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.

Step 1: Create a Django Project

If you haven't created a Django project yet, you can do so using the following command:

django-admin startproject myproject

Replace "myproject" with your desired project name.

Step 2: Create a Django App

Inside your project, create a Django app using the following command:

```
cd myproject
python manage.py startapp myapp
```

Step 3: Define a Function-Based View

Open the `views.py` file inside your app directory (`myapp`) and define a simple function-based view. For example:

myapp/views.py

from django.http import HttpResponse

def hello(request):

    return HttpResponse("Hello, Django!")

Step 4: Configure URL Patterns

Open the `urls.py` file inside your app directory (`myapp`) and configure the URL pattern for your view:

myapp/urls.py

```
from django.urls import path
from .views import hello

urlpatterns = [
    path('hello/', hello, name='hello'),
]
```

Step 5: Include App URLs in Project URLs

Open the `urls.py` file in your project directory (`myproject`) and include the URLs of your app:

```
# myproject/urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')),
]
```

Step 6: Run the Development Server

Now, run the development server using the following command:

```
python manage.py runserver
```

Class-Based Views (CBVs):

1. **Reusability and Mixins:**
    - CBVs encourage code reuse through the use of mixins. You can create a base class with common functionality and then inherit from it to build specialized views.
2. **Organized Code:**
    - CBVs allow you to organize your code in a more object-oriented way. Views can be grouped into classes, making it easier to manage and extend.
3. **Built-in Generic Views:**
    - Django provides a set of generic class-based views that cover common use cases (e.g., displaying a list of objects, displaying a single object). This can reduce the amount of boilerplate code you need to write.
4. **Functionality Through Inheritance:**
    - CBVs allow you to leverage the power of inheritance to extend or override behavior easily. This can lead to more maintainable and extensible code.
5. **Mixins:**

- CBVs support the use of mixins, which are reusable components that can be combined with views to add specific functionality.

Choosing Between FBVs and CBVs:

1. **Project Requirements:**
    - For simpler projects or views, FBVs might be sufficient and easier to work with. For larger and more complex projects, CBVs might provide a more organized structure.
2. **Developer Preference:**
    - Some developers have a personal preference for one approach over the other. It's essential to consider the team's familiarity and comfort with each style.
3. **Codebase Consistency:**
    - Consistency in the codebase is crucial. If a project already uses one approach, it might be beneficial to stick with that to maintain a consistent code style.

Step 1: Create a Django Project

If you haven't created a Django project yet, you can do so using the following command:

```
django-admin startproject myproject
```

Step 2: Create a Django App

Inside your project, create a Django app using the following command:

```
cd myproject
```

```
python manage.py startapp myapp
```

Step 3: Define a Class-Based View

Open the `views.py` file inside your app directory (`myapp`) and define a simple class-based view. For example:

```
# myapp/views.py
from django.views import View
from django.http import HttpResponse

class HelloView(View):
    def get(self, request):
        return HttpResponse("Hello, Django!")
```

Step 4: Configure URL Patterns

Open the `urls.py` file inside your app directory (`myapp`) and configure the URL pattern for your class-based view:

```
# myapp/urls.py
from django.urls import path
from .views import HelloView

urlpatterns = [
```

```
    path('hello/', HelloView.as_view(), name='hello'),
]
```

Step 5: Include App URLs in Project URLs

Open the `urls.py` file in your project directory (`myproject`) and include the URLs of your app:

```
# myproject/urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')),
]
```

Step 6: Run the Development Server

Now, run the development server using the following command:

```
python manage.py runserver
```