Course Module 1: Course Unit 10: Django (Fundamentals)

Week 13

Django (Fundamentals)

Objectives:

- Classify Django fundamentals.
- Categorize Django fundamentals.
- Generate a Django model using Django fundamentals.

Django Fundamentals

Django Fundamentals

- Django Architecture
- Django MTV Architecture
- Django Project Layout and File Structure
- **Django Models**
- Django Views
- Django Templates
- Django URLs and URL Conf

Django Models

Django Models

- the built-in feature that Django uses to create tables, their fields, and various constraints.
- In short, Django Models is the SQL of Database one uses with Django.

SQL (Structured Query Language) is complex and involves a lot of different queries for creating, deleting, updating or any other stuff related to database.

- Django models simplify the tasks and organize tables into models. Generally, each model maps to a single database table.

Django Models

Creating a Model

Syntax:

```
from django.db import models


class ModelName(models.Model):
        field_name = models.Field(**options)
```

Django Models

Model **GeeksModel**
title
description
image

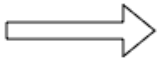


Table **appname_geeksmodel**	
ID	1
title	GeeksForGeeks
description	Best Website
image	obj.png

Example:

```
# import the standard Django Model
# from built-in library
from django.db import models

# declare a new model with a name "GeeksModel"
class GeeksModel(models.Model):
        # fields of the model
    title = models.CharField(max_length = 200)
    description = models.TextField()
    last_modified = models.DateTimeField(auto_now_add = True)
    img = models.ImageField(upload_to = "images/")

        # renames the instances of the model
        # with their title name
    def __str__(self):
        return self.title
```

Django Models

Whenever we create a Model, Delete a Model, or update anything in any of models.py of our project. We need to run two commands ***makemigrations*** and ***migrate***.

- makemigrations basically generates the SQL commands for preinstalled apps (which can be viewed in installed apps in settings.py) and your newly created app's model which you add in installed apps
- migrate executes those SQL commands in the database file

python manage.py makemigrations SQL Query to create above Model as a Table is created

python manage.py migrate creates the table in the database

Django Models

Render a model in Django Admin Interface

To render a model in Django admin, we need to modify app/admin.py. Go to admin.py in the created app and enter the following code. Import the corresponding model from models.py and register it to the admin interface.

```
from django.contrib import admin

# Register your models here.
from .models import GeeksModel

admin.site.register(GeeksModel)
```

Django Models

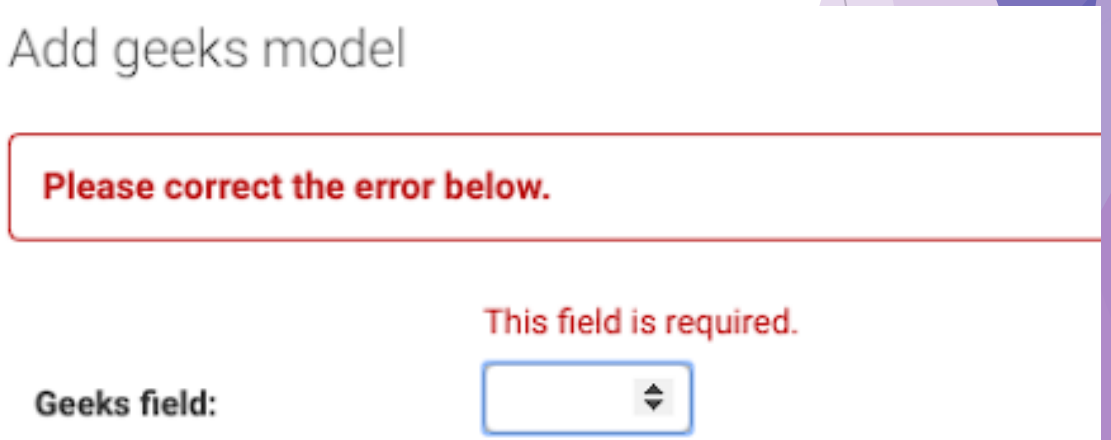
Validation on Fields in a Model

Built-in Field Validations in Django models are the default validations that come predefined to all Django fields. Every field comes in with built-in validations from Django validators.

For example, IntegerField comes with built-in validation that it can only store integer values and that too in a particular range.

```
from django.db import models
from django.db.models import Model
# Create your models here.

class GeeksModel(Model):
    geeks_field = models.IntegerField()

    def __str__(self):
        return self.geeks_field
```

After running makemigrations and migrate on Django and rendering above model, let us try to create an instance using string "**GfG is Best**".



one can not enter a string in an IntegerField. Similarly every field has its own validations.

Django Models

Basic model data types and fields list

The most important part of a model and the only required part of a model is the list of database fields it defines. Fields are specified by class attributes. Here is a list of all Field types used in Django.

Field Name	Description
AutoField	It An IntegerField that automatically increments.
BigAutoField	It is a 64-bit integer, much like an AutoField except that it is guaranteed to fit numbers from 1 to 9223372036854775807.
BigIntegerField	It is a 64-bit integer, much like an IntegerField except that it is guaranteed to fit numbers from -9223372036854775808 to 9223372036854775807.

Django Models - GeeksforGeeks

Django Models

BinaryField	A field to store raw binary data.
BooleanField	A true/false field. The default form widget for this field is a CheckboxInput.
CharField	It is string filed for small to large-sized input
DateField	A date, represented in Python by a datetime.date instance
	It is used for date and time, represented in Python by a datetime.datetime instance.
DecimalField	It is a fixed-precision decimal number, represented in Python by a Decimal instance.

Django Models - GeeksforGeeks

Django Models

DurationField	A field for storing periods of time.
EmailField	It is a CharField that checks that the value is a valid email address.
FileField	It is a file-upload field.
FloatField	It is a floating-point number represented in Python by a float instance.
ImageField	It inherits all attributes and methods from FileField, but also validates that the uploaded object is a valid image.
IntegerField	It is an integer field. Values from -2147483648 to 2147483647 are safe in all databases supported by Django.

Django Models - GeeksforGeeks

Django Models

GenericIPAddressField	An IPv4 or IPv6 address, in string format (e.g. 192.0.2.30 or 2a02:42fe::4).
NullBooleanField	Like a BooleanField, but allows NULL as one of the options.
PositiveIntegerField	Like an IntegerField, but must be either positive or zero (0).
PositiveSmallIntegerField	Like a PositiveIntegerField, but only allows values under a certain (database-dependent) point.
SlugField	Slug is a newspaper term. A slug is a short label for something, containing only letters, numbers, underscores or hyphens. They're generally used in URLs.

Django Models - GeeksforGeeks

Django Models

SmallIntegerField	It is like an IntegerField, but only allows values under a certain (database-dependent) point.
TextField	A large text field. The default form widget for this field is a Textarea.
TimeField	A time, represented in Python by a datetime.time instance.
URLField	A CharField for a URL, validated by URLValidator.
UUIDField	A field for storing universally unique identifiers. Uses Python's UUID class. When used on PostgreSQL, this stores in a uuid datatype, otherwise in a char(32).

Django Models - GeeksforGeeks

Django Models

Relationship Fields

Django also defines a set of fields that represent relations.

Field Name	Description
ForeignKey	A many-to-one relationship. Requires two positional arguments: the class to which the model is related and the on_delete option.
ManyToManyField	A many-to-many relationship. Requires a positional argument: the class to which the model is related, which works exactly the same as it does for ForeignKey, including recursive and lazy relationships.
OneToOneField	A one-to-one relationship. Conceptually, this is similar to a ForeignKey with unique=True, but the "reverse" side of the relation will directly return a single object.

Django Models - GeeksforGeeks

Django Models

Field Options

Field Options are the arguments given to each field for applying some constraint or imparting a particular characteristic to a particular Field. For example, adding an argument null = True to CharField will enable it to store empty values for that table in relational database.
Here are the field options and attributes that an CharField can use.

Field Options	Description
Null	If **True**, Django will store empty values as **NULL** in the database. Default is **False**.
Blank	If **True**, the field is allowed to be blank. Default is **False**.
db_column	The name of the database column to use for this field. If this isn't given, Django will use the field's name.

Django Models - GeeksforGeeks

Django Models

Default	The default value for the field. This can be a value or a callable object. If callable it will be called every time a new object is created.
help_text	Extra "help" text to be displayed with the form widget. It's useful for documentation even if your field isn't used on a form.
primary_key	If True, this field is the primary key for the model.
editable	If **False**, the field will not be displayed in the admin or any other ModelForm. They are also skipped during model validation. Default is **True**.

Django Models - GeeksforGeeks

Django Models

error_messages	The error_messages argument lets you override the default messages that the field will raise. Pass in a dictionary with keys matching the error messages you want to override.
help_text	Extra "help" text to be displayed with the form widget. It's useful for documentation even if your field isn't used on a form.
verbose_name	A human-readable name for the field. If the verbose name isn't given, Django will automatically create it using the field's attribute name, converting underscores to spaces.
validators	A list of validators to run for this field. See the validators documentation for more information.
Unique	If True, this field must be unique throughout the table.

Django Models - GeeksforGeeks

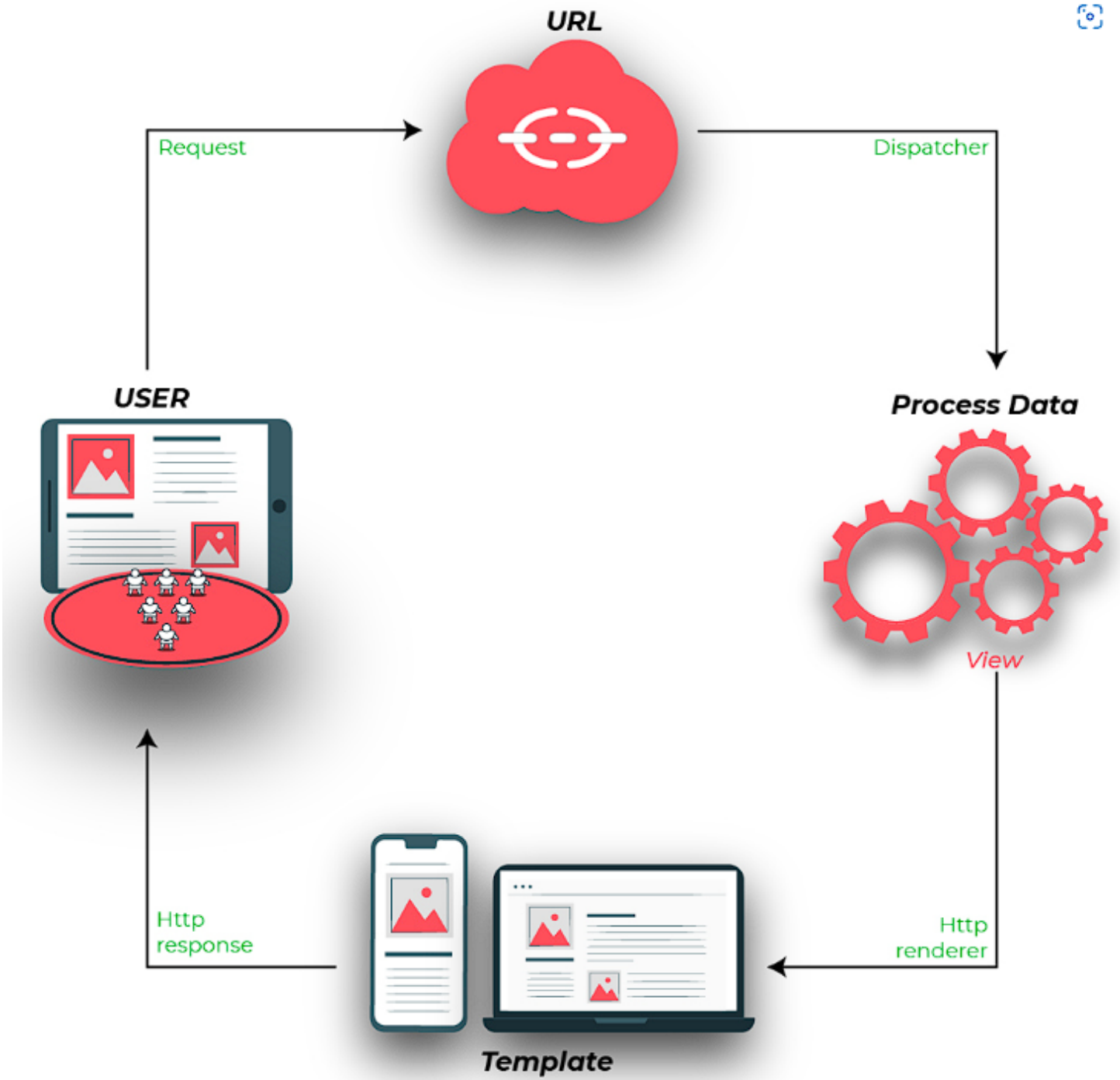
Django Fundamentals

- Django Architecture
- Django MTV Architecture
- Django Project Layout and File Structure
- Django Models
- **Django Views**
- Django Templates
- Django URLs and URL Conf

Django Views

Django Views

- Django Views are one of the vital participants of MVT Structure of Django.
- A view function is a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, anything that a web browser can display.
- Django views are part of the user interface — they usually render the HTML/CSS/Javascript in your Template files into what you see in your browser when you render a web page.


URL
Request
Dispatcher
USER
Process Data
View
Http
response
Http
renderer
Template

How to create and use a Django view

- First, we import the class **HttpResponse** from the django.http module, along with Python's datetime library.

```
# import Http Response from django
from django.http import HttpResponse
# get datetime
import datetime


# create a function
def geeks_view(request):
    # fetch date and time
    now = datetime.datetime.now()
    # convert to string
    html = "Time is {}".format(now)
    # return response
    return HttpResponse(html)
```

- Next, we define a function called geeks_view. This is the view function. Each view function takes an HttpRequest object as its first parameter, which is typically named request.

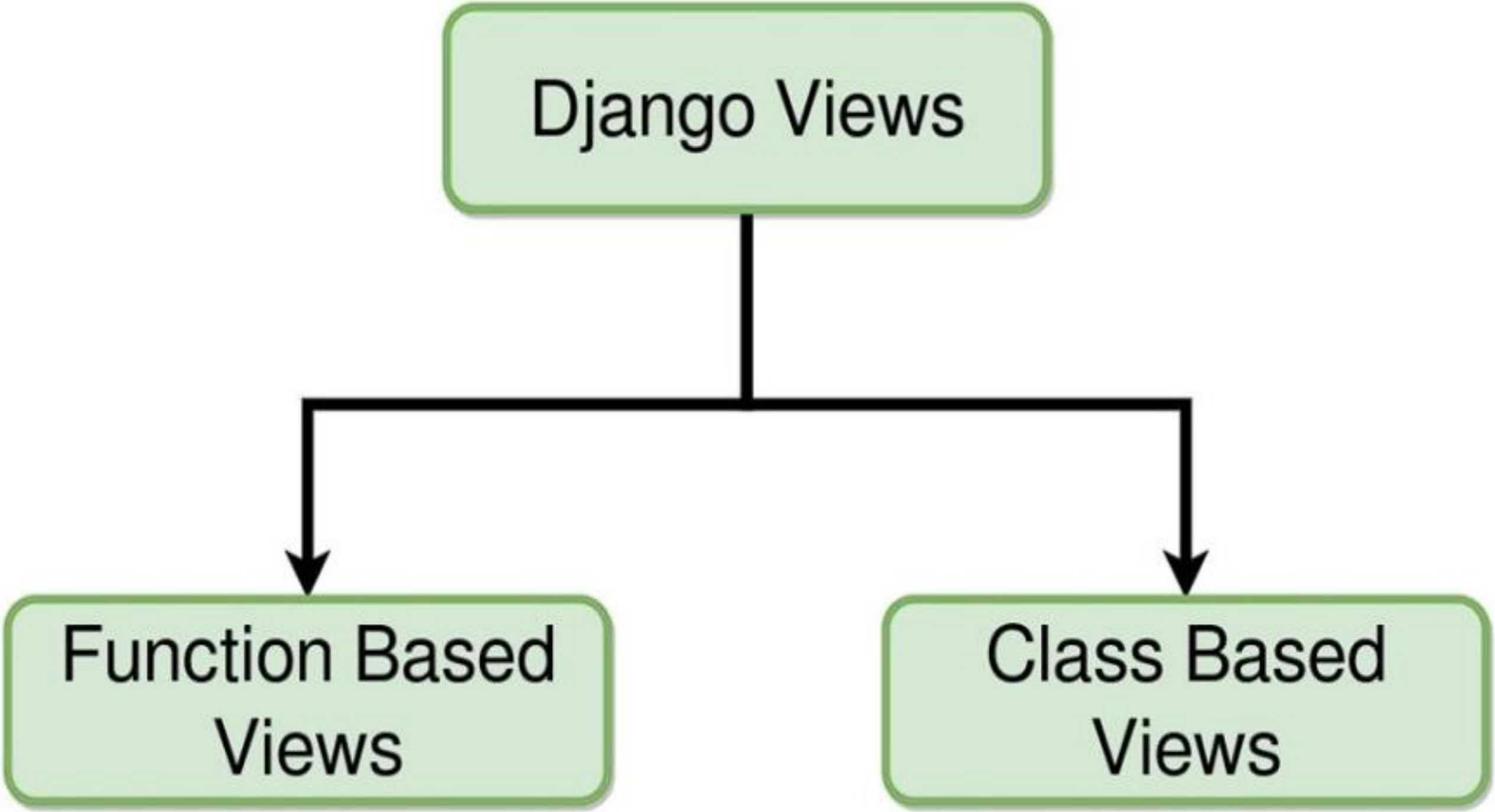
```
# import Http Response from django
from django.http import HttpResponse
# get datetime
import datetime


# create a function
def geeks_view(request):
    # fetch date and time
    now = datetime.datetime.now()
    # convert to string
    html = "Time is {}".format(now)
    # return response
    return HttpResponse(html)
```

- The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an **HttpResponse** object.

```python
# import Http Response from django
from django.http import HttpResponse
# get datetime
import datetime


# create a function
def geeks_view(request):
    # fetch date and time
    now = datetime.datetime.now()
    # convert to string
    html = "Time is {}".format(now)
    # return response
    return HttpResponse(html)
```

```python
from django.urls import path

# importing views from views..py
from .views import geeks_view

urlpatterns = [
    path('', geeks_view),
]
```

Time is 2020-01-23 08:36:24.142498


Django Views
Function Based Views
Class Based Views

Django Views

Function Based Views

- Function based views are writer using a function in python which receives as an argument HttpRequest object and returns an HttpResponse Object.
- Function based views are generally divided into 4 basic strategies, i.e., CRUD (Create, Retrieve, Update, Delete). CRUD is the base of any framework one is using for development.

Django Views

Class Based Views

- Class-based views provide an alternative way to implement views as Python objects instead of functions.
- 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.

Django Views

Class-based views are simpler and efficient to manage than function-based views.

A function-based view with tons of lines of code can be converted into class-based views with few lines only.

This is where Object-Oriented Programming comes into impact.

Django Fundamentals

- Django Architecture
- Django MTV Architecture
- Django Project Layout and File Structure
- Django Models
- Django Views
- **Django Templates**
- Django URLs and URL Conf

Django Templates

Django Templates

- Templates are the third and most important part of Django's MVT Structure.
- A template in Django is basically written in HTML, CSS, and Javascript in a .html file.
- Django framework efficiently handles and generates dynamically HTML web pages that are visible to the end-user. Django mainly functions with a backend so, in order to provide a frontend and provide a layout to our website, we use templates.

Django Templates

There are two methods of adding the template to our website depending on our needs.

- We can use a single template directory which will be spread over the entire project.
- For each app of our project (App-level templates), we can create a different template directory. Generally used in big projects or in case we want to provide a different layout to each component of our webpage.

Django Template Language

- This is one of the most important facilities provided by Django Templates.

A Django template is a text document or a Python string marked-up using the Django template language. Some constructs are recognized and interpreted by the template engine.

The main characteristics of Django Template language:

- Variables
- Tags
- Filters
- Comments

Django Template Language

Variables

- Variables output a value from the context, which is a dict-like object mapping keys to values. The context object we sent from the view can be accessed in the template using variables of Django Template.

Syntax: {{ variable_name }}

Django Templates - GeeksforGeeks

```
My first name is {{ first_name }}. My last name is {{ last_name }}.
```

With a context of {'first_name': 'Naveen', 'last_name': 'Arora'}, this template renders to:

```
My first name is Naveen. My last name is Arora.
```

Django Template Language

Django Templates - GeeksforGeeks

Tags

- Tags provide arbitrary logic in the rendering process. For example, a tag can output content, serve as a control structure e.g. an “if” statement or a “for” loop, grab content from a database, or even enable access to other template tags.

Syntax: {% tag_name %}

```
{% csrf_token %}
```

Most tags accept arguments, for example :

```
{% cycle 'odd' 'even' %}
```

Commonly used Tags		
Comment	cycle	extends
if	for loop	for ... empty loop
Boolean Operators	firstof	include
lorem	now	url

Django Template Language

Filters

Django Template Engine provides filters that are used to transform the values of variables and tag arguments. Tags can't modify the value of a variable whereas filters can be used for incrementing the value of a variable or modifying it to one's own need.

Syntax: {{ variable_name | filter_name }}

Filters can be "chained." The output of one filter is applied to the next. {{ text|escape|linebreaks }} is a common idiom for escaping text contents, then converting line breaks to <p> tags.

Example

```
{{ value | length }}
```

If value is **['a', 'b', 'c', 'd']**, the output will be **4**.

Django Templates - GeeksforGeeks

Django Template Language

Comments

Template ignores everything between {% comment %} and {% end comment %}. An optional note may be inserted in the first tag. For example, this is useful when commenting out code for documenting why the code was disabled.

Syntax:

{% comment 'comment_name' %}
{% endcomment %}

Example :

```
{% comment "Optional note" %}
        Commented out text with {{ create_date|date:"c" }}
{% endcomment %}
```

Django Templates - GeeksforGeeks

Django Fundamentals

- Django Architecture
- Django MTV Architecture
- Django Project Layout and File Structure
- Django Models
- Django Views
- Django Templates
- **Django URLs and URL Conf**

Django URLs and URL Conf

Django URLs and URL Conf

- URL stands for Uniform Resource Locator. It is the address used by your server to search for the right webpage.

- All the servers take the URL which you searched in the browser and via that server, provides you the correct result and if they don’t find anything matching the URL, it will show 404 FILE NOT FOUND ERROR.

Django URLs and URL Conf

- Django interprets URLs in a rather different way, the URLs in Django are in the format of regular expressions, which are easily readable by humans than the traditional URLs of PHP frameworks.

A **Regular Expression** also called RegEx is a format for search pattern which is much cleaner and easy to read for the humans and is very logical.

Django URLs and URL Conf

- The **urls.py** file in Django is like the address book of your Django website. It stores all the web addresses for your website. It connects that to some view component or any other urls-conf file for a certain application.

- A **URLconf** is similar to a table of contents for our Django-powered web site. It’s a mapping between URL patterns and the view functions that need to be called for those URLs.

Applications???

Any questions???

REFERENCES:

- Django Tutorial. (2022). https://data-flair.training/blogs/django-tutorial/
- Django Tutorial. (2020). https://www.geeksforgeeks.org/django-tutorial/
- Django Tutorial. (2022). https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction
- Django Tutorial. (n.d.). https://www.tutorialspoint.com/django/index.htm
- Django Tutorial. (n.d.). https://www.w3schools.com/django/index.php
- Getting started with Django. (n.d.). https://www.djangoproject.com/start/