



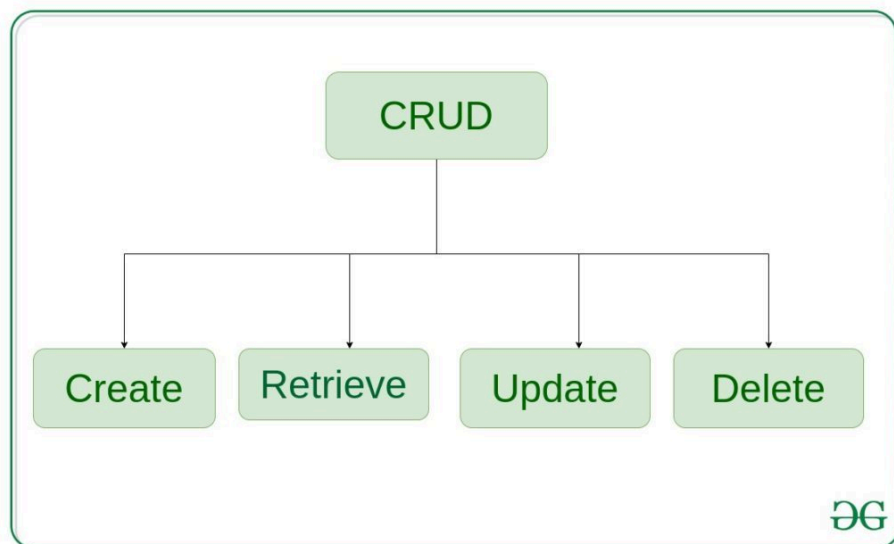
Django Class Based Views

Last Updated : 23 Sep, 2024

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. It is often called a Class-Based **included framework** because it provides built-in facilities for every functionality. Class Based Generic Views are an advanced set of Built-in views that are used for the implementation of selective view strategies such as Create, Retrieve, Update, and Delete. Class-based views simplify the use by separating GET, and POST requests for a view. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

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

This article revolves around the complete implementation of **Class Based Views in Django** (Create, Retrieve, Update, Delete). Let's discuss what actually CRUD means,



[CreateView](#) – create or add new entries in a table in the database.

[Retrieve Views](#) – read, retrieve, search, or view existing entries as a list([ListView](#)) or retrieve a particular entry in detail ([DetailView](#))

[UpdateView](#) – update or edit existing entries in a table in the database

[DeleteView](#) – delete, deactivate, or remove existing entries in a table in the database

[FormView](#) – render a form to template and handle data entered by user

Django Class Based Views CRUD Operations

Illustration of **How to create and use CRUD views** using an Example. Consider a project named geeksforgeeks having an app named geeks.

Refer to the following articles to check how to create a project and an app in Django.

- [How to Create a Basic Project using MVT in Django?](#)
- [How to Create an App in Django ?](#)

After you have a project and an app, let's create a model of which we will be creating instances through our view. In geeks/models.py,

Python3



```
1 # import the standard Django Model
2 # from built-in library
3 from django.db import models
4
5 # declare a new model with a name "GeeksModel"
6 class GeeksModel(models.Model):
7
8     # fields of the model
9     title = models.CharField(max_length = 200)
10    description = models.TextField()
11
12    # renames the instances of the model
13    # with their title name
14    def __str__(self):
```

```
15         return self.title
```

After creating this model, we need to run two commands in order to create Database for the same.

Python manage.py [makemigrations](#)

Python manage.py [migrate](#)

Now we will create a Django ModelForm for this model. Refer this article for more on modelform – [Django ModelForm – Create form from Models](#). create a file forms.py in geeks folder,

Python3

```
1  from django import forms
2  from .models import GeeksModel
3
4
5  # creating a form
6  class GeeksForm(forms.ModelForm):
7
8      # create meta class
9      class Meta:
10         # specify model to be used
11         model = GeeksModel
12
13         # specify fields to be used
14         fields = [
15             "title",
16             "description",
17         ]
```

Using Class Based Views

At its core, a class-based view allows you to respond to different HTTP request methods with different class instance methods, instead of with conditionally branching code inside a single view function.

So where the code to handle HTTP GET in a view function would look something like:

Python3

```
1 from django.http import HttpResponse
2
3 def my_view(request):
4     if request.method == 'GET':
5         # <view logic>
6         return HttpResponse('result')
```

In a class-based view, this would become:

Python3

```
1 from django.http import HttpResponse
2 from django.views import View
3
4 class MyView(View):
5     def get(self, request):
6         # <view logic>
7         return HttpResponse('result')
```

Similarly in urls.py, one needs to use as_view() method to differentiate a class based view from function based view.

Python3

```
1 # urls.py
2 from django.urls import path
3 from myapp.views import MyView
4
5 urlpatterns = [
6
7     path('about/', MyView.as_view()),
8
9 ]
```

CreateView

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 – Function based Views Django. Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create Create View for and the fields. Then Class based CreateView will automatically try to find a template in `app_name/modelname_form.html`. In our case it is `geeks/templates/geeks/geeksmodel_form.html`. Let's create our class based view. In `geeks/views.py`,

Python3

```
1 from django.views.generic.edit import CreateView
2 from .models import GeeksModel
3
4 class GeeksCreate(CreateView):
5
6     # specify the model for create view
7     model = GeeksModel
8
9     # specify the fields to be displayed
10
11     fields = ['title', 'description']
```

Now create a url path to map the view. In `geeks/urls.py`,

Python3

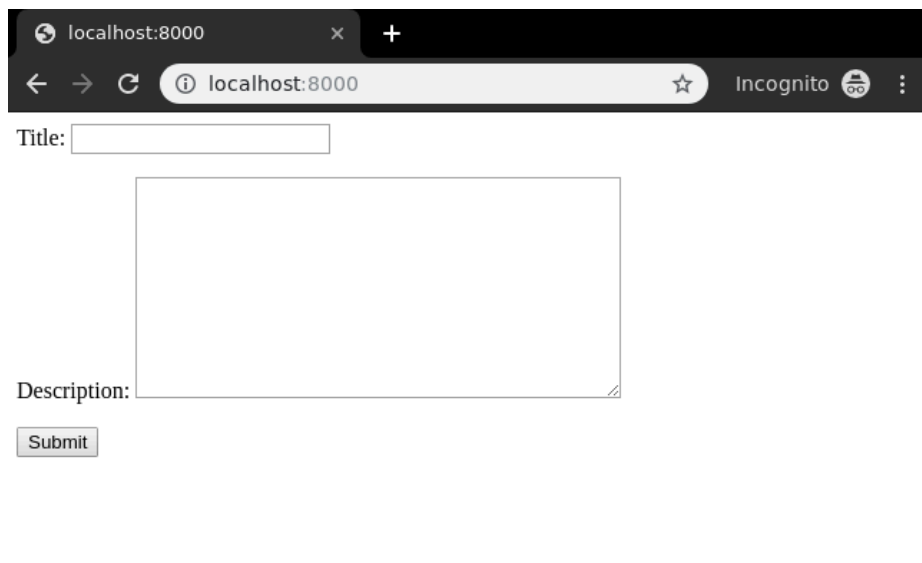
```
1 from django.urls import path
2
3 # importing views from views..py
4 from .views import GeeksCreate
5 urlpatterns = [
6     path('', GeeksCreate.as_view() ),
7 ]
```

Create a template in `templates/geeks/geeksmodel_form.html`,

HTML

```
1 <form method="POST" enctype="multipart/form-data">
2
3     <!-- Security token -->
4     {% csrf_token %}
5
6     <!-- Using the formset -->
7     {{ form.as_p }}
8
9     <input type="submit" value="Submit">
10 </form>
```

Let's check what is there on <http://localhost:8000/>



The screenshot shows a web browser window with the address bar set to 'localhost:8000'. The page displays a simple form with the following elements:

- A 'Title:' label followed by a single-line text input field.
- A 'Description:' label followed by a multi-line text area.
- A 'Submit' button located below the text area.

To check complete implementation of Class based CreateView, visit [Createview – Class Based Views Django](#).

Retrieve Views

ListView

List View refers to a view (logic) to display multiple instances of a table in the database. We have already discussed basics of List View in [List View – Function based Views Django](#). Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create ListView for, then Class based ListView will automatically try to find a template in `app_name/modelname_list.html`. In our case it is

geeks/templates/geeks/geeksmode_list.html. Let's create our class based view.
In geeks/views.py,

Python3

```
1 from django.views.generic.list import ListView
2 from .models import GeeksModel
3
4 class GeeksList(ListView):
5
6     # specify the model for list view
7     model = GeeksModel
```

Now create a url path to map the view. In geeks/urls.py,

Python3

```
1 from django.urls import path
2
3 # importing views from views..py
4 from .views import GeeksList
5 urlpatterns = [
6     path('', GeeksList.as_view()),
7 ]
```

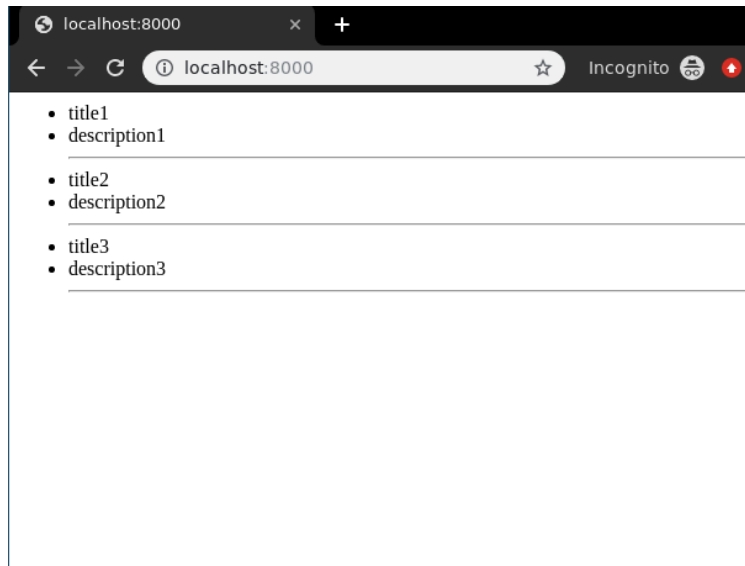
Create a template in templates/geeks/geeksmode_list.html,

HTML

```
1 <ul>
2     <!-- Iterate over object_list -->
3     {% for object in object_list %}
4     <!-- Display Objects -->
5     <li>{{ object.title }}</li>
6     <li>{{ object.description }}</li>
7
8     <hr/>
9     <!-- If object_list is empty -->
10    {% empty %}
```

```
11     <li>No objects yet.</li>
12     {% endfor %}
13 </ul>
```

Let's check what is there on <http://localhost:8000/>



To check complete implementation of Class based ListView, visit [ListView – Class Based Views Django](#)

DetailView

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 – Function based Views Django](#). Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create DetailView for, then Class based DetailView will automatically try to find a template in `app_name/modelname_detail.html`. In our case it is `geeks/templates/geeks/geeksmodel_detail.html`. Let's create our class based view. In `geeks/views.py`,

Python3

```
1 from django.views.generic.detail import DetailView
2
3 from .models import GeeksModel
4
5 class GeeksDetailView(DetailView):
6     # specify the model to use
```



```
7 model = GeeksModel
```

Now create a url path to map the view. In `geeks/urls.py`,

Python3

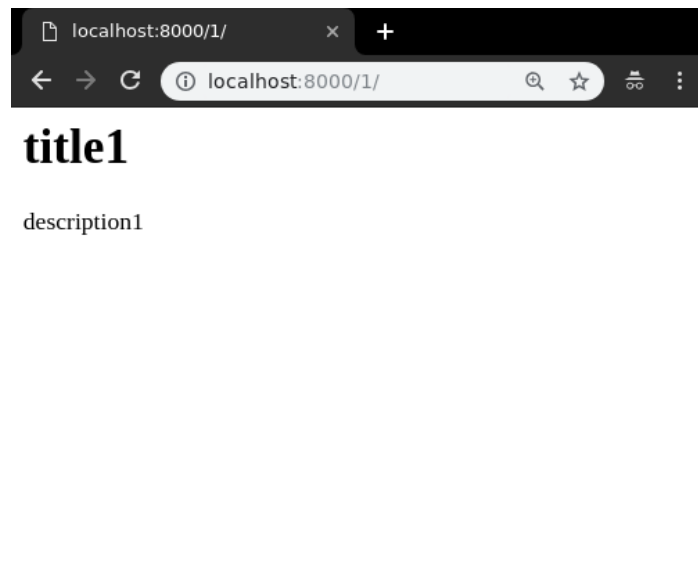
```
1 from django.urls import path
2
3 # importing views from views..py
4 from .views import GeeksDetailView
5 urlpatterns = [
6     # <pk> is identification for id field,
7     # slug can also be used
8     path('<pk>/', GeeksDetailView.as_view()),
9 ]
```

Create a template in `templates/geeks/geeksmodel_detail.html`,

HTML

```
1 <h1>{{ object.title }}</h1>
2
3
4
5 <p>{{ object.description }}</p>
```

Let's check what is there on <http://localhost:8000/1/>



To check complete implementation of Class based Detail View, visit [DetailView – Class Based Views Django](#)

UpdateView

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 database for example, updating an article at geeksforgeeks. We have already discussed basics of Update View in [Update View – Function based Views Django](#). Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create UpdateView for, then Class based UpdateView will automatically try to find a template in `app_name/modelname_form.html`. In our case it is `geeks/templates/geeks/geeksmodel_form.html`. Let's create our class based view. In `geeks/views.py`,

Python3

```
1  # import generic UpdateView
2  from django.views.generic.edit import UpdateView
3
4  # Relative import of GeeksModel
5  from .models import GeeksModel
6
7  class GeeksUpdateView(UpdateView):
8      # specify the model you want to use
9      model = GeeksModel
10
11     # specify the fields
12     fields = [
13         "title",
14         "description"
15     ]
16
17     # can specify success url
18     # url to redirect after successfully
19     # updating details
20     success_url = "/"
```

Now create a url path to map the view. In `geeks/urls.py`,

Python3

```
1 from django.urls import path
2
3 # importing views from views..py
4 from .views import GeeksUpdateView
5 urlpatterns = [
6     # <pk> is identification for id field,
7     # <slug> can also be used
8     path('<pk>/update', GeeksUpdateView.as_view()),
9 ]
```

Create a template in `templates/geeks/geeksmodel_form.html`,

HTML

```
1 <form method="post">
2     {% csrf_token %}
3     {{ form.as_p }}
4     <input type="submit" value="Save">
5 </form>
```

Let's check what is there on <http://localhost:8000/1/update/>



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/1/update'. The page content includes a form with a 'Title:' label and a text input field containing 'title1'. Below this is a 'Description:' label and a larger text area containing 'description1'. At the bottom of the form is a 'Save' button.

To check complete implementation of Class based UpdateView, visit [UpdateView – Class Based Views Django](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/).

DeleteView

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 an article at geeksforgeeks. We have already discussed basics of Delete View in [Delete View – Function based Views Django](#). Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create DeleteView for, then Class based DeleteView will automatically try to find a template in `app_name/modelname_confirm_delete.html`. In our case it is `geeks/templates/geeks/geeksmodel_confirm_delete.html`. Let's create our class based view. In `geeks/views.py`,

Python3

```
1  # import generic UpdateView
2  from django.views.generic.edit import DeleteView
3
4  # Relative import of GeeksModel
5  from .models import GeeksModel
6
7  class GeeksDeleteView(DeleteView):
8      # specify the model you want to use
9      model = GeeksModel
10
11     # can specify success url
12     # url to redirect after successfully
13     # deleting object
14     success_url = "/"
```

Now create a url path to map the view. In `geeks/urls.py`,

Python3

```
1  from django.urls import path
2
3  # importing views from views..py
4  from .views import GeeksDeleteView
```

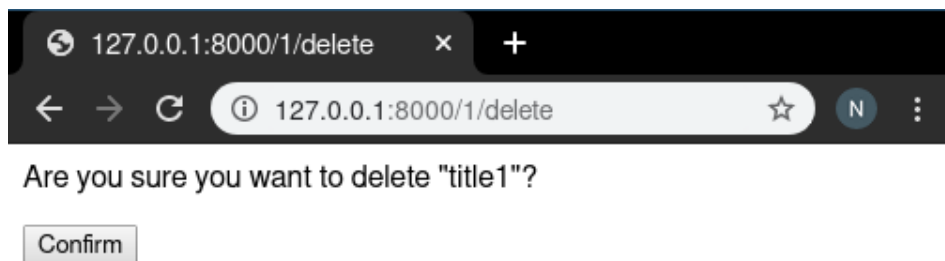
```
5 urlpatterns = [  
6     # <pk> is identification for id field,  
7     # slug can also be used  
8     path('<pk>/delete/', GeeksDeleteView.as_view()),  
9 ]
```

Create a template in templates/geeks/geeksmodel_confirm_delete.html,

HTML

```
1 <form method="post">{% csrf_token %}  
2  
3  
4  
5 <p>Are you sure you want to delete "{{ object }}"?</p>  
6  
7  
8  
9     <input type="submit" value="Confirm">  
10 </form>
```

Let's check what is there on <http://localhost:8000/1/delete>



To check complete implementation of Class based DeleteView, visit [DeleteView – Class Based Views Django](#)

FormView

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 automatically setup everything from A to Z. One just needs to specify which form to create FormView for and template_name, then Class based FormView will automatically render that form. Let's create our class based view. In geeks/views.py,

Python3

```
1 # import generic FormView
2 from django.views.generic.edit import FormView
3
4 # Relative import of GeeksForm
5 from .forms import GeeksForm
6
7 class GeeksFormView(FormView):
8     # specify the Form you want to use
9     form_class = GeeksForm
10
11     # specify name of template
12     template_name = "geeks / geeksmodel_form.html"
13
14     # can specify success url
15     # url to redirect after successfully
16     # updating details
17     success_url = "/thanks/"
```

Create a template for this view in geeks/geeksmodel_form.html,

HTML

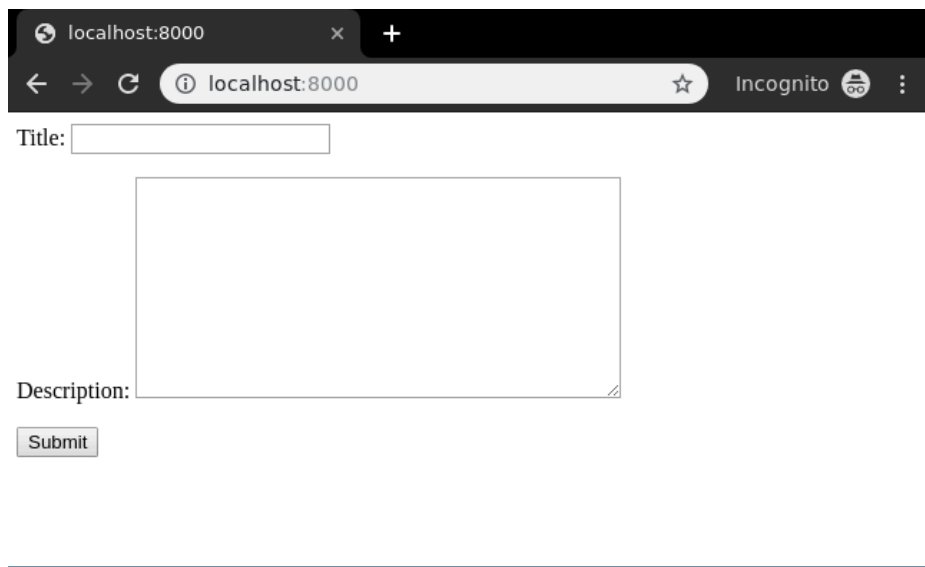
```
1 <form method="post">
2     {% csrf_token %}
3     {{ form.as_p }}
4     <input type="submit" value="Save">
5 </form>
```

Map a url to this view in geeks/urls.py,

Python3

```
1 from django.urls import path
2
3 # importing views from views..py
4 from .views import GeeksFormView
5 urlpatterns = [
6     path('', GeeksFormView.as_view()),
7 ]
```

Now visit <http://127.0.0.1:8000/>,



The screenshot shows a web browser window with the address bar set to 'localhost:8000'. The page displays a simple form with two input fields and a submit button. The first field is labeled 'Title:' and is a single-line text input. The second field is labeled 'Description:' and is a larger text area. Below the text area is a button labeled 'Submit'.

To check complete implementation of Class based FormView, visit [FormView – Class Based Views Django](#)

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!

N Nave...

27

Previous Article

Delete View - Function based Views
Django

Next Article

Createview - Class Based Views Django

Similar Reads

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

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...

7 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

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

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 :[Django](#)[Python](#)[Django-views](#)[Python Django](#)**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



Company

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

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

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Computer Science

Languages

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

Data Science & ML

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

Python Tutorial

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

DevOps

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

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Interview Preparation

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

GeeksforGeeks Videos

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

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved