**Video file: 86**

**Code file : ch77**

**Class Based View Base Class Based View View in Django 5**

Class based see below after function based we can also work in class based view and working with that

In last proper code with explain

Both ,function that I write below can perform same work but their method are different

First:

Urls.py:

    #first  1.view that we define in views.py

    path('newsfun/',views.newsfunview,name='newsfun'),

Views.py:

#here is our first view

def newsfunview(request):

    template\_name = 'myapp/news.html'

    context = {'info':'Subscribe to Geeky Shows YT Channel'}

    return render(request,template\_name,context)

second:

urls.py:

   #second 2.view that we define in views.py

    path('newsfun/',views.newsfunview,{'template\_name':'myapp/news.html'},name='newsfun'),

views.py:

#here is our second view

def newsfunview(request,template\_name):

    template\_name = template\_name

    context = {'info':'Subscribe to Geeky Shows YT Channel'}

    return render(request,template\_name,context)

if we use second we have one benefit :

we can render different page through urls with this same view:

    #second 2.view that we define in views.py

    path('newsfun/',views.newsfunview,{'template\_name':'myapp/news.html'},name='newsfun'),

    path('newsfun2/',views.newsfunview,{'template\_name':'myapp/news2.html'},name='news\_fun2\_view'),

here are some funtion based view that we used:

urls.py:

from django.urls import path

from myapp import views

urlpatterns = [

    path('fun1/',views.myfunview1,name='myfunview1'),

    path('fun2/',views.myfunview2,name='myfunview2'),

    path('homefun/',views.homefunview,name='home\_fun\_view'),

    path('aboutfun/',views.aboutfunview,name='aboutfunview'),

    # #first  1.view that we define in views.py

    # path('newsfun/',views.newsfunview,name='newsfun'),

    #second 2.view that we define in views.py

    path('newsfun/',views.newsfunview,{'template\_name':'myapp/news.html'},name='newsfun'),

    path('newsfun2/',views.newsfunview,{'template\_name':'myapp/news2.html'},name='news\_fun2\_view'),

    path('contactfun/',views.contactfunview,name='contact\_fun\_view'),

]

Views.py:

from django.shortcuts import render,HttpResponse

from myapp.forms import ContactForm

# Create your views here.

def myfunview1(request):

    return HttpResponse("Hello Function based view")

def myfunview2(request):

    return HttpResponse("<h1> Function Based View </h1>")

def homefunview(request):

    return render(request,'myapp/home.html')

def aboutfunview(request):

    context = {'msg': 'Welcome to Geeky shows Function Based View'}

    return render(request,'myapp/about.html',context)

# #here is our first view

# def newsfunview(request):

#     template\_name = 'myapp/news.html'

#     context = {'info':'Subscribe to Geeky Shows YT Channel'}

#     return render(request,template\_name,context)

#here is our second view

def newsfunview(request,template\_name):

    template\_name = template\_name

    context = {'info':'Subscribe to Geeky Shows YT Channel'}

    return render(request,template\_name,context)

def contactfunview(request):

    form = ContactForm(request.POST)

    if form.is\_valid():

        print(form.cleaned\_data['name'])

        return HttpResponse("Thank you for Submitted")

    else:

        form = ContactForm()

    return render(request,'myapp/contact.html',{'form':form})

Now we see all of them in base class based view :

## 1. What is a View in Django?

* A **view** is a Python function or class that takes a web request and returns a web response.
* Responses can be **HTML, JSON, XML, file, or even just plain text**.
* Views are linked to URLs via the urls.py file.

## 2. Function Based Views (FBV)

👉 **Definition**:

* A **function based view** is simply a Python function that receives an HttpRequest object and returns an HttpResponse object.

👉 **When to use**:

* Simple logic.
* Small projects.
* When fewer HTTP methods (like only GET/POST) are required.

### Example

# views.py

from django.http import HttpResponse

from django.shortcuts import render

def myfunview1(request):

    return HttpResponse("Hello Function Based View")

def myfunview2(request):

    return HttpResponse("<h1> Function Based View </h1>")

def homefunview(request):

    return render(request, 'myapp/home.html')

def aboutfunview(request):

    context = {'msg': 'Welcome to Geeky Shows Function Based View'}

    return render(request, 'myapp/about.html', context)

👉 **Mapped in urls.py**:

# urls.py

from django.urls import path

from myapp import views

urlpatterns = [

    path('fun1/', views.myfunview1, name='fun1'),

    path('fun2/', views.myfunview2, name='fun2'),

    path('homefun/', views.homefunview, name='homefun'),

    path('aboutfun/', views.aboutfunview, name='aboutfun'),

]

✅ Pros: Easy to read, simple for beginners.  
❌ Cons: For complex views, functions become large and difficult to manage.

## 3. Class Based Views (CBV)

👉 **Definition**:

* A **class based view** is a Python class that inherits from django.views.View.
* You define methods like get(), post(), etc. inside the class to handle HTTP methods.

👉 **When to use**:

* Complex logic.
* Reusability (Inheritance).
* Large projects.

### 3.1 Basic Example

from django.http import HttpResponse

from django.views import View

class MyClassView1(View):

    def get(self, request):

        return HttpResponse("Hello Class Based View")

class MyClassView2(View):

    def get(self, request):

        return HttpResponse("<h1> Class Based View </h1>")

👉 **urls.py**

urlpatterns = [

    path('cl1/', views.MyClassView1.as\_view(), name='classview1'),

    path('cl2/', views.MyClassView2.as\_view(), name='classview2'),

]

⚠️ Important:

* Always use .as\_view() in urls.py when calling a class-based view.
* Without it, Django won’t treat it as a view.

### 3.2 Passing Data in CBV

class MyClassView3(View):

    data = "Pass value through MyClassView3"

    def get(self, request):

        return HttpResponse(self.data)

# Child Class (Inheritance)

class MyChildClassView3(MyClassView3):

    def get(self, request):

        return HttpResponse(self.data + " - from Child Class")

👉 urls.py

path('cl3/', views.MyClassView3.as\_view(), name='classview3'),

path('childcl3/', views.MyChildClassView3.as\_view(), name='childclassview3'),

✅ This shows **inheritance** and **reusability** of CBVs.

### 3.3 Rendering Templates in CBV

class HomeClassView(View):

    def get(self, request):

        return render(request, 'myapp/home.html')

class AboutClassView(View):

    def get(self, request):

        context = {'msg': 'Welcome to Geeky Shows Class Based View'}

        return render(request, 'myapp/about.html', context)

### 3.4 Passing Template Name via URL

# Function Based View

def newsfunview(request, template\_name):

    context = {'info': 'Subscribe to Geeky Shows YT Channel'}

    return render(request, template\_name, context)

# Class Based View

class NewsClassView(View):

    template\_name = ''

    def get(self, request):

        context = {'info': 'Subscribe to Geeky Shows YT Channel'}

        return render(request, self.template\_name, context)

👉 urls.py

path('newsfun/', views.newsfunview, {'template\_name': 'myapp/news.html'}, name='newsfun'),

path('newsfun2/', views.newsfunview, {'template\_name': 'myapp/news2.html'}, name='newsfun2'),

path('newsclass1/', views.NewsClassView.as\_view(template\_name='myapp/news.html'), name='newsclass1'),

path('newsclass2/', views.NewsClassView.as\_view(template\_name='myapp/news2.html'), name='newsclass2'),

## 4. Handling Forms with Views

### Function Based View Example

def contactfunview(request):

    form = ContactForm(request.POST or None)

    if form.is\_valid():

        print(form.cleaned\_data['name'])

        return HttpResponse("Thank you for submitting")

    return render(request, 'myapp/contact.html', {'form': form})

### Class Based View Example

class ContactClassView(View):

    def get(self, request):

        form = ContactForm()

        return render(request, 'myapp/contact.html', {'form': form})

    def post(self, request):

        form = ContactForm(request.POST)

        if form.is\_valid():

            print(form.cleaned\_data['name'])   # ✅ use cleaned\_data (not cleaneddata)

            return HttpResponse("Thank you for submitting")

        return render(request, 'myapp/contact.html', {'form': form})

passing context:

class ContactClassView(View):

    def get(self, request):

        form = ContactForm()

        context = {

            'form': form,

            'title': 'Contact Us',

            'message': 'Fill out the form below to reach us.'

        }

        return render(request, 'myapp/contact.html', context)

    def post(self, request):

        form = ContactForm(request.POST)

        if form.is\_valid():

            print(form.cleaned\_data['name'])

            return HttpResponse("Thank you for submitting")

        context = {

            'form': form,

            'title': 'Contact Us',

            'message': 'Something went wrong, please try again.'

        }

        return render(request, 'myapp/contact.html', context)

## 5. Difference: FBV vs CBV

| **Feature** | **Function Based View** | **Class Based View** |
| --- | --- | --- |
| Structure | Python function | Python class |
| Handles HTTP Methods | Inside same function (if-else on request.method) | Separate methods (get(), post(), etc.) |
| Reusability | Low | High (Inheritance, Mixins) |
| Simplicity | Very simple | Slightly complex |
| Use Case | Small projects | Large projects |

## 6. Best Practices

* Use **FBV** for simple, one-time logic.
* Use **CBV** for reusable, extendable, and large projects.
* Always use .as\_view() when mapping CBVs in urls.py.
* Use **generic CBVs** (like TemplateView, ListView, CreateView) in real projects to reduce boilerplate code.