**1) Differences between List vs Sets?**

A) **List:** Lists are just like dynamic sized arrays, declared in other languages (vector in C++ and Array List in Java).

Lists need not be homogeneous always which makes it the most powerful tool in Python.

The main characteristics of lists are –

1) The list is a datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.

2) List are mutable .i.e. it can be converted into another data type and can store any data element in it.

3) List can store any type of element.

**Set:** In Python, Set is an unordered collection of data type that is iterable, mutable, and has no duplicate elements.

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for

Checking whether a specific element is contained in the set. The main characteristics of set are –

1) Sets are an unordered collection of elements or unintended collection of items in python.

2) Here the order in which the elements are added into the set is not fixed, it can change frequently.

3) It is defined under curly braces {}

4) Sets are mutable, however, only immutable objects can be stored in it.

List Set

Lists is Mutable Set is Mutable

It is ordered collection of items it is Unordered collection of items

Items in list can be replaced or changed Items in set cannot be changed or replaced

2) What is Memory Management in python?

A) Memory management in Python involves a private heap containing all Python objects and data structures.at the lowest level, a raw memory allocator ensures that there is enough room in the private heap for storing all Python-related data by interacting with the memory manager of the operating system.

Memory allocation can be defined as allocating a block of space in the computer memory to a program. In Python memory allocation and deallocation method is automatic as the Python developers created a garbage collector for Python so that the user does not have to do manual garbage collection.

3) What is Pickling and unpickling?

Python pickle module is used for serializing and de-serializing python object structures. The process to converts any kind of python objects (list, dict, etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshalling.

We can converts the byte stream (generated through pickling) back into python objects by a process called as unpickling.

Why Pickle?: In real world scenario, the use pickling and unpickling are widespread as they allow us to easily transfer data from one server/system to another and then store it in a file or database.

Precaution: It is advisable not to unpickle data received from an untrusted source as they may pose security threat. However, the pickle module has no way of knowing or raise alarm while pickling malicious data.

Only after importing pickle module we can do pickling and unpickling. Importing pickle can be done using the following command −

import pickle

**4)** What is Method overloading in python?

Like other languages (for example, method overloading in C++) do, python does not support method overloading by default. But there are different ways to achieve method overloading in Python.

The problem with method overloading in Python is that we may overload the methods but can only use the latest defined method This process of calling the same method in different ways is called method overloading. Two methods cannot have the same name in Python; hence method overloading is a feature that allows the same operator to have different meanings.

Overloading is a method or operator that can do different functionalities with the same name.

\* In the process of method overloading, all the functions or methods must contain the  
same name with different signatures.  
\* It comes under compile-time polymorphism.Compile-time polymorphism allows us to have more than one method share the  
same name with different signatures and different return types.  
\* One may or may not require inheritance in the case of method overloading.  
\* One can perform method overloading between methods that reside within a class.  
\* We use method overloading for adding more to the behavior of all the methods.  
\* A user won’t need more than one class for implementing method overloading.

5).**Class methods and instance methods?**

Instance Methods:

The first method on MyClass, called method, is a regular instance method. That’s the basic,

no-frills method type you’ll use most of the time. You can see the method takes one parameter, self, which points to an instance of MyClass when the method is called (but of course instance methods can accept more than just one parameter).

Through the self parameter, instance methods can freely access attributes and other methods on the same object. This gives them a lot of power when it comes to modifying an object’s state.

Not only can they modify object state, instance methods can also access the class itself through the self.\_\_class\_\_ attribute. This means instance methods can also modify class state.

Class Methods:

Let’s compare that to the second method, MyClass.classmethod. I marked this method with a @classmethod decorator to flag it as a class method.

Instead of accepting a self parameter, class methods take a cls parameter that points to the class—and not the object instance—when the method is called.

Because the class method only has access to this cls argument, it can’t modify object instance state. That would require access to self. However, class methods can still modify class state that applies across all instances of the class.

**6).Instance variables can access class variables?**

A Python class variable is shared by all object instances of a class. Class variables are declared when a class is being constructed. They are not defined inside any methods of a class.

Because a class variable is shared by instances of a class, the Python class owns the variable. As a result, all instances of the class will be able to access that variable. Class variables are shared by all instances that access the class.

**7).Append and extend in python?**

**Append**:

append():Adds its argument as a single element to the end of a list. The length of the list increases by one.

list.append(1)

**Extend:**

extend(): Iterates over its argument and adding each element to the list and extending the list.

The length of the list increases by number of elements in it’s argument.

list.extend(1,2,3)

**8).Decorators in python?**

Decorators are a very powerful and useful tool in Python since it allows programmers to modify the behaviour of function or class.

Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it.

Python has an interesting feature called decorators to add functionality to an existing code.

This is also called metaprogramming because a part of the program tries to modify another part of the program at compile time.

**9).Call by refrence and call by value in python?**

While calling a function, we pass values of variables to it. Such functions are known as “Call By Values”.

While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References.

**10).Shallow copy and Deep copy**

Shallow Copy:

A shallow copy is a copy of an object that stores the reference of the original elements.

It creates the new collection object and then occupying it with reference to the child objects found in the original.

It makes copies of the nested objects' reference and doesn't create a copy of the nested objects.

So if we make any changes to the copy of the object will reflect in the original object. We will use the copy() function to implement it.

Deep Copy:

A deep copy is a process where we create a new object and add copy elements recursively. We will use the deecopy() method which present in copy module.

The independent copy is created of original object and its entire object

**11)Circular Import in python?**

Circular Dependecy:

When two or more modules rely on each other, this is referred to as a circular dependency.

This is due to the fact that each module is defined in terms of the others.

Circular Import:

Circular importing is a form of circular dependency that is created with the import statement in Python.

**12).Avoid circular import in python?**

Just change your test file to another name such as aaa.py , other than request.py . Do not use names that are already used by other libs. This works because you can import file b without Python executing any of the import statements in file b , and thus you elude any circular imports.

**13).Generator and Iterator in python?**

Iterator:

An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc.

Iterators are implemented using a class and a local variable for iterating is not required here, It follows lazy evaluation

where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically

which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and

when we are using a large dataset then, wastage of RAM space will be reduced the need to load the entire dataset at the same time will not be there.

**Using an iterator:**

iter() keyword is used to create an iterator containing an iterable object.

next() keyword is used to call the next element in the iterable object.

After the iterable object is completed, to use them again reassign them to the same object.

**Generators:**

It is another way of creating iterators in a simple way where it uses the keyword “yield” instead of returning it in a defined function.

Generators are implemented using a function. Just as iterators, generators also follow lazy evaluation.

Here, the yield function returns the data without affecting or exiting the function.

It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won’t store the entire

sequence in the memory.

**14).\_init.py\_ uses in python?**

The \_\_init\_\_.py file makes Python treat directories containing it as modules.

Furthermore, this is the first file to be loaded in a module, so you can use it to execute code that you want to run each time a module is loaded, or specify the submodules to be exported.

**15).Method resolution order in python?**

Method Resolution Order :

Method Resolution Order(MRO) it denotes the way a programming language resolves a method or attribute. Python supports classes inheriting from other classes.

The class being inherited is called the Parent or Superclass, while the class that inherits is called the Child or Subclass.

In python, method resolution order defines the order in which the base classes are searched when executing a method.

First, the method or attribute is searched within a class and then it follows the order we specified while inheriting.

This order is also called Linearization of a class and set of rules are called MRO(Method Resolution Order). While inheriting from another class,

the interpreter needs a way to resolve the methods that are being called via an instance. Thus we need the method resolution order.

**16).how import works in python?**

When a module is imported, Python runs all of the code in the module file. When a package is imported, Python runs all of the code in the package’s \_\_init\_\_.py file, if such a file exists. All of the objects defined in the module or the package’s \_\_init\_\_.py file are made available to the importer.

**17)**Differences Between Django vs Flask

Django and Flask are two of the most popular web framework for Python. Flask showed up as an alternative to Django, as designers needed to have more flexibility that would permit them to decide how they want to implement things, while on the other hand, Django does not permits alteration of their modules to such degree. Flask is truly so straightforward and direct that working in it permits an experienced Python designer making ventures inside truly tight timeframes.

 Flask and Django use cases: Flask is extraordinary for building basic locales with inactive substance, like blogs; it gives all the usefulness you would like and permits for customization to a tremendous degree whereas Django is incredible for building complex destinations with energetic substance, with adaptability in intellect; enormous ventures that require out-of-the-box arrangements can be sent truly quick. Flask is idealized for creating straightforward web apps Django permits sending more complicated web apps quicker, as its modules are preconfigured to supply quick app improvement and arrangement.

| Django | Flask |
| --- | --- |
| Django could be a Python-based free, open source system which takes after the MVT(model view Template) approach of structural design | Flask could be a Python-based smaller scale system without any set of specific instruments or outside libraries. It too doesn’t have a database layer or arrangements for shape approval and makes utilize of expansions. |
| Urls.py is utilized to set the association properties and demands are handled by the primary coordinating see of regex list | URI is most regularly than not set by the see decorator and centralized setup is additionally conceivable. Some time recently the designs are coordinated with the URIs, the last mentioned is sorted in a default arrange |
| Doesn’t exclude setting flexibility | It is accepted that all the conceivable stages to organize a Flask code rises to the applications number show in flask as of now |
| Extend Layout is Conventional extend structure | Extend Layout is Arbitrary structure |
| Django gives an all-inclusive encounter: you get an admin board, database interfacing, an ORM, and registry structure for your apps and ventures out of the box. | Flask gives straightforwardness, adaptability and fine-grained control. It is unopinionated |

# 18)How to Create a basic API using Django RestFramework ?

Django REST Framework is a wrapper over default Django Framework, basically used to create APIs of various kinds. There are three stages before creating a API through REST framework, Converting a Model’s data to JSON/XML format (Serialization), Rendering this data to the view, Creating a URL for mapping to the viewset.  
This article revolves around how to create a basic API using Django REST Framework. It assumes you are familiar with Django basics – [Django tutorial](https://www.geeksforgeeks.org/django-tutorial/). Also, installation of [Django REST Framework](https://geeksforgeeks.org/django-rest-framework-installation/). Assuming you have created a project named geeksforgeeks with Django, let’s initiate Django REST Framework.

Steps

* [Add rest\_framework to INSTALLED\_APPS](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/#add_rest_framework)
* [Create a app and model](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/#create-app)
* [Serialization](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/#serialization)
* [Creating a viewset](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/#create-viewset)
* [Define URLs of API](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/#define-urls)
* [Run server and check API](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/#runserver)

# 19)Flask and Databases

To add database functionality to a Flask app, we will use SQLAlchemy.

SQLAlchemy is a Python SQL toolkit and object relational mapper (ORM) that enables Python to communicate with the SQL database system you prefer: MySQL, PostgreSQL, SQLite, and others. An ORM converts data between incompatible systems (object structure in Python, table structure in SQL database). SQLAlchemy is basically a bridge between Python and a SQL database.

Flask-SQLAlchemy is an extension for Flask that adds SQLAlchemy to your Flask app.

* [SQLAlchemy documentation](https://www.sqlalchemy.org/)
* [Flask-SQLAlchemy documentation](https://flask-sqlalchemy.palletsprojects.com/)
* [Code for this chapter](https://github.com/macloo/python-adv-web-apps/tree/master/python_code_examples/flask/databases)

## Setup: Flask-SQLAlchemy

We will install the Flask-SQLAlchemy extension to enable us to work with a SQL database in Flask. There are many extensions for Flask; each one adds a different set of functions and capabilities. See the [list of Flask extensions](https://flask.palletsprojects.com/en/1.1.x/extensions/) for more.

In Terminal, change into your Flask projects folder and activate your virtual environment there. Then install the extension at the command prompt — where you see $ (Mac) or C:\Users\yourname> (Windows) —

pip install flask-sqlalchemy

We will use SQLite for database examples here. Although it’s not necessary to use SQLAlchemy to interact with a SQLite database, learning to use SQLAlchemy gives you a skill set that can be applied to any SQL database system.

SQLAlchemy can bridge between Python and various different SQL database systems — some of which need an additional module, or library, to be installed. SQLite does not require an additional module — the sqlite3 module is included in Python 3.x.

# 20)Connect Flask to a Database with Flask-SQLAlchemy

Flask is a micro web framework written in python. Micro-framework is normally a framework with little to no dependencies on external libraries. Though being a micro framework almost everything can be implemented using python libraries and other dependencies when and as required.

In this article, we will be building a Flask application that takes data in a form from the user and then displays it on another page on the website. We can also delete the data. We won’t focus on the front-end part rather we will be just coding the backend for the web application.

## Installing Flask

In any directory where you feel comfortable create a folder and open the command line in the directory. Create a python virtual environment using the command below.

python -m venv <name>

Once the command is done running activate the virtual environment using the command below.

<name>\scripts\activate

Now, install Flask using pip(package installer for python). Simply run the command below.

pip install Flask

## Creating app.py

Once the installation is done create a file name app.py and open it in your favorite editor. To check whether Flask has been properly installed you can run the following code.

# 21)Token-Based Authentication With Flask

## Introduction

[JSON Web Tokens](https://jwt.io/) (or JWTs) provide a means of transmitting information from the client to the server in a [stateless](https://en.wikipedia.org/wiki/Stateless_protocol), secure way.

On the server, JWTs are generated by signing user information via a secret key, which are then securely stored on the client. This form of auth works well with modern, single page applications. For more on this, along with the pros and cons of using JWTs vs. session and cookie-based auth,

Keep in mind that since a JWT is [signed rather than encrypted](http://stackoverflow.com/questions/454048/what-is-the-difference-between-encrypting-and-signing-in-asymmetric-encryption) it should never contain sensitive information like a user’s password.

# 22)The difference between get and filter methods in Django

get(\*\*kwargs)

Returns the object matching the given lookup parameters, which should be in the format described in Field lookups.

get() raises Multiple Objects Returned if more than one object was found. The Multiple

Objects Returned exception is an attribute of the model class.

get() raises a DoesNotExist exception if an object wasn't found for the given parameters. This exception is also an attribute of the model class.

filter(\*\*kwargs)

Returns a new QuerySet containing objects that match the given lookup parameters.

Basically use get() when you want to get a single unique object, and filter() when you want to get all objects that match your lookup parameters.

# 23)User authentication in Django ?

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This document explains how things work.

## Overview

* The auth system consists of:
* Users
* Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
* Groups: A generic way of applying labels and permissions to more than one user.
* Messages: A simple way to queue messages for given users.
* Deprecated in Django 1.2:
* Deprecated since version 1.2: The Messages component of the auth system will be removed in Django 1.4.

## Installation

Authentication support is bundled as a Django application in django.contrib.auth. To install it, do the following:

1. Put 'django.contrib.auth' and 'django.contrib.contenttypes' in your [INSTALLED\_APPS](https://django.readthedocs.io/en/1.3.X/ref/settings.html#std:setting-INSTALLED_APPS) setting. (The [Permission](https://django.readthedocs.io/en/1.3.X/topics/auth.html#django.contrib.auth.models.Permission) model in [django.contrib.auth](https://django.readthedocs.io/en/1.3.X/topics/auth.html" \l "module-django.contrib.auth" \o "django.contrib.auth: Django's authentication framework.) depends on [django.contrib.contenttypes](https://django.readthedocs.io/en/1.3.X/ref/contrib/contenttypes.html" \l "module-django.contrib.contenttypes" \o "django.contrib.contenttypes: Provides generic interface to installed models.).)
2. Run the command manage.py syncdb.

Note that the default settings.py file created by [django-admin.py startproject](https://django.readthedocs.io/en/1.3.X/ref/django-admin.html#django-admin-startproject) includes 'django.contrib.auth' and 'django.contrib.contenttypes' in [INSTALLED\_APPS](https://django.readthedocs.io/en/1.3.X/ref/settings.html#std:setting-INSTALLED_APPS) for convenience. If your [INSTALLED\_APPS](https://django.readthedocs.io/en/1.3.X/ref/settings.html#std:setting-INSTALLED_APPS) already contains these apps, feel free to run [manage.py syncdb](https://django.readthedocs.io/en/1.3.X/ref/django-admin.html#django-admin-syncdb) again; you can run that command as many times as you’d like, and each time it’ll only install what’s needed.

The [syncdb](https://django.readthedocs.io/en/1.3.X/ref/django-admin.html" \l "django-admin-syncdb) command creates the necessary database tables, creates permission objects for all installed apps that need ‘em, and prompts you to create a superuser account the first time you run it.

Once you’ve taken those steps, that’s it.

## Users

class models.User

# 24)Authentication and Authorization

## Introduction to Security

Authentication is the mechanism whereby systems may securely identify their users. Eve supports several authentication schemes: Basic Authentication, Token Authentication, HMAC Authentication. [OAuth2 integration](https://docs.python-eve.org/en/stable/authentication.html#oauth2-integration) is easily accomplished.

Authorization is the mechanism by which a system determines what level of access a particular (authenticated) user should have access to resources controlled by the system. In Eve, you can restrict access to all API endpoints, or only some of them. You can protect some HTTP verbs while leaving others open. For example, you can allow public read-only access while leaving item creation and edition restricted to authorized users only. You can also allow GET access for certain requests and POST access for others by checking the method parameter. There is also support for role-based access control.

Security is one of those areas where customization is very important. This is why you are provided with a handful of base authentication classes. They implement the basic authentication mechanism and must be subclassed in order to implement authorization logic. No matter which authentication scheme you pick the only thing that you need to do in your subclass is override the check\_auth() method.

check authorization in django

# 25)User authentication in Django

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This section of the documentation explains how the default implementation works out of the box, as well as how to [extend and customize](https://docs.djangoproject.com/en/4.0/topics/auth/customizing/) it to suit your project’s needs.

## Overview

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

The auth system consists of:

* Users
* Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
* Groups: A generic way of applying labels and permissions to more than one user.
* A configurable password hashing system
* Forms and view tools for logging in users, or restricting content
* A pluggable backend system

The authentication system in Django aims to be very generic and doesn’t provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages:

* Password strength checking
* Throttling of login attempts
* Authentication against third-parties (OAuth, for example)
* Object-level permissions

## Installation

Authentication support is bundled as a Django contrib module in django.contrib.auth. By default, the required configuration is already included in the settings.py generated by [django-admin startproject](https://docs.djangoproject.com/en/4.0/ref/django-admin/" \l "django-admin-startproject), these consist of two items listed in your [INSTALLED\_APPS](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-INSTALLED_APPS) setting:

1. 'django.contrib.auth' contains the core of the authentication framework, and its default models.
2. 'django.contrib.contenttypes' is the Django [content type system](https://docs.djangoproject.com/en/4.0/ref/contrib/contenttypes/), which allows permissions to be associated with models you create.

and these items in your [MIDDLEWARE](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-MIDDLEWARE) setting:

1. [SessionMiddleware](https://docs.djangoproject.com/en/4.0/ref/middleware/#django.contrib.sessions.middleware.SessionMiddleware) manages [sessions](https://docs.djangoproject.com/en/4.0/topics/http/sessions/) across requests.
2. [AuthenticationMiddleware](https://docs.djangoproject.com/en/4.0/ref/middleware/#django.contrib.auth.middleware.AuthenticationMiddleware) associates users with requests using sessions.

With these settings in place, running the command manage.py migrate creates the necessary database tables for auth related models and permissions for any models defined in your installed apps.

Token based in rest api

#### Obtaining the API token

To get the API token for a user, an HTTP POST request should be sent to the Token resource. In the post body, username and password are specified in JSON format, and the response body contains a token key with an actual API Token as the value

**26)serlizers in Django**

# Serializers – Django REST Framework

Serializers in Django REST Framework are responsible for converting objects into data types understandable by javascript and front-end frameworks. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data. The serializers in REST framework work very similarly to Django’s Form and ModelForm classes. The two major serializers that are most popularly used are [ModelSerializer](https://www.geeksforgeeks.org/modelserializer-in-serializers-django-rest-framework/) and [HyperLinkedModelSerialzer](https://www.geeksforgeeks.org/hyperlinkedmodelserializer-in-serializers-django-rest-framework/).  
This article revolves around how to use serializers from scratch in Django REST Framework to advanced serializer fields and arguments. It assumes one is familiar with [How to start a project with Django REST Framework ?](https://www.geeksforgeeks.org/how-to-create-a-basic-api-using-django-rest-framework/)

* [Creating and Using Serializers](https://www.geeksforgeeks.org/serializers-django-rest-framework/#create-use-serializer)
* [ModelSerializer](https://www.geeksforgeeks.org/serializers-django-rest-framework/#modelserializer)
* [HyperLinkedModelSerializer](https://www.geeksforgeeks.org/serializers-django-rest-framework/#hyperlinkedmodelserializer)
* [Serializer Fields](https://www.geeksforgeeks.org/serializers-django-rest-framework/#serializer-fields)
* [Core arguments in serializer fields](https://www.geeksforgeeks.org/serializers-django-rest-framework/#core-arguments)

# 27)How to Create and Use Signals in Django ?

Signals are used to perform any action on modification of a model instance. The signals are utilities that help us to connect events with actions. We can develop a function that will run when a signal calls it. In other words, Signals are used to perform some action on modification/creation of a particular entry in Database. For example, One would want to create a profile instance, as soon as a new user instance is created in Database

There are 3 types of signal.

1. pre\_save/post\_save: This signal  works before/after the method save().
2. pre\_delete/post\_delete: This signal  works before after delete a model’s instance (method delete()) this signal is thrown.
3. pre\_init/post\_init: This signal is thrown before/after instantiating a model (\_\_init\_\_() method).

### How to use Signals ion Django?

For example, if we want to create a profile of a user as soon as the user is created using post\_save signal

Models.py

28)How-to Use Custom View @decorators in Django?

## Prevent unauthorised users from accessing views and provide feedback using the @decorator method, such as for RECAPTCHA protected form submissions

 Decorators are a way to restrict access to views based on the request method or control caching behaviour. This is particularly useful when you want to separate logged-in users from unauthenticated users or create an admin page that only privileged users can access.

Django has several built-in decorators, but their main issue is that they do not provide user feedback. Django also has a built-in messages framework that uses the SESSION\_COOKIE to store messages and display them after submitting or reloading webpages as a means of user feedback. We’re going to incorporate this in our custom decorators. We’re also going to use Google’s RECAPTCHA to protect our form submissions from spam and abuse and include this in a decorator.

29)Acid in Django?

What are the ACID properties of a transaction?

In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction: **atomicity, consistency, isolation, and durability**. All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

ACID properties of transactions

In the context of transaction processing, the acronym *ACID* refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.

**Atomicity**

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

**Consistency**

Data is in a consistent state when a transaction starts and when it ends.

For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

**Isolation**

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.

For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

**Durability**

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

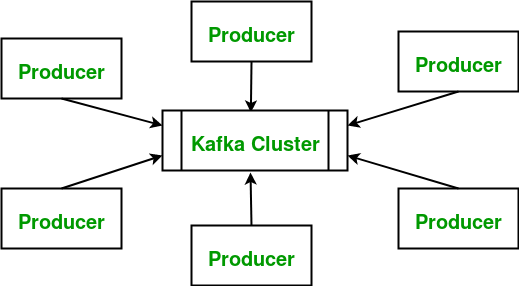
For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

# 30)Apache Kafka | Introduction

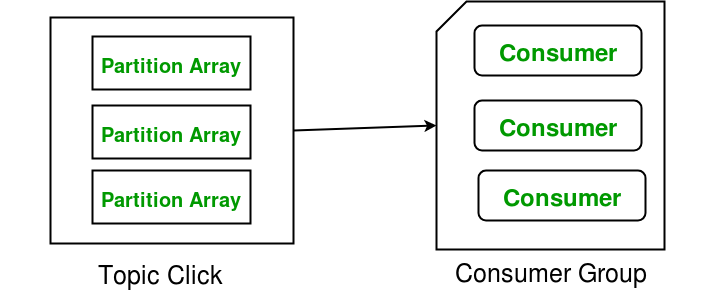
Apache Kafka is a publish-subscribe messaging system. A messaging system let you send messages between processes, applications, and servers. Broadly Speaking, Apache Kafka is a software where topics (A topic might be a category) can be defined and further processed. Applications may connect to this system and transfer a message onto the topic. A message can include any kind of information ,from any event on your Personal blog or can be a very simple text message that would trigger any other event.

**Kafka Broker**

A Kafka cluster usually consists of one or more servers (called as kafka brokers), which are running Kafka over them. **Producers** are processes that publish data (push messages over trigger) into Kafka topics within the specified broker. A consumer of topics pulls messages off a Kafka topic.



**Kafka Topic :**A Topic basically is a category or a feed name to which messages are stored and published during operations. Messages are mostly byte arrays that can store any object in any format. Yes , That’s the best thing about kafka. Any object can be stored as byte array. Also, as we discussed before, all Kafka messages are organized into topics. If you wish to send a message you send it to a specific topic and if you wish to read a message you read it from a specific topic.  
**Consumers and consumer groups :**Consumers can always read messages starting from a specific offset and are allowed to read from any offset point they choose in between. This allows consumers to join the cluster at any point in time.This makes functioning and working really smooth.  
**Partitions allow you to parallelise a topic by splitting the data in a particular topic across multiple brokers.**



**Apache Kafka and Important Server concepts**

* **Topic partition**: Kafka topics are divided into a number of partitions, which allows you to split data across multiple brokers.
* **Consumer Group**: A consumer group includes the set of consumer processes that are subscribing to a specific topic.
* **Node**: A node is a single computer in the Apache Kafka cluster.
* **Replicas:** A replica of a partition is a “backup” of a partition. Replicas never read or write data. They are used to prevent data loss.
* **Producer**: Application that sends the messages.
* **Consumer**: Application that receives the messages.

**Real time Applications**

* **Twitter**: Registered users can read and post tweets, but unregistered users can only read tweets. Twitter uses Storm-Kafka as a part of their stream processing infrastructure.
* **LinkedIn**: Apache Kafka is used at LinkedIn for activity stream data and operational metrics. Kafka messaging system helps LinkedIn with various products like LinkedIn Newsfeed, LinkedIn Today for online message consumption and in addition to offline analytics systems like Hadoop.
* **Netflix**: Netflix is an American multinational provider of on-demand Internet streaming media. Netflix uses Kafka for real-time monitoring and event processing.
* **Box**: At Box, Kafka is used for the production analytics pipeline & real time monitoring infrastructure.

**31)Micro services in flask**

Flask is a lightweight ‘web application framework’ written in Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. Flask doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to work with. Flask is super useful for building Microservices. You can utilize any number of its built-in extensions to design and deploy Microservices at high velocity. It will help you to get your offerings to market fast.

**32)deployemnt in projects**

## **What Is Deployment Planning?**

In general, deployment refers to moving an object to a place where some action can be performed on it. **In the case of software development, deployment means making an application ready for delivery.**

 First, the project manager creates a deployment plan. After that, the project team reviews it, before deployment.

Here’s the basic structure I prefer in deployment planning.

1. Preliminaries + briefing + reserving resources
   * Check point, Go/NoGo
2. Timespan from 5 to 1 days before deployment
   * Check point, Go/NoGo
3. Cut over
   * Check point, Go/NoGo
4. Production test/authentication
   * Check point, Go/NoGo
5. First day in production use
   * Check point
6. Second day in production use
   * Check point

The plan will include hundreds of activities for dozens of people, so communication and clear task list on what to do and when, are critical.

**33)Max salary of each department in sql**

# SQL Query to Find the Highest Salary of Each Department

Structured Query Language or SQL is a standard Database language that is used to create, maintain and retrieve the data from relational databases like MySQL, Oracle, etc. In this article, we will be using the**Microsoft SQL Server**.

Here we are going to see how to get the highest salary of each department. Here, we will first create a database named “geeks” then we will create a table “department” in that database. After, that we will execute our query on that table.

# 34)Indexing in sql

# Indexes are used to speed-up query process in SQL Server, resulting in high performance. They are similar to textbook indexes. In textbooks, if you need to go to a particular chapter, you go to the index, find the page number of the chapter and go directly to that page. Without indexes, the process of finding your desired chapter would have been very slow.

The same applies to indexes in databases. Without indexes, a DBMS has to go through all the records in the table in order to retrieve the desired results. This process is called table-scanning and is extremely slow. On the other hand, if you create indexes, the database goes to that index first and then retrieves the corresponding table records directly.

There are two types of Indexes in SQL Server:

1. Clustered Index
2. Non-Clustered Index

35)Instance Variable

# Different ways to access Instance Variable in Python ?

**Instance attributes** are those attributes that are not shared by objects. Every object has its own copy of the instance attribute i.e. for every object, instance attribute is different.

**There are two ways to access the instance variable of class:**

* Within the class by using [self](https://www.geeksforgeeks.org/self-in-python-class/) and object reference.
* Using [getattr()](http://hello%20my%20name%20is:%20harry%20my%20roll%20number%20is:%201001%20HARRY/" \t "_blank) method

**36)Static Variable**

# Class or Static Variables in Python

All objects share class or static variables. An instance or non-static variables are different for different objects (every object has a copy). For example, let a Computer Science Student be represented by class **CSStudent**. The class may have a static variable whose value is “cse” for all objects. And class may also have non-static members like **name** and **roll**. In [C++](https://www.geeksforgeeks.org/c-plus-plus/) and [Java](https://www.geeksforgeeks.org/java/), we can use static keywords to make a variable a class variable. The variables which don’t have a preceding static keyword are instance variables. See [this](https://www.geeksforgeeks.org/static-keyword-in-java/) for the Java example and [this](https://www.geeksforgeeks.org/stati/) for the C++ example.  
The **Python** approach is simple; it doesn’t require a static keyword.

*All variables which are assigned a value in the class declaration are class variables. And variables that are assigned values inside methods are instance variables.*

# 37)template in djagno dashboard

The [Django template language](https://docs.djangoproject.com/en/4.0/ref/templates/language/) is Django’s own template system. Until Django 1.8 it was the only built-in option available. It’s a good template library even though it’s fairly opinionated and sports a few idiosyncrasies. If you don’t have a pressing reason to choose another backend, you should use the DTL, especially if you’re writing a pluggable application and you intend to distribute templates. Django’s contrib apps that include templates, like [django.contrib.admin](https://docs.djangoproject.com/en/4.0/ref/contrib/admin/), use the DTL.

For historical reasons, both the generic support for template engines and the implementation of the Django template language live in the **django.template** namespace.