Django shortcut functions ---

The package django.shortcuts collects helper functions and classes that “span” multiple levels of MVC. In

other words, these functions/classes introduce controlled coupling for convenience’s sake.

**1:render()**

render(request, template\_name, context=None, content\_type=None, status=None, using=None)

Combines a given template with a given context dictionary and returns an HttpResponse object with

that rendered text. Django does not provide a shortcut function which returns a TemplateResponse because the constructor of TemplateResponse offers the same level of convenience as render().

**Required arguments**

**request**

The request object is used to generate this response.

**template\_name**

The full name of a template to use or sequence of template names. If a sequence is given, the first

template that exists will be used.

**Optional arguments**

**context**

A dictionary of values to add to the template context. By default, this is an empty dictionary. If a

value in the dictionary is callable; the view will call it just before rendering the template.

**content\_type**

The MIME type to use for the resulting document. Defaults to 'text/html'.

**status**

The status code for the response. Defaults to 200.

**using**

The NAME of a template engine to use for loading the template.

## Example

The following example renders the template myapp/index.html with the MIME type application/

xhtml+xml:

from django.shortcuts import render

def my\_view(request):

# View code here...

return render(request,"myapp/index.html",{"foo": "bar",},content\_type = "application/xhtml+xml",)

This example is equivalent to:

from django.http import HttpResponse

from django.template import loader

def my\_view(request):

# View code here...

t = loader.get\_template("myapp/index.html")

c = {"foo": "bar"}

return HttpResponse(t.render(c, request), content\_type = "application /xhtml+xml")

----------------------------------------------------------------------------------------------------------------------------------------------

**2:redirect()**

redirect(to, \*args, permanent=False, \*\*kwargs)

Returns an HttpResponseRedirect to the appropriate URL for the arguments passed.

The arguments could be:

• A model: the model’s get\_absolute\_url() function will be called.

• A view name, possibly with arguments: reverse() will be used to reverse-resolve the name.

• An absolute or relative URL, which will be used as-is for the redirect location.

By default, issues a temporary redirect; pass permanent=True to issue a permanent redirect.

Examples

You can use the redirect() function in a number of ways.

**1.** By passing some object; that object’s get\_absolute\_url() method will be called to figure out the redirect URL:

from django.shortcuts import redirect

def my\_view(request):

...

obj = MyModel.objects.get(...)

return redirect(obj)

**2.** By passing the name of a view and optionally some positional or keyword arguments; the URL will be

reverse resolved using the reverse() method:

def my\_view(request):

...

return redirect("some-view-name", foo="bar")

**3.** By passing a hardcoded URL to redirect to:

def my\_view(request):

...

return redirect("/some/url/")

This also works with full URLs:

def my\_view(request):

...

return redirect("https://example.com/")

By default, redirect() returns a temporary redirect. All of the above forms accept a permanent argument;

if set to True a permanent redirect will be returned:

def my\_view(request):

...

obj = MyModel.objects.get(...)

return redirect(obj, permanent=True)

----------------------------------------------------------------------------------------------------------------------------------------------

**3:get\_object\_or\_404()**

**get\_object\_or\_404(klass, \*args, \*\*kwargs)**

**aget\_object\_or\_404(klass, \*args, \*\*kwargs)**

Asynchronous version: **aget\_object\_or\_404()**

Calls get() on a given model manager, but it raises Http404 instead of the model’s DoesNotExist exception.

Arguments

klass

A Model class, a Manager, or a QuerySet instance from which to get the object.

\*args

Q objects.

\*\*kwargs

Lookup parameters, which should be in the format accepted by get() and filter().

Example

The following example gets the object with the primary key of 1 from MyModel:

from django.shortcuts import get\_object\_or\_404

def my\_view(request):

obj = get\_object\_or\_404(MyModel, pk=1)

This example is equivalent to:

from django.http import Http404

def my\_view(request):

try:

obj = MyModel.objects.get(pk=1)

except MyModel.DoesNotExist:

raise Http404("No MyModel matches the given query.")

The most common use case is to pass a Model, as shown above. However, you can also pass a QuerySet

instance:

queryset = Book.objects.filter(title\_\_startswith="M")

get\_object\_or\_404(queryset, pk=1)

The above example is a bit contrived since it’s equivalent to doing:

get\_object\_or\_404(Book, title\_\_startswith="M", pk=1)

but it can be useful if you are passed the queryset variable from somewhere else. Finally, you can also use a **Manager**. This is useful for example if you have a **custom manager**:

get\_object\_or\_404(Book.dahl\_objects, title="Matilda")

You can also use **related managers**:

author = Author.objects.get(name="Roald Dahl")

get\_object\_or\_404(author.book\_set, title="Matilda")

Note: As with get(), a MultipleObjectsReturned exception will be raised if more than one object is found.

----------------------------------------------------------------------------------------------------------------------------------------------

**4:get\_list\_or\_404()**

**get\_list\_or\_404(klass, \*args, \*\*kwargs)**

**aget\_list\_or\_404(klass, \*args, \*\*kwargs)**

Asynchronous version: **aget\_list\_or\_404()**

Returns the result of filter() on a given model manager cast to a list, raising Http404 if the resulting

list is empty.

Arguments

klass

A Model, Manager or QuerySet instance from which to get the list.

\*args

Q objects.

\*\*kwargs

Lookup parameters, which should be in the format accepted by get() and filter().

# Example

The following example gets all published objects from MyModel:

from django.shortcuts import get\_list\_or\_404

def my\_view(request):

my\_objects = get\_list\_or\_404(MyModel, published=True)

This example is equivalent to:

from django.http import Http404

def my\_view(request):

my\_objects = list(MyModel.objects.filter(published=True))

if not my\_objects:

raise Http404("No MyModel matches the given query.")----------------------------------- END ----------------------------------------