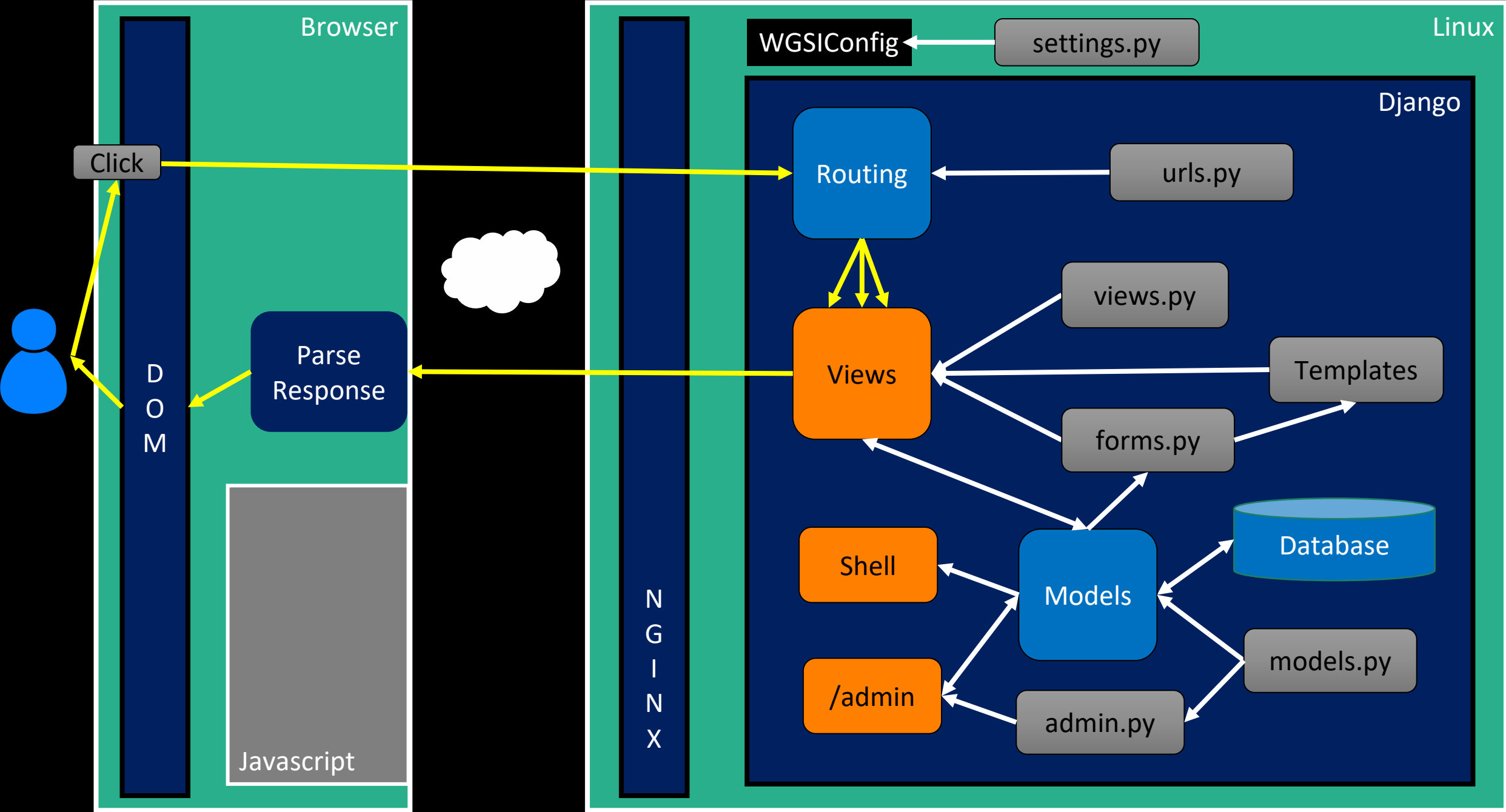


Charles Severance  
[www.dj4e.com](http://www.dj4e.com)

# Django Generic Views





## `dj4e-samples/gview/urls.py`

```
from django.urls import path
from . import views
from django.views.generic import TemplateView
app_name = 'gview'

# Note use of plural for list view and singular for detail view
urlpatterns = [
    path('', TemplateView.as_view(template_name='gview/main.html')),
    path('cats', views.CatListView.as_view(), name='cats'),
    path('cat/<int:pk_from_url>', views.CatDetailView.as_view(), name='cat'),
    path('dogs', views.DogListView.as_view(), name='dogs'),
    path('dog/<int:pk>', views.DogDetailView.as_view(), name='dog'),
    path('horses', views.HorseListView.as_view(), name='horses'),
    path('horse/<int:pk>', views.HorseDetailView.as_view(), name='horse'),
    path('cars', views.CarListView.as_view(), name='cars'),
    path('car/<int:pk>', views.CarDetailView.as_view(), name='car'),
    path('wacky', views.WackyEquinesView.as_view(), name='whatever'),
]
```

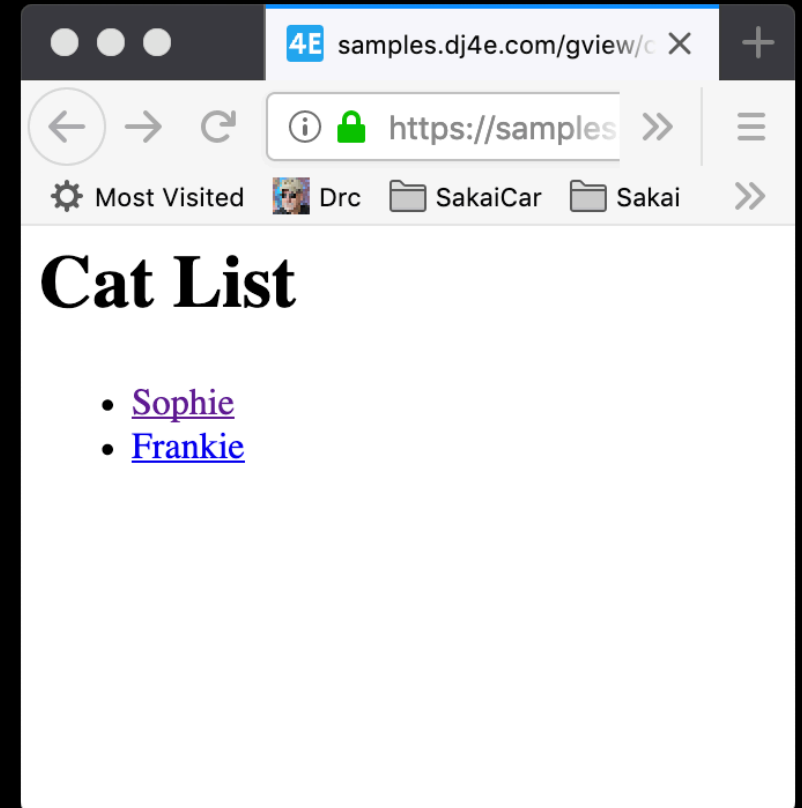
[dj4e-samples/gview/views.py](https://samples.dj4e.com/gview/views.py)

```
class CatListView(View):
    def get(self, request) :
        stuff = Cat.objects.all()
        cntx = { 'cat_list': stuff }
        return render(request, 'gview/cat_list.html', cntx)
```

[dj4e-samples/gview/templates/gview/cat\\_list.html](https://samples.dj4e.com/gview/templates/gview/cat_list.html)

```
<h1>Cat List</h1>
<p>
{% if cat_list %}
<ul>
    {% for cat in cat_list %}
        <li>
            <a href="{% url 'gview:cat' cat.id %}">{{ cat.name }}</a>
        </li>
    {% endfor %}
</ul>
{% else %}
    <p>There are no cats in the database.</p>
{% endif %}
</p>
```

<https://samples.dj4e.com/gview/cats>



[dj4e-samples/gview/templates/gview/cat\\_detail.html](https://samples.dj4e.com/gview/templates/gview/cat_detail.html)

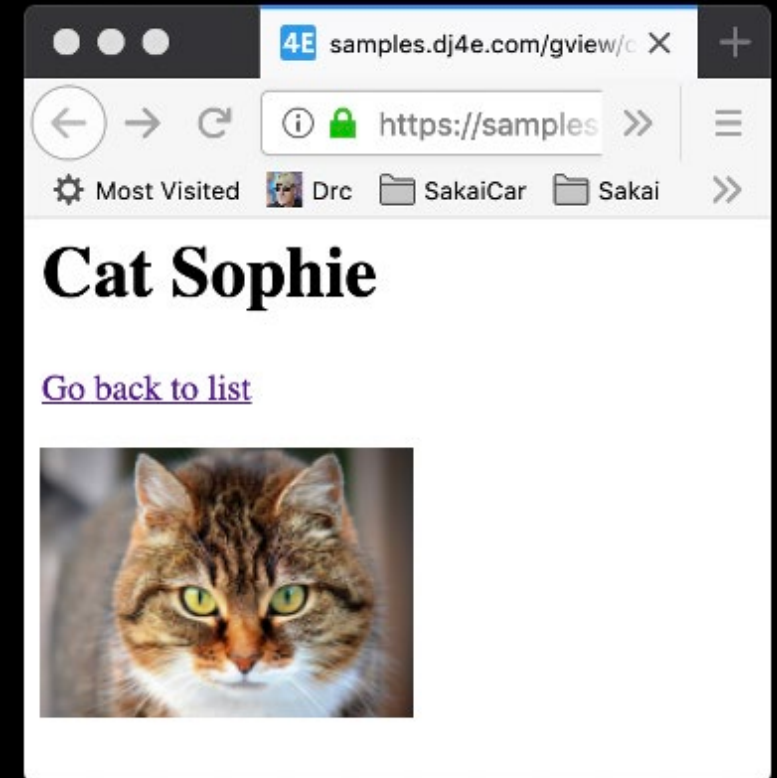
```
<h1>Cat {{ cat.name }}</h1>
<p>
<a href="{% url 'gview:cats' %}">Go back to list</a>
</p>
<p>

</p>
```

[dj4e-samples/gview/views.py](https://samples.dj4e.com/gview/views.py)

```
class CatDetailView(View):
    def get(self, request, pk_from_url) :
        obj = Cat.objects.get(pk=pk_from_url)
        cntx = { 'cat': obj }
        return render(request, 'gview/cat_detail.html', cntx)
```

<https://samples.dj4e.com/gview/cat/1>



# Concept: Don't Repeat Yourself (DRY)

Don't repeat yourself (DRY, or sometimes do not repeat yourself) is a principle of software development aimed at reducing repetition of software patterns] replacing it with abstractions or using data normalization to avoid redundancy. The principle has been formulated by Andy Hunt and Dave Thomas in their book The Pragmatic Programmer.

When the DRY principle is applied successfully, a modification of any single element of a system does not require a change in other logically unrelated elements. Additionally, elements that are logically related all change predictably and uniformly, and are thus kept in sync.

[https://en.wikipedia.org/wiki/Don%27t\\_repeat\\_yourself](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)

# Built-in class-based generic views

Writing Web applications can be monotonous, because we repeat certain patterns again and again. Django's *generic views* were developed to ease that pain. They take certain common idioms and patterns found in view development and abstract them so that you can quickly write common views of data without having to write too much repetitive code.

We can recognize certain common tasks, like displaying a list of model objects, and write code that displays a list of *any* model object. Django ships with generic views to display list and detail pages for a single model object.

<https://docs.djangoproject.com/en/3.0/topics/class-based-views/generic-display/>

# Convention over Configuration

Convention over configuration is a software design paradigm used by software frameworks that attempts to decrease the number of decisions that a developer using the framework is required to make without necessarily losing flexibility.

When the convention matches the desired behavior, it behaves as expected without having to write configuration files. Only when the desired behavior deviates from the implemented convention is explicit configuration required.

[https://en.wikipedia.org/wiki/Convention\\_over\\_configuration](https://en.wikipedia.org/wiki/Convention_over_configuration)



# Convention Over Configuration

- If
  - If the app\_name is `gview`
  - And the view extends `django.views.generic.list.ListView`
  - And the view uses the model `Horse`
- Then
  - The will automatically render a view named `gview/horse_list.html`
  - Passing a `list` of Horse objects in the variable `horse_list` into the template

<https://docs.djangoproject.com/en/3.0/ref/class-based-views/generic-display/#django.views.generic.list.ListView>

[dj4e-samples/gview/views.py](https://samples.dj4e.com/gview/views.py)

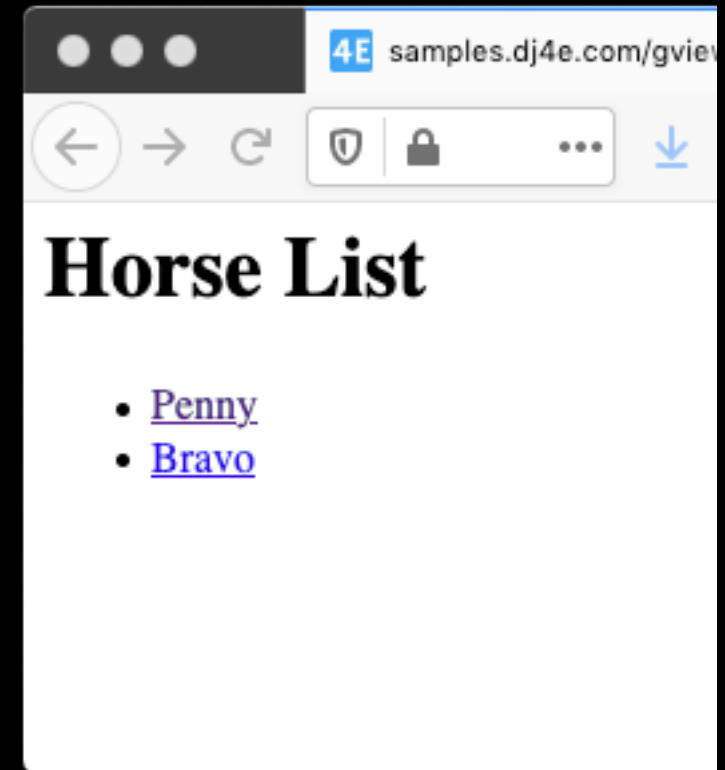
```
from django.views import generic

class HorseListView(generic.ListView):
    model = Horse
```

[dj4e-samples/gview/templates/gview/horse\\_list.html](https://samples.dj4e.com/gview/templates/gview/horse_list.html)

```
<h1>Horse List</h1>
<p>
{% if horse_list %}
<ul>
    {% for horse in horse_list %}
        <li>
            <a href="{% url 'gview:horse' horse.id %}">{{ horse.name }}</a>
        </li>
    {% endfor %}
</ul>
{% else %}
    <p>There are no horses in the database.</p>
{% endif %}
</p>
```

<https://samples.dj4e.com/gview/horses>



## [dj4e-samples/gview/views.py](#)

```
from django.views import generic
from gview.models import Cat, Dog, Horse, Car

class HorseDetailView(generic.DetailView):
    model = Horse
```

## [dj4e-samples/gview/templates/gview/horse\\_detail.html](#)

```
<h1>Horse {{ horse.name }}</h1>
<p>

</p>
<p>
<a href="{% url 'gview:horses' %}">Go back to list</a>
</p>
```

`gview.views.HorseDetailView`

`model = gviews.models.Horse`

`django.views.generic.DetailView`

<https://docs.djangoproject.com/en/3.0/topics/class-based-views/generic-display/>

[dj4e-samples/gview/views.py](https://samples.dj4e.com/gview/views.py)

```
from django.views import generic
from gview.models import Cat, Dog, Horse, Car

class HorseDetailView(generic.DetailView):
    model = Horse
```

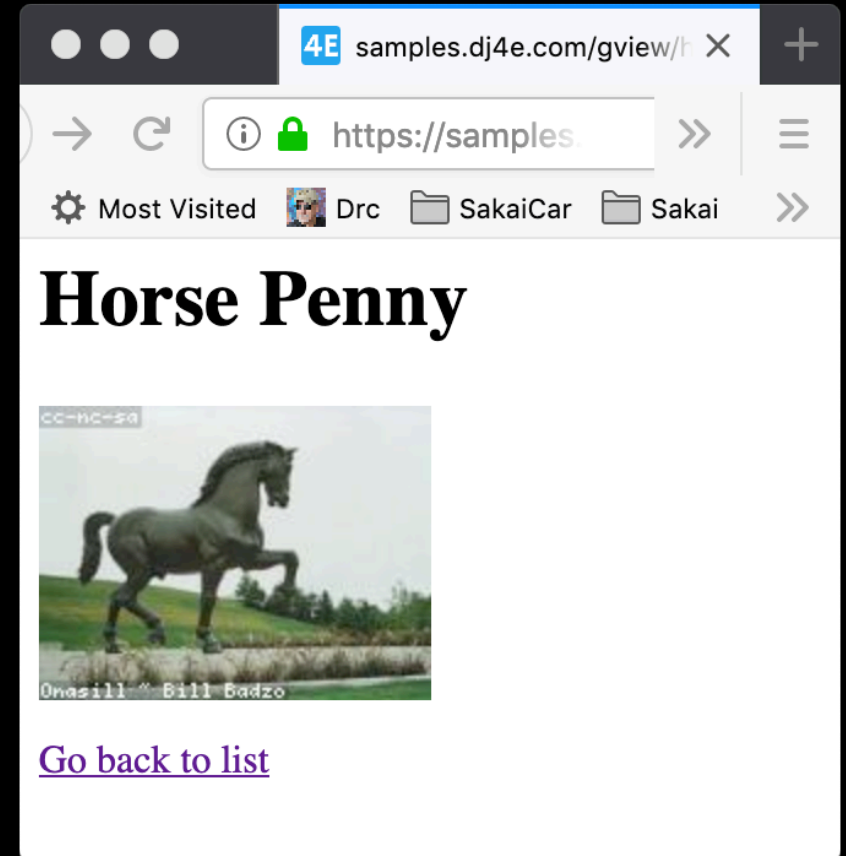
[dj4e-samples/gview/templates/gview/horse\\_detail.html](https://samples.dj4e.com/gview/templates/gview/horse_detail.html)

```
<h1>Horse {{ horse.name }}</h1>
<p>

</p>
<p>
<a href="{% url 'gview:horses' %}">Go back to list</a>
</p>
```

Lots of convention – no repetition

<https://samples.dj4e.com/gview/horse/1>



[dj4e-samples/gview/views.py](#)

```
class CatListView(View):
    def get(self, request) :
        stuff = Cat.objects.all()
        cntx = { 'cat_list': stuff }
        return render(request, 'gview/cat_list.html', cntx)

from django.views import generic

class HorseListView(generic.ListView):
    model = Horse
```

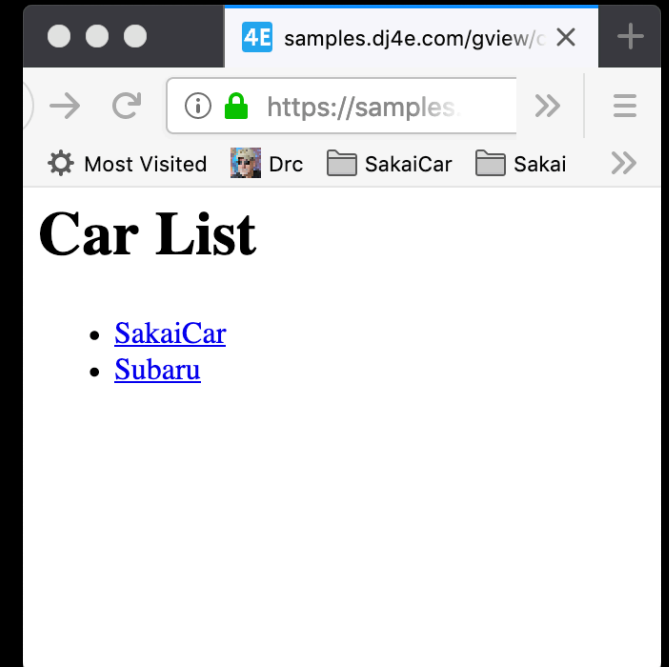
<https://docs.djangoproject.com/en/3.0/topics/class-based-views/generic-display/>

[dj4e-samples/gview/views.py](https://samples.django.com/gview/views.py)

```
# Lets review how inheritance works to avoid repeating ourselves
# It is all about convention
class DJ4EListView(View):
    def get(self, request) :
        modelname = self.model._meta.verbose_name.title().lower()
        stuff = self.model.objects.all()
        cntx = { modelname+'_list': stuff }
        return render(request, 'gview/'+modelname+'_list.html', cntx)

# Lets reuse those "generic" classes
class CarListView(DJ4EListView):
    model = Car
```

<https://samples.django.com/gview/cars>



<https://docs.djangoproject.com/en/3.0/topics/class-based-views/generic-display/>

# Overriding Convention

# Convention over Configuration

Convention over configuration is a software design paradigm used by software frameworks that attempts to decrease the number of decisions that a developer using the framework is required to make without necessarily losing flexibility.

When the convention matches the desired behavior, it behaves as expected without having to write configuration files. Only when the desired behavior deviates from the implemented convention is explicit configuration required.

[https://en.wikipedia.org/wiki/Convention\\_over\\_configuration](https://en.wikipedia.org/wiki/Convention_over_configuration)




# Departing from Convention in a View

- You can add instance variables to the `as_view()` in the `urls.py`
- You can add instance variables to the class in `views.py`
- You can override methods in the class in `views.py`

`dj4e-samples/gview/urls.py`

```
app_name = 'gview'
urlpatterns = [
    path('', TemplateView.as_view(template_name='gview/main.html')),
    path('cats', views.CatListView.as_view(), name='cats'),
    ...
]
```



## **class django.views.generic.list.ListView**

A page representing a list of objects. While this view is executing, `self.object_list` will contain the list of objects (usually, but not necessarily a queryset) that the view is operating upon.

### **Method Flowchart**

1. `setup()`
2. `dispatch()`
3. `http_method_not_allowed()`
4. `get_template_names()`
5. `get_queryset()`
6. `get_context_object_name()`
7. `get_context_data()`
8. `get()`
9. `render_to_response()`

<https://docs.djangoproject.com/en/3.0/ref/class-based-views/generic-display/#django.views.generic.list.ListView>

[dj4e-samples/gview/views.py](#)

```
# Lets explore how (badly) we can override things...
class WackyEquinesView(generic.ListView):
    model = Car
    template_name = 'gview/wacky.html'

    def get_queryset(self, **kwargs):
        crazy = Horse.objects.all()      # Convention: Car
        return crazy

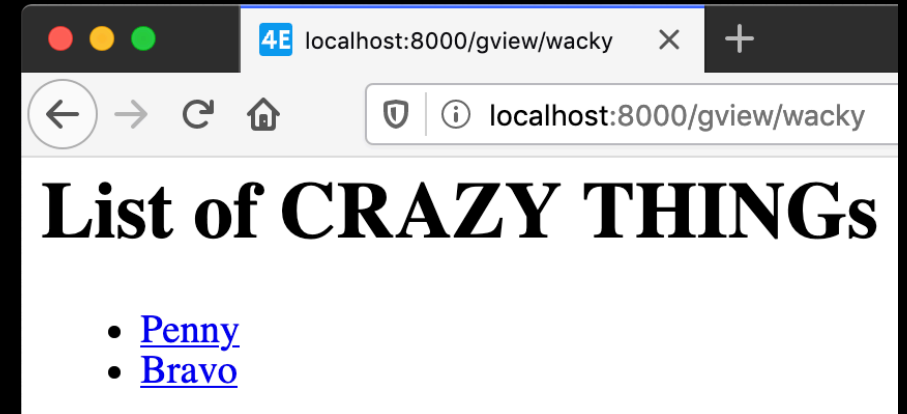
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['crazy_thing'] = 'CRAZY THING'
        return context
```

[dj4e-samples/gview/templates/gview/wacky.html](#)

```
<h1>List of {{ crazy_thing }}s</h1>
<p>
{% if horse_list %}
<ul>
    {% for xyz in horse_list %}
        <li>
            <a href="{% url 'gview:horse' xyz.id %}">{{ xyz.name }}</a>
        </li>
    {% endfor %}
...

```

<https://samples.dj4e.com/gview/wacky>



# Summary

- Generic views allow us to produce lots of similar pages without cutting, pasting and editing boiler plate
- Quicker development
- Consistent User Experience
- Less lines of code means fewer mistakes

# Acknowledgements / Contributions

These slides are Copyright 2019- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) as part of [www.dj4e.com](http://www.dj4e.com) and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

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