1. Start the Django shell from the **c:/learning\_log** directory:

bashCopy code

python manage.py shell

1. Inside the Django shell, change the current working directory to **c:/learning\_log/learning\_logs**:

pythonCopy code

import os os.chdir('learning\_logs'

Now you should be in the **c:/learning\_log/learning\_logs** directory within the Django shell. Keep in mind that changing the working directory like this is specific to the current shell session and won't affect the outer environment.

(ll\_env) c:\learning\_log>python manage.py shell

Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> **import os**

>>> **os.chdir('learning\_logs')**

>>>Top of Form

Remember, the decision to use class-based views or function-based views is up to you based on your project's requirements. If you want to use class-based views, you would need to define **My\_View** as a class that inherits from **View** and use **.as\_view()** in the **urlpatterns**. If you prefer function-based views, as in your case, you can include the function directly.

Top of Form

Base views[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#base-views)

The following three classes provide much of the functionality needed to create Django views. You may think of them as *parent* views, which can be used by themselves or inherited from. They may not provide all the capabilities required for projects, in which case there are Mixins and Generic class-based views.

Many of Django’s built-in class-based views inherit from other class-based views or various mixins. Because this inheritance chain is very important, the ancestor classes are documented under the section title of **Ancestors (MRO)**. MRO is an acronym for Method Resolution Order.

**View**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#view)

***class*django.views.generic.base.View[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.View" \o "Permalink to this definition)**

The base view class. All other class-based views inherit from this base class. It isn’t strictly a generic view and thus can also be imported from **django.views**.

**Method Flowchart**

1. [**setup()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.setup)
2. [**dispatch()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch)
3. [**http\_method\_not\_allowed()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.http_method_not_allowed)
4. [**options()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.options)

**Example views.py**:

**from** **django.http** **import** HttpResponse

**from** **django.views** **import** View

**class** **MyView**(View):

**def** get(self, request, \*args, \*\*kwargs):

**return** HttpResponse("Hello, World!")

**Example urls.py**:

**from** **django.urls** **import** path

**from** **myapp.views** **import** MyView

urlpatterns = [

path("mine/", MyView.as\_view(), name="my-view"),

]

**Attributes**

**http\_method\_names[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.View.http_method_names" \o "Permalink to this definition)**

The list of HTTP method names that this view will accept.

Default:

["get", "post", "put", "patch", "delete", "head", "options", "trace"]

**Methods**

***classmethod*as\_view(*\*\*initkwargs*)**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.as_view)

Returns a callable view that takes a request and returns a response:

response = MyView.as\_view()(request)

The returned view has **view\_class** and **view\_initkwargs** attributes.

When the view is called during the request/response cycle, the [**setup()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.setup) method assigns the **[HttpRequest](https://docs.djangoproject.com/en/4.2/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** to the view’s **request** attribute, and any positional and/or keyword arguments [captured from the URL pattern](https://docs.djangoproject.com/en/4.2/topics/http/urls/#how-django-processes-a-request) to the **args** and **kwargs** attributes, respectively. Then [**dispatch()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch) is called.

If a **View** subclass defines asynchronous (**async def**) method handlers, **as\_view()** will mark the returned callable as a coroutine function. An **ImproperlyConfigured** exception will be raised if both asynchronous (**async def**) and synchronous (**def**) handlers are defined on a single view-class.

**Changed in Django 4.1:**

Compatibility with asynchronous (**async def**) method handlers was added.

**setup(*request*, *\*args*, *\*\*kwargs*)**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.setup)

Performs key view initialization prior to [**dispatch()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch).

If overriding this method, you must call **super()**.

**dispatch(*request*, *\*args*, *\*\*kwargs*)**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch)

The **view** part of the view – the method that accepts a **request** argument plus arguments, and returns an HTTP response.

The default implementation will inspect the HTTP method and attempt to delegate to a method that matches the HTTP method; a **GET** will be delegated to **get()**, a **POST** to **post()**, and so on.

By default, a **HEAD** request will be delegated to **get()**. If you need to handle **HEAD** requests in a different way than **GET**, you can override the **head()** method. See [Supporting other HTTP methods](https://docs.djangoproject.com/en/4.2/topics/class-based-views/#supporting-other-http-methods) for an example.

**http\_method\_not\_allowed(*request*, *\*args*, *\*\*kwargs*)**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.http_method_not_allowed)

If the view was called with an HTTP method it doesn’t support, this method is called instead.

The default implementation returns **HttpResponseNotAllowed** with a list of allowed methods in plain text.

**options(*request*, *\*args*, *\*\*kwargs*)**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.options)

Handles responding to requests for the OPTIONS HTTP verb. Returns a response with the **Allow** header containing a list of the view’s allowed HTTP method names.

If the other HTTP methods handlers on the class are asynchronous (**async def**) then the response will be wrapped in a coroutine function for use with **await**.

**Changed in Django 4.1:**

Compatibility with classes defining asynchronous (**async def**) method handlers was added.

**TemplateView[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "templateview" \o "Permalink to this headline)**

***class*django.views.generic.base.TemplateView[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.TemplateView" \o "Permalink to this definition)**

Renders a given template, with the context containing parameters captured in the URL.

**Ancestors (MRO)**

This view inherits methods and attributes from the following views:

* [**django.views.generic.base.TemplateResponseMixin**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/mixins-simple/#django.views.generic.base.TemplateResponseMixin)
* [**django.views.generic.base.ContextMixin**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/mixins-simple/#django.views.generic.base.ContextMixin)
* [**django.views.generic.base.View**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View)

**Method Flowchart**

1. [**setup()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.setup)
2. [**dispatch()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch)
3. [**http\_method\_not\_allowed()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.http_method_not_allowed)
4. [**get\_context\_data()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/mixins-simple/#django.views.generic.base.ContextMixin.get_context_data)

**Example views.py**:

**from** **django.views.generic.base** **import** TemplateView

**from** **articles.models** **import** Article

**class** **HomePageView**(TemplateView):

template\_name = "home.html"

**def** get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context["latest\_articles"] = Article.objects.all()[:5]

**return** context

**Example urls.py**:

**from** **django.urls** **import** path

**from** **myapp.views** **import** HomePageView

urlpatterns = [

path("", HomePageView.as\_view(), name="home"),

]

**Context**

* Populated (through **[ContextMixin](https://docs.djangoproject.com/en/4.2/ref/class-based-views/mixins-simple/" \l "django.views.generic.base.ContextMixin" \o "django.views.generic.base.ContextMixin)**) with the keyword arguments captured from the URL pattern that served the view.
* You can also add context using the **[extra\_context](https://docs.djangoproject.com/en/4.2/ref/class-based-views/mixins-simple/" \l "django.views.generic.base.ContextMixin.extra_context" \o "django.views.generic.base.ContextMixin.extra_context)** keyword argument for **[as\_view()](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.View.as_view" \o "django.views.generic.base.View.as_view)**.

**RedirectView[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "redirectview" \o "Permalink to this headline)**

***class*django.views.generic.base.RedirectView[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView" \o "Permalink to this definition)**

Redirects to a given URL.

The given URL may contain dictionary-style string formatting, which will be interpolated against the parameters captured in the URL. Because keyword interpolation is *always* done (even if no arguments are passed in), any **"%"** characters in the URL must be written as **"%%"** so that Python will convert them to a single percent sign on output.

If the given URL is **None**, Django will return an **HttpResponseGone** (410).

**Ancestors (MRO)**

This view inherits methods and attributes from the following view:

* [**django.views.generic.base.View**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View)

**Method Flowchart**

1. [**setup()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.setup)
2. [**dispatch()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch)
3. [**http\_method\_not\_allowed()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.View.http_method_not_allowed)
4. [**get\_redirect\_url()**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.RedirectView.get_redirect_url)

**Example views.py**:

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

**from** **django.views.generic.base** **import** RedirectView

**from** **articles.models** **import** Article

**class** **ArticleCounterRedirectView**(RedirectView):

permanent = **False**

query\_string = **True**

pattern\_name = "article-detail"

**def** get\_redirect\_url(self, \*args, \*\*kwargs):

article = get\_object\_or\_404(Article, pk=kwargs["pk"])

article.update\_counter()

**return** super().get\_redirect\_url(\*args, \*\*kwargs)

**Example urls.py**:

**from** **django.urls** **import** path

**from** **django.views.generic.base** **import** RedirectView

**from** **article.views** **import** ArticleCounterRedirectView, ArticleDetailView

urlpatterns = [

path(

"counter/<int:pk>/",

ArticleCounterRedirectView.as\_view(),

name="article-counter",

),

path("details/<int:pk>/", ArticleDetailView.as\_view(), name="article-detail"),

path(

"go-to-django/",

RedirectView.as\_view(url="https://www.djangoproject.com/"),

name="go-to-django",

),

]

**Attributes**

**url[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.url" \o "Permalink to this definition)**

The URL to redirect to, as a string. Or **None** to raise a 410 (Gone) HTTP error.

**pattern\_name[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.pattern_name" \o "Permalink to this definition)**

The name of the URL pattern to redirect to. Reversing will be done using the same args and kwargs as are passed in for this view.

**permanent**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.RedirectView.permanent)

Whether the redirect should be permanent. The only difference here is the HTTP status code returned. If **True**, then the redirect will use status code 301. If **False**, then the redirect will use status code 302. By default, **permanent** is **False**.

**query\_string[¶](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.query_string" \o "Permalink to this definition)**

Whether to pass along the GET query string to the new location. If **True**, then the query string is appended to the URL. If **False**, then the query string is discarded. By default, **query\_string** is **False**.

**Methods**

**get\_redirect\_url(*\*args*, *\*\*kwargs*)**[**¶**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/#django.views.generic.base.RedirectView.get_redirect_url)

Constructs the target URL for redirection.

The **args** and **kwargs** arguments are positional and/or keyword arguments [captured from the URL pattern](https://docs.djangoproject.com/en/4.2/topics/http/urls/#how-django-processes-a-request), respectively.

The default implementation uses **[url](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.url" \o "django.views.generic.base.RedirectView.url)** as a starting string and performs expansion of **%** named parameters in that string using the named groups captured in the URL.

If **[url](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.url" \o "django.views.generic.base.RedirectView.url)** is not set, **get\_redirect\_url()** tries to reverse the **[pattern\_name](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.pattern_name" \o "django.views.generic.base.RedirectView.pattern_name)** using what was captured in the URL (both named and unnamed groups are used).

If requested by **[query\_string](https://docs.djangoproject.com/en/4.2/ref/class-based-views/base/" \l "django.views.generic.base.RedirectView.query_string" \o "django.views.generic.base.RedirectView.query_string)**, it will also append the query string to the generated URL. Subclasses may implement any behavior they wish, as long as the method returns a redirect-ready URL string.

[**B**](https://docs.djangoproject.com/en/4.2/ref/class-based-views/)