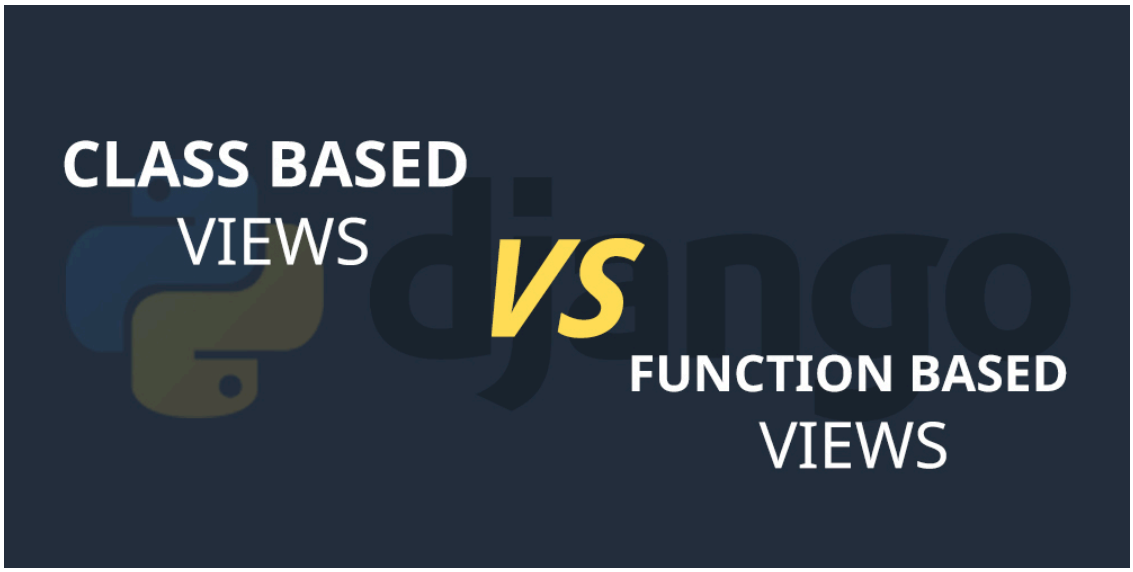
Class Based vs Function Based Views – Which One is Better to Use in Django?

Last Updated : 25 Sep, 2024

Django...We all know the popularity of this python framework all over the world. This framework has made life easier for developers. It has become easier for developers to build a full-fledged web application in Django. If you're an experienced Django developer then surely you might have been aware of the flow of the project. How things run in the boilerplate of Django and how data gets rendered to the user.



Django works on the MVT concept we mainly work on two types of views in it... class-based views and function-based views. If you're new to the Django framework then surely you might have been using FBVs (Function Based

Django Views Model Template Forms Jinja Python SQLite Flask Json Postman Interview Ques

Initially, **Django** started with the Function Based Views but later Django added the concept of class-based views to avoid the redundancy of code in the boilerplate. It is a debate among developers which one is better to use in Django... class-based views or function-based views? Today in this blog we are going to discuss this topic in-depth to get to know the pros and cons of both of the views.

You can accomplish your task using both of them. Some tasks can be best implemented using CBVs and some of them can be implemented in FBVs. Django views have mainly three requirements...

- They are callable. You can write the views either using function-based or class-based. While using CBVs you inherit the method as_view() that uses the dispatch() method to call the method that is appropriate depending on the HTTP verb (get, post), etc.
- As a first positional argument, Django views should accept HttpRequest.
- It should return the HttpResponse object, or it should raise an exception.

Now let's compare both of the views and see the pros and cons of both of them.

1. Function-Based Views

Function-based views are good for beginners. It is very easy to understand in comparison to class-based views. Initially when you want to focus on core fundamentals, using the function-based views gives the advantage to understand it. Let's discuss some pros and cons of it.

Pros:

- Easy to read, understand and implement.
- Explicit code flow
- Straightforward usage of decorators.
- Good for the specialized functionality.

Cons:

- Code redundancy and hard to extend
- Conditional branching will be used to handle HTTP methods.

As we have discussed function-based views are easy to understand but due to the code redundancy in a large Django project, you will find similar kinds of functions in the views. You will find a similar kind of code is repeated unnecessarily.

Here is an example of a function-based view...

Python

```python
def example_create_view(request, pk):
  template_name = 'form.html'
  form_class = FormExample

  form = form_class

  if request.method == 'POST':
    form = form_class(request.POST)
    if form.is_valid():
      form.save()
      return HttpResponseRedirect(reverse('list-view'))

  return render(request, template_name, {'form': form})
```

All the above cons of FBVs you won't find in class-based views. You won't have to write the same code over and over in your boilerplate.

2. Class-Based Views

Class-based views are the alternatives of function-based views. It is implemented in the projects as Python objects instead of functions. Class-based views don't replace function-based views, but they do have certain advantages over function-based views. Class-based views take care of basic functionalities such as deleting an item or add an item.

Using the class-based view is not easy if you're a beginner. You will have to go through the documentation, and you will have to study it properly. Once you understand the function-based view in Django and your concepts are clear, you can move to the class-based views. Let's discuss the class-based views in detail.

Pros

- The most significant advantage of the class-based view is inheritance. In the class-based view, you can inherit another class, and it can be modified for the different use cases.
- It helps you in following the DRY principle. You won't have to write the same code over and over in your boilerplate. Code reusability is possible in class-based views.
- You can extend class-based views, and you can add more functionalities using Mixins.
- Another advantage of using a class-based view is code structuring. In class-based views, you can use different class instance methods (instead of conditional branching statements inside function-based views) to generate different HTTP requests.
- Built-in generic class-based views.

Cons

- Complex to implement and harder to read
- Implicit code flow.
- Extra import or method override required in view decorators.

Below is an example of a class-based view...

Python

```
class MyCreateView(View):
  template_name = 'form.html'
  form_class = MyForm

  def get(self, request, *args, **kwargs):
    form = self.form_class
    return render(request, template_name, {'form': form})

  def post(self, request, *args, **kwargs):
    form = self.form_class(request.POST)
    if form.is_valid():
      form.save()
      return HttpResonseRedirect(reverse('list-view'))
    else:
      return render(request, self.template_name, {'form':
form})
```

We have a little abstraction and method as_view() is calling the dispatch() to determine which class method needs to be executed, depending on the HTTP

request. as_view() let you override the class attributes in your URLs confs. You can do something like the below...

Python

```python
urlpatterns = [
    url(r'^new/$', MyCreateView.as_view(), name='original-create-view')
    url(r'^new_two/$', MyCreateView.as_view(template_name='other_form.html',
                    form_class='MyOtherForm'), name='modified-create-view')
  ]
```

Once you start using the Django generic class-based views, you will be able to over-write the helper method like get_form_class and get_template_names. You can insert the additional logic at these points instead of just overriding the class attribute.

One of the good examples of it is...ModelFormMixin. form_valid method is overridden. With the updated value stored in self.object() form_valid method is overridden.

3. Django Generic Class-Based View

Creating a new object, form handling, list views, pagination, archive views all these things are the common use cases in a Web application. It comes in Django core, you can implement them from the module django.views.generic. Generic class-based views are a great choice to perform all these tasks. It speeds up the development process.

Django provides a set of views, mixins, and generic class-based views. Taking the advantage of it you can solve the most common tasks in web development.

The main goal is not to reduce the boilerplate. It saves you from writing the same code again and again. Modify MyCreateView to inherit from django.views.generic.CreateView.

Python

```python
from django.views.generic import CreateView
class MyCreateView(CreateView):
    model = MyModel
    form_class = MyForm
```

You might be thinking that where all the code disappears. The answer is that it's all in django.views.generic.CreateView. You get a lot of functionality and shortcuts when you inherit from CreateView. You also buy into a sort of 'convention over configuration.' style arrangement. Let's discuss few more details...

By default template should reside in **/<modelname>/<modelname>_form.html.** You can change it by setting the class attribute **template_name** and **template_name_suffix.**

- We also need to declare the model and form_class attributes. Methods you inherit from CreateView rely on them.
- You will have to declare success_url as a class attribute on the view or you will have to specify **get_absolute_url()** in the model. This is important for the view in your boilerplate else the view won't know where to redirect to following a successful form submission.

- Define the fields in your form or specify the fields class attribute on the view. Here in this example, you can choose to do the latter.

Look at the example given below to check how it will look like.

Python

```python
from django import forms
from . models import MyModel
class MyModelForm(forms.ModelForm):
  class Meta:
    model = MyModel
    fields = ['name', 'description']
```

Conclusion

It is still a debate among developers that which one is good to use. Class-based views or function-based views? We have discussed the pros and cons for both of them but it totally depends on the context and the needs. We have mentioned that class-based views don't replace function-based views. In some cases, function-based views are better and in some cases, class-based views are better.

In the implementation of the list view, you can get it working by subclassing the ListView and overriding the attributes. In a scenario where you need to perform the more complex operation, handling multiple forms at once, a function-based view will be a better choice for you.

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

 anuu... 8

Previous Article

Django Class Based Views

Next Article

Django Templates

Similar Reads

How to Use permission_required Decorators with Django Class-Based Views

In Django, permissions are used to control access to views and resources. When working with function-based views (FBVs), decorators like @permission_require...

4 min read

Createview - Class Based Views Django

Create View refers to a view (logic) to create an instance of a table in the database. We have already discussed basics of Create View in Create View –...

3 min read

ListView - Class Based Views Django

List View refers to a view (logic) to display multiple instances of a table in the database. We have already discussed the basics of List View in List View –...

4 min read

UpdateView - Class Based Views Django

UpdateView refers to a view (logic) to update a particular instance of a table from the database with some extra details. It is used to update entries in the databas...

3 min read

DetailView - Class Based Views Django

Detail View refers to a view (logic) to display one instances of a table in the database. We have already discussed basics of Detail View in Detail View –...

3 min read

DeleteView - Class Based Views Django

Delete View refers to a view (logic) to delete a particular instance of a table from the database. It is used to delete entries in the database for example, deleting a...

3 min read

FormView - Class Based Views Django

FormView refers to a view (logic) to display and verify a Django Form. For example, a form to register users at Geeksforgeeks. Class-based views provide ...

3 min read

Class based views - Django Rest Framework

Class-based views help in composing reusable bits of behavior. Django REST Framework provides several pre-built views that allow us to reuse common...

12 min read

Django Class Based Views

Django is a Python-based web framework that allows you to quickly create web applications. It has a built-in admin interface which makes it easy to work with it...

10 min read

Create View - Function based Views Django

Create View refers to a view (logic) to create an instance of a table in the database. It is just like taking an input from a user and storing it in a specified...

3 min read

Article Tags : Difference Between GBlog Python Python Django +1 More

Practice Tags : python

Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

GET IT ON Google Play Download on the App Store

Company

About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns

Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation

OOAD
System Design Bootcamp
Interview Questions
Puzzles

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved