Class Base View

Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views.

I. Base Class-Based Views / Base View
II. Generic Class-Based Views / Generic View

Advantages:-

1. Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
2. Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

Base Class-Based View

Base class-based views can be thought of 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 which extend what base views can do.

1. View
2. Template View
3. RedirectView

View:- The master class-based base view. 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.

django.views.generic.base.View

C:\Users\GS\AppData\Local\Programs\Python\Python38-32\Lib\site-packages\django\views\generic

Attribute:-
http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']
The list of HTTP method names that this view will accept.

Method:-

1. **setup(self, request, *args, **kwargs)** - It initializes view instance attributes: self.request, self.args, and self.kwargs prior to dispatch()

2. **dispatch(self, request, *args, **kwargs)** - The view part of the view – the method that accepts a request argument plus arguments, and returns a 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.

3. **http_method_not_allowed(self, request, *args, **kwargs)** - If the view was called with a 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

4. **options(self, request, *args, **kwargs)** - It 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.

5. **as_view(cls, **initkwargs)** - It returns a callable view that takes a request and returns a response.

6. **_allowed_methods(self)**

Example:-

Function Based View (views.py)	Class Based View (views.py)

```
from django.http import HttpResponse

def myview(request):
    return HttpResponse('Function Based View')
```
	```
	from django.views import View

	class MyView(View):
	    def get(self, request):
	        return HttpResponse('Class Based View')
	```

```
Urlpatterns = [
    Path('func/', views.myview, name='func'),
]
```
	```
	Urlpatterns = [
	    Path('cl/', views.MyView.as_view(),
	            name= 'cl'),
	]
	```

TemplateView:-

django.views.generic.base. Template View

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

This view inherits methods and attributes from the following views:

- django.views.generic.base. TemplateResponseMixin
- django.views.generic.base.ContextMixin
- django.views.generic.base.View

class TemplateView(TemplateResponseMixin, ContextMixin, View):

TemplateResponseMixin:-

It provides a mechanism to construct a TemplateResponse, given suitable context. The template to use is configurable and can be further customized by subclasses.

Attributes:-

template_name - The full name of a template to use as defined by a string. Not defining a template_name will raise a django.core.exceptions.ImproperlyConfigured exception.

template_engine - The NAME of a template engine to use for loading the template. template_engine is passed as the using keyword argument to response_class. Default is None, which tells Django to search for the template in all configured engines.

response_class - The response class to be returned by render_to_response method. Default is TemplateResponse. The template and context of TemplateResponse instances can be altered later (e.g. in template response middleware).
If you need custom template loading or custom context object instantiation, create a TemplateResponse subclass and assign it to response_class.

content_type - The content type to use for the response. content_type is passed as a keyword argument to response_class. Default is None - meaning that Django uses 'text/html'.

Method:-

render_to_response(context, **response_kwargs)- It returns a self.response_class instance.
If any keyword arguments are provided, they will be passed to the constructor of the response class. Calls get_template_names() to obtain the list of template names that will be searched looking for an existent template.

get_template_names() - It returns a list of template names to search for when rendering the template. The first template that is found will be used.

If template_name is specified, the default implementation will return a list containing template_name (if it is specified).

ContextMixin:-

A default context mixin that passes the keyword arguments received by **get_context_data()** as the template context.

Attribute:-

extra_context - A dictionary to include in the context. This is a convenient way of specifying some context in **as_view().**

Method:-

get_context_data(kwargs)**- It returns a dictionary representing the template context. The keyword arguments provided will make up the returned context.

TemplateView Define:-

views.py

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

class HomeView(TemplateView):
template_name = 'school/home.html'
  def get_context_data(self, **kwargs):
      context = super().get_context_data(**kwargs)
      context[‘name’] = ‘Rakib’
      context[‘roll’] = 1515
      return context
```

urls.py

```
from school import views
urlpatterns = [
        path('home/', views.HomeView.as_view(extra_ context ={‘course’: ‘Django’}), name='home'),
]
```

RedirectView:-

django.views.generic.base.RedirectView

It 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).

This view inherits methods and attributes from the following view:

- django.views.generic.base. View

Attributes:-
url - The URL to redirect to, as a string. Or None to raise a 410 (Gone) HTTP error.

pattern_name - The name of the URL pattern to redirect to. Reversing will be done using the same and kwargs as are passed in for this view.
args
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 - 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) - It constructs the target URL for redirection.

The args and kwargs arguments are positional and/or keyword arguments captured from the URL pattern, respectively.

The default implementation uses url as a starting string and performs expansion of % named parameters in that string using the named groups captured in the URL.

If url is not set, get_redirect_url() tries to reverse the pattern_name using what was captured in the URL (both named and unnamed groups are used).

If requested by 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.

Example:

views.py

```
from django.shortcuts import render
from django.views.generic.base import RedirectView, Template View

class ResultShows RedirectView(RedirectView):
    url='https://geekyshows.com'

class ResultRedirectView(RedirectView):
    pattern_name = 'mindex'
    permanent = False
    query_string = False

   def get_redirect_url(self, *args, **kwargs):
        print(kwargs)
        kwargs['pk'] = 16
        return super().get_redirect_url(*args, **kwargs)
```

urls.py

```
from django.urls import path
from django.views.generic.base import RedirectView, Template View
from school import views

urlpatterns = [
   path('admin/, admin.site.urls),

   path('', views. TemplateView.as_view(template_name='school/home.html'), name='blankindex'),

   path('home/, views. RedirectView.as_view(url='/'), name='home'),

   path('index/'', views.RedirectView.as_view(pattern_name='home'), name='index'),

   path('geekyshows/'),views.RedirectView.as_view(url='https://geekyshows.com'), name='go-to-geekyshows'

  path('geekyshows/', views.GeekyShowsRedirectView.as_view(), name='go-to-geekyshows'),

  path('home/<int:pk>/, views.GeekRedirectView.as_view(), name='geek'),

  path('<int:pk>/', views.TemplateView.as_view(template_name='school/home.html'), name='mindex'),

  #path('home/<slug:post>/', views.GeekRedirectView.as_view(), name='geek'),

  #path(''<slug:post>/, views. Template View.as_view(template_name='school/home.html'), name='index'),

]
```