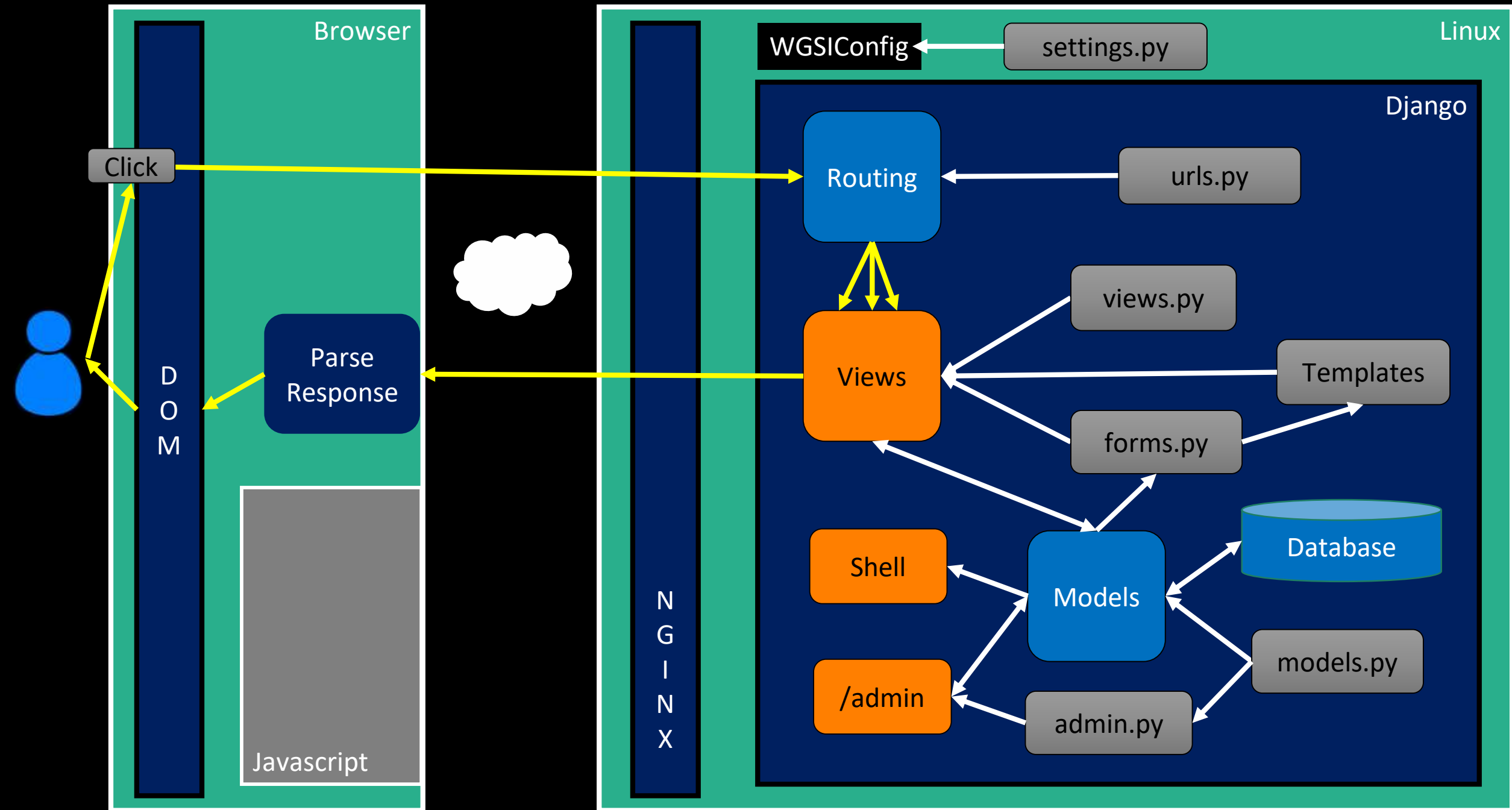# Table of Contents

This slide deck consists of slides used in 7 lecture videos in Week 2. Below is a list of shortcut hyperlinks for you to jump into specific sections.

Charles Severance
www.dj4e.com

# Views and Templates

# Views are the core of our application

- Django looks at the incoming request URL and uses urls.py to select a view

- The view from views.py
  - Handle any incoming data in the request and copy it to the database through the model
  - Retrieve data to put on the page from the database though the model
  - Produce the HTML that will become the response and return it to the browser

```
https://samples.dj4e.com/
```

# Reading the URL

- When Django receives an HTTP request it parses it, uses some of the URL for routing purposes and passes parts of the URL to your code

Django Application (also folder)

View within application

`https://samples.dj4e.com/views/funky`

Key / value parameter (GET)

`https://samples.dj4e.com/views/danger?guess=42`

`https://samples.dj4e.com/views/rest/24`

URL Path Parameter

# URL Dispatcher

A clean, elegant URL scheme is an important detail in a high-quality Web application. Django lets you design URLs however you want, with no framework limitations.

To design URLs for an app, you create a Python module informally called a URLconf (URL configuration). This module is pure Python code and is a mapping between URL path expressions to Python functions (your views).

This mapping can be as short or as long as needed. It can reference other mappings. And, because it's pure Python code, it can be constructed dynamically.

https://docs.djangoproject.com/en/3.0/topics/http/urls/

# Three patterns for views (in urls.py)

- Requests are routed to a pre-defined class from Django itself

- Requests are routed to a function in views.py that takes the http request as a parameter and returns a response

- Requests are routed to a class in views.py that has get() and post() methods that take the http request as a parameter and return a response

```python
from django.urls import path
from . import views
from django.views.generic import TemplateView

# https://docs.djangoproject.com/en/3.0/topics/http/urls/
app_name='views'
urlpatterns = [
    # pre-defined class from Django
    path('', TemplateView.as_view(template_name='views/main.html')),
    # function from views.py
    path('funky', views.funky),
    path('danger', views.danger),
    path('game', views.game),
    path('rest/<int:guess>', views.rest),
    path('bounce', views.bounce),
    # our class from views.py
    path('main', views.MainView.as_view()),
    path('remain/<slug:guess>', views.RestMainView.as_view()),
]
```

# Viewing the Views

```
path('', TemplateView.as_view(template_name='views/main.html'))
```

**views/urls.py**

**views/templates/views/main.html**

```html
<html><body><p>This is the views main.html sample</p>
<p>
<ul>
    <li>This page is coming from a file in views/templates/main.html</li>
    <li><a href="funky">Use a view function</a></li>
        ...
</ul>
</p>
<p>This sample code is available at
<a href="https://github.com/csev/dj4e-samples" target="_blank">
https://github.com/csev/dj4e-samples</a>
</p>
</body></html>
```

# Request and Response Objects

Django uses request and response objects to pass information throughout your Django application.

When a page is requested by a browser, Django creates an **HttpRequest** object that contains metadata about the request.

Then Django loads the appropriate view, passing the **HttpRequest** as the first argument to the view function. Each view is responsible for returning an **HttpResponse** object.

The Application Programming Interfaces (APIs) for **HttpRequest** and **HttpResponse** objects, are defined in the **django.http** module.

https://docs.djangoproject.com/en/3.0/ref/request-response/

# class HttpRequest

**Attributes**
All attributes should be considered read-only, unless stated otherwise.

**HttpRequest.scheme**
A string representing the scheme of the request (http or https usually).

**HttpRequest.body**

The raw HTTP request body as a bytestring. This is useful for processing data in different ways than conventional HTML forms: binary images, XML payload etc. For processing conventional form data, use HttpRequest.POST.

# class HttpResponse

In contrast to HttpRequest objects, which are created automatically by Django, HttpResponse objects are your responsibility.

Each view you write is responsible for instantiating, populating, and returning an HttpResponse.

**Passing strings**

Typical usage is to pass the contents of the page, as a string or bytestring, to the HttpResponse constructor.

https://docs.djangoproject.com/en/3.0/ref/request-response/#django.http.HttpResponse

https://samples.dj4e.com/views/funky

```
path('funky', views.funky),
```

```python
from django.http import HttpResponse
from django.http import HttpResponseRedirect

# Create your views here.


def funky(request):
    response = """<html><body><p>This is the funky function sample</p>
    <p>This sample code is available at
    <a href="https://github.com/csev/dj4e-samples">
    https://github.com/csev/dj4e-samples</a></p>
    </body></html>"""
    return HttpResponse(response)
```
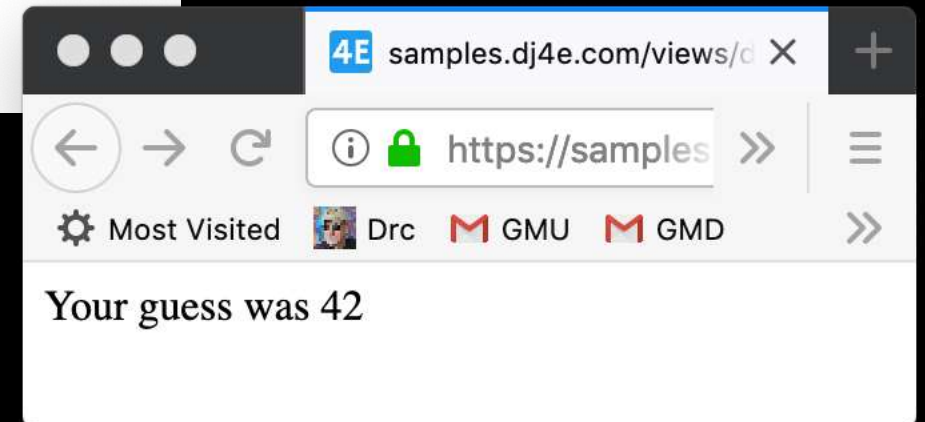
https://samples.dj4e.com/views/danger?guess=42

```
path('danger', views.danger),
```
views/urls.py

```python
from django.http import HttpResponse
from django.http import HttpResponseRedirect

# Create your views here.

def danger(request) :
    response = """<html><body>
    <p>Your guess was """+request.GET['guess']+"""</p>
    </body></html>"""
    return HttpResponse(response)
```
views/views.py

samples.dj4e.com/views/d

https://samples

Most Visited    Drc    GMU    GMD

Your guess was 42

# Danger – What Danger?

I use the django.http.escape() function whenever I sense danger...
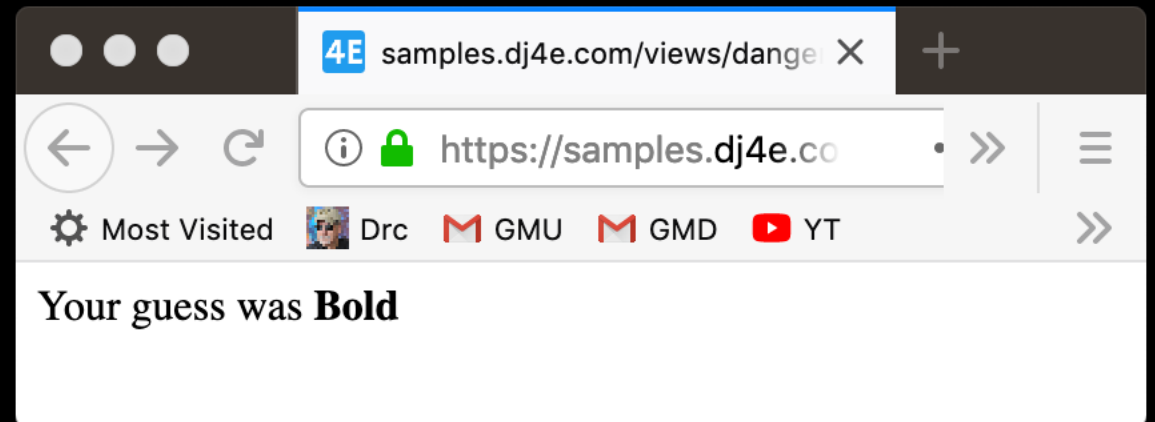
# Why is that view named danger?

- It is dangerous to take data from the user and include it in the HTTP Response without "escaping" the output.
- HTML + JavaScript is a programming language and you don't want your users "sending code" to other user's browsers

- Cross-Site Scripting (XSS)
  - https://en.wikipedia.org/wiki/Cross-site_scripting
  - https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references

https://samples.dj4e.com/views/danger?guess=%3Cb%3EBold%3C%2Fb%3E

```
def danger(request) :
    response = """<html><body>
    <p>Your guess was """+request.GET['guess']+"""</p>
    </body></html>"""
    return HttpResponse(response)
```
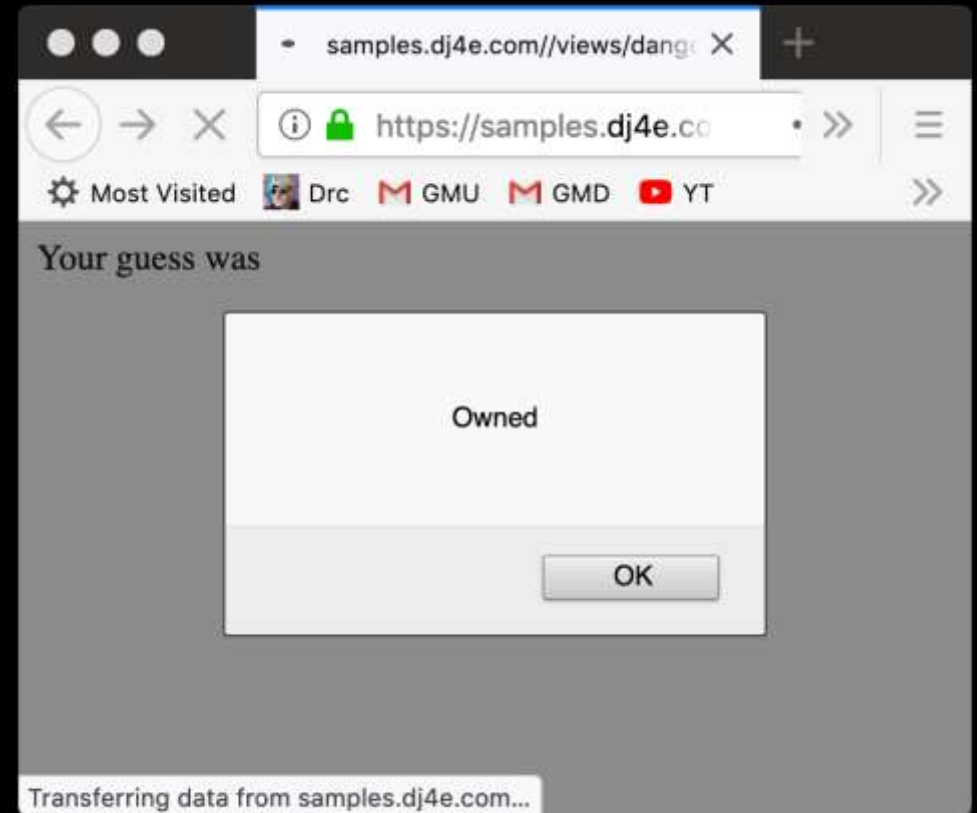
## Response Source

```
<html><body>
<p>Your guess was <b>Bold</b></p>
</body></html>
```



Your guess was **Bold**

https://samples.dj4e.com/views/danger?guess=%3Cscript%3Ealert%28%27Owned%27%29%3B%3C%2Fscript%3E

Response Source

```
<html><body>
<p>Your guess was
<script>alert('Owned');</script></p>
</body></html>
```

https://samples.dj4e.com/views/game?guess=%3Cb%3EBold%3C%2Fb%3E

# HTML entities
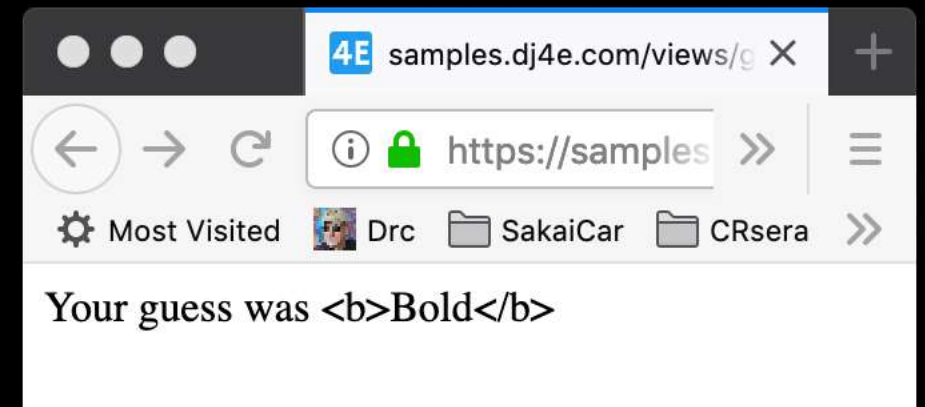
```python
from django.utils.html import escape

def game(request) :
    response = """<html><body>
    <p>Your guess was """+escape(request.GET['guess'])+"""</p>
    </body></html>"""
    return HttpResponse(response)
```

```html
<html><body>
    <p>Your guess was &lt;b&gt;Bold&lt;/b&gt;</p>
    </body></html>
```

samples.dj4e.com/views/g

https://samples

Most Visited    Drc    SakaiCar    CRsera

Your guess was <b>Bold</b>

# Parsing the URL after the Application and View

**https://samples.dj4e.com/views/rest/41**

```
urlpatterns = [
    path('rest/<int:guess>', views.rest),
]
```

**<type:parameter-name>**

```
from django.http import HttpResponse
from django.utils.html import escape

def rest(request, guess) :
    response = """<html><body>
    <p>Your guess was """+escape(guess)+"""</p>
    </body></html>"""
    return HttpResponse(response)
```

# Class Views – Inheritance

```
path('main', views.MainView.as_view()),
```

```python
from django.http import HttpResponse
from django.utils.html import escape
from django.views import View

class MainView(View) :
    def get(self, request):
        response = """<html><body><p>Hello world MainView in HTML</p>
        <p>This sample code is available at
        <a href="https://github.com/csev/dj4e-samples">
        https://github.com/csev/dj4e-samples</a></p>
        </body></html>"""
        return HttpResponse(response)
```

https://www.py4e.com/lessons/Objects

# Parameters to Class Views

`https://samples.dj4e.com/views/remain/abc123-42-xyzzy`

```python
path('remain/<slug:guess>', views.RestMainView.as_view()),
```

```python
from django.http import HttpResponse
from django.utils.html import escape
from django.views import import View

class RestMainView(View) :
    def get(self, request, guess):
        response = """<html><body>
        <p>Your guess was """+escape(guess)+"""</p>
        </body></html>"""
        return HttpResponse(response)
```

https://www.py4e.com/lessons/Objects

# HTTP Status Codes

- http://www.dr-chuck.com/page1.htm - 200 OK
- http://www.dj4e.com/nowhere.htm - 404 Not Found
- 500 Server Error
- http://www.drchuck.com/ - 302 Found / Moved
  Also known as "redirect"

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

# HTTP Location Header

- You can send a "Redirect" response instead of a page response to communicate a "Location:" header to the browser

- The location header includes a URL that the browser is supposed to forward itself to.

- It was originally used for web sites that moved from one URL to another.

http://en.wikipedia.org/wiki/URL_redirection

# Sending a Redirect from a View

**https://samples.dj4e.com/views/bounce**

```
path('bounce', views.bounce)
```

```python
from django.http import HttpResponse
from django.http import HttpResponseRedirect

# This is a command to the browser
def bounce(request) :
    return HttpResponseRedirect('https://www.dj4e.com/simple.htm')
```
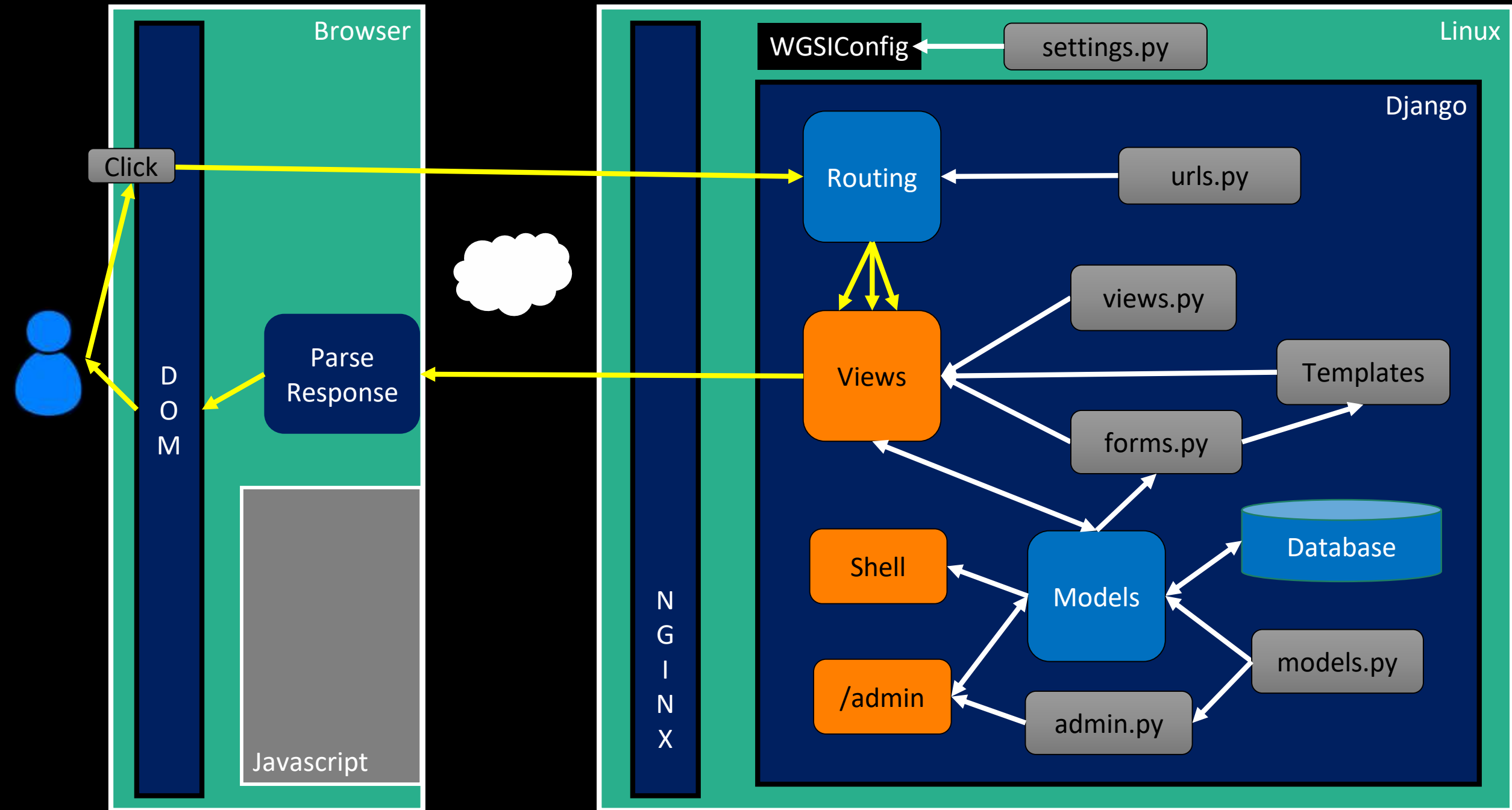
https://docs.djangoproject.com/en/3.0/ref/request-response/#django.http.HttpResponseRedirect

# Templates to Organize HTML

https://github.com/csev/dj4e-samples/tree/master/tmpl

# Templates

Being a web framework, Django needs a convenient way to generate HTML dynamically. The most common approach relies on templates. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

A Django project can be configured with one or several template engines (or even zero if you don't use templates). Django ships built-in backends for its own template system, creatively called the Django template language (DTL), and for the popular alternative Jinja2.

A template is simply a text file. It can generate any text-based format (HTML, XML, CSV, etc.). A template contains **variables**, which get replaced with values when the template is evaluated, and **tags**, which control the logic of the template.

https://docs.djangoproject.com/en/3.0/topics/templates/

# What is a Template?

- Concatenation and escaping can get tiresome and lead to very obtuse looking view code

```python
from django.http import HttpResponse
from django.utils.html import escape
from django.views import View

class RestMainView(View) :
    def get(self, request, guess):
        response = """<html><body>
        <p>Your guess was """+escape(guess)+"""</p>
        </body></html>"""
        return HttpResponse(response)
```
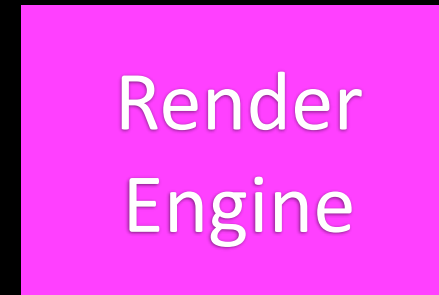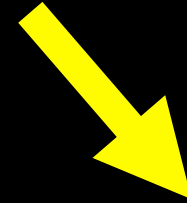
https://samples.dj4e.com/views/rest/24

# Template Render Process

Render Data

Template

Render Engine

Rendered Output

# Template Render Process

`{ 'dat' : 'Fun > Stuff' }`

```
<h1>Hi!</h1>
<pre>
{{ dat }}
</pre>
```

Render Engine

```
<h1>Hi!</h1>
<pre>
Fun &gt; Stuff
</pre>
```

https://samples.dj4e.com/tmpl/game/200

```
path('game/<slug:guess>', views.GameView.as_view())
```

```python
from django.shortcuts import render
from django.views import View

class GameView(View) :
    def get(self, request, guess) :
        x = {'guess' : int(guess) }
        return render(request, 'tmpl/cond.html', x)
```

https://samples.dj4e.com/tmpl/game/200

dj4e-samples/tmpl/templates/tmpl/cond.html

```html
<html>
<head>
    <title>A conditional template</title>
</head>
<body>
    <p>Your guess was {{ guess }}</p>
    {% if guess < 42 %}
        <p>Too low</p>
    {% elif guess > 42 %}
        <p>Too high</p>
    {% else %}
        <p>Just right</p>
    {% endif %}
</body>
</html>
```

```python
from django.views import View

class GameView(View) :
    def get(self, request, guess) :
        x = {'guess' : int(guess) }
        return render(request, 'tmpl/cond.html', x)
```
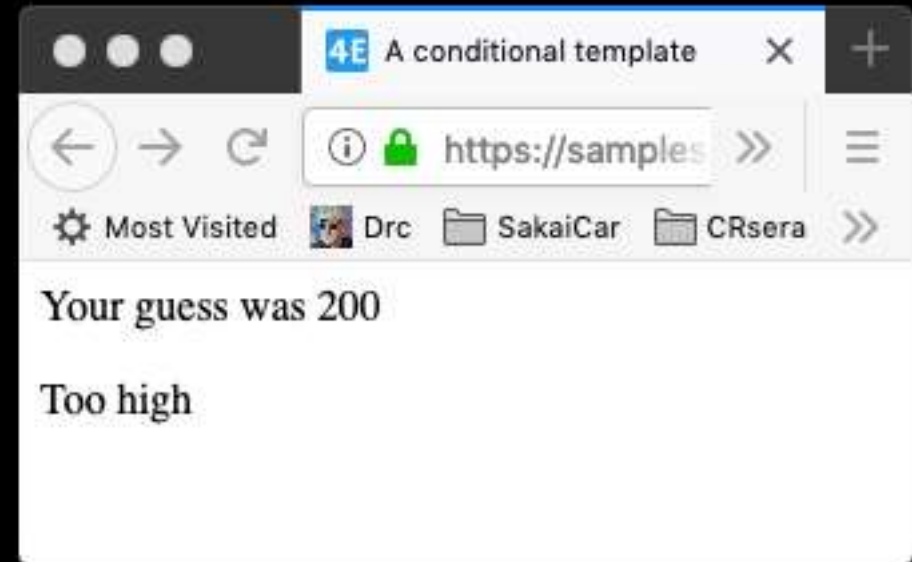


Your guess was 200

Too high

# Where are Templates?

- A Django project is made up of one or more applications in folders

```
dj4e-samples$ ls
LICENSE.md          form            pics
README.md           forums          requirements.txt
autos               getpost         rest
bookmany            gview           route
bookone             hello           scripts
crispy              home            session
db.sqlite3          manage.py       tmpl
dj4e-samples        many            tracks
favs                menu            users
favsql              myarts          views
```

# Templates in Folders

- It is common to reuse the "name" of a template file in several applications
- We use a technique called "namespace" so that each application can load its own templates without template name collision

```
dj4e-samples$ ls */templates/*/detail.html favs/templates/favs/detail.html
favsql/templates/favsql/detail.html
forums/templates/forums/detail.html
pics/templates/pics/detail.html
dj4e-samples$
```

https://en.wikipedia.org/wiki/Namespace
https://docs.djangoproject.com/en/3.0/topics/http/urls/#url-namespaces

# Templates in Name Spaces

- For the namespace to work, we need to put templates in a path that includes the application name twice. Weird but necessary. ☹

```
dj4e-samples$ ls */templates/*/detail.html favs/templates/favs/detail.html
favsql/templates/favsql/detail.html
forums/templates/forums/detail.html
pics/templates/pics/detail.html
dj4e-samples$
```

https://en.wikipedia.org/wiki/Namespace
https://docs.djangoproject.com/en/3.0/topics/http/urls/#url-namespaces

# Django template language (DTL)

https://docs.djangoproject.com/en/3.0/ref/templates/language/

# Template Tags / Code

**Substitution**

```
{{ zap }}
{{ zap|safe }}
```

**Calling code**

```
{% url 'cat-detail' cat.id %}
{% author.get_absolute_url %}
```

**Logic**

```
{% if zap > 100 %}
{% endif %}
```

**Blocks**

```
{% block content %}
{% endblock %}
```

https://samples.dj4e.com/tmpl/simple

```python
from django.shortcuts import render

def simple(request):
    return render(request, 'tmpl/simple.html')
```

dj4e-samples/tmpl/templates/tmpl/simple.html

```html
<html>
<head>
    <title>A simple page</title>
</head>
<body>
    <h1>This is pretty simple</h1>
</body>
</html>
```
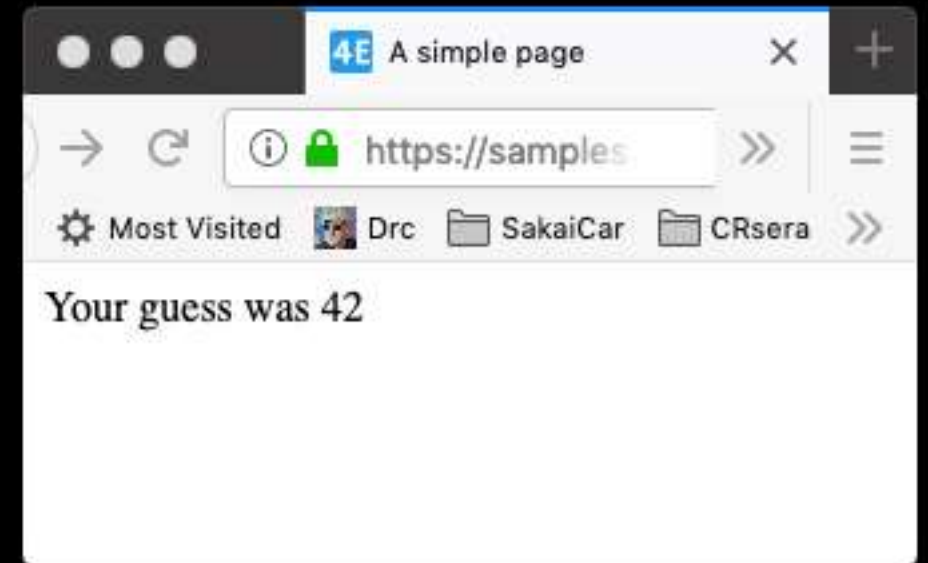
https://samples.dj4e.com/tmpl/guess

```python
def guess(request) :
    context = {'zap' : '42' }
    return render(request, 'tmpl/guess.html', context)
```

dj4e-samples/tmpl/templates/tmpl/guess.html

```html
<html>
<head>
    <title>A simple page</title>
</head>
<body>
    <p>Your guess was {{ zap }}</p>
</body>
</html>
```



4E A simple page

https://samples

Most Visited  Drc  SakaiCar  CRsera

Your guess was 42

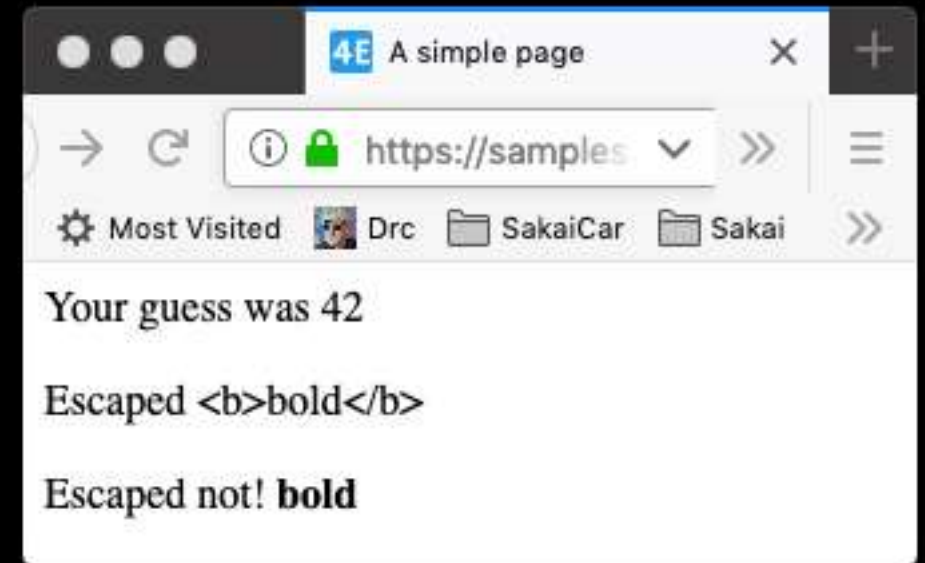https://samples.dj4e.com/tmpl/special

```
def special(request) :
    context = {'txt' : '<b>bold</b>',
               'zap' : '42' }
    return render(request, 'tmpl/special.html', context)
```

dj4e-samples/tmpl/templates/tmpl/special.html

```
<body>
    <p>Your guess was {{ zap }}</p>
    <p>Escaped {{ txt }}</p>
    <p>Escaped not! {{ txt|safe }}</p>
</body>
```



4E A simple page

https://samples

Most Visited   Drc   SakaiCar   Sakai

Your guess was 42

Escaped <b>bold</b>

Escaped not! **bold**
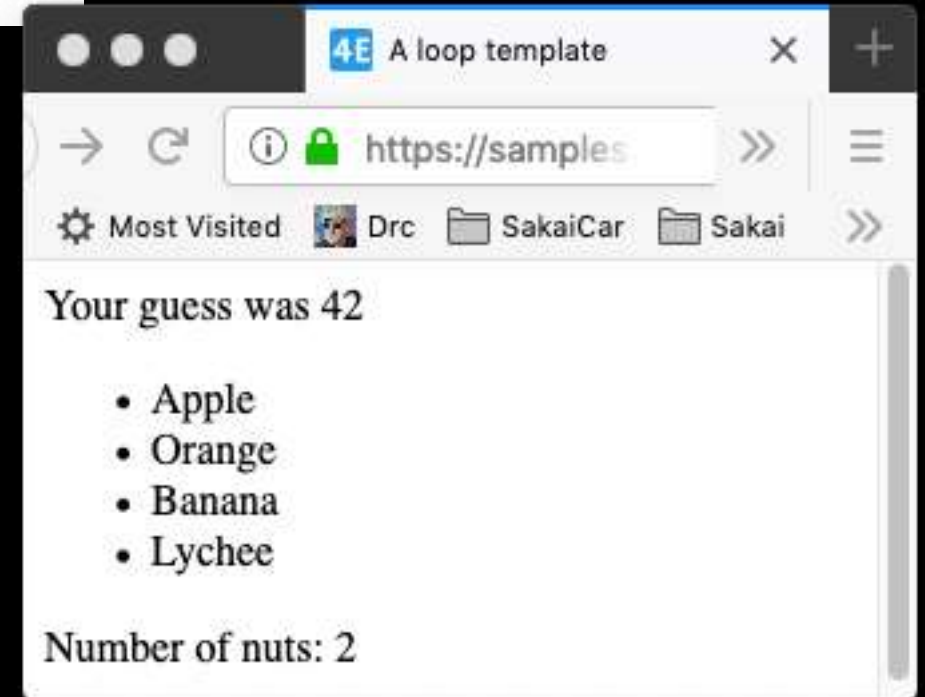
https://samples.dj4e.com/tmpl/loop

```python
def loop(request) :
    f = ['Apple', 'Orange', 'Banana', 'Lychee']
    n = ['peanut', 'cashew']
    x = {'fruits' : f, 'nuts' : n, 'zap' : '42' }
    return render(request, 'tmpl/loop.html', x)
```

dj4e-samples/tmpl/templates/tmpl/loop.html

```html
<ul>
{% for x in fruits  %}
<li>{{ x }}</li>
{% endfor %}
</ul>
{% if nuts %}
    <p>Number of nuts: {{ nuts|length }}</p>
{% else %}
    <p>No nuts.</p>
{% endif %}
```



4E A loop template

https://samples

Most Visited  Drc  SakaiCar  Sakai

Your guess was 42

- Apple
- Orange
- Banana
- Lychee

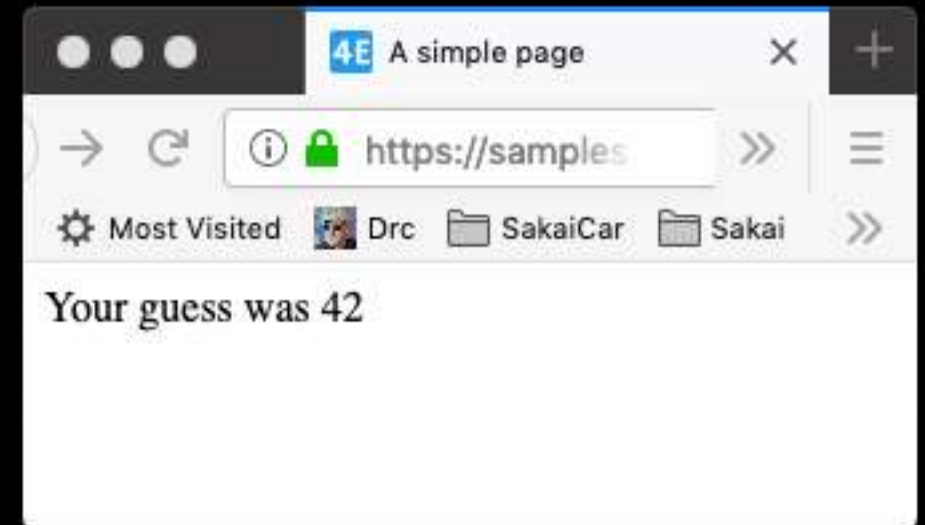Number of nuts: 2

https://samples.dj4e.com/tmpl/nested

```
def nested(request) :
    x = {'outer' : { 'inner' : '42' } }
    return render(request, 'tmpl/nested.html', x)
```

dj4e-samples/tmpl/templates/tmpl/nested.html

```html
<body>
    <p>Your guess was {{ outer.inner }}</p>
</body>
```

Your guess was 42

https://samples.dj4e.com/tmpl/game/200
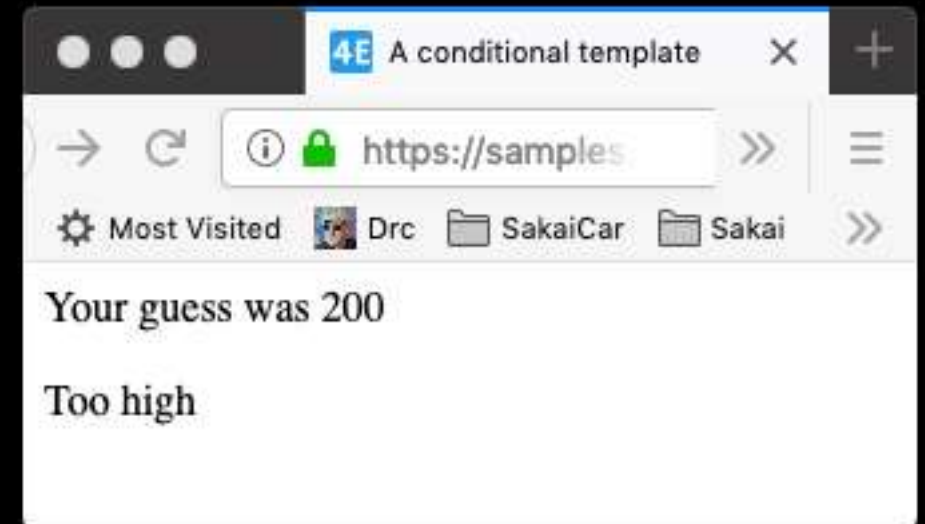
```
path('game/<slug:guess>', views.GameView.as_view())
```

```python
class GameView(View) :
    def get(self, request, guess) :
        x = {'guess' : int(guess) }
        return render(request, 'tmpl/cond.html', x)
```

dj4e-samples/tmpl/templates/tmpl/cond.html

```html
<p>Your guess was {{ guess }}</p>
{% if guess < 42 %}
    <p>Too low</p>
{% elif guess > 42 %}
    <p>Too high</p>
{% else %}
    <p>Just right</p>
{% endif %}
```

4E A conditional template

https://samples

Most Visited   Drc   SakaiCar   Sakai

Your guess was 200

Too high

# Template Inheritance

https://docs.djangoproject.com/en/3.0/ref/urlresolvers/

# Review Python Object-Oriented Programming



https://www.py4e.com/lessons/Objects

# Inheritance



- When we make a new template - we can extend an existing template and then add our own little bit to make our new class

- Another form of store and reuse

- Don't Repeat Yourself (DRY)

https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

# Template Inheritance

**tmpl/templates/tmpl/base.html**

```html
<html>
<head>
    <title>Base template</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
</html>
```

**tmpl/templates/tmpl/cond.html**

```html
<html>
<head>
    <title>A conditional template</title>
</head>
<body>
    <p>Your guess was {{ guess }}</p>
    {% if guess < 42 %}
      <p>Too low</p>
    {% elif guess > 42 %}
      <p>Too high</p>
    {% else %}
      <p>Just right</p>
    {% endif %}
</body>
</html>
```
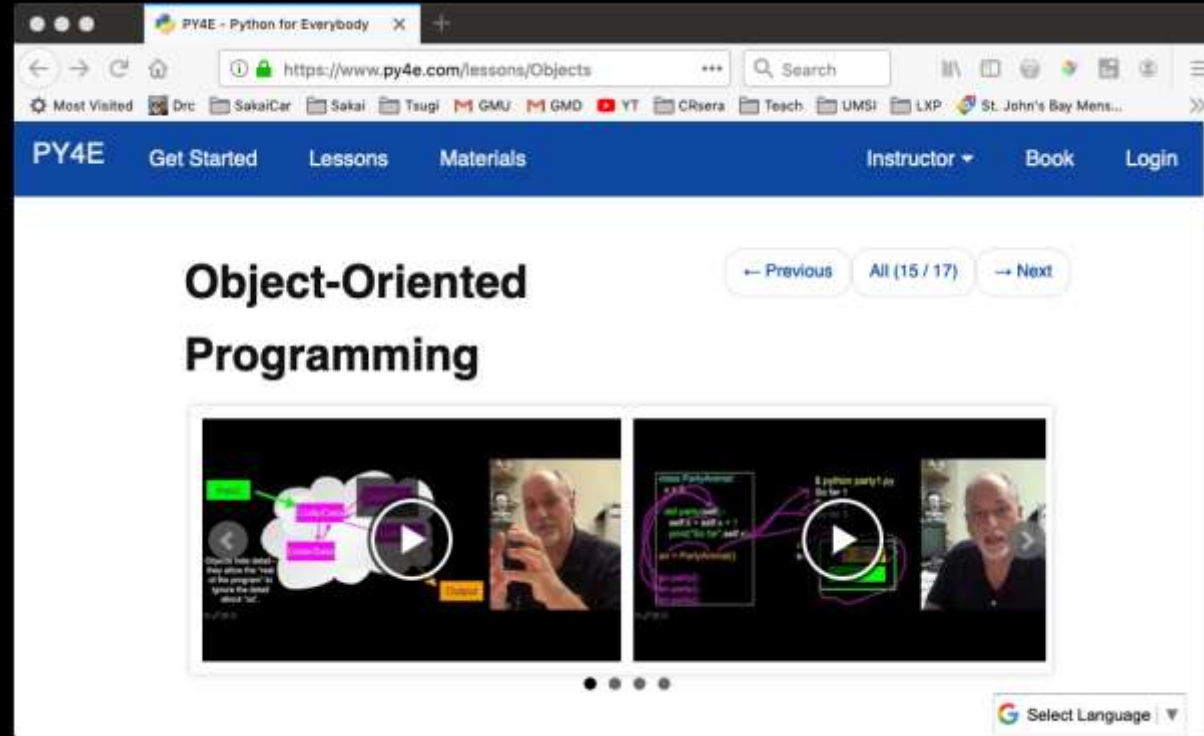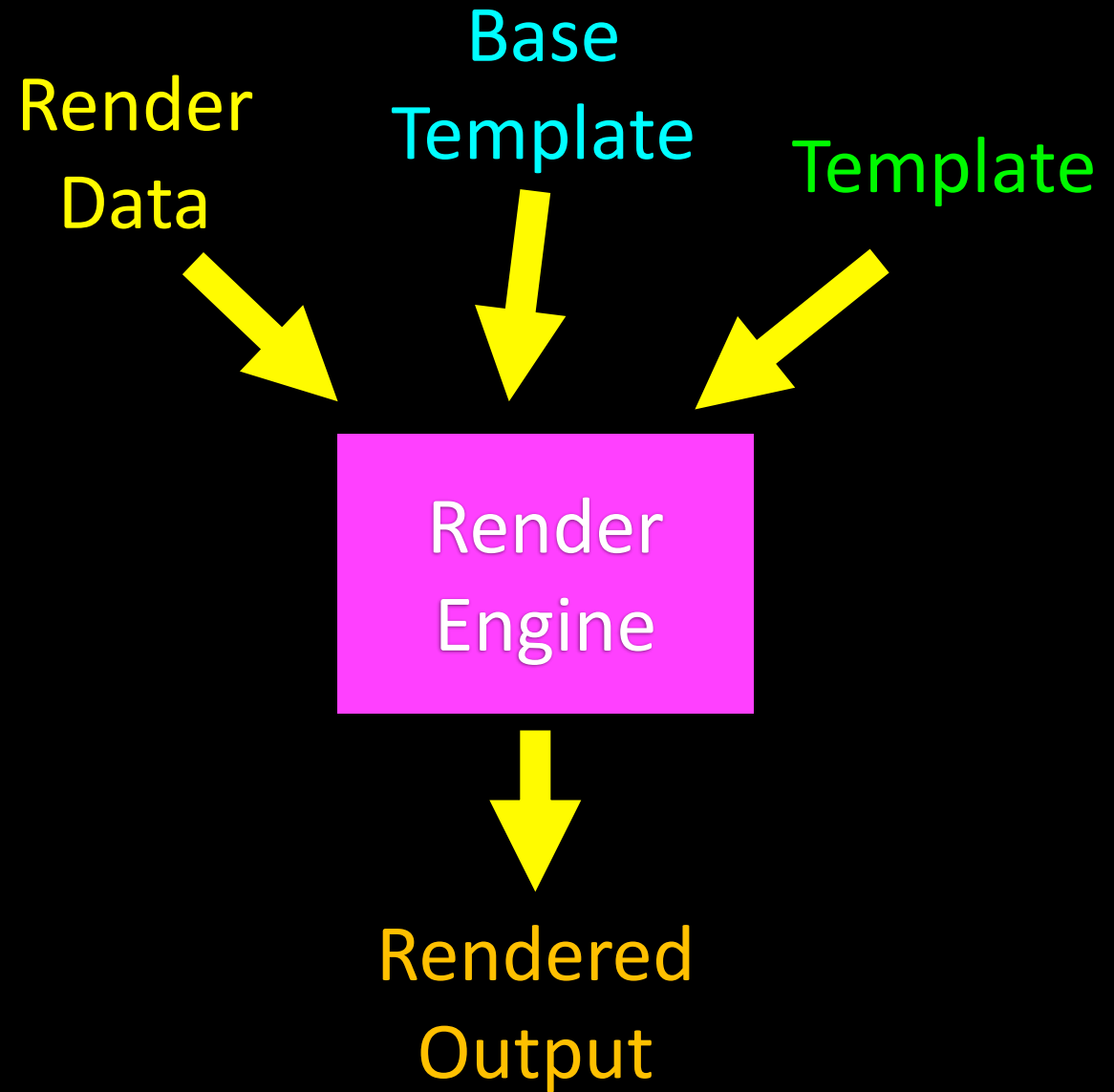
**tmpl/templates/tmpl/cond2.html**

```html
{% extends "tmpl/base.html" %}

{% block content %}
    <p>Your guess was {{ guess }}</p>
    {% if guess < 42 %}
        <p>Too low</p>
    {% elif guess > 42 %}
        <p>Too high</p>
    {% else %}
        <p>Just right</p>
    {% endif %}
{% endblock %}
```

https://samples.dj4e.com/tmpl/game2/200

```python
class GameView2(View) :
    def get(self, request, guess) :
        x = {'guess' : int(guess) }
        return render(request, 'tmpl/cond2.html', x)
```

**tmpl/templates/tmpl/base.html**

```html
<html>
<head>
    <title>A template</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
</html>
```
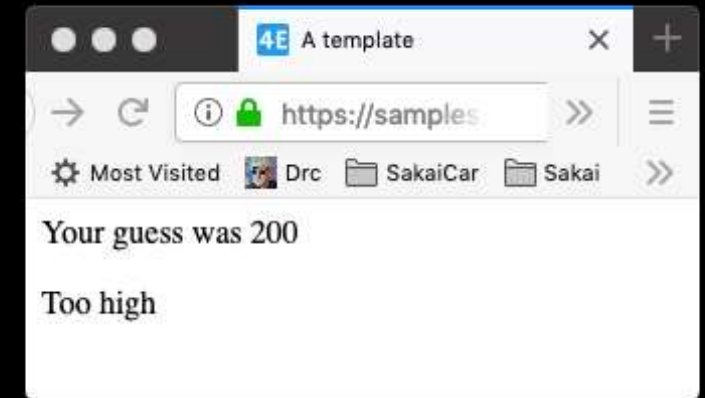
**tmpl/templates/tmpl/cond2.html**

```html
{% extends "tmpl/base.html" %}

{% block content %}
    <p>Your guess was {{ guess }}</p>
    {% if guess < 42 %}
        <p>Too low</p>
    {% elif guess > 42 %}
        <p>Too high</p>
    {% else %}
        <p>Just right</p>
    {% endif %}
{% endblock %}
```

A template

https://samples

Most Visited   Drc   SakaiCar   Sakai

Your guess was 200

Too high

# URL Mapping / Reversing

https://samples.dj4e.com/route/

https://docs.djangoproject.com/en/3.0/topics/http/urls/#reverse-resolution-of-urls
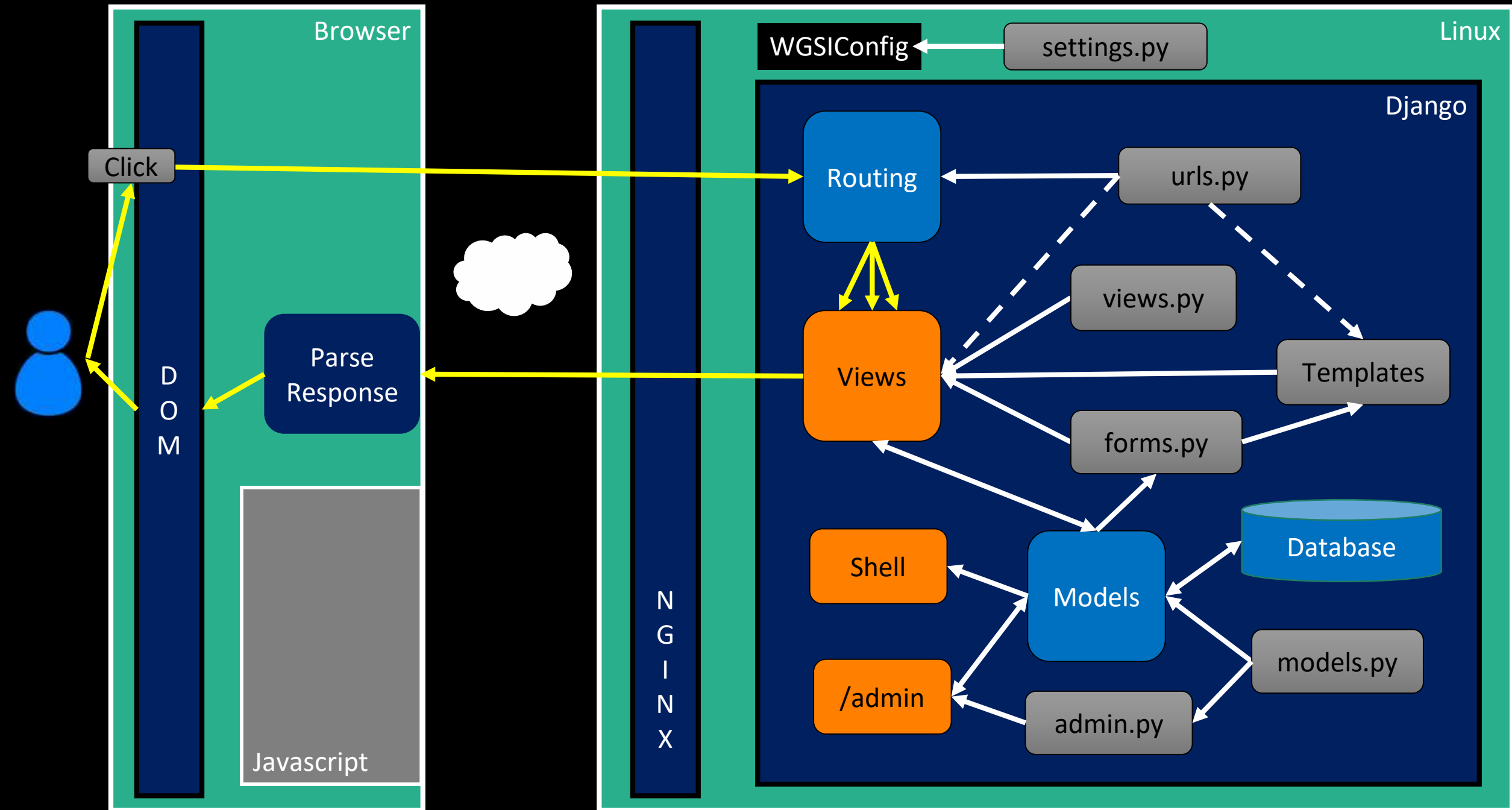
# Reverse Resolution of URLs

A common need when working on a Django project is the possibility to obtain URLs in their final forms either for embedding in generated content or for handling of the navigation flow on the server side (redirections, etc.)

It is strongly desirable to avoid hard-coding these URLs . Equally dangerous is devising ad-hoc mechanisms to generate URLs that are parallel to the design described by the URLconf, which can result in the production of URLs that become stale over time.

In other words, what's needed is a DRY mechanism. Among other advantages it would allow evolution of the URL design without having to go over all the project source code to search and replace outdated URLs.
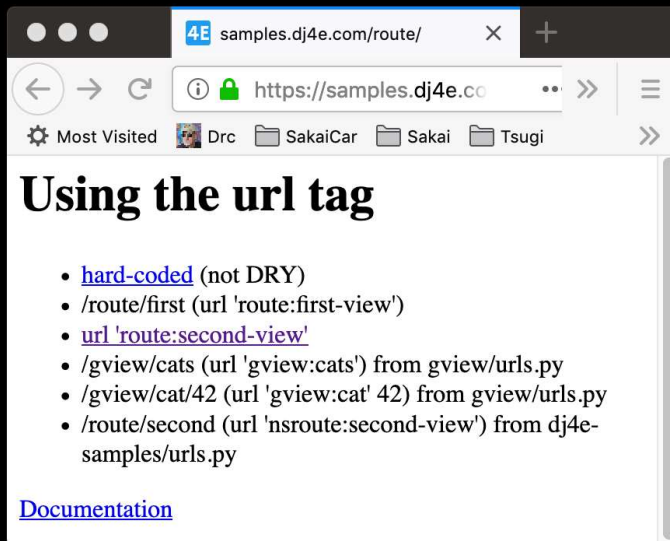
The primary piece of information we have available to get a URL is an identification (e.g. the name) of the view in charge of handling it. Other pieces of information that necessarily must participate in the lookup of the right URL are the types (positional, keyword) and values of the view arguments.

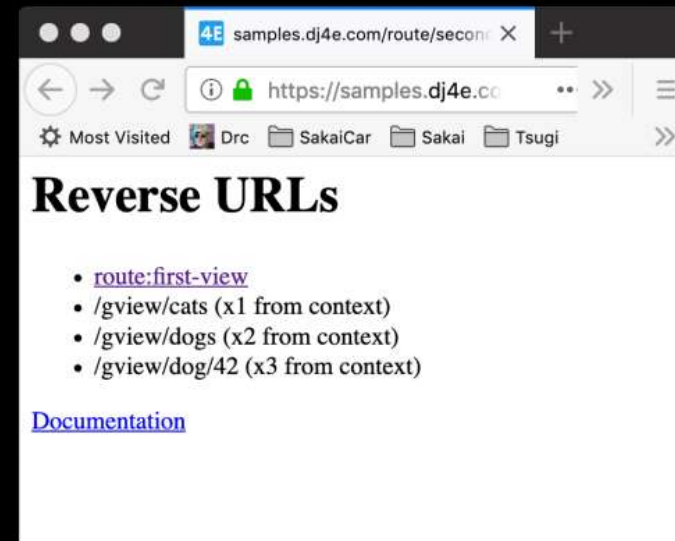https://docs.djangoproject.com/en/3.0/topics/http/urls/#reverse-resolution-of-urls

```
urlpatterns = [
    path('', TemplateView.as_view(template_name='route/main.html')),
    path('first', views.FirstView.as_view(), name='first-view'),
    path('second', views.SecondView.as_view(), name='second-view'),
]
```

https://samples.dj4e.com/route/                    https://samples.dj4e.com/route/second



**Using the url tag**

- hard-coded (not DRY)
- /route/first (url 'route:first-view')
- url 'route:second-view'
- /gview/cats (url 'gview:cats') from gview/urls.py
- /gview/cat/42 (url 'gview:cat' 42) from gview/urls.py
- /route/second (url 'nsroute:second-view') from dj4e-samples/urls.py

Documentation



**Reverse URLs**

- route:first-view
- /gview/cats (x1 from context)
- /gview/dogs (x2 from context)
- /gview/dog/42 (x3 from context)

Documentation

```
app_name = 'route'
urlpatterns = [
    path('', TemplateView.as_view(template_name='route/main.html')),
    path('first', views.FirstView.as_view(), name='first-view'),
    path('second', views.SecondView.as_view(), name='second-view'),
]
```

dj4e-samples/route/templates/route/main.html

```
<li>
    <a href="/route/second">
    hard-coded</a> (not DRY)
</li>
<li>
    {% url 'route:first-view' %}
    (url 'route:first-view')
</li>
<li>
    <a href="{% url 'route:second-view' %}">
    url 'route:second-view'</a>
</li>
```

route:first-view

application
name

view
name

Using the url
tag in a
template

https://samples.dj4e.com/route/

dj4e-samples/route/templates/route/main.html

```
<li>
    <a href="/route/second-view">
    hard-coded</a> (not DRY)
</li>
<li>
    {% url 'route:first-view' %}
    (url 'route:first-view')
</li>
<li>
    <a href="{% url 'route:second-view' %}">
    url 'route:second-view'</a>
</li>
```
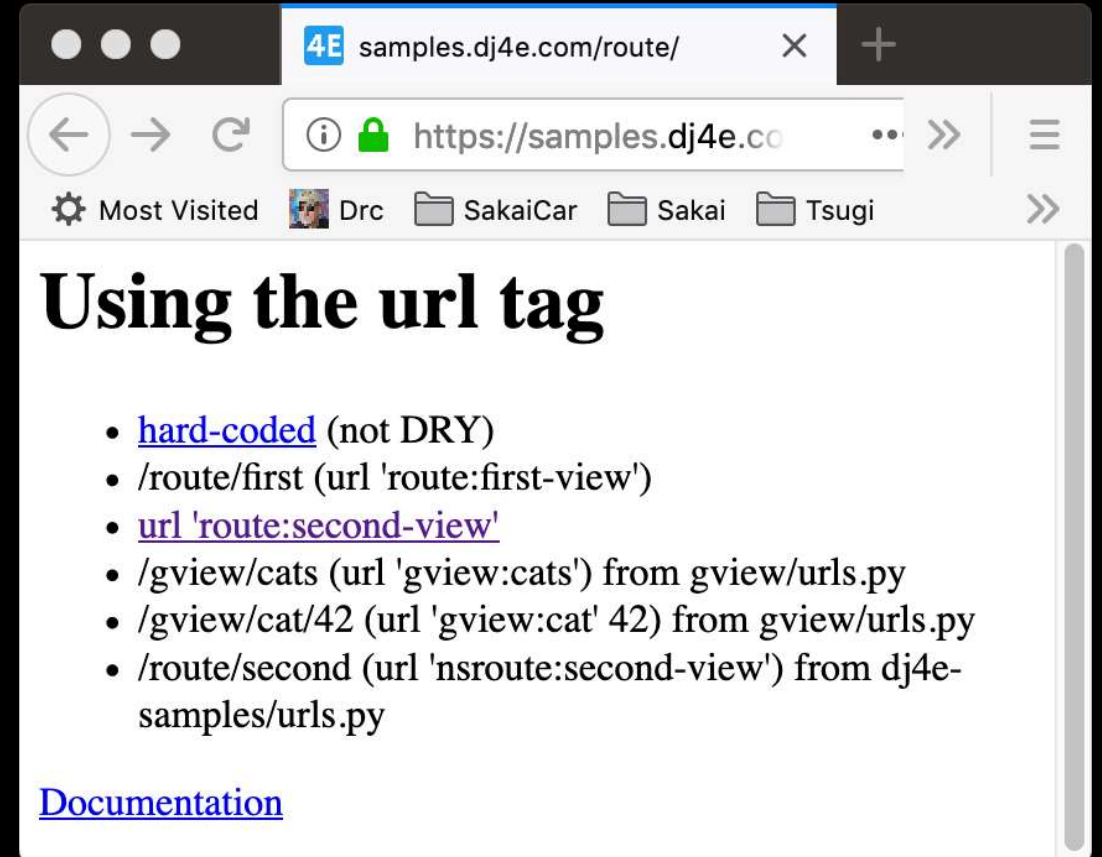
# Using the url tag

- hard-coded (not DRY)
- /route/first (url 'route:first-view')
- url 'route:second-view'
- /gview/cats (url 'gview:cats') from gview/urls.py
- /gview/cat/42 (url 'gview:cat' 42) from gview/urls.py
- /route/second (url 'nsroute:second-view') from dj4e-samples/urls.py

Documentation

dj4e-samples/gview/urls.py

```python
app_name = 'gview'
urlpatterns = [
    path('cats', views.CatListView.as_view(), name='cats'),
    path('cat/<int:pk_from_url>', views.CatDetailView.as_view(), name='cat'),
]
```

dj4e-samples/route/templates/route/main.html

```html
<li>
    {% url 'gview:cats' %}
    (url 'gview:cats') from gview/urls.py
</li>
<li>
    {% url 'gview:cat' 42 %}
    (url 'gview:cat' 42) from gview/urls.py
</li>
```
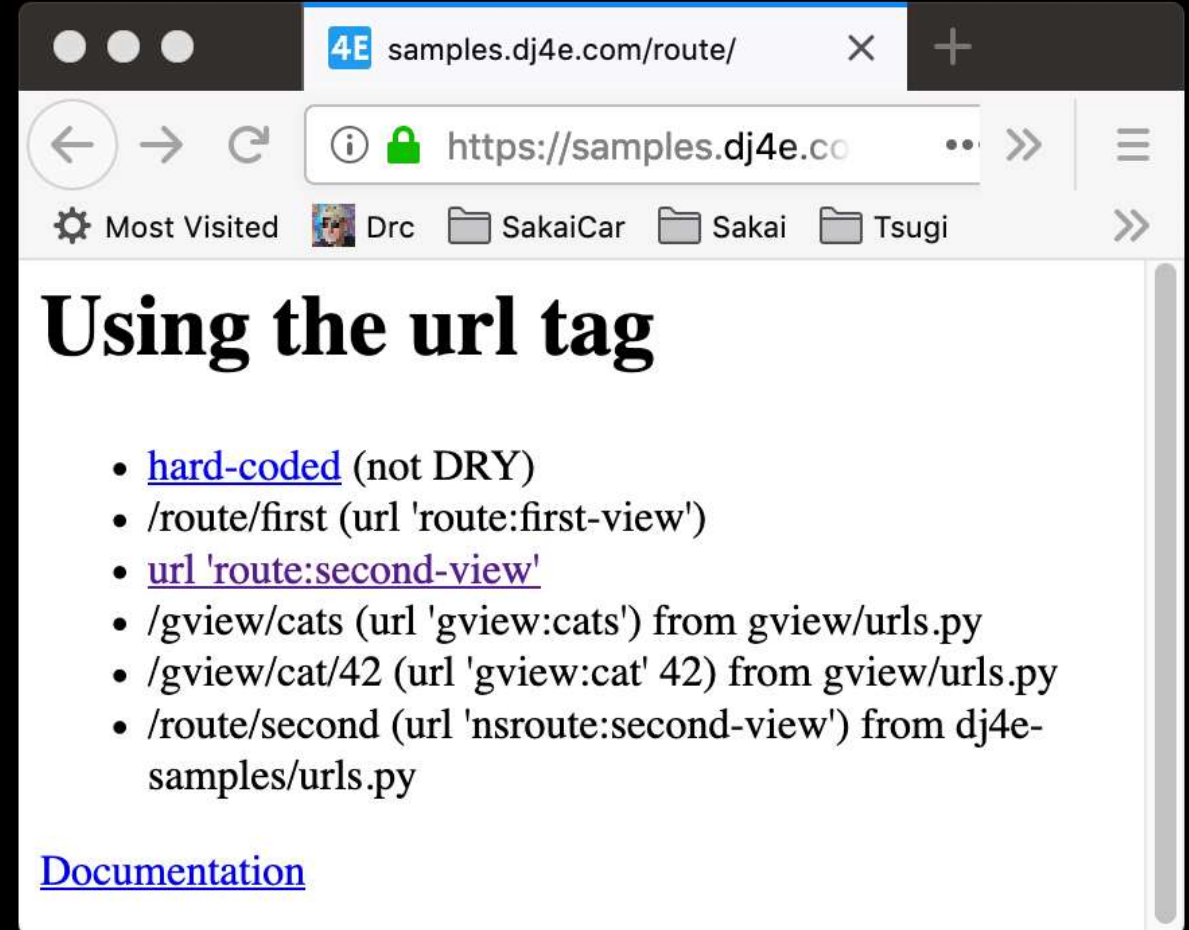
Parameter

'gview:cat' 42

application     view
name            name

Other applications and parameters

dj4e-samples/route/templates/route/main.html

```
<li>
    {% url 'gview:cats' %}
  (url 'gview:cats')
</li>
<li>
    {% url 'gview:cat' 42 %}
  (url 'gview:cat' 42)
</li>
```



# Using the url tag

- hard-coded (not DRY)
- /route/first (url 'route:first-view')
- url 'route:second-view'
- /gview/cats (url 'gview:cats') from gview/urls.py
- /gview/cat/42 (url 'gview:cat' 42) from gview/urls.py
- /route/second (url 'nsroute:second-view') from dj4e-samples/urls.py

Documentation

dj4e-samples/dj4e-samples/urls.py

```python
urlpatterns = [
    path('', include('home.urls')),
    path('admin/', admin.site.urls),  # Keep
    url(r'^oauth/', include('social_django.urls', namespace='social')),
    path('hello/', include('hello.urls')),
    path('route/', include('route.urls', namespace='nsroute')),
]
```

dj4e-samples/route/templates/route/main.html

```html
<li>
   <a href="{% url 'route:second-view' %}">
    url 'route:second-view'</a>
</li>
    ...
<li>
   {% url 'nsroute:second-view' %}
   (url 'nsroute:second-view')
</li>
```

A "second"
name space

```
path('second', views.SecondView.as_view(), name='second-view'),
```

dj4e-samples/route/urls.py

dj4e-samples/route/views.py

```python
from django.shortcuts import render
from django.urls import reverse
from django.views import View

class SecondView(View):
  def get(self, request) :
    u = reverse('gview:cats')
    u2 = reverse('gview:dogs')
    u3 = reverse('gview:dog', args=['42'] )
    ctx = {'x1' : u, 'x2': u2, 'x3': u3 }
    return render(request, 'route/second.html', ctx)
```

dj4e-samples/route/templates/route/main.html

```html
<li>
    <a href="{% url 'route:first-view' %}">
    route:first-view</a>
</li>
<li>
    {{ x1 }} (x1 from context)
</li>
<li>
    {{ x2 }} (x2 from context)
</li>
<li>
    {{ x3 }} (x3 from context)
</li>
```

dj4e-samples/route/templates/route/main.html

```
class SecondView(View):
  def get(self, request) :
    u = reverse('gview:cats')
    u2 = reverse ('gview:dogs')
    u3 = reverse('gview:dog', args=['42'] )
    ctx = {'x1' : u, 'x2': u2, 'x3': u3 }
    return render(request, 'route/second.html', ctx)
```

dj4e-samples/route/templates/route/main.html

```
<li>
    {{ x1 }} (x1 from context)
</li>
<li>
    {{ x2 }} (x2 from context)
</li>
<li>
    {{ x3 }} (x3 from context)
</li>
```

https://samples.dj4e.com/route/second



# Reverse URLs

- route:first-view
- /gview/cats (x1 from context)
- /gview/dogs (x2 from context)
- /gview/dog/42 (x3 from context)

Documentation

# Summary

- Views are where we bring the application components together to handle requests from browsers and produce responses for the browsers

- Templates take a context and merge it into a template to produce HTML
  - Values can be substituted without without "escaping"
  - Coding in templates

# Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here

# Additional Source Information

- Snowman Cookie Cutter" by Didriks is licensed under CC BY https://www.flickr.com/photos/dinnerseries/23570475099

- Portions of the text of these slides is adapted from the text www.djangoproject.org web site. Those slides which use text from that site have a reference to the original text on that site. Django is licensed under the three-clause BSD license.