What problem did the GIL solve for Python :

Python has something that no other language has that is a reference counter. With the help of the reference counter, we can count the total number of references that are made internally in python to assign a value to a data object. Due to this counter, we can count the references and when this count reaches to zero the variable or data object will be released automatically. This reference counter variable needed to be protected, because sometimes two threads increase or decrease its value simultaneously by doing that it may lead to memory leaked so in order to protect thread we add locks to all data structures that are shared across threads but sometimes by adding locks there exists a multiple locks which lead to another problem that is deadlock. In order to avoid memory leaked and deadlocks problem, we used single lock on the interpreter that is Global Interpreter Lock(GIL).
When do threads switch? Whenever a thread begins sleeping or awaiting network I/O, there is a chance for another thread to take the GIL and execute Python code. This is cooperative multitasking. CPython also has preemptive multitasking: If a thread runs uninterrupted for 1000 bytecode instructions in Python 2, or runs 15 milliseconds in Python 3, then it gives up the GIL and another thread may run. Think of this like time slicing in the olden days when we had many threads but one CPU.

Applications now a days are serving millions of users at the same time. With a lot of requests comes a wide range of problems concurrency problems is one of them. How to handle this?(ticket booking)

What do we have here:
1. When we use select_for_update on our queryset we tell the database to lock the object until the transaction is done(atomic).
2. Locking a row in the database requires a database transaction — we use Django's decorator transaction.atomic() to scope the transaction.
3. We use a classmethod instead of an instance method — to acquire the lock we need to tell the database to lock it. To achieve that we need to be the ones fetching the object from the database.
4. All the operations on the slot are executed within the database transaction.

What is an efficient way of inserting thousands of records into an SQLite table using Django?

in django 1.4, bulk_create was added, allowing you to create lists of your model objects and then commit them all at once.
Django's default behavior is to run in autocommit mode. Each query is immediately committed to the database, unless a transaction is active. See below for details.

Django uses transactions or savepoints automatically to guarantee the integrity of ORM operations that require multiple queries, especially delete() and update() queries.
Set ATOMIC_REQUESTS to True in the configuration of each database for which you want to enable this behavior.

It works like this. Before calling a view function, Django starts a transaction. If the response is produced without problems, Django commits the transaction. If the view produces an exception, Django rolls back the transaction.

You may perform subtransactions using savepoints in your view code, typically with the atomic() context manager.

Handle multiple requests python

Handling Multiple Requests on Flask - in Flask 4 worker processes will be able to handle more than 4 requests is that they will fire off threads to handle more and more requests. The actual request limit depends on HTTP server chosen, I/O, OS, hardware, network connection etc.

Handling multiple http request in Python, As there is a significant amount of networking I/O involved, threading should improve the overall performance significantly. You can try using a The GET method indicates that you're trying to get or retrieve data from a specified resource.

How Does Python Handle Multiple Web Requests ?

App Engine runs multiple instances of your application, and each instance has its own web server for handling requests. Any request can be routed to any instance, so consecutive requests from the same user are not necessarily sent to the same instance. An instance can handle multiple requests concurrently. The number of instances can be adjusted automatically as traffic changes

Web server vs Application Server

A web server accepts and fulfills requests from clients for static content (i.e., HTML pages, files, images, and videos) from a website. Web servers handle HTTP requests and responses only. An application server exposes business logic to the clients, which generates dynamic content.
Web servers: Nginx, Apache, Gunicorn
Application Servers: Tornado, Django,
For Web applications, an application server will be running in the same environment as its web server(s), and application servers are there to support the construction of dynamic pages and implement services like clustering, fail-over, and load-balancing, so developers can focus on implementing the business logic.

Key Differences Between Python 2 and Python 3

Print: In Python 2, "print" is treated as a statement rather than a function.
Integer Division: Python 2 treats numbers that you type without any digits after the decimal point as integers, which can lead to some unexpected results during division. For example, if you type the expression 3 / 2 in Python 2 code, the result of the evaluation will be 1, not 1.5 as you might expect. you would have to write 3.0 / 2.0 to tell Python that you want it to return a float,
List Comprehension Loop Variables: In previous versions of Python, giving the variable that is iterated over in a list comprehension the same name as a global variable could lead to the value of the global variable being changed — something you usually don't want. This irritating bug has been fixed in Python 3, so you can use a variable name you already used for the control variable in your list comprehension without worrying about it leaking out and messing with the values of the variables in the rest of your code.
Unicode Strings: Python 3 stores strings as Unicode by default, whereas Python 2 requires you to mark a string with a "u" if you want to store it as Unicode. Unicode strings are more versatile than ASCII strings, which are the Python 2 default, as they can store letters from foreign languages as well as emoji and the standard Roman letters and numerals. You can still label your Unicode strings with a "u"
Raising Exceptions: Python 3 requires different syntax for raising exceptions. If you want to output an error message to the user, you need to use the syntax:
raise IOError("your error message")

This syntax works in Python 2 as well. The following code works only in Python 2, not Python 3:

raise IOError, "your error message"

What is GIL?

Global Interpreter Lock The GIL is implemented because the memory management of CPython in not thread safe. If two threads access the same data, the GIL will add a lock on the data for one thread and the other thread has to wait for the lock to be released.
The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

Does python support multithreading?

Threading is Allowed in Python, the only problem is that the GIL will make sure that just one thread is executed at a time (no parallelism).

Threading in python is used to run multiple threads (tasks, function calls) at the same time. Note that this does not mean that they are executed on different CPUs. Python threads will NOT make your program faster if it already uses 100 % CPU time. In that case, you probably want to look into parallel programming.

So basically if you want to multi-thread the code to speed up calculation it won't speed it up as just one thread is executed at a time, but if you use it to interact with a database for example it will.

What's the difference between Python threading and multiprocessing?

With threading, concurrency is achieved using multiple threads, but due to the GIL only one thread can be running at a time. In multiprocessing, the original process is forked process into multiple child processes bypassing the GIL. Each child process will have a copy

How are Python multithreading and multiprocessing related?

Both multithreading and multiprocessing allow Python code to run concurrently. Only multiprocessing will allow your code to be truly parallel. However, if your code is IO-heavy (like HTTP requests), then multithreading will still probably speed up your code.of the entire program's memory.

What is Context Managers?

Context managers allow you to allocate and release resources precisely when you want to. The most widely used example of context managers is the with statement. Suppose you have two related operations which you'd like to execute as a pair, with a block of code in between. Context managers allow you to do specifically that. For example:

```
with open('some_file', 'w') as opened_file:
    opened_file.write('Hola!')
```

The above code opens the file, writes some data to it and then closes it. If an error occurs while writing the data to the file, it tries to close it. The above code is equivalent to:

```
file = open('some_file', 'w')
try:
    file.write('Hola!')
finally:
    file.close()
```

How do Context Managers work?

The logic behind context managers is actually pretty easy. Let's explore it by writing our own context manager. For an object to be recognized as a context manager, it must implement 2 methods: __enter__ and __exit__:

```
class File(object):
    def __init__(self, file_name, method):
        self.file_obj = open(file_name, method)
    def __enter__(self):
        return self.file_obj
    def __exit__(self, type, value, traceback):
        self.file_obj.close()

with File('demo.txt', 'w') as opened_file:
    opened_file.write('Hola!')
```

How to reduce response time of rest api in django

Web API performance: profiling Django REST framework, In order to time this we can wrap APIView.dispatch in a superclass that forces the response to render before returning it. Rather than use Python's Many to many PATCH not immediately showing up in response Django REST Framework 2.4.3 7 Compress JSON payload in Django for both request from client and response from server (REST API).

Guide to Django Performance Testing and Optimization, In this article, I will use these methods to reduce the response time of a query data through an API endpoint, we install Django REST Framework and define For better performance, you can ask the server to send only the fields you really need and get a partial response instead. To request a partial response, use the fields request parameter to specify the fields you want returned. You can use this parameter with any request that returns response data.

Throttling, When a developer chooses Python, Django, or Django Rest While both are valid ways to improve the overall response time of an API request Using the Response class simply provides a nicer interface for returning content-negotiated Web API responses, that can be rendered to multiple formats. Unless you want to heavily customize REST framework for some reason, you should always use an APIView class or @api

What is namespace?

A namespace is a system to have a unique name for each and every object in Python. An object might be a variable or a method. Python itself maintains a namespace in the form of a Python dictionary.
On the similar lines, Python interpreter understands what exact method or variable one is trying to point to in the code, depending upon the namespace. So, the division of the word itself gives little more information. Its Name (which means name, an unique identifier) + Space(which talks something related to scope). Here, a name might be of any Python method or variable and space depends upon the location from where is trying to access a variable or a method.

Types of namespaces :
these are built in namespaces. When a user creates a module, a global namespace gets created, later creation of local functions creates the local namespace. The built-in namespace encompasses global namespace and global namespace encompasses local namespace.

If Python is interpreted, what are .pyc files?

Python is an interpreted language, as opposed to a compiled one, though the distinction can be blurry because of the presence of the bytecode compiler.
This means that source files can be run directly without explicitly creating an executable which is then run.

What is __init__.py ?

Files name __init__.py are used to mark directories on disk as Python package directories.
If we wish to import code directly from a package as opposed to a module, we can use the __init__.py. In this approach, the __init__.py file houses the most visible functionality for the package. It pieces together the functionality from the sub-modules.

What do you mean by *args and **kwargs?

In cases when we don't know how many arguments will be passed to a function, like when we want to pass a list or a tuple of values, we use *args.

The words args and kwargs are a convention, and we can use anything in their place. The return type of *args is tuple and **kwargs is dictionary

What is a closure in Python?

A closure in Python is said to occur when a nested function references a value in its enclosing scope. The whole point here is that it remembers the value.

```
1. def A(x):
2.     def B():
3.     print(x)
4.     return B
5. A(7)()
```

Differentiate between deep and shallow copy.

The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

- A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.

- A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

```
>>> import copy
>>> b=copy.deepcopy(a)
```

A shallow copy, however, copies one object's reference to another. So, if we make a change in the copy, it will affect the original object. For this, we have the function copy(). We use it like:

What is the iterator?

An **iter*able*** is anything you're able to loop over.
An **iter*ator*** is the object that does the actual iterating.
You can get an iterator from any iterable by calling the built-in iter function on the iterable.

1. iter()- To create an iterator
2. next()- To iterate to the next element

```
a=iter([2,4,6,8,10])
next(a)
```

After a completion of values it throws a STOPITERATION error.

```
class yrange:
  def __init__(self, n):
    self.i = 0
    self.n = n

  def __iter__(self):
    return self

  def __next__(self):
    if self.i < self.n:
      i = self.i
      self.i += 1
      return i
    else:
      raise StopIteration()
```

What is a generator?

Python generator produces a sequence of values to iterate on. This way, it is kind of an iterable. We define a function that 'yields' values one by one, and then use a for loop to iterate on it.

```
def squares(n):
  i=1
  while(i<=n):
    yield i**2
    i+=1
for i in squares(7):
  print(I, end= ",")
```

1, 4, 9, 16, 25, 36, 49

A common use case of generators is to work with data streams or large files, like CSV files.

```
def csv_reader(file_name):
    for row in open(file_name, "r"):
        yield row
```

file.read().split() loads everything into memory at once, causing the MemoryError.

What is difference between iterator and generator?

They do, but there are subtle differences:

generators are usually slightly slower when run to completion. Their advantage is that they don't need to store all elements in memory at once, and can stop halfway through.

- For a generator, we create a function. For an iterator, we use in-built functions iter() and next().
- For a generator, we use the keyword 'yield' to yield/return an object at a time.
- A generator may have as many 'yield' statements as you want.
- A generator will save the states of the local variables every time 'yield' will pause the loop. An iterator does not use local variables; it only needs an iterable to iterate on.
- Using a class, you can implement your own iterator, but not a generator.

What is a decorator?

A decorator is a function that adds functionality to another function without modifying it. It wraps another function to add functionality to it. Take an example.

```
>>> def decor(func):
        def wrap():
            print("$$$$$$$$$$$$$$$$")
            func()
            print("$$$$$$$$$$$$$$$$")
    return wrap
```

```
>>> @decor
def sayhi():
    print("Hi")
>>> sayhi()
```

What is tuple unpacking?

Suppose we have a tuple nums=(1,2,3). We can unpack its values into the variables a, b, and c. Here's how:

```
>>> nums=(1,2,3)
>>> a,b,c=nums
>>> a
```

What is a frozen set in Python?

First, let's discuss *what a set is*. A set is a collection of items, where there cannot be any duplicates. A set is also unordered(We cant do indexing).

```
>>> myset={1,3,2,2}
>>> myset
```

{1, 2, 3}
However, a set is mutable. A frozen set is immutable. This means we cannot change its values. This also makes it eligible to be used as a key for a dictionary.

```
>>> myset=frozenset([1,3,2,2])
>>> myset
```

frozenset({1, 2, 3})

```
>>> type(myset)
<class 'frozenset'>
```

What is the Dogpile effect?

In case the cache expires, what happens when a client hits a website with multiple requests is what we call the dogpile effect. To avoid this, we can use a semaphore lock. When the value expires, the first process acquires the lock and then starts to generate the new value.

Explain garbage collection with Python.

The following points are worth nothing for the garbage collector with CPython-

- Python maintains a count of how many references there are to each object in memory
- When a reference count drops to zero, it means the object is dead and Python can free the memory it allocated to that object
- The garbage collector looks for reference cycles and cleans them up
- Python uses heuristics to speed up garbage collection
- Recently created objects might as well be dead
- The garbage collector assigns generations to each object as it is created
- It deals with the younger generations first.

What is Monkey Patching?

Monkey patching refers to modifying a class or module at runtime (dynamic modification). It extends Python code at runtime.

For example:

```
from pkg.module import MyClass
def sayhello(self):
print("Hello")

MyClass.sayhello=sayhello
```

How will you use Python to read a random line from a file?

We can borrow the choice() method from the random module for this.

```
>>> import random
>>> lines=open('tabs.txt').read().splitlines()
>>> random.choice(lines)
```

What is JSON? Describe in brief how you'd convert JSON data into Python data?

JSON stands for JavaScript Object Notation. It is a highly popular data format, and it stores data into NoSQL databases. JSON is generally built on the following two structures:

- A collection of <name,value> pairs

- An ordered list of values.

Python supports JSON parsers. In fact, JSON-based data is internally represented as a dictionary in Python. To convert JSON data into Python data, we use the load() function from the JSON module.

Differentiate between split(), sub(), and subn() methods of the remodule.

The re module is what we have for processing regular expressions with Python. Let's talk about the three methods we mentioned-

- split()- This makes use of a regex pattern to split a string into a list
- sub()- This looks for all substrings where the regex pattern matches, and replaces them with a different string
- subn()- Like sub(), this returns the new string and the number of replacements made

How would you display a file's contents in reversed order?

Let's first get to the Desktop. We use the chdir() function/method form the os module for this.

```
>>> import os
>>> os.chdir('C:\\Users\\lifei\\Desktop')
```

The file we'll use for this is Today.txt, and it has the following contents:

OS, DBMS, DS, ADA HTML, CSS, jQuery, JavaScript
Python, C++, Java This sem's subjects Debugger itertools container

Let's read the contents into a list, and then call reversed() on it:

```
>>> for line in reversed(list(open('Today.txt'))):
print(line.rstrip())
```

container itertools Debugger This sem's subjects Python, C++, Java
HTML, CSS, jQuery, JavaScript OS, DBMS, DS, ADA

Without the rstrip(), we would get blank lines between the output.

Whenever you exit Python, is all memory de-allocated?

The answer here is no. The modules with circular references to other objects, or to objects referenced from global namespaces, aren't always freed on exiting Python.

Plus, it is impossible to de-allocate portions of memory reserved by the C library.

Optionally, what statements can you put under a try-except block?

We have two of those:

else- To run a piece of code when the try-block doesn't create an exception.
finally- To execute some piece of code regardless of whether there is an exception.

```
>>> try:
print("Hello")
except:
print("Sorry")
else:
print("Oh then")
finally:
print("Bye")
```

Hello
Oh then
Bye

Can you explain the filter(), map(), and reduce() functions?

filter()- This function lets us keep the values that satisfy some conditional logic. Let's take an example.

```
1. >>> set(filter(lambda x:x>4, range(7)))
```

{5, 6}

This filters in the elements from 0 to 6 that are greater than the number 4.

map()- This function applies a function to each element in the iterable.

```
1. >>> set(map(lambda x:x**3, range(7)))
```

{0, 1, 64, 8, 216, 27, 125}

This calculates the cube for each element in the range 0 to 6 and stores them in a set.

reduce()- This function reduces a sequence pair-wise, repeatedly until we arrive at a single value.

```
1. >>> reduce(lambda x,y:y-x, [1,2,3,4,5])
```

3

How will you share global variables across modules?

To do this for modules within a single program, we create a special module, then import the config module in all modules of our application. This lets the module be global to all modules.

Q.30. List some pdb commands.

Some pdb commands include-

- <b> — Add breakpoint
- <c> — Resume execution
- <s> — Debug step by step
- <n> — Move to next line
- <l> — List source code
- <p> — Print an expression

Write a regular expression that will accept an email id. Use the re module.

```
1. >>> import re
2. >>> e=re.search(r'[0-9a-zA-Z.]+@[a-zA-Z]+\.(com|co\.in)$','abc@gmail.com')
3. >>> e.group()
```

'abc@gmail.com'

What is pickling and unpickling?

To create portable serialized representations of Python objects, we have the module 'pickle'. It accepts a Python object (remember, everything in Python is an object). It then converts it into a string representation and uses the dump() function to dump it into a file. We

call this pickling. In contrast, retrieving objects from this stored string representation is termed 'unpickling'.

What is the MRO in Python?

MRO stands for Method Resolution Order. Talking of multiple inheritances, whenever we search for an attribute in a class, Python first searches in the current class. If found, its search is satiated. If not, it moves to the parent class. It follows an approach that is left-to-right and depth-first. It goes Child, Mother, Father, Object.

We can call this order a linearization of the class Child; the set of rules applied are the Method Resolution Order (MRO). We can borrow the __mro__ attribute or the mro() method to get this.

If a function does not have a return statement, is it valid?

Very conveniently. A function that doesn't return anything returns a None object. Not necessarily does the return keyword mark the end of a function; it merely ends it when present in the function. Normally, a block of code marks a function and where it ends, the function body ends.

So, this was the last category of our Python programming interview questions and answers. Hope this helped you. If you have more python interview questions for experienced or freshers or any interview experience do share with us through comments.

Explain join() and split() in Python.

join() lets us join characters from a string together by a character we specify.

```
1. >>> ','.join('12345')
```

'1,2,3,4,5'

split() lets us split a string around the character we specify.

```
1. >>> '1,2,3,4,5'.split(',')
```

['1', '2', '3', '4', '5']

Differentiate between the append() and extend() methods of a list.

methods append() and extend() work on lists. While append() adds an element to the end of the list, extend adds another list to the end of a list.
Let's take two lists.

```
1. >>> list1,list2=[1,2,3],[5,6,7,8]
```

This is how append() works:

```
1. >>> list1.append(4)
2. >>> list1
```

[1, 2, 3, 4]

And this is how extend() works:

```
1. >>> list1.extend(list2)
2. >>> list1  o/p [1, 2, 3, 4, 5, 6, 7, 8]
```

How would you randomize the contents of a list in-place?

For this, we'll import the function shuffle() from the module random.

```
>>> from random import shuffle
>>> shuffle(mylist)
```

What is the enumerate() function in Python?

enumerate() iterates through a sequence and extracts the index position and its corresponding value too.
Let's take an example.

```
>>> for i,v in enumerate(['Python','C++','Scala']):
print(i,v)
```

0 Python
1 C++
2 Scala

Where will you use while rather than for?

Although we can do with for all that we can do with while, there are some places where a while loop will make things easier-

- For simple repetitive looping
- When we don't need to iterate through a list of items- like database records and characters in a string.

Q.32. Take a look at this piece of code:

```
A0= dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1= range(10)
A2= sorted([i for i in A1 if i in A0])
A3= sorted([A0[s] for s in A0])
A4= [i for i in A1 if i in A3]
A5= {i:i*i for i in A1}
A6= [[i,i*i] for i in A1]
A0,A1,A2,A3,A4,A5,A6
```

What are the values of variables A0 to A6? Explain.

Here you go:

A0= {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
A1= range(0, 10)
A2= []
A3= [1, 2, 3, 4, 5]
A4= [1, 2, 3, 4, 5]
A5= {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
A6= [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

What are the file-related modules we have in Python?

We have the following libraries and modules that let us manipulate text and binary files on our file systems-

Os, os.path, shutil

Explain inheritance in Python.

When one class inherits from another, it is said to be the child/derived/sub class inheriting from the parent/base/super class. It inherits/gains all members (attributes and methods).

Inheritance lets us reuse our code, and also makes it easier to create and maintain applications. Python supports the following kinds of inheritance:

- Single Inheritance- A class inherits from a single base class.
- Multiple Inheritance- A class inherits from multiple base classes.
- Multilevel Inheritance- A class inherits from a base class, which, in turn, inherits from another base class.
- Hierarchical Inheritance- Multiple classes inherit from a single base class.
- Hybrid Inheritance- Hybrid inheritance is a combination of two or more types of inheritance.

Explain lambda expressions. When would you use one?

When we want a function with a single expression, we can define it anonymously. A *lambda expression* may take input and returns a value. To define the above function as a lambda expression, we type the following code in the interpreter:

```
1. >>> (lambda a,b:a if a>b else b)(3,3.5)
```

3.5

Here, a and b are the inputs. a if a>b else b is the expression to return. The arguments are 3 and 3.5.

It is possible to not have any inputs here.

```
1. >>> (lambda :print("Hi"))()
```

Hi

What is the pass statement in Python?

There may be times in our code when we haven't decided what to do yet, but we must type something for it to be syntactically correct. In such a case, we use the pass statement.

```
1.      >>> def func(*args):
2.      Pass
```

Explain help() and dir() functions in Python.

The help() function displays the documentation string and help for its argument.

```
1. >>> import copy
2. >>> help(copy.copy)
```

Help on function copy in module copy:

copy(x)

Shallow copy operation on arbitrary Python objects.

See the module's __doc__ string for more info.

The dir() function displays all the members of an object(any kind).

```
1. >>> dir(copy.copy)
```

What does the function zip() do?

One of the less common functions with beginners, zip() returns an iterator of tuples.

```
1. >>> list(zip(['a','b','c'],[1,2,3]))
```

[('a', 1), ('b', 2), ('c', 3)]

Here, it pairs items from the two lists and creates tuples with those. But it doesn't have to be lists.

```
1. >>> list(zip(('a','b','c'),(1,2,3)))
```

[('a', 1), ('b', 2), ('c', 3)]

Explain Python List Comprehension.

The list comprehension in python is a way to declare a list in one line of code. Let's take a look at one such example.

```
1. >>> [i for i in range(1,11,2)]
```

[1, 3, 5, 7, 9]

```
1. >>> [i*2 for i in range(1,11,2)]
```

[2, 6, 10, 14, 18]

How is a .pyc file different from a .py file?

While both files hold bytecode, .pyc is the compiled version of a Python file. It has platform-independent bytecode. Hence, we can execute it on any platform that supports the .pyc format. Python automatically generates it to improve performance(in terms of load time, not speed).

What makes Python object-oriented?

Python is object-oriented because it follows the Object-Oriented programming paradigm. This is a paradigm that revolves around classes and their instances (objects). With this kind of programming, we have the following features:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

What is Encapsulation?

Ans: we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation.

What is Polymorphism?

Ans: Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

example: Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). we could use same method to color any shape. This concept is called Polymorphism.

What is class?

Ans: A class is a blueprint for the objects. For example, Ram, Shyam are all objects so we can define a template (blueprint) class Human for these objects.

What is Object?

Ans: An object (instance) is an instantiation of a class.

example: The example for object of parrot class can be:

obj = Parrot()

Here, obj is object of class Parrot.

What is Abstraction?

Ans: Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.

What is the use of regular expression in Python?

Ans: A **regular expression** is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. The **Python** module re provides full support for **regular expressions in Python**.

1) How do you match a regular expression in Python?

1. Import the regex module with import re.
2. Create a Regex object with the re.compile() function. ...
3. Pass the string you want to search into the Regex object's search() method. ...
4. Call the Match object's group() method to return a string of the actual matched text.

2) Which pattern matching modifiers permits whitespace and comments inside the regular expression?

Ans: The modifier re.X allows whitespace and comments inside the regular expressions.

3) Which function creates a Python object?

Ans: The function re.compile(str) is the only function which creates a Python object.

4) **Which function returns a dictionary mapping group names to group numbers?**
Ans: The function re.compile.groupindex returns a dictionary mapping group names to group numbers.

`

1. What is the difference between Flask and Django?

Comparison Factor	Django	Flask
Project Type	Supports large projects	Built for smaller projects
Templates, Admin and ORM	Built-in	Requires installation
Ease of Learning	Requires more learning and practice	Easy to learn
Flexibility	Allows complete web development without the need for third-party tools	More flexible as the user can select any third-party tools according to their choice and requirements
Visual Debugging	Does not support Visual Debug	Supports Visual Debug
Type of framework	Batteries included	Simple, lightweight
Bootstrapping-tool	Built-it	Not available

2. What is Django?

Django is a free and open source web application framework, written in Python. A web framework is a set of components that helps you to develop websites faster and easier.

When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc.

Frameworks exist to save you from having to reinvent the wheel and to help alleviate some of the overhead when you're building a new site.

3. What are the features of Django?

1) Fast: Django was designed to help developers take applications from concept to completion as quickly as possible.

2) Fully loaded: Django includes dozens of extras we can use to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks.

3) Security: Django takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.

4) **Scalability**: Some of the busiest sites on the planet use Django's ability to quickly and flexibly scale to meet the heaviest traffic demands.

`

5) **Versatile**: Companies, organizations and governments have used Django to build all sorts of things — from content management systems to social networks to scientific computing platforms.

4. How do you check for the version of Django installed on your system?

To check for the version of Django installed on your system, you can open the command prompt and enter the following command:

python -m django –version

You can also try to import Django and use the get_version() method as follows:

Import Django

print(django.get_version())

5. What are the advantages of using Django?

- Django's stack is loosely coupled with tight cohesion

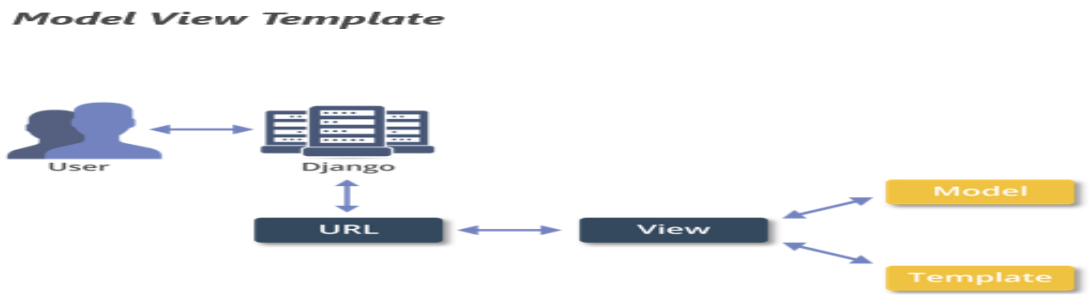
  It means that we can work on different components parallelly and then can integrate much more easily.

- The Django apps make use of very less code
- Allows quick development of websites
- Follows the DRY or the Don't Repeat Yourself Principle which means, one concept or a piece of data should live in just one place
- Consistent at low as well as high levels
- Behaviors are not implicitly assumed, they are rather explicitly specified
- SQL statements are not executed too many times and are optimized internally
- Can easily drop into raw SQL whenever required

6. Explain Django architecture.

Django follows the MVT or Model View Template architecture which is based on the MVC or Model View Controller architecture. The main difference between these two is that Django itself takes care of the controller part.

`



According to Django, the 'view' basically describes the data presented to the user. It *does not deal with how the data looks* but rather *what the data actually is*. Views are basically callback functions for the specified URL's and these callback functions describe which data is presented.

The 'templates' on the other hand deal with the presentation of data, thereby, separating the content from its presentation. In Django, views delegate to the templates to present the data.

The 'controller' here is Django itself which sends the request to the appropriate view in accordance with the specified URL. This is why Django is referred to as MTV rather than MVC architecture.

7. Give a brief about 'django-admin'.

Django comes with a fully customizable in-built admin interface, which lets us see and make changes to all the data in the database of registered apps and models. To use a database table with the admin interface, we need to register the model in the admin.py file.

django-admin is the command-line utility of Django for administrative tasks. Using the django-admin you can perform a number of tasks some of which are listed out in the following table:

Task	Command
To display the usage information and the list of the commands provided by each application	django-admin help
To display the list of available commands	django-admin help –command
To display the description of a given command and the list of its available options	django-admin help <command>
Determining the version of Django	django-admin version
Creating new migrations based on the changes made in models	django-admin makemigrations
Synchronizing the database state with the current set of models and migrations	django-admin migrate
Starting the development server	django-admin runserver
Sending a test email in order to confirm the email sending through	django-admin sendtestemail

`

Django is working	
To start the Python interactive interpreter	django-admin shell
To show all the migrations in your project	django-admin showmigrations

8. How do you connect your Django project to the database?

Django comes with a default database which is SQLite. To connect your project to this database, use the following commands:

1. python manage.py migrate (migrate command looks at the INSTALLED_APPS settings and creates database tables accordingly)
2. python manage.py makemigrations (tells Django you have created/ changed your models)
3. python manage.py sqlmigrate <name of the app followed by the generated id> (sqlmigrate takes the migration names and returns their SQL)

9. What are the various files that are created when you create a Django Project? Explain briefly.

When you create a project using the startproject command, the following files will be created:

File Name	Description
manage.py	A command-line utility that allows you to interact with your Django project
__init__.py	An empty file that tells Python that the current directory should be considered as a Python package
settings.py	Consists of the settings for the current project
urls.py	Contains the URL's for the current project
wsgi.py	This is an entry-point for the web servers to serve the project you have created

10. What are 'Models' in Django?

A model is a Python class in Django that is derived from the django.db.models.Model class. A model is used in Django to represent a table in a database. It is used to interact with and get results from the database tables of our application.

Models are a single and definitive source for information about your data. It consists of all the

`

essential fields and behaviors of the data you have stored. Often, each model will map to a single specific database table.

In Django, models serve as the abstraction layer that is used for structuring and manipulating your data. Django models are a subclass of the django.db.models.Model class and the attributes in the models represent database fields.

11. What are 'views' in Django?

A view in Django is a class and/or a function that receives a request and returns a response. A view is usually associated with urlpatterns, and the logic encapsulated in a view is run when a request to the URL associated with it is run. A view, among other things, gets data from the database using models, passes that data to the templates, and sends back the rendered template to the user as an HttpResponse.

Django views serve the purpose of encapsulation. They encapsulate the logic liable for processing a user's request and for returning the response back to the user. Views in Django either return an HttpResponse or raise an exception such as Http404. HttpResponse contains the objects that consist of the content that is to be rendered to the user. Views can also be used to perform tasks such as read records from the database, delegate to the templates, generate a PDF file, etc.

12. What are 'templates' in Django?

Django's template layer renders the information to be presented to the user in a designer-friendly format. Using templates, you can generate HTML dynamically. The HTML consists of both static as well as dynamic parts of the content. You can have any number of templates depending on the requirement of your project. It is also fine to have none of them.Django has its own template system called the Django template language (DTL). Regardless of the backend, you can also load and render templates using Django's standard admin.

13. What is the difference between a Project and an App?

An app is basically a Web Application that is created to do something for example, a database of employee records. A project, on the other hand, is a collection of apps of some particular website. Therefore, a single project can consist of 'n' number of apps and a single app can be in multiple projects.

14. What are the different inheritance styles in Django?

Inheritance style	Description
Abstract base classes	Used when you want to use the parent class to hold information that you don't want to type for each child model. Here, the parent class is never used in solitude
Multi-table inheritance	Used when you have to subclass an existing model and want each model to have its own database table
Proxy models	Used if you only want to modify the Python-level behavior of a model, without changing the 'models' fields in any way

`

15. What are static files?

Static files in Django are those files that serve the purpose of additional files such as the CSS, images or JavaScript files. These files are managed by django.contrib.staticfiles. These files are created within the project app directory by creating a subdirectory named as static.

16. What are 'signals'?

Signals are messages for applications. The signals are present to transmit the occurring of an event. They can notify other apps that a particular event has occurred. These are pretty useful when the applications exist which are interested in a mutual event.

There are events like saving data in the database like deleting something from the database. There are various backend actions that are performed.

Signals are core functionality and may not be used in every project. Still, it's nice to have an application that can help us use it.

Django's signal is especially good as it transmits objects between applications. The native implementation is very efficient. There are various signals provided by Django like:

1. django.db.models.signals.pre_save
2. django.db.models.signals.post_save
3. django.db.models.signals.pre_delete
4. django.db.models.signals.m2m_changed
5. django.core.signals.request_started

17. Briefly explain Django Field Class.

'Field' is basically an abstract class that actually represents a column in the database table. The Field class, is in turn, a subclass of RegisterLookupMixin. In Django, these fields are used to create database tables (db_type()) which are used to map Python types to the database using *get_prep_value()* and vice versa using *from_db_value()* method. Therefore, fields are fundamental pieces in different Django APIs such as models and querysets.

18. How to do you create a Django project?

To create a Django project, cd into the directory where you would like to create your project and type the following command:

- django-admin startproject xyz

NOTE: Here, xyz is the name of the project. You can give any name that you desire.

`

19. What is mixin?

Mixin is a type of multiple inheritance wherein you can combine behaviors and attributes of more than one parent class. Mixins provide an excellent way to reuse code from multiple classes. For example, generic class-based views consist of a mixin called TemplateResponseMixin whose purpose is to define *render_to_response()* method. When this is combined with a class present in the View, the result will be a TemplateView class.

One drawback of using these mixins is that it becomes difficult to analyze what a child class is doing and which methods to override in case of its code being too scattered between multiple classes.

20. What are 'sessions'?

Sessions are fully supported in Django. Using the session framework, you can easily store and retrieve arbitrary data based on the per-site-visitors. This framework basically stores data on the server-side and takes care of sending and receiving cookies. These cookies consist of a session ID but not the actual data itself unless you explicitly use a cookie-based backend.

21. What do you mean by context?

Context in Django is a dictionary mapping template variable name given to Python objects. This is the conventional name, but you can give any other name of your choice if you wish to do it.

A context in Django is a dictionary, in which keys represent variable names and values represent their values. This dictionary (context) is passed to the template which then uses the variables to output the dynamic content.

22. When can you use iterators in Django ORM?

Iterators in Python are basically containers that consist of a countable number of elements. Any object that is an iterator implements two methods which are, the __init__() and the __next__() methods. When you are making use of iterators in Django, the best situation to do it is when you have to process results that will require a large amount of memory space. To do this, you can make use of the iterator() method which basically evaluates a QuerySet and returns the corresponding iterator over the results.

23. Explain the caching strategies of Django?

Caching basically means to save the output of an expensive calculation in order to avoid performing the same calculation again. Django provides a robust cache system which in turn helps you save dynamic web pages so that they don't have to be evaluated over and over again for each request. Some of the caching strategies of Django are listed down in the following table:

Strategy	Description
Memcached	Memory-based cache server which is the fastest and most efficient

`

Filesystem caching	Cache values are stored as separate files in a serialized order
Local-memory caching	This is actually the default cache in case you have not specified any other. This type of cache is per-process and thread-safe as well
Database caching	Cache data will be stored in the database and works very well if you have a fast and well-indexed database server

24. Explain the use of Middlewares in Django.

Middleware is a framework that is light and low-level plugin system for altering Django's input and output globally. It is basically a framework of hooks into the request/ response processing of Django. Each component in middleware has some particular task. For example, the *AuthenticationMiddleware* is used to associate users with requests using sessions. Django provides many other middlewares such as cache middleware to enable site-wide cache, common middleware that performs many tasks such as forbidding access to user agents, URL rewriting, etc, GZip middleware which is used to compress the content for browsers, etc.

25. What is the significance of manage.py file in Django?

The manage.py file is automatically generated whenever you create a project. This is basically a command-line utility that helps you to interact with your Django project in various ways. It does the same things as django-admin but along with that, it also sets the DJANGO_SETTINGS_MODULE environment variable in order to point to your project's settings. Usually, it is better to make use of manage.py rather than the django-admin in case you are working on a single project.

26. Explain the use of 'migrate' command in Django?

In Django, migration is a way of applying changes made in models that are adding the field, deleting model, etc. into a database. To manage database schema changes migrations are a great way. You will find Django has created migration files inside the migration folder for each model. Each table is mapped to the model. Most migrations are designed to be automatic, but you need to know when to migrate, and when to run migrations.

Django has various commands to interact with migrations and Django's handling of a database schema. These commands perform migration-related tasks. After creating models you can use these commands.

- **makemigration** - This command is used to create a migration file.
- **migrate** - This command creates a table according to the schema defined in migration file.
- **sqlmigrate** - This command is used to show a raw SQL query of the applied migration.
- **showmigrations** - This command list all the migrations and their status

`

27. How to view and filter items from the database?

In order to view all the items from your database, you can make use of the 'all()' function in your interactive shell as follows:

- XYZ.objects.all() where XYZ is some class that you have created in your models

- XYZ.objects.filter(pk=1)
- XYZ.objects.get(id=1)

28. Explain how a request is processed in Django?

We know that all web application uses HTTP/HTTPS protocol. The basic principle of the HTTP protocol is Client sends a request to the server and the data server sends a response back to the client based on request data. We need a web server and WSGIserver while setting up a Django application on the server. Without a web server, WSGI server will result in a more number of requests resulting in gradually slow down the application performance. The web server will help in balancing the requests load on the server.

The client is a piece of software which sends a request by following HTTP/HTTPS protocol and mostly we consider Web Browser as a client. "Nginx, uWSGI and Django" or "Nginx, gunicorn and Django" or "Apache, mod_wsgi and Nginx" are the three types of combinations in Django application deployment on the server and we can use any one of the combinations. Now let us discuss on how a request is processed and response is created and sent to the client.

When a client sends request it first passed to a web server. The request contains configuration rules to be dispatch to the WSGI server. WSGI server sends the request to the Django application.

For dealing with request-response lifecycle, Django application has the following layers -

- Request middlewares
- URL Router
- Views
- Context processors
- Template Renderers
- Response middleware

Any request that comes in is handled by Request middleware. There can be multiple middlewares and can find it in settings.py. While processing the request Django Request middlewares follow order. Django has some default middlewares and we can also write or customize middleware. Middleware process the request and submits it to the URL Router or URL dispatcher.

URL Router receives a request from the middleware and from the request it collects the URL path. From the URL path, the URL router tries to match the request path with available URL patterns. These

`

patterns are in the regular expression form. Once the URL path is matched with URL patterns the request is submitted to the View associated with URL.

Views are business logic layer, which processes the business logic using request and request data. A request is processed in the view, it is sent to Context processor. Context processor adds context data that helps Template Renders to deliver the template to generate HTTP response and again this request will be sent back to Response middleware to process. Request middleware adds or modifies header information or body information before sending it to the client again.

29. How did Django come into existence?

Django basically grew from a very practical need. World Online developers namely **Adrian Holovaty and Simon Willison** started using Python to develop its websites. As they went on building interactive sites, they began to pull out a generic Web development framework that allowed them to build Web applications more and more quickly. In summer 2005, World Online decided to open-source the resulting software, which is, Django.

30. How to use file-based sessions?

In order to make use of file-based sessions, you will need to set the SESSION_ENGINE setting to “django.contrib.sessions.backends.file”.

Session data is stored on the server. Django creates a random unique string called session-id or SID and Django associates that ID or SID with the data. As a value to the browser, the server sends a cookie named session-id containing SID as value. On-page request, the browser sends a request containing a cookie with SID to the server. To retrieve this session data and make it accessible in code, Django uses this SID. Django generated SID is 32 characters long random string. Django support for anonymous sessions. Django session framework allows to store and retrieve arbitrary data on per site visitor basis.

Seesions are implemented via middleware. To enable session functionality, edit MIDDLEWARE_CLASSES setting and make sure 'django.contrib.sessions.middleware.SessionMiddleware' is added.

31. Explain the Django URLs in brief?

Django allows you to design your own URLs however you like. The aim is to maintain a clean URL scheme without any framework limitations. In order to create URLs for your app, you will need to create a Python module informally called the URLconf or URL configuration which is pure Python code and is also a mapping between the URL path expressions to the Python methods. The length of this mapping can be as long or short as required and can also reference other mappings. When processing a request, the requested URL is matched with the URLs present in the urls.py file and the corresponding view is retrieved.

32. Give the exception classes present in Django.

Django uses its own exceptions as well as those present in Python. Django core exceptions are

`

present in *django.core.exceptions* class some of which are mentioned in the table below:

Exception	Description
AppRegistryNotReady	Raised when you try to use your models before the app loading process (initializes the ORM) is completed.
ObjectDoesNotExist	This is the base class for DoesNotExist exceptions
EmptyResultSet	This exception may be raised if a query won't return any result
FieldDoesNotExist	This exception is raised by a model's _meta.get_field() function in case the requested field does not exist
MultipleObjectsReturned	This is raised by a query if multiple objects are returned and only one object was expected

33. Is Django stable?

Yes, Django is quite stable. Many companies like Instagram, Discus, Pinterest, and Mozilla have been using Django for a duration of many years now. Not just this, Websites that are built using Django have weathered traffic spikes of over 50 thousand hits per second.

34. Does the Django framework scale?

Yes. Hardware is much cheaper when compared to the development time and this is why Django is designed to make full use of any amount of hardware that you can provide it. Django makes use of a "shared-nothing" architecture meaning you can add hardware at any level i.e database servers, caching servers or Web/ application servers.

35. Is Django a CMS?

No, Django is not a Content Management System (CMS), rather it is a Web application framework and a programming tool that helps you to build websites.

DjangoCMS is a web publishing platform built with Django and offers out of the box support for the common features you expect from a CMS. Django CMS can be easily customized and extended by developers to create a site precise to their needs.

Django CMS is a thoroughly tested platform that supports both large and small web sites. Django CMS is robust internationalization support for creating multilingual sites. It can be configured to handle different requirements. Content editing is possible and provides rapid access to the content management interface. Django CMS supports a variety of editors with advanced text editing features. It is a flexible plugins system. Thorough documentation is available for Django

36. What Python version should be used with Django?

The following table gives you the details of the versions of Python that you can use for Django:

`

Django Version	Python Versions
1.11	2.7, 3.4, 3.5, 3.6, 3.7 (added in 1.11.17)
2.0	3.4, 3.5, 3.6, 3.7
2.1, 2.2	3.5, 3.6, 3.7

Python 3 is actually the most recommended because it is fast, has more features and is better supported. In the case of Python 2.7, Django 1.1 can be used along with it but only till the year 2020.

37. Does Django support NoSQL?

NoSQL basically stands for "not only SQL". This is considered as an alternative to the traditional RDBMS or the relational Databases. Officially, Django does not support NoSQL databases. However, there are third-party projects, such as Django non-rel, that allow NoSQL functionality in Django. Currently, you can use MongoDB and Google App Engine.

38. How can you customize the functionality of the Django admin interface?

There are a number of ways to do this. You can piggyback on top of an add/ change form that is automatically generated by Django, you can add JavaScript modules using the *js parameter*. This parameter is basically a list of URLs that point to the JavaScript modules that are to be included in your project within a <script> tag. In case you want to do more rather than just playing around with from, you can exclusively write views for the admin.

39. Is Django better than Flask?

Django is a framework that allows you to build large projects. On the other hand, Flask is used to build smaller websites but flask is much easier to learn and use compared to Django. Django is a full-fledged framework and no third-party packages are required. Flask is more of a lightweight framework that allows you to install third-party tools as and how you like. So, the answer to this question basically depends on the user's need and in case the need is very heavy, the answer is definitely, Django.

40. Give an example of a Django view.

A view in Django either returns an HttpResponse or raises an exception such as Http404. HttpResponse contains the objects that consist of the content that is to be rendered to the user.

41. What should be done in case you get a message saying "Please enter the correct username and password" even after entering the right details to log in to the admin section?

In case you have entered the right details and still not able to login to the admin site, cross verify if the user account has *is_active* and *is_staff* attributes set to True. The admin site allows only those users for whom these values are set to True.

`

42. What should be done in case you are not able to log in even after entering the right details and you get no error message?

In this case, the login cookie is not being set rightly. This happens if the domain of the cookie sent out by Django does not match the domain in your browser. For this, you must change the *SESSION_COOKIE_DOMAIN* setting to match that of your browser.

43. How can you limit admin access so that the objects can only be edited by those users who have created them?

Django's *ModelAdmin* class provides customization hooks using which, you can control the visibility and editability of objects in the admin. To do this, you can use the *get_queryset()* and *has_change_permission().*

44. What to do when you don't see all objects appearing on the admin site?

Inconsistent row counts are a result of missing Foreign Key values or if the Foreign Key field is set to null=False. If the *ForeignKey* points to a record that does not exist and if that foreign is present in the *list_display* method, the record will not be shown the admin changelist.

45. What do you mean by the csrf_token?

The csrf_token is used for protection against Cross-Site Request Forgeries. This kind of attack takes place when a malicious website consists of a link, some JavaScript or a form whose aim is to perform some action on your website by using the login credentials of a genuine user.

46. Does Django support multiple-column Primary Keys?

No. Django only supports single-column Primary Keys.

47. How can you see the raw SQL queries that Django is running?

First, make sure that your DEBUG setting is set to True. Then, type the following commands:

from django.db import connection

connection.queries

48. Is it mandatory to use the model/ database layer?

No. The model/ database layer is actually decoupled from the rest of the framework.

49. How to make a variable available to all the templates?

You can make use of the *RequestContext* in case all your templates require the same objects, such as, in the case of menus. This method takes an HttpRequest as its first parameter and it automatically populates the context with a few variables, according to theengine's context_processors configuration option

`

50. What is the difference between a project and an app in Django?

In Django, a project is the entire application and an app is a module inside the project that deals with one specific requirement. E.g., if the entire project is an ecommerce site, then inside the project we will have several apps, such as the retail site app, the buyer site app, the shipment site app, etc.

51. Explain Django's Request/Response Cycle.

In the Request/Response Cycle, first, a request is received by the Django server. Then, the server looks for a matching URL in the urlpatterns defined for the project. If no matching URL is found, then a response with 404 status code is returned. If a URL matches, then the corresponding code in the view file associated with the URL is executed to build and send a response.

52. What is the use of the include function in the urls.py file in Django?

As in Django there can be many apps, each app may have some URLs that it responds to. Rather than registering all URLs for all apps in a single urls.py file, each app maintains its own urls.py file, and in the project's urls.py file we use each individual urls.py file of each app by using the include function.

53. Why is Django called a loosely coupled framework?

Django is called a loosely coupled framework because of its MVT architecture, which is a variant of the MVC architecture. It helps in separating the server code from the client-related code. Django's models and views take care of the code that needs to be run on the server like getting records from database, etc., and the templates are mostly HTML and CSS that just need data from models passed in by the views to render them. Since these components are independent of each other, Django is called a loosely coupled framework.

54. How do we register a model with Django admin?

To register a model with Django's admin interface, we make changes to our apps admin.py file. We have to open the admin.py file in the app folder in which our models are. For example, if we have an app named 'polls' and we wish to register a model named 'Question', then we need to open 'polls/admin.py' and import the Question model and write: admin.site.register(Question). This will register our Question model with the admin site.

55. How do we generate a super user in Django?

In our project folder that contains Django's manage.py script, we have to open the command prompt and type the **python manage.py createsuperuser** command. Then, we need to enter the username, the email, and finally the password, twice (for conformation). This will create a super user for our Django project.

56. Mention some disadvantages of Django.

- Django' modules are bulky.

`

- It is completely based on Django ORM.
- Components are deployed together.
- We must know the full system to work with it.

57. What is Sessions Framework in Django?

The Sessions framework in Django is used to store arbitrary information about the user on the server in the database. This is done because HTTP is a stateless protocol, i.e., it does not store information between subsequent requests. Django uses a cookie containing a special session ID to identify each browser and its associated session with the site.

58. What is a cookie in Django?

A cookie is a small piece of information that is stored in the client browser. It is used to store user's data in a file permanently (or for the specified time). Cookie has its expiry date and time and gets removed automatically when it gets expired. Django provides in-built methods to set and fetch cookies.

59. What is a QuerySet in Django?

A QuerySet in Django is basically a collection of objects from our database. QuerySets are used by the Django ORM. When we use our models to get a single record or a group of records from the database, they are returned as QuerySets.

60. What is Django REST Framework?

Django REST framework is a powerful and flexible toolkit for building Web APIs.

It is popular because of its features over Django frameworks like Serialisation, Authentication policies and Web-browsable API.

Some of the reasons why you should use the REST framework are :

- For developers, the Web browsable API is huge usability.
- Authentication policies including packages for OAuth1a and OAuth2.
- Serialization for both ORM and non-ORM data sources.

`

- Customizable - just use regular function-based views.
- Extensive documentation, along with great community support.
- Used and trusted by internationally recognized companies.

61. What is a Meta Class in Django?

A Meta class is simply an inner class that provides metadata about the outer class in Django. It defines such things as available permissions, associated database table name, singular and plural versions of the name, etc.

62. When should we generate and apply migrations in a Django project and why?

We should do that whenever we create or make changes to the models of one of the apps in our project. This is because a migration is used to make changes to the database schema, and it is generated based on our models.

63. What is serialization in Django?

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

django.core serializers are meant for the purpose of serializing entire model instances into XML, JSON, or YAML, and vice versa. They don't do anything besides just serializing.

DRF's serializers are specifically for converting model instances into JSON objects *when dealing with data from HTML forms or API requests*. Thus, serialization is not always a smooth or straightforward process, as you may be passed illegitimate or incomplete data, or fields of the form may not correspond in an obvious way to the fields of the corresponding model(s). For this reason, DRF allows you to create custom subclasses of serializers.Serializer in order to clean and validate data that is passed to the server. This also allows you to customize the manner in which the data is stored in the model instance.

64. What are generic views?

When building a web application there are certain kind of views that we build again and again, such as a view that displays all records in the database (e.g., displaying all books in the books table), etc. These kinds of views perform the same functions and lead to repeated code. To solve this issue, Django uses class-based generic views. When using generic views, all we have to do is inherit the desired class from django.views.generic module and provide some information like model, context_object_name, etc.

`

65. Which class do we inherit from in order to perform unit tests?

We inherit from django.test.TestCase in order to perform unit tests as it contains all the methods we need to perform unit tests like assertEquals, assertTrue, etc.

66. Which class do we inherit from in order to perform functional tests?

We inherit from django.test.LiveServerTestCase. It launches a live Django server in the background on setup and shuts it down on teardown. This allows the use of automated test clients, e.g., the Selenium client, to execute a series of functional tests inside a browser and simulate a real user's actions.

67. What is the django.test.Client class used for?

The Client class acts as a dummy web browser, allowing us to test our views and interact with our Django-powered application programmatically. This is especially helpful for doing integration testing.

68. Explain context variable lookups in Django.

Context variables are variables passed in templates. The DTL (Django Templating Language) replaces these variables. A context dictionary is used to perform the replacement. We pass the context dictionary alongside the render() alongside template information.

DTL has its own way of filling these variables and it's in order. The template system can handle complex ***Python data structures*** as variables. The context variable lookups come in the role when the data is a dictionary, class object, etc.

The context lookup will fill the value in this order.

- **Dictionary Values:** The template system will look for dictionaries matching the variable name. It matches the key name with the name after the dot(.) operator.
- **Class Object Attribute values:** If it didn't find any dictionaries it will look for a class object. First lookups are done for attributes.
- **Class Object Methods:** It looks for a particular attribute of that name. If the attribute is not found then it looks for methods.
- **List-Index lookup:** At last, it will look for any lists with corresponding names. If it didn't find any lists then it considers variable as an Invalid Variable.

69. How does DTL handle invalid variables?

`

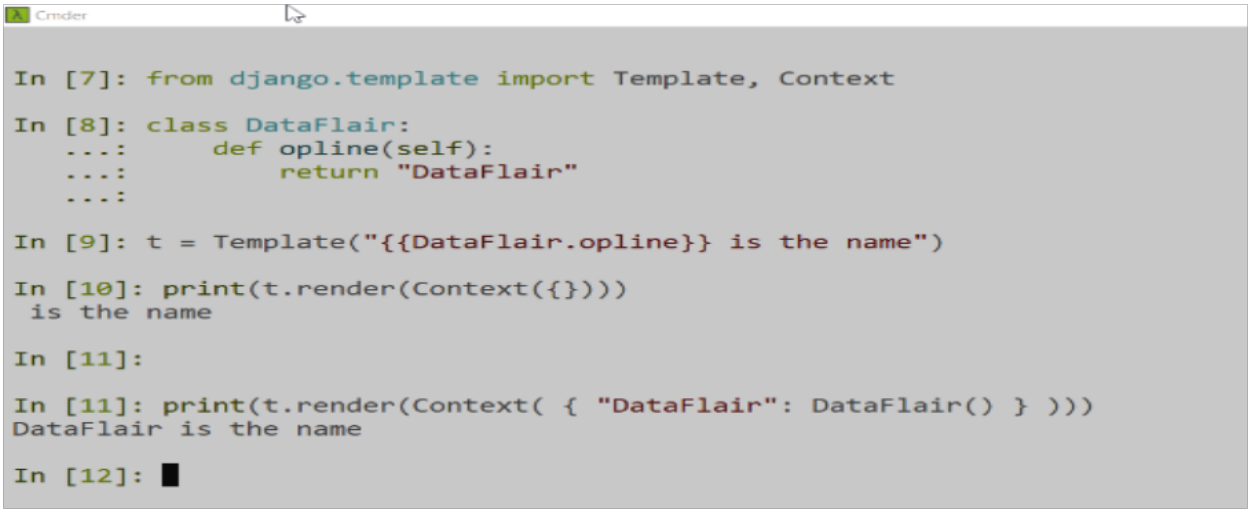
Ans. There are various instances where a variable might not exist in context variables. In case DTL cannot find the variable in context, it considers that variable as an invalid variable.

The DTL handles the value for invalid variables. The DTL System will insert the value from a setting in these cases. The setting is an attribute of the engine named as string_if_invalid. By default, it's an empty string. That empty string fills in the value in HTML render.

70. Can we call methods in templates? Elucidate with example.

Ans. Yes, method calls are possible in templates itself. it's the same thing as accessing an attribute from an object or dictionary.

For example, we have a Python class:



We make a template **t at line 9**.

In this template **t**, we will get the output of the **online** method in **DataFlair** Class. The corresponding output can be seen in the image. It is after line 11.

As we can see it prints the output returned by the string. We simply created a context dictionary and passed it to the template. The context dictionary has a key DataFlair which contained a DataFlair class object. To note, we execute the method when passing the object in the context variable. It is not recommended to pass methods like that. The method execution takes place before rendering.

Thus, we can call methods in the template.

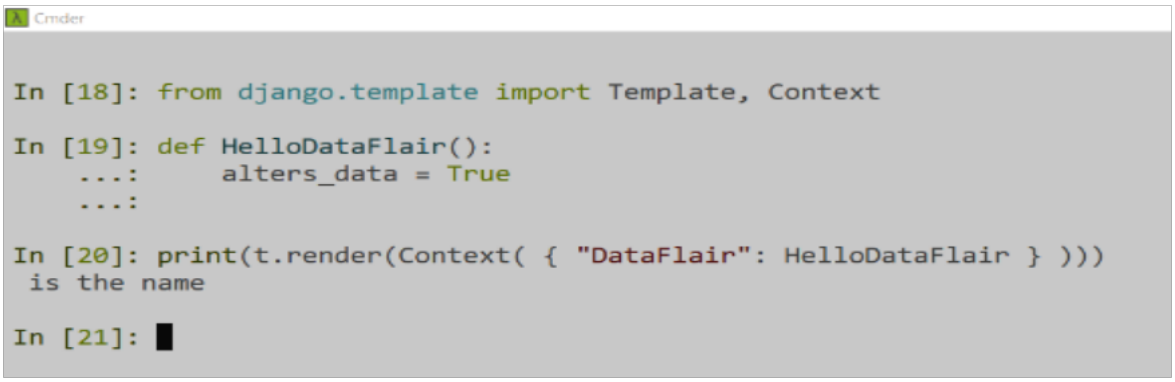
`

71. How can we stop the execution of the method while rendering?

It is true that the template system shall not be given access to some methods. Methods that can change data in the database are one of them. To prevent the execution of the method, we can keep a lock on the method definition itself.

DTL also checks for some variables in methods automatically. The variable we want to put in our method is **alters_data = True**. If we keep that variable in our method, the template system won't render it. Instead, it will treat the context variable as an invalid variable.

For example – Suppose we have a method that contains this variable.



Taking the template from the previous question, we just pass this method. Now, when we render this template, it will print output as shown.

Thus, the method was not executed and none object rendered.

72. What is forloop variable? Explain and implement.

The **{% for %}** in the template system comes with a pre-defined template variable. The forloop variable has various attributes that are frequently used. The forloop variable attributes are:

- **forloop.counter:** The forloop counter will return the integer. The number of times the loop has run is displayed in it. The forloop counter starts counting from 1.
- **forloop.counter0:** It is the same variable as the forloop counter but the counting starts from 0.
- **forloop.revcounter:** The forloop reverse counter returns an integer. The number of times the loop will run or the number of items remaining. Its numbering ends on 1.
- **forloop.revcounter0:** It is the same variable as forloop reverse counter. The numbering ends on 0

`

rather than 1.

- **forloop.first:** This is a special variable that returns True or False. This variable is true when the loop runs for the first time and false for all other times.
- **forloop.last:** This variable is similar to forloop.first. the only difference is that it marks the end of the list. It returns true when the list has ended and false otherwise.

In the below example, we have implemented For loop in the template.

```
Cmder

In [25]: from django.template import Template, Context

In [26]: rawst = """
    ...: <ul>
    ...: {% for item in list %}
    ...: <li><pre>
    ...: counter : {{ forloop.counter }}
    ...: coutner0 : {{ forloop.counter0 }}
    ...: revcounter : {{ forloop.revcounter }}
    ...: revcounter0 : {{ forloop.revcounter0 }}
    ...: forloop.first : {{ forloop.first }}
    ...: value : {{ item }}
    ...: </pre></li>
    ...: {% endfor %}
    ...: </ul>
    ...: """

In [27]: t = Template(rawst)

In [28]: li = ["DataFlair", "Karan", "Mittal"]

In [29]:
```

The output of the template is shown.

`

```
In [29]: print(t.render(Context({"list":li})))

<ul>

<li><pre>
counter : 1
coutner0 : 0
revcounter : 3
revcounter0 : 2
forloop.first : True
value : DataFlair
</pre></li>

<li><pre>
counter : 2
coutner0 : 1
revcounter : 2
revcounter0 : 1
forloop.first : False
value : Karan
</pre></li>

<li><pre>
counter : 3
coutner0 : 2
revcounter : 1
revcounter0 : 0
forloop.first : False
value : Mittal
</pre></li>
```

As you can see, the variables and their values. That's the functioning of forloop variable and its attributes.

73. What is reverse URL resolution in Django?

Ans. Reverse URL resolution is the method of obtaining urls in their final forms. The URLs can be embedded in generated webpages. Like in the navigation bar, there are various scenarios. Django provides various methods to achieve this which otherwise could have been hardcoded.

Hardcoding these URLs is a time-consuming and error-prone task. Therefore, these ways are used instead of getting Url's final form. The reverse resolution of URL is commonly achieved:

- In templates, by using **url template** tag.
- In Python code/view function, by using the **reverse().**
- In some high-level model class, by using **get_absolute_url()** method.

74. Explain user object in the Django admin system with its 3 flags?

Ans. User objects are our standard objects which have attributes like username, password, e-mail and name

`

fields. The user objects have a set of fields that define the permissions for that user.

All users are not allowed to do everything and that's where Django's permission system comes in. Its an integrated part of ***Django admin***.

There are 3 flags which determine the user activity, they are:

- **Active Flag:** The active flag sets whether the user is active or not. this flag shall be on if the user wants to login otherwise system won't log in the user. Even if the credentials entered by the user are correct this flag still needs to be active.
- **Staff Flag:** This flag differentiates between staff members and public users. It checks whether the user can access the admin-site. If he/she is not a staff, the response by the server is an error. Often a normal redirect is used in these case scenarios.
- **Superuser Flag:** The superuser is the ultimate user of the admin site. It can Create, Update, Delete any object. If this flag is on then all regular permissions are available for this user. It can perform any sort of operation that exists in Django-Admin.

75. What are object permissions in admin?

Ans. Each object which can be edited in the Django admin has three permissions. **Create Permission** allows the user to create the objects one time. An **Edit Permission** allows the user to update the created objects. A **Delete Permission** allows the user to delete objects.

These permissions have an impact on the database too.

76. What are custom validation rules in form data?

Ans. Custom validation rules are the customized rules used for form validation. Suppose we have a feedback form. There are fields like **messages, email, and subject**. If we get message data of 1 or 2 words that are of no use to us. To check the issue, we use custom validation rules.

We simply define a method in our forms.py by the name **clean_message()**. This is the Django way to do it. The method's name for custom validation should start with a **clean_fieldname()**. Django form system automatically looks for this type of method. Thus, these are called custom validation rules in Django.

It is always important to return the cleaned data of the field in a custom validation method. If not done, the method will return none instead resulting in loss of data.

77. What is the functionality of django.contrib.auth app? What kind of system exists in the auth app?

Ans. The auth app is a built-in Django application. It provides the developer with an authorization system. The auth app handles both authentication and authorization of users.

The authentication system consists of:

1. Users

`

2. Permissions
3. Groups
4. Password Hashing System
5. View functions and an interface
6. Customizable and plugins for the backend system

78. What is the difference between authentication and authorization?

Ans. Authentication and authorization are two different terms. The authentication means the verification of the entity(user) in Django's context. Authentication will verify that the user is what they claim to be.

Authorization is the step after authentication. It sets the actions that can be performed by the authenticated user. Authorization is a grouping of users and allowing limited actions.

Both of these functionalities are achieved by **django.contrib.auth application**. It is a built-in application in Django.

79. What's the difference between these two commands? What result will their execution generate?

Ans. Both commands are used to create or initiate a Django project. The difference comes by adding the period (.). The period makes it so the project initiation differs.

The command without the period(.) will initiate the project in a new directory. That means all the files will be

`

in an additional sub-directory. The later will initiate the project and keep all the files in the same directory.

80. Explain the usage of django.contrib.auth.get_user_model()?

Ans. The user model is the base model. Developers need it customized for their application. It should also contain all the security features that Django offers. Usually, they only want the inherited class and add or remove the field option there.

Now, get_user_model() will let us reference to user model. This method allows developers to use the custom user model. It is changed in setting AUTH_USER_MODEL in **settings.py**.
The **get_user_model()** allows the developer to implement the changes. It can reference the custom user model where needed while avoiding global implementation. Since Django's user model is tightly bundled with admin and auth. It is beneficial to customize Django's user model.

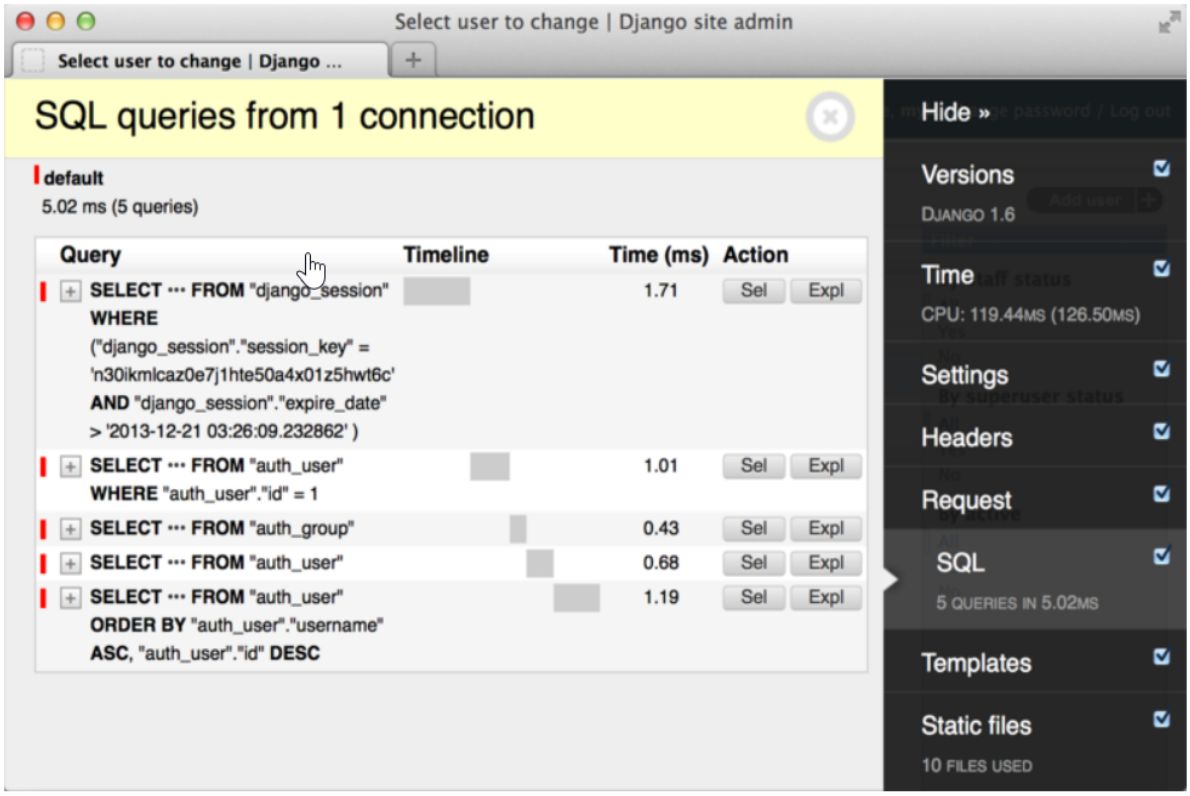
The get_user_model() comes in here and resolved that. The get_user_model() will return an active user or a custom user. In case none of the two are present, it will return the user object.

81. Why do we need performance benchmarking? What is a tool that can be used to do the same?

Ans. The performance measurement tools is thus an important part. If you want to improve the performance of your web platform, it needs monitoring and analysis. Performance-benchmarking tools help us analyze website performance.

For Django, there is a great tool available: **django-debug-toolbar**. It will monitor and provide you with various panels.

`



These panels show information for very niche things. Like the SQL panel here will show how much time it took to execute the SQL. The tool is significant and can give you leads to where you can improve the performance. This is currently the no.1 benchmarking tool which integrates easily with Django.

This is an internal tool. One, that needs to be included in code. There are other ways of performance benchmarking. There are third-party services like Yahoo's Yslow and Google PageSpeed. They will analyze the web platform as an HTTP Client. They give the performance as experienced by the user. These are free and many more services are available both free and paid.

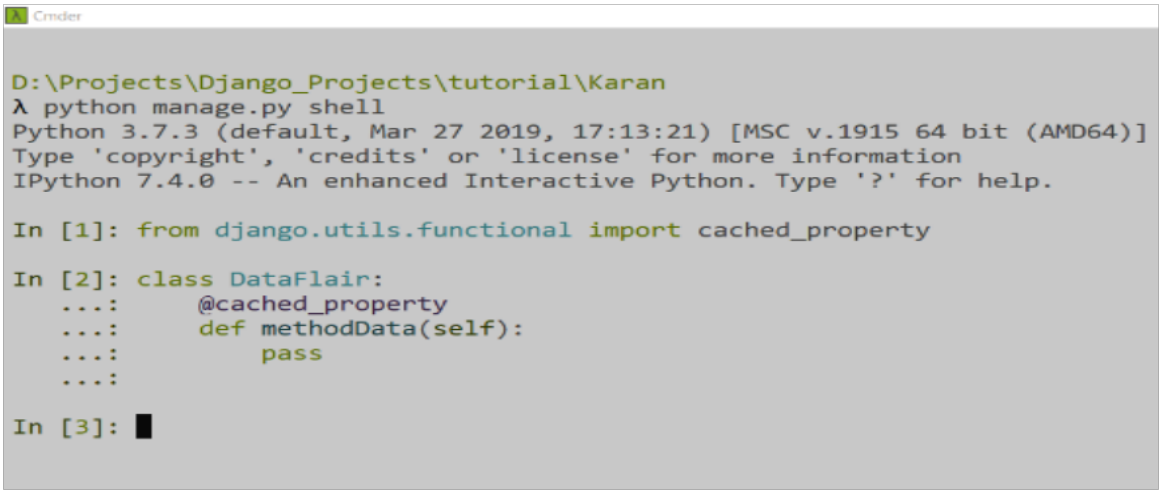
82. What is the use of cached_property() decorator?

Ans. The **cached_property()** decorator is a built-in decorator. This decorator comes under the **django.utils.functional** module. The decorator's property is to take the method as an argument. The method's result is cached by the decorator for that instance.

`

This is useful when a method is computationally expensive. Also, its generated value is valid for a while or for the session.

We can use this decorator just before we define the computationally expensive method. That method's answer is cached for the instance.



Replace the pass with functional code. Also, import the method from **django.utils.functional**. These modules are provided by Django to handle tedious tasks.

These also support backward compatibility. The utils module of Django is a very powerful set of tools.

83. What is the concept of laziness? How Django implements laziness?

Ans. Laziness is a complement of caching. We use both the processes to improve the efficiency of the program.

We ***implement caching*** by storing the result of a computationally expensive result. But, we implement laziness by not computing it until it's actually required. The concept of laziness is doing things when it's necessary. This approach has its own benefits. The more expensive the computation, the more we avoid it. It does work when it becomes a necessity.

Django is quite lazy. The most used example of laziness is querysets. We have used querysets quite often. The queryset objects are passed around even before they have the value. This approach is where we pass objects. Defining methods before the definition of its data, we call it laziness. It makes very easy for us to interact with databases. Python support and encourages laziness. That makes it up for its performance

`

issues many times. It's a concept made by lazy people implemented in computers.

84. Django provides various optimizations for static files. Explain one of those solutions.

Ans. Static files are one of the main contents consuming the bandwidth of the client. There are times when a low network speed results in a poor experience for the consumer. There are various solutions to deal with this in Django.

Django developers came up with a solution using the browser's caching behavior.
The **ManifestStaticFilesStorage** class is the implementation of the solution.

The ManifestStaticFilesStorage class uses the browser caching ability. Using this class will improve site performance at low network speeds. The class when serving the static file appends a content-dependent tag with the file. The tag lets the browser store the file for a longer time. Django changes tag only when there is some change in the file. Otherwise, the browser uses the stored file.

The ManifestStaticFilesStorage serves the files with the names added with the MD5 Hash. Django generates the Hash from the content of the files. For example:

The filename is **DataFlairStyle.css**. The file stored by ManifestStaticFilesStorage Class is **DataFlairStyle.55e7cbb9ba48.css**.

Django serves the custom file to the browsers. Then if the file content changes do the MD5 hash. Thus, the browser will know which file to download. Browsers mostly cache the static files. It's easier to just change the name if anything is changed in the backend. The browser will download it again if the file name changes.

That's how we can improve the static files serving.

85. Explain the management commands of django.contrib.staticfiles app.

Ans. The staticfiles app comes with three useful management commands. The commands are:

1. collectstatic
2. findstatic
3. runserver –nostatic

We can execute the collectstatic command like this:

1. python manage.py collectstatic

As its name suggests, this command will collect all the static files from the directories. Django scans the directories in the setting STATICFILES_DIRS. All the static files found are then collected in directory STATIC_ROOT.

1. python manage.py findstatic staticfile [staticfile]

`

This command will return the path of static files found. The arguments contain the file name. We can append the filename within the directory to make the search more efficient.

Django searches files in directories set in STATICFILES_DIRS setting.

1. python manage.py runserver

This command can be confusing. The runserver is not a staticfiles management command by default. If the django.contrib.staticfiles is installed in Django, the runserver command is overridden.

The runserver command gets various options like:

1. **–nostatic :** Serve requests without loading static files.
2. **–insecure :** This command will force the app to serve static files even if the debug command is off.

These are the 3 management commands which we get with staticfiles app in django.

86. What is Pagination?

Ans. Pagination is a concept where we separate or divide data into different pages. It's divided into multiple pages. The pages are provided as per user-request.

In the context of web applications: Suppose you search for anything on Google. Though Google claims they have found thousands of results in the resulting page, they give you only 10-20 results. Then you press the next page and you get more results. That is pagination.

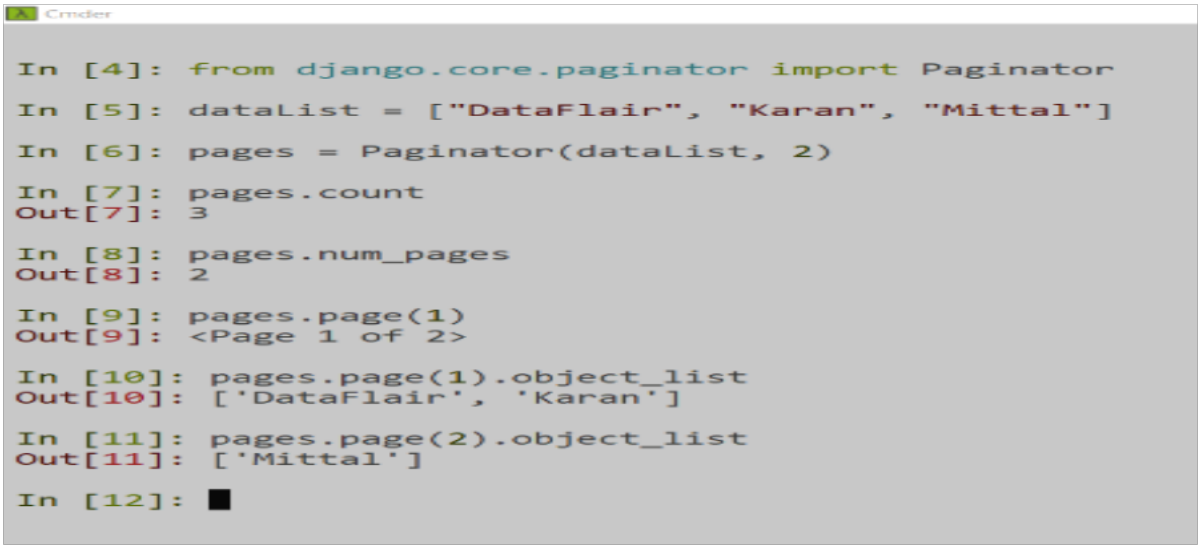
Showing user only limited data is good for both of them. The user can sort the results easily and smoothly reach the information he/she wants. The bandwidth cost is not much for some results. Since the transfer-data is small, it gives results on low network speeds.

Pagination is also good for the server. The server can simply store the data in queryset or database and wait for the client to request more. This reduces the template overhead which would have been to generate a response. Generating a response with 1000 data is much more expensive than 10-20 data. Pagination can drastically improve performance if done the right way.

87. How do we implement pagination in Django?

Ans. We implement pagination in Django with the help of **django.core.paginator** classes. These classes help us manage and use paginated data.

`



From the code above, we have a list named **datalist**. That list is then divided into 2 pages. We can think of pages as numbered elements or parts of the object. The division occurred when we pass the list in the class constructor. The second parameter specifies the number of objects in one page. The pagination can be directly implemented in views. The first argument can be a list, tuple or a Queryset.

In lines 7 and 8, we used some built-in attributes of Paginator class. The **count** will give the number of objects. It is the total number of objects. The **num_pages** will provide the number of pages made.

In this image, we are accessing the page data. We can access the page data with **pages.page_range**. This function works the way as a range in for loop. We can use object_list to access the contents on the page. It gives the list of objects. It's not necessary though that the data need to be a list. There are more functions in pagination.

88. Explain password management in Django? In what format, Django stores passwords?

Ans. Django has various built-in tools to manage and store passwords. The way Django store passwords are encryption through various levels. Django uses the **PBKDF2(Password Based Derivation Function 2)** which is an industry-standard. The function results are also iterated over. Our passwords are finally stored in a way like this:

<algorithm>$<iterations>$<salt>$<hash>

This is the format in which Django stores a password. We specify the algorithm used before the dollar $ sign. The second part contains the number of iterations. The number of iterations over the algorithm

`

is generally more than a million times. A random salt is included after that. The hash of the resulting password is the last part of the string.

The default hash used by Django is SHA256. NIST recommends these standards. Django's password system is customizable. We can give it the customized algorithms but, in most cases, it's not preferred. Default implementations are what most websites ever need. The default security is already very high.

89. How does Django handle weak passwords? How it implements password validation?

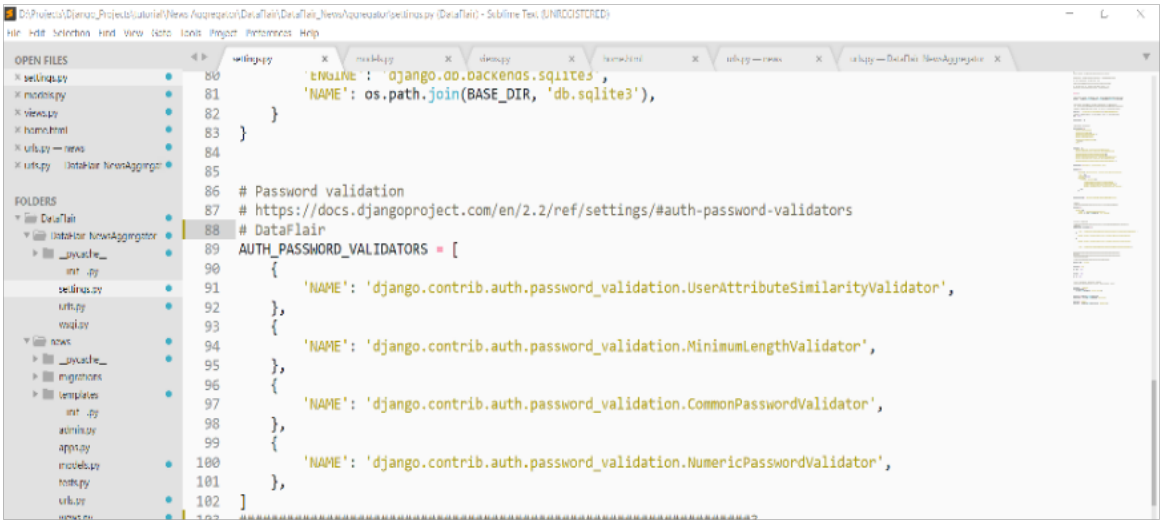
Ans. Password validation is another very useful feature in Django. People often resort to using weak passwords. To prevent them from doing so, Django provides a solution. Django's password validation system is a part of **django.contrib.auth** application.

The Django has PASSWORD_VALIDATORS for this purpose. The password validators will check the passwords before saving it. The validators like it must contain 8 characters. They are the validators that check the data like validating all characters are not digits. There are even validators that check passwords among the very common ones. For example passwords like **test123, hello123,** etc raise validation error.

We can even write our custom password validators in the case. The custom validators can even combine the user-information and test password. They can tell whether passwords are a combination of user-information or not.

There are various ways. The main requirement by any validator is its description. The validator must provide a description to the user as to what is wrong with the password.

Django handles password validation by the **AUTH_PASSWORD_VALIDATORS** setting.(default)



`

90. Explain some built-in validators in Django.

Ans. The validator in Django checks the value. When the input value doesn't meet certain constraints, Django raises **ValidationError**. **is_valid()** method handles ValidationError. The method then returns the value as **False**. Thus, we can ascertain whether the Form is providing valid data or not.

There are various built-in validators. We can access them using **django.core.validators** module. This module has a collection of validators for different fields in Django.

Some of the common validators we use include:

1. EmailValidator: This validator is the default one checking emails in models. This validator checks whether the input email is valid or not. If the email is not valid, it returns so with a message.

An important attribute of email validation is **whitelist**. The whitelist attribute of EmailValidator class contains a list of domains. Django will accept these domains regardless if they follow the general email conventions. The default value of that list includes localhost.

2. URLValidator: The URLValidator ensures that the inserted value looks like a URL. Now, it has some set of regular expressions. If the string passes that it shall be a URL. It doesn't dial the URL in the browser actually accessing the link. If the validator doesn't return anything, we get an **invalid** code.

The **scheme** parameter is important here. You can specify the scheme of URLs like **http, https, FTP,** etc. The accepted URLs shall have these schemes or protocols. This is default value in the scheme parameter.

3. MaxValueValidator: As the name suggests, the maximum value that a field accepts. It checks the value, not length.

4. MinLengthValidator: This validator will check the length of the input. It raises ValidationError if the error exceeds a certain length.

There are other validators too, as Django provides a lot. We can use various parameters according to use. Using Django validators is always better than using custom validators. They provide a standard validation and are well integrated with the rest of Django. Although, writing custom validators is as easy.

91. What are the various steps involved in form validation in Django?

Ans. The ***form validation*** is a complex process comprising several steps. An end result is a form where all the data is either partially or fully clean. The form can also be completely wrong. Form validation is the process of validating data before performing any database operations.

The steps involved in form validation include:

1. The first method executed is **to_python()** method. It will convert the data into the appropriate Python type to perform validation.

`

2. The second method is **validate()** method. It handles field-specific validation.
3. The third method is **run_validators()** method. It calls all the field's validators. The collective errors are then returned by this method.

These three methods are actually executed in order. The **clean()** method calls all three methods in the appropriate order. This is a form class method. It calls all three **to_python(),**
validate() & **run_validators()** methods. The comprised results and validation errors are returned.A

That's how we get a cleaned form or validated form.

92. What are some good practices when writing custom validators? How do we raise ValidationErrors as recommended by Django?

Ans. ***Raising errors in Django*** shows how well planned a web-developer can be. There are certain ways recommended by Django to raise ValidationErrors.

1. The ValidationErrors shall be descriptive and easy to comprehend.
2. The error code shall be a descriptive one.
3. Django encourages the use of the **params parameter** when returning variables.
4. The **params** argument shall contain mapped data. Dictionaries are used in these cases.
5. We use the gettext function or **(_('text'))** to enable translation.

Example Code:

```
In [4]: from django.forms import ValidationError

In [5]: raise ValidationError(
   ...:  _("DataFlair %(error)s"),
   ...:  code="DataFlair-raised-error",
   ...:  params={"error":"The raised error"},
   ...:  )
```

We have implemented all the concepts while raising validation error. A well thought validation error helps you better control the flow.

93. What is the code layout for creating custom templates and filters?

Ans. Custom template tags are often used by programmers if the Django has a built-in tag. The custom template system is often defined in a different application. They are then made available to other ***Django templates***.

The code layout works like this.

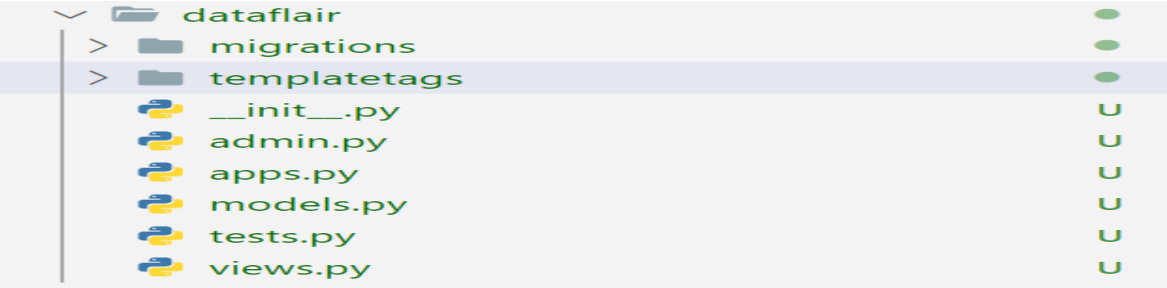
`

1. Make a new app. Install that application in the Django project.

Code:

1. python manage.py startapp dataflair

2. Make a new directory **templatetags** at the same level as **views.py, models.py,** etc…



3. Inside the **templatetags** directory, make a new file called **__init__.py**

4. Then write your template tags or custom logic in a Python file. You can name it anything relevant to custom templates.

5. The custom templates can be used by other templates. We use **{% load "module-name" %}** to load the templates.

The **load** tag will load all the custom templates defined in the file **customDataFlair**.

{% load customDataFlair %}

`

The template tags from the module will work if this is imported. There can be multiple load statements in the same template.

94. What is the use of {autoescape} tag in DTL?

Ans. HTML escaping is important and developers should keep that in mind while coding. The HTML escaping means escaping character like “, &, <, >. The browser engine can interpret these characters as HTML Mark-up. These can sometimes lead to glitches. Also, in the worst scenario, the user gets a broken site. To prevent that to happen, we use HTML escaping. The Django developers have come up with a solution.

The solution to perform HTML escaping built-in is achieved by **autoescape** tag in DTL. The tag will identify the problem characters and replace them with escape characters. **For example:** “ represents **"**. Other characters too have their escape sequences.

The **{% autoescape %}** tag has one argument. The argument can be on or off. That indicated whether auto-escaping is on or off. The code resides between **{% autoescape %}** and **{% endautoescape %}** tags.

1. {% autoescape on %}
2. {{ dataflair.value }}
3. {% endautoescape %}

This code will automatically look for any escape characters in **dataflair.value** variable. Then it will replace the escape characters if found.

This function can help us achieve a valid and bugless HTML. Django encourages the use of autoescape character.

95. What are formsets in Django?

Ans. Formsets provide a layer of abstraction to developers. They enable developers to work with multiple forms on the same page. The formsets follow the analogy of datagrid. Django provides various functions in the forms module for this purpose.

Suppose we have a class.

`

```
In [16]: from django import forms

In [17]: class DataFlairForm(forms.Form):
    ...:     title = forms.CharField()
    ...:     author = forms.CharField()
    ...:     pub_date = forms.DateField()
    ...:

In [18]: from django.forms import formset_factory

In [19]: DjangoObjects = formset_factory(DataFlairForm)

In [20]: for form in DjangoObjects():
    ...:     print(form)
    ...:
<tr><th><label for="id_form-0-title">Title:</label></th><td><input type="text" name="form-0-title" id="id_form-0-title"></td></tr>
<tr><th><label for="id_form-0-author">Author:</label></th><td><input type="text" name="form-0-author" id="id_form-0-author"></td></tr>
<tr><th><label for="id_form-0-pub_date">Pub date:</label></th><td><input type="text" name="form-0-pub_date" id="id_form-0-pub_date"></td
></tr>

In [21]:
```

If we want the user to create several objects at once, we use formsets. Django provides a class **formset_factory** for this purpose.

We can make our form a formset as shown in the line 19.

Now, we have made DjangoObjects a formset of **DataFlairForm** form. The **formset_factory** is a class present to help us in doing so.

Now, we can make an instance of this variable. The instance of the article will provide us with a list of forms. The for loop can iterate over that list. We can get multiple forms on a single page using this. You can see the output on line 20.

96. How do you listen to signals in Django?

Ans. We use signals to transmit messages between applications. The messages will give notification of an

`

event. The Signals package provides various functions to make signals.

There are also functions to receive or listen to signals.

To receive a signal we use Signal.connect() method in an app. The method has some important parameters.

The signal will call receiver function when connect() function gets a message. The message signifies the occurrence of event. Django handles the signaling and activating of all these things.

The connect() method takes 4 parameters:

1. receiver
2. sender
3. weak
4. dispatch_uid

The receiver refers to a call back function. Django calls the receiver function when it gets the signal.

The sender parameter takes a specific sender. A particular sender is set to receive signals from. The weak attribute takes two values, True or False. Django stores signals as weak references. Garbage collectors handle weak references. When the receiver function is local, it can be garbage collected. To prevent this, weak is set to false.

The last parameter is a unique id. We use it to distinguish between duplicate signal receivers.

Setting these parameters in a function will make the signal receiver. The receiver can then execute a function when an event occurs.

1. How to find the query associated with a queryset?

Sometime you want to know how a Django ORM makes our queries execute or what is the corresponding SQL of the code you are writing. This is very strightforward. Youn can get str of any queryset.query to get the sql.

You have a model called Event. For getting all records, you will write something like Event.objects.all(), then do str(queryset.query)

```
>>> queryset = Event.objects.all()
>>> str(queryset.query)
SELECT "events_event"."id", "events_event"."epic_id",
   "events_event"."details", "events_event"."years_ago"
   FROM "events_event"
```

```
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> queryset = Event.objects.all()
>>> print(queryset.query)
SELECT "events_event"."id", "events_event"."epic_id", "events_event"."details",
>>>
```

Example 2

```
>>> queryset = Event.objects.filter(years_ago__gt=5)
>>> str(queryset.query)
SELECT "events_event"."id", "events_event"."epic_id", "events_event"."details",
"events_event"."years_ago" FROM "events_event"
WHERE "events_event"."years_ago" > 5
```

Next Previous

---

2. How to do OR queries in Django ORM?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

If you are using django.contrib.auth, you will have a table called auth_user. It will have fields as username, first_name, last_name and more.

A common requirement is performing OR filtering with two ore more conditions. Say you want find all users with firstname starting with 'R' and last_name starting with 'D'.

Django provides two options.

- queryset_1 | queryset_2
- filter(Q(<condition_1>)|Q(<condition_2>)

2.1. The query in detail

The SQL query for the above condition will look something like

```
SELECT username, first_name, last_name, email FROM auth_user WHERE first_name LIKE 'R%'
OR last_name LIKE 'D%';
```

	username	first_name	last_name	email
1	Ricky	Ricky	Dayal	ricky@gmail.com
2	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
3	Radha	Radha	George	radha@gmail.com
4	Raghu	Raghu	Khan	raghu@rediffmail.com
5	rishab	Rishabh	Deol	rishabh@yahoo.com

Similarly our ORM query would look like

```
queryset = User.objects.filter(
      first_name__startswith='R'
   ) | User.objects.filter(
   last_name__startswith='D'
)
queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: Radha>, <User: Raghu>, <User: rishab>]>
```

You can also look at the generated query.

```
In [5]: str(queryset.query)
Out[5]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
"auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name",
"auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff",
"auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user"
WHERE ("auth_user"."first_name"::text LIKE R% OR "auth_user"."last_name"::text LIKE D%)'
```

Alternatively, you can use the Q objects.

```
from django.db.models import Q
qs = User.objects.filter(Q(first_name__startswith='R')|Q(last_name__startswith='D'))
```

If you look at the generated query, the result is exactly the same

```
In [9]: str(qs.query)
Out[9]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
 "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name",
  "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff",
  "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user"
  WHERE ("auth_user"."first_name"::text LIKE R% OR "auth_user"."last_name"::text LIKE D%)'
```

3. How to do AND queries in Django ORM?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

If you are using django.contrib.auth, you will have a table called auth_user. It will have fields as username, first_name, last_name and more.

You would frequently need to want to perform AND operation, to find querysets which match multiple criteria.

Say you want to find users with firstname starting with 'R' AND last_name starting with 'D'.

Django provides three options.

- filter(<condition_1>, <condition_2>)
- queryset_1 & queryset_2
- filter(Q(<condition_1>) & Q(<condition_2>))

3.1. The query in detail

Our SQL query for the above condition will look something like

```
SELECT username, first_name, last_name, email FROM auth_user WHERE first_name LIKE 'R%' AND last_name LIKE 'D%';
```

	username	first_name	last_name	email
1	Ricky	Ricky	Dayal	ricky@gmail.com
2	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
3	rishab	Rishabh	Deol	rishabh@yahoo.com

The default way to combine multiple conditions in filter is AND, so you can just do.

```
queryset_1 = User.objects.filter(
   first_name__startswith='R',
   last_name__startswith='D'
)
```

Alternatively, you can explicitly use the *&* operator on querysets.

```
queryset_2 = User.objects.filter(
   first_name__startswith='R'
) & User.objects.filter(
   last_name__startswith='D'
)
```

For complete customisability, you can use the Q objects.

```
queryset_3 = User.objects.filter(
   Q(first_name__startswith='R') &
   Q(last_name__startswith='D')
)

queryset_1
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: rishab>]>
```

You can look at the generated query and verify that they are all same.

```
In [10]: str(queryset_2.query)
Out[10]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
"auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name",
```

```
"auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active",
"auth_user"."date_joined" FROM "auth_user" WHERE ("auth_user"."first_name"::text LIKE R% AND
"auth_user"."last_name"::text LIKE D%)'

In [11]: str(queryset_1.query) == str(queryset_2.query) == str(queryset_3.query)
Out[11]: True
```

4. How to do a NOT query in Django queryset?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

If you are using django.contrib.auth, you will have a table called auth_user. It will have fields as username, first_name, last_name and more.

Say you want to fetch all users with id NOT < 5. You need a NOT operation.

Django provides two options.

- exclude(<condition>)
- filter(~Q(<condition>))

4.1. The query in detail

Our SQL query for the above condition will look something like

```
SELECT id, username, first_name, last_name, email FROM auth_user WHERE NOT id < 5;
```

	id	username	first_name	last_name	email
1	5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
2	6	Billy	Billy	Sharma	billy@gmail.com
3	7	Radha	Radha	George	radha@gmail.com
4	8	sohan	Sohan	Upadhyay	sohan@aol.com
5	9	Raghu	Raghu	Khan	raghu@rediffmail.com
6	10	rishab	Rishabh	Deol	rishabh@yahoo.com

Method 1 using exclude

Method 2 using Q() method

```
>>> from django.db.models import Q
>>> queryset = User.objects.filter(~Q(id__lt=5))
>>> queryst
<QuerySet [<User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>, <User: rishab>]>
```

5. How to do union of two querysets from same or different models?

The UNION operator is used to combine the result-set of two or more querysets. The querysets can be from the same or from different models. When they querysets are from different models, the fields and their datatypes should match.

Let's continue with our auth_user model and generate 2 querysets to perform union operation

```
>>> q1 = User.objects.filter(id__gte=5)
>>> q1
<QuerySet [<User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>, <User: rishab>]>
>>> q2 = User.objects.filter(id__lte=9)
>>> q2
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>]>
>>> q1.union(q2)
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>, <User: rishab>]>
>>> q2.union(q1)
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>, <User: rishab>]>
```

Now try this

```
>>> q3 = EventVillain.objects.all()
>>> q3
<QuerySet [<EventVillain: EventVillain object (1)>]>
>>> q1.union(q3)
django.db.utils.OperationalError: SELECTs to the left and right of UNION do not have the same number of result columns
```

The union operation can be performed only with the querysets having same fields and the datatypes. Hence our last union operation encountered error. You can do a union on two models as long as they have same fields or same subset of fields.

Since Hero and Villain both have the name and gender, we can use values_list to limit the selected fields then do a union.

```
Hero.objects.all().values_list(
   "name", "gender"
).union(
Villain.objects.all().values_list(
   "name", "gender"
))
```

This would give you all Hero and Villain objects with their name and gender.

6. How to select some fields only in a queryset?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

The auth_user model has a number of fields in it. But sometimes, you do not need to use all the fields. In such situations, we can query only desired fields.

Django provides two ways to do this

- *values* and *values_list* methods on queryset.
- *only_method*

Say, we want to get first_name and last_name of all the users whose name starts with **R**. You do not want the fetch the other fields to reduce the work the DB has to do.

```
>>> queryset = User.objects.filter(
   first_name__startswith='R'
).values('first_name', 'last_name')
>>> queryset
<QuerySet [{'first_name': 'Ricky', 'last_name': 'Dayal'}, {'first_name': 'Ritesh', 'last_name':
'Deshmukh'}, {'first_name': 'Radha', 'last_name': 'George'}, {'first_name': 'Raghu', 'last_name':
'Khan'}, {'first_name': 'Rishabh', 'last_name': 'Deol'}]
```

You can verify the generated sql using str(queryset.query), which gives.

```
SELECT "auth_user"."first_name", "auth_user"."last_name"
FROM "auth_user" WHERE "auth_user"."first_name"::text LIKE R%
```

The output will be list of dictionaries.

Alternatively, you can do

```
>>> queryset = User.objects.filter(
    first_name__startswith='R'
).only("first_name", "last_name")
```

str(queryset.query), gives us

```
SELECT "auth_user"."id", "auth_user"."first_name", "auth_user"."last_name"
FROM "auth_user" WHERE "auth_user"."first_name"::text LIKE R%
```

The only difference between only and values is only also fetches the id.

7. How to do a subquery expression in Django?

Django allows using SQL subqueries. Let's start with something simple, We have a UserParent model which has OnetoOne relation with auth user. We will find all the UserParent which have a UserParent.

```
>>> from django.db.models import Subquery
>>> users = User.objects.all()
>>> UserParent.objects.filter(user_id__in=Subquery(users.values('id')))
<QuerySet [<UserParent: UserParent object (2)>, <UserParent: UserParent object (5)>,
<UserParent: UserParent object (8)>]>
```

Now for something more complex. For each Category, we want to find the most benevolent Hero.

The models look something like this.

```
class Category(models.Model):
    name = models.CharField(max_length=100)


class Hero(models.Model):
    # ...
    name = models.CharField(max_length=100)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)

    benevolence_factor = models.PositiveSmallIntegerField(
        help_text="How benevolent this hero is?",
        default=50
    )
```

You can find the most benevolent Hero like this

```
hero_qs = Hero.objects.filter(
    category=OuterRef("pk")
).order_by("-benevolence_factor")
Category.objects.all().annotate(
    most_benevolent_hero=Subquery(
        hero_qs.values('name')[:1]
    )
)
```

If you look at the generated sql, you will see

```
SELECT "entities_category"."id",
       "entities_category"."name",

  (SELECT U0."name"
   FROM "entities_hero" U0
   WHERE U0."category_id" = ("entities_category"."id")
   ORDER BY U0."benevolence_factor" DESC
   LIMIT 1) AS "most_benevolent_hero"
FROM "entities_category"
```

Let's break down the queryset logic. The first part is

```
hero_qs = Hero.objects.filter(
    category=OuterRef("pk")
).order_by("-benevolence_factor")
```

We are ordering the Hero object by benevolence_factor in DESC order, and using category=OuterRef("pk") to declare that we will be using it in a subquery.

Then we annotate with most_benevolent_hero=Subquery(hero_qs.values('name')[:1]), to get use the subquery with a Category queryset. The hero_qs.values('name')[:1] part picks up the first name from subquery.

8. How to filter a queryset with criteria based on comparing their field values

Django ORM makes it easy to filter based on fixed values. To get all User objects with first_name starting with 'R', you can do User.objects.filter(first_name__startswith='R').

What if you want to compare the first_name and last name? You can use the F object. Create some users first.

```
In [27]: User.objects.create_user(email="shabda@example.com", username="shabda",
first_name="Shabda", last_name="Raaj")
Out[27]: <User: shabda>

In [28]: User.objects.create_user(email="guido@example.com", username="Guido",
```

```
first_name="Guido", last_name="Guido")
Out[28]: <User: Guido>
```

Now you can find the users where first_name==last_name

```
In [29]: User.objects.filter(last_name=F("first_name"))
Out[29]: <QuerySet [<User: Guido>]>
```

F also works with calculated field using annotate. What if we wanted users whose first and last names have same letter?

You can set the first letter from a string using Substr("first_name", 1, 1), so we do.

```
In [41]: User.objects.create_user(email="guido@example.com", username="Tim",
first_name="Tim", last_name="Teters")
Out[41]: <User: Tim>
#...
In [46]: User.objects.annotate(first=Substr("first_name", 1, 1), last=Substr("last_name", 1,
1)).filter(first=F("last"))
Out[46]: <QuerySet [<User: Guido>, <User: Tim>]>
```

F can also be used with __gt, __lt and other expressions.

9. How to filter FileField without any file?

A FileField or ImageField stores the path of the file or image. At the DB level they are same as a CharField.

So to find FileField without any file we can query as under.

```
no_files_objects = MyModel.objects.filter(
    Q(file='')|Q(file=None)
)
```

13. How to find distinct field values from queryset?

id	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@gmail.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com
11	johny	John	Smith	john@example.com
12	paul	Paul	Jones	paul@example.com
13	johny1	John	Smith	johny@example.com

You want to find users whose names have not been repeated. You can do this like this

```
distinct = User.objects.values(
    'first_name'
).annotate(
    name_count=Count('first_name')
).filter(name_count=1)
records = User.objects.filter(first_name__in=[item['first_name'] for item in distinct])
```

This is different from User.objects.distinct("first_name").all(), which will pull up the first record when it encounters a distinct first_name.

10. How to perform join operations in django ORM?

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Join can be carried out in many ways. Some are shown below.

```
>>> a1 = Article.objects.select_related('reporter') // Using select_related
>>> a1
<QuerySet [<Article: International News>, <Article: Local News>, <Article: Morning news>,
<Article: Prime time>, <Article: Test Article>, <Article: Weather Report>]>
>>> print(a1.query)
SELECT "events_article"."id", "events_article"."headline", "events_article"."pub_date",
"events_article"."reporter_id", "events_article"."slug", "auth_user"."id", "auth_user"."password",
"auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username",
"auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff",
"auth_user"."is_active", "auth_user"."date_joined" FROM "events_article" INNER JOIN "auth_user"
ON ("events_article"."reporter_id" = "auth_user"."id") ORDER BY "events_article"."headline" ASC
>>> a2 = Article.objects.filter(reporter__username='John')
>>> a2
<QuerySet [<Article: International News>, <Article: Local News>, <Article: Prime time>, <Article:
Test Article>, <Article: Weather Report>]>
>>> print(a2.query)
```

```
SELECT "events_article"."id", "events_article"."headline", "events_article"."pub_date",
"events_article"."reporter_id", "events_article"."slug" FROM "events_article" INNER JOIN
"auth_user" ON ("events_article"."reporter_id" = "auth_user"."id") WHERE "auth_user"."username"
= John ORDER BY "events_article"."headline" ASC
```

11. How to find second largest record using Django ORM ?

You would across situations when you want to find second highest user depending on their age or salary.

Though the ORM gives the flexibility of finding first(), last() item from the queryset but not nth item. You can do it using the slice operator.

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

We can find Nth records from the query by using slice operator.

```
>>> user = User.objects.order_by('-last_login')[1] // Second Highest record w.r.t 'last_login'
>>> user.first_name
'Raghu'
>>> user = User.objects.order_by('-last_login')[2] // Third Highest record w.r.t 'last_login'
>>> user.first_name
'Sohan'
```

User.objects.order_by('-last_login')[2] only pulls up the required object from db using LIMIT ... OFFSET. If you look at the generated sql, you would see something like this.

```
SELECT "auth_user"."id",
    "auth_user"."password",
    "auth_user"."last_login",
    "auth_user"."is_superuser",
    "auth_user"."username",
    "auth_user"."first_name",
    "auth_user"."last_name",
    "auth_user"."email",
    "auth_user"."is_staff",
    "auth_user"."is_active",
```

```
    "auth_user"."date_joined"
FROM "auth_user"
ORDER BY "auth_user"."last_login" DESC
LIMIT 1
OFFSET 2
```

12. Find rows which have duplicate field values

id	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@gmail.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com
11	johny	John	Smith	john@example.com
12	paul	Paul	Jones	paul@example.com
13	johny1	John	Smith	johny@example.com

Say you want all users whose first_name matches another user.

You can find duplicate records using the technique below.

```
>>> duplicates = User.objects.values(
   'first_name'
   ).annotate(name_count=Count('first_name')).filter(name_count__gt=1)
>>> duplicates
<QuerySet [{'first_name': 'John', 'name_count': 3}]>
```

If you need to fill all the records, you can do

```
>>> records = User.objects.filter(first_name__in=[item['first_name'] for item in duplicates])
>>> print([item.id for item in records])
[2, 11, 13]
```

17. How to use arbitrary database functions in querysets?

Django comes with functions like Lower, Coalesce and Max, but it can't support all database functions, expecially ones which are database specific.

Django provides Func which allows using arbitrary database functions, even if Django doesn't

provide them.

Postgres has fuzzystrmatch, which provides several functions to determine similarities. Install the extension in your postgres DB with create extension fuzzystrmatch

We will use the levenshtein function. Lets first create some Hero objects.

```
Hero.objects.create(name="Zeus", description="A greek God", benevolence_factor=80,
category_id=12, origin_id=1)
Hero.objects.create(name="ZeuX", description="A greek God", benevolence_factor=80,
category_id=12, origin_id=1)
Hero.objects.create(name="Xeus", description="A greek God", benevolence_factor=80,
category_id=12, origin_id=1)
Hero.objects.create(name="Poseidon", description="A greek God", benevolence_factor=80,
category_id=12, origin_id=1)
```

We want to find out the Hero objects which have name similar to Zeus. You can do

```
from django.db.models import Func, F
Hero.objects.annotate(like_zeus=Func(F('name'), function='levenshtein',
template="%(function)s(%(expressions)s, 'Zeus')"))
```

The like_zeus=Func(F('name'), function='levenshtein', template="%(function)s(%(expressions)s, 'Zeus')") took two arguments which allowed the database representation,
viz, function and template. If you need to reuse the function, you can define a class like this.

```
class LevenshteinLikeZeus(Func):
    function='levenshtein'
    template="%(function)s(%(expressions)s, 'Zeus')"
```

And then use Hero.objects.annotate(like_zeus=LevenshteinLikeZeus(F("name")))

You can then filter on this annotated field like this.

```
In [16]: Hero.objects.annotate(
   ...:         like_zeus=LevenshteinLikeZeus(F("name"))
   ...:     ).filter(
   ...:         like_zeus__lt=2
   ...:     )
   ...:
Out[16]: <QuerySet [<Hero: Zeus>, <Hero: ZeuX>, <Hero: Xeus>]>
```

14. How to use Q objects for complex queries?

In previous chapters we used Q objects for OR and AND and NOT operations. Q objects provides you complete control over the where clause of the query.

If you want to OR your conditions.

```
>>> from django.db.models import Q
>>> queryset = User.objects.filter(
    Q(first_name__startswith='R') | Q(last_name__startswith='D')
)
>>> queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: Radha>, <User: Raghu>, <User: rishab>]>
```

If you want to AND your conditions.

```
>>> queryset = User.objects.filter(
    Q(first_name__startswith='R') & Q(last_name__startswith='D')
)
>>> queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: rishab>]>
```

If you want to find all users whose first_name starts with 'R', but not if the last_name has 'Z'

```
>>> queryset = User.objects.filter(
    Q(first_name__startswith='R') & ~Q(last_name__startswith='Z')
)
```

If you look at the generated query, you would see

```
SELECT "auth_user"."id",
       "auth_user"."password",
       "auth_user"."last_login",
       "auth_user"."is_superuser",
       "auth_user"."username",
       "auth_user"."first_name",
       "auth_user"."last_name",
       "auth_user"."email",
       "auth_user"."is_staff",
       "auth_user"."is_active",
       "auth_user"."date_joined"
FROM "auth_user"
WHERE ("auth_user"."first_name"::text LIKE R%
      AND NOT ("auth_user"."last_name"::text LIKE Z%))
```

You can combine the Q objects in more complex ways to generate complex queries.

15. How to group records in Django ORM?

Grouping of records in Django ORM can be done using aggregation functions like Max, Min, Avg, Sum. Django queries help to create, retrieve, update and delete objects. But sometimes we need to get aggregated values from the objects. We can get them by example shown below

```
>>> from django.db.models import Avg, Max, Min, Sum, Count
>>> User.objects.all().aggregate(Avg('id'))
{'id__avg': 7.571428571428571}
```

```
>>> User.objects.all().aggregate(Max('id'))
{'id__max': 15}
>>> User.objects.all().aggregate(Min('id'))
{'id__min': 1}
>>> User.objects.all().aggregate(Sum('id'))
{'id__sum': 106}
```

2. How to do OR queries in Django ORM?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

If you are using `django.contrib.auth`, you will have a table called `auth_user`. It will have fields as `username`, `first_name`, `last_name` and more.

A common requirement is performing `OR` filtering with two ore more conditions. Say you want find all users with firstname starting with 'R' and last_name starting with 'D'.

Django provides two options.

- `queryset_1 | queryset_2`
- `filter(Q(<condition_1>)|Q(<condition_2>)`

2.1. The query in detail

The SQL query for the above condition will look something like

```
SELECT username, first_name, last_name, email FROM auth_user WHERE first_name LIKE 'R%' OR last_name LIKE 'D%';
```

	username	first_name	last_name	email
1	Ricky	Ricky	Dayal	ricky@gmail.com
2	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
3	Radha	Radha	George	radha@gmail.com
4	Raghu	Raghu	Khan	raghu@rediffmail.com
5	rishab	Rishabh	Deol	rishabh@yahoo.com

Similarly our ORM query would look like

```
queryset = User.objects.filter(
      first_name__startswith='R'
   ) | User.objects.filter(
   last_name__startswith='D'
)
queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: Radha>, <User: Raghu>, <User: rishab>]>
```

You can also look at the generated query.

```
In [5]: str(queryset.query)
Out[5]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
"auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name",
"auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff",
"auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user"
WHERE ("auth_user"."first_name"::text LIKE R% OR "auth_user"."last_name"::text LIKE D%)'
```

Alternatively, you can use the `Q` objects.

```
from django.db.models import Q
qs = User.objects.filter(Q(first_name__startswith='R')|Q(last_name__startswith='D'))
```

If you look at the generated query, the result is exactly the same

```
In [9]: str(qs.query)
Out[9]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
 "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name",
  "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff",
  "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user"
  WHERE ("auth_user"."first_name"::text LIKE R% OR "auth_user"."last_name"::text LIKE D%)'
```

1. How to create multiple objects in one shot?

There are conditions when we want to save multiple objects in one go. Say we want to add multiple categories at once and we don't want to make many queries to the database. We can use bulk_create for creating multiple objects in one shot.

Here is an example.

```
>>> Category.objects.all().count()
2
>>> Category.objects.bulk_create(
   [Category(name="God"),
    Category(name="Demi God"),
    Category(name="Mortal")]
)
[<Category: God>, <Category: Demi God>, <Category: Mortal>]
>>> Category.objects.all().count()
5
```

bulk_create takes a list of unsaved objects.

2. How to copy or clone an existing model object?

There is no built-in method for copying model instances, it is possible to create new instance with all fields values copied.

If an instance is saved with instance's pk set to None, the instance is used to create a new record in the DB. That means every field other than the PK is copied.

```
In [2]: Hero.objects.all().count()
Out[2]: 4

In [3]: hero = Hero.objects.first()
In [4]: hero.pk = None
In [5]: hero.save()
In [6]: Hero.objects.all().count()
Out[6]: 5
```

5. How to perform truncate like operation using Django ORM?

Truncate statement in SQL is meant to empty a table for future use. Though Django doesn't provide a builtin to truncate a table, but still similar result can be achived using delete() method. For example:

```
>>> Category.objects.all().count()
7
>>> Category.objects.all().delete()
(7, {'entity.Category': 7})
>>> Category.objects.all().count()
0
```

This works, but this uses DELETE FROM ... SQL statement. If you have a large number of records, this can be quite slow. You can add a classmethod to Category if you want to enable truncate.

```
class Category(models.Model):
    # ...

    @classmethod
    def truncate(cls):
        with connection.cursor() as cursor:
            cursor.execute('TRUNCATE TABLE "{0}" CASCADE'.format(cls._meta.db_table))
```

Then you can call Category.truncate() to a real database truncate.

6. What signals are raised by Django during object creation or update?

Django provides signals which allows hooking into a model objects creation and deletion lifecycle. The signals provided by Django are

- pre_init
- post_init
- pre_save
- post_save
- pre_delete
- post_delete

Among these, the most commonly used signals are pre_save and post_save. We will look into them in detail.

6.1. Signals vs overriding .save

Since signals can be used for similar effects as overriding .save, which one to use is a frequent source of confusion. Here is when you should use which.

- If you want other people, eg. third party apps, to override or customize the object save behaviour, you should raise your own signals
- If you are hooking into the save behavior of an app you do not control, you should hook into the post_save or pre_save
- If you are customizing the save behaviour of apps you control, you should override save.

Lets take an example of a UserToken model. This a class used for providing authentication and should get created whenever a User is created.

```
class UserToken(models.Model):
    token = models.CharField(max_length=64)

    # ...
```

7. How to convert string to datetime and store in database?

We can convert a date-string and store it in the database using django in many ways. Few of them are discussed below. Lets say we have a date-string as “2018-03-11” we can not directly store it to our date field, so we can use some dateparser or python library for it.

```
>>> user = User.objects.get(id=1)
>>> date_str = "2018-03-11"
>>> from django.utils.dateparse import parse_date // Way 1
>>> temp_date = parse_date(date_str)
>>> a1 = Article(headline="String converted to date", pub_date=temp_date, reporter=user)
>>> a1.save()
>>> a1.pub_date
datetime.date(2018, 3, 11)
>>> from datetime import datetime // Way 2
```

```
>>> temp_date = datetime.strptime(date_str, "%Y-%m-%d").date()
>>> a2 = Article(headline="String converted to date way 2", pub_date=temp_date, reporter=user)
>>> a2.save()
>>> a2.pub_date
datetime.date(2018, 3, 11)
```

1. How to order a queryset in ascending or descending order?

Ordering of the queryset can be achieved by order_by method. We need to pass the field on which we need to Order (ascending/descending) the result. Query looks like this

```
>>> User.objects.all().order_by('date_joined') # For ascending
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>, <User: Billy>, <User: Radha>, <User: Raghu>, <User: rishab>, <User: johny>, <User: paul>, <User: johny1>, <User: alien>]>
>>> User.objects.all().order_by('-date_joined') # For descending; Not '-' sign in order_by method
<QuerySet [<User: alien>, <User: johny1>, <User: paul>, <User: johny>, <User: rishab>, <User: Raghu>, <User: Radha>, <User: Billy>, <User: Ritesh>, <User: sharukh>, <User: Ricky>, <User: John>, <User: yash>]>
```

You can pass multiple fields to order_by

```
User.objects.all().order_by('date_joined', '-last_login')
```

Looking at the SQL

```
SELECT "auth_user"."id",
    -- More fields
    "auth_user"."date_joined"
FROM "auth_user"
ORDER BY "auth_user"."date_joined" ASC,
     "auth_user"."last_login" DESC
```

2. How to order a queryset in case insensitive manner?

Whenever we try to do order_by with some string value, the ordering happens alphabetically and w.r.t case. Like

```
>>> User.objects.all().order_by('username').values_list('username', flat=True)
<QuerySet ['Billy', 'John', 'Radha', 'Raghu', 'Ricky', 'Ritesh', 'johny', 'johny1', 'paul', 'rishab', 'sharukh', 'sohan', 'yash']>
```

If we want to order queryset in case insensitive manner, we can do like this .

```
.. code-block:: ipython

>>> from django.db.models.functions import Lower
```

```
>>> User.objects.all().order_by(Lower('username')).values_list('username', flat=True)
<QuerySet ['Billy', 'John', 'johny', 'johny1', 'paul', 'Radha', 'Raghu', 'Ricky', 'rishab', 'Ritesh', 'sharukh', 'sohan', 'yash']>
```

Alternatively, you can annotate with Lower and then order on annotated field.

```
User.objects.annotate(
    uname=Lower('username')
).order_by('uname').values_list('username', flat=True)
```

4. How to order on a field from a related model (with a foreign key)?

You have two models, Category and Hero.

```
class Category(models.Model):
    name = models.CharField(max_length=100)

class Hero(models.Model):
    # ...
    name = models.CharField(max_length=100)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
```

You want to order Hero by category and inside each category by the Hero name. You can do.

```
Hero.objects.all().order_by(
    'category__name', 'name'
)
```

Note the double underscore(__) in 'category__name'. Using the double undertscore, you can order on a field from a related model.

If you look at the SQL.

```
SELECT "entities_hero"."id",
       "entities_hero"."name",
       -- more fields
FROM "entities_hero"
INNER JOIN "entities_category" ON ("entities_hero"."category_id" = "entities_category"."id")
ORDER BY "entities_category"."name" ASC,
         "entities_hero"."name" ASC
```

1. How to model one to one relationships?

One-to-one relationships occur when there is exactly one record in the first table that

corresponds to one record in the related table. Here we have an example where we know that each individual can have only one Biological parents i.e., Mother and Father. We already have auth user model with us, we will add a new model UserParent as described below.

```
from django.contrib.auth.models import User

class UserParent(models.Model):
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        primary_key=True,
    )
    father_name = models.CharField(max_length=100)
    mother_name = models.CharField(max_length=100)

>>> u1 = User.objects.get(first_name='Ritesh', last_name='Deshmukh')
>>> u2 = User.objects.get(first_name='Sohan', last_name='Upadhyay')
>>> p1 = UserParent(user=u1, father_name='Vilasrao Deshmukh', mother_name='Vaishali Deshmukh')
>>> p1.save()
>>> p1.user.first_name
'Ritesh'
>>> p2 = UserParent(user=u2, father_name='Mr R S Upadhyay', mother_name='Mrs S K Upadhyay')
>>> p2.save()
>>> p2.user.last_name
'Upadhyay'
```

The on_delete method is used to tell Django what to do with model instances that depend on the model instance you delete. (e.g. a ForeignKey relationship). The on_delete=models.CASCADE tells Django to cascade the deleting effect i.e. continue deleting the dependent models as well.

```
>>> u2.delete()
```

Will also delete the related record of UserParent.

2. How to model one to many relationships?

In relational databases, a one-to-many relationship occurs when a parent record in one table can potentially reference several child records in another table. In a one-to-many relationship, the parent is not required to have child records; therefore, the one-to-many relationship allows zero child records, a single child record or multiple child records. To define a many-to-one relationship, use *ForeignKey.*:

```
class Article(models.Model):
    headline = models.CharField(max_length=100)
    pub_date = models.DateField()
    reporter = models.ForeignKey(User, on_delete=models.CASCADE, related_name='reporter')

    def __str__(self):
        return self.headline

    class Meta:
        ordering = ('headline',)

>>> u1 = User(username='johny1', first_name='Johny', last_name='Smith',
email='johny@example.com')
>>> u1.save()
>>> u2 = User(username='alien', first_name='Alien', last_name='Mars',
email='alien@example.com')
>>> u2.save()
>>> from datetime import date
>>> a1 = Article(headline="This is a test", pub_date=date(2018, 3, 6), reporter=u1)
>>> a1.save()
>>> a1.reporter.id
13
>>> a1.reporter
<User: johny1>
```

If you try to assign an object before saving it you will encounter a ValueError

```
>>> u3 = User(username='someuser', first_name='Some', last_name='User',
email='some@example.com')
>>> Article.objects.create(headline="This is a test", pub_date=date(2018, 3, 7), reporter=u1)
Traceback (most recent call last):
...
ValueError: save() prohibited to prevent data loss due to unsaved related object 'reporter'.
>>> Article.objects.create(headline="This is a test", pub_date=date(2018, 3, 7), reporter=u1)
>>> Article.objects.filter(reporter=u1)
<QuerySet [<Article: This is a test>, <Article: This is a test>]>
```

The above queryset shows User u1 with multiple Articles. Hence One to Many.

3. How to model many to many relationships?

A many-to-many relationship refers to a relationship between tables in a database when a parent row in one table contains several child rows in the second table, and vice versa.

Just to make it more interactive, we will talk about a twitter app. By just using few fields and

ManyToMany field we can make a simple twitter app.

We basically have 3 basic things in Twitter, tweets, followers, favourite/unfavourite.

We have two models to make everything work. We are inheriting django's auth_user.:

```
class User(AbstractUser):
    tweet = models.ManyToManyField(Tweet, blank=True)
    follower = models.ManyToManyField(settings.AUTH_USER_MODEL, blank=True)
    pass

class Tweet(models.Model):
    tweet = models.TextField()
    favourite = models.ManyToManyField(settings.AUTH_USER_MODEL, blank=True,
related_name='user_favourite')

    def __unicode__(self):
        return self.tweet
```

What will the above model be able to do ?

```
1) User will able to follow/unfollow other users.
2) User will able to see tweets made by other users whom user is following.
3) User is able to favorite/unfavorite tweets.
```

Few operations using ManyToManyfield which can be done are:

```
>>> t1 = Tweet(tweet="I am happy today")
>>> t1.save()
>>> t2 = Tweet(tweet="This is my second Tweet")
>>> t2.save()
>>> u1 = User(username='johny1', first_name='Johny', last_name='Smith',
email='johny@example.com')
>>> u1.save()
>>> u2 = User(username='johny1', first_name='Johny', last_name='Smith',
email='johny@example.com')
>>> u2.save()
>>> u3 = User(username='someuser', first_name='Some', last_name='User',
email='some@example.com')
>>> u3.save()
```

We have created few tweets and few users, that didn't involve any use of M2M field so far. Lets continue linking them in next step.

```
>>> u2.tweet.add(t1)
>>> u2.save()
>>> u2.tweet.add(t2)
```

```
>>> u2.save()
// User can follow other users.
>>> u2.follow.add(u1)
>>> u2.save()
// Tweets are linked to the users. Users have folloewd each other. Now we can make users do favourite/unfavourite of the tweets.
>>> t1.favourite.add(u1)
>>> t1.save()
>>> t1.favourite.add(u3)
>>> t1.save()
// For removing any users vote
>>> t1.favourite.remove(u1)
>>> t1.save(
```

4. How to include a self-referencing ForeignKey in a model

Self-referencing foreign keys are used to model nested relationships or recursive relationships. They work similar to how One to Many relationships. But as the name suggests, the model references itself.

Self reference Foreignkey can be achived in two ways.

```
class Employee(models.Model):
    manager = models.ForeignKey('self', on_delete=models.CASCADE)

# OR

class Employee(models.Model):
    manager = models.ForeignKey("app.Employee", on_delete=models.CASCADE)
```

5. How to convert existing databases to Django models?

Django comes with a utility called inspectdb that can create models by introspecting an existing database. You can view the output by running this command

```
$ python manage.py inspectdb
```

Befor running this you will have to configure your database in the settings.py file. The result will be a file containing a model for each table. You may want to save that file

```
$ python manage.py inspectdb > models.py
```

The output file will be saved to your current directory. Move that file to the correct app and you have a good starting point for further customizations.

6. How to add a model for a database view?

A database view is a searchable object in a database that is defined by a query. Though a view doesn't store data, some refer to a views as "virtual tables," you can query a view like you can a table. A view can combine data from two or more table, using joins, and also just contain a subset of information. This makes them convenient to abstract, or hide, complicated queries.

In our SqliteStuio we can see 26 tables and no views.

Lets create a simple view.

```
create view temp_user as
   select id, first_name from auth_user;
```

After the view is created, we can see 26 tables and 1 view.

We can create its related model in our app, by managed = False and db_table="temp_user"

```
class TempUser(models.Model):
   first_name = models.CharField(max_length=100)

   class Meta:
      managed = False
      db_table = "temp_user"

// We can query the newly created view similar to what we do for any table.
>>> TempUser.objects.all().values()
<QuerySet [{'first_name': 'Yash', 'id': 1}, {'first_name': 'John', 'id': 2}, {'first_name': 'Ricky', 'id': 3},
{'first_name': 'Sharukh', 'id': 4}, {'first_name': 'Ritesh', 'id': 5}, {'first_name': 'Billy', 'id': 6},
{'first_name': 'Radha', 'id': 7}, {'first_name': 'Raghu', 'id': 9}, {'first_name': 'Rishabh', 'id': 10},
{'first_name': 'John', 'id': 11}, {'first_name': 'Paul', 'id': 12}, {'first_name': 'Johny', 'id': 13},
{'first_name': 'Alien', 'id': 14}]>
// You cannot insert new reord in a view.
>>> TempUser.objects.create(first_name='Radhika', id=15)
Traceback (most recent call last):
```

```
...
django.db.utils.OperationalError: cannot modify temp_user because it is a view
```

8. How to specify the table name for a model?

To save you time, Django automatically derives the name of the database table from the name of your model class and the app that contains it. A model's database table name is constructed by joining the model's "app label" – the name you used in manage.py startapp – to the model's class name, with an underscore between them.

We have two apps in our demo application i.e., entities and events so all the models in them will have app names as the prefixes followed by _ then the model name.

For renaming them we cab use db_table parameter

```
class TempUser(models.Model):
    first_name = models.CharField(max_length=100)
    . . .
    class Meta:
        db_table = "temp_user"
```

9. How to specify the column name for model field?

Naming of a column in the model can be achieved py passing a db_column parameter with some name. If we don't pass this parameter django creates a column with the field name which we give.

```
class ColumnName(models.Model):
    a = models.CharField(max_length=40,db_column='column1')
    column2 = models.CharField(max_length=50)

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

Above we can db_column has higher priority over field name. First column is named as column1 but not as a.

10. What is the difference between null=True and blank=True?

The default value of both null and blank is False. Both of these values work at field level i.e., whether we want to keep a field null or blank.

null=True will set the field's value to NULL i.e., no data. It is basically for the databases column value.

```
date = models.DateTimeField(null=True)
```

blank=True determines whether the field will be required in forms. This includes the admin and your own custom forms.

```
title = models.CharField(blank=True) // title can be kept blank. In the database ("") will be stored.
```

null=True blank=True This means that the field is optional in all circumstances.

```
epic = models.ForeignKey(null=True, blank=True)
// The exception is CharFields() and TextFields(), which in Django are never saved as NULL. Blank values are stored in the DB as an empty string ('').
```

Also there is a special case, when you need to accept NULL values for a BooleanField, use NullBooleanField.

11. How to use a UUID instead of ID as primary key?

Whenever we create any new model, there is an ID field attached to it. The ID field's data type will be Integer by default.

To make id field as UUID, there is a new field type UUIDField which was added in django version 1.8+.

Example

```
import uuid
from django.db import models

class Event(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    details = models.TextField()
    years_ago = models.PositiveIntegerField()

>>> eventobject = Event.objects.all()
>>> eventobject.first().id
'3cd2b4b0c36f43488a93b3bb72029f46'
```

12. How to use slug field with django for more readability?

Slug is a part of a URL which identifies a particular page on a website in a form readable by users. For making it work django offers us a slugfield. It can be implimented as under. We already had a model Article we will be adding slugfield to it to make it user readable.

```
from django.utils.text import slugify
class Article(models.Model):
    headline = models.CharField(max_length=100)
    . . .
    slug = models.SlugField(unique=True)

    def save(self, *args, **kwargs):
        self.slug = slugify(self.headline)
        super(Article, self).save(*args, **kwargs)
    . . .

>>> u1 = User.objects.get(id=1)
>>> from datetime import date
>>> a1 = Article.objects.create(headline="todays market report", pub_date=date(2018, 3, 6),
reporter=u1)
>>> a1.save()
// slug here is auto-generated, we haven't created it in the above create method.
>>> a1.slug
'todays-market-report'
```

Slug is useful because:

- it's human friendly (eg. /blog/ instead of /1/).
- it's good SEO to create consistency in title, heading and URL.

13. How to add multiple databases to the django application ?

The configuration of database related stuff is mostly done in settings.py file. So to add multiple database to our django project we need add them in DATABASES dictionary.

```
DATABASE_ROUTERS = ['path.to.DemoRouter']
DATABASE_APPS_MAPPING = {'user_data': 'users_db',
                         'customer_data':'customers_db'}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
```

```
    'users_db': {
        'NAME': 'user_data',
        'ENGINE': 'django.db.backends.postgresql',
        'USER': 'postgres_user',
        'PASSWORD': 'password'
    },
    'customers_db': {
        'NAME': 'customer_data',
        'ENGINE': 'django.db.backends.mysql',
        'USER': 'mysql_cust',
        'PASSWORD': 'root'
    }
}
```

With multiple databases it will be good to talk about Database Router. The default routing scheme ensures that if a database isn't specified, all queries fall back to the default database. Database Router defaults to [].

```
class DemoRouter:
    """
    A router to control all database operations on models in the
    user application.
    """
    def db_for_read(self, model, **hints):
        """
        Attempts to read user models go to users_db.
        """
        if model._meta.app_label == 'user_data':
            return 'users_db'
        return None

    def db_for_write(self, model, **hints):
        """
        Attempts to write user models go to users_db.
        """
        if model._meta.app_label == 'user_data':
            return 'users_db'
        return None

    def allow_relation(self, obj1, obj2, **hints):
        """
        Allow relations if a model in the user app is involved.
        """
        if obj1._meta.app_label == 'user_data' or \
           obj2._meta.app_label == 'user_data':
           return True
        return None
```

```
def allow_migrate(self, db, app_label, model_name=None, **hints):
    """
    Make sure the auth app only appears in the 'users_db'
    database.
    """
    if app_label == 'user_data':
        return db == 'users_db'
    return None
```

Respective models would be modified as

```
class User(models.Model):
    username = models.Charfield(ax_length=100)
    . . .
        class Meta:
        app_label = 'user_data'

class Customer(models.Model):
    name = models.TextField(max_length=100)
    . . .
        class Meta:
        app_label = 'customer_data'
```

Few helpful commands while working with multiple databases.

```
$ ./manage.py migrate --database=users_db
```

1. How to assert that a function used a fixed number of queries?

We can count number of queries for testing by using assertNumQueries() method.

```
def test_number_of_queries(self):
    User.objects.create(username='testuser1', first_name='Test', last_name='user1')
    # Above ORM create will run only one query.
    self.assertNumQueries(1)
    User.objects.filter(username='testuser').update(username='test1user')
    # One more query added.
    self.assertNumQueries(2)
```

2. How to speed tests by reusing database between test runs?

When we execute the command python manage.py test, a new db is created everytime. This doesn't matter much if we don't have many migrations.

But when we have many migrations, it takes a long time to recreate the database between the test runs. To avoid such situations, we may reuse the old database.

You can prevent the test databases from being destroyed by adding the --keepdb flag to the test command. This will preserve the test database between runs. If the database does not exist, it will first be created. If any migrations have been added since the last test run, they will be applied in order to keep it up to date.

```
$ python manage.py test --keepdb
```

3. How to reload a model object from the database?

Models can be reloaded from the databse using refresh_from_db() method. THis proves helpful during testing. For example.

```
class TestORM(TestCase):
    def test_update_result(self):
        userobject = User.objects.create(username='testuser', first_name='Test', last_name='user')
        User.objects.filter(username='testuser').update(username='test1user')
        # At this point userobject.val is still testuser, but the value in the database
        # was updated to test1user. The object's updated value needs to be reloaded
        # from the database.
        userobject.refresh_from_db()
        self.assertEqual(userobject.username, 'test1user')
```

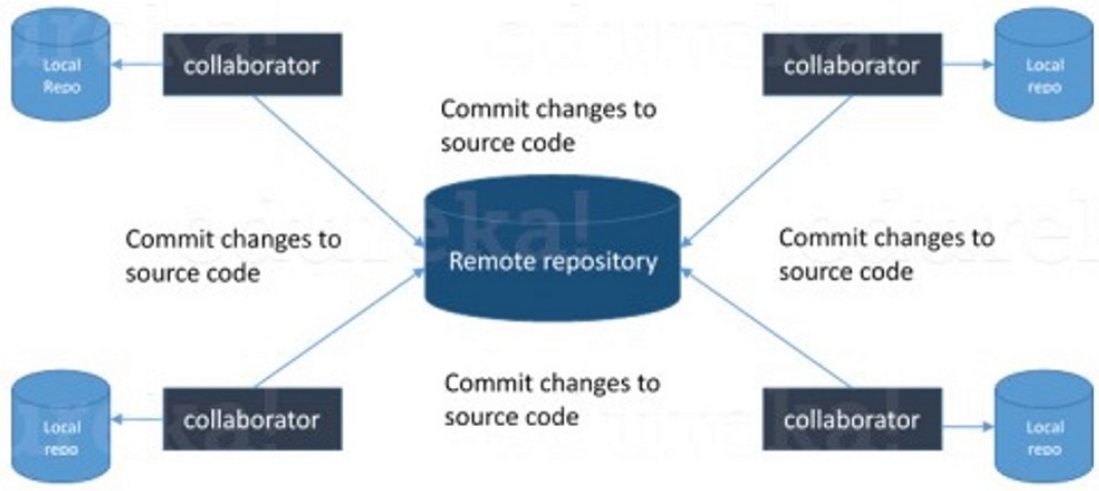
1. What is the difference between Git and SVN?

Git	SVN
Git is a Decentralized Version Control tool	SVN is a Centralized Version Control tool
It belongs to the 3rd generation of Version Control tools	It belongs to the 2nd generation of Version Control tools
Clients can clone entire repositories on their local systems	Version history is stored on a server-side repository
Commits are possible even if offline	Only online commits are allowed
Push/pull operations are faster	Push/pull operations are slower
Works are shared automatically by commit	Nothing is shared automatically

2. What is Git?

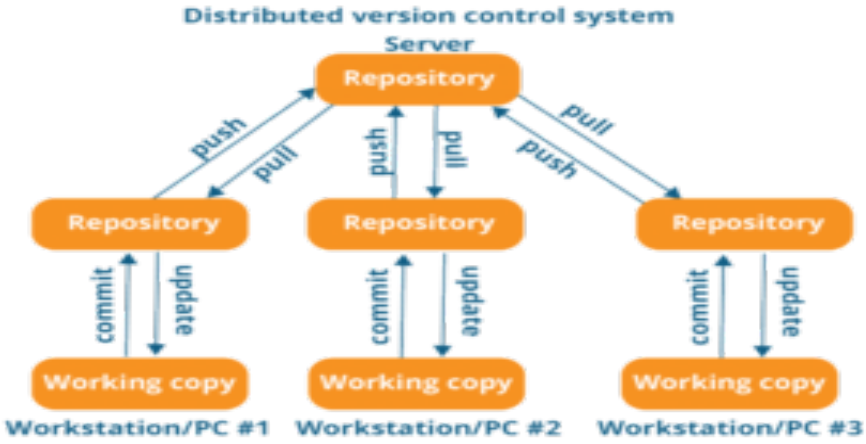
I will suggest you attempt this question by first telling about the architecture of git as shown in the below diagram just try to explain the diagram by saying:

- Git is a Distributed Version Control system(DVCS). It lets you track changes made to a file and allows you to revert back to any particular change that you wish.
- It is a distributed architecture that provides many advantages over other Version Control Systems (VCS) like SVN. One of the major advantages is that it does not rely on a central server to store all the versions of a project's files.
- Instead, every developer "clones" a copy of a repository I have shown in the diagram with "Local repository" and has the full history of the project available on his hard drive. So when there is a server outage all you need to do to recover is one of your teammate's local Git repository.
- There is a central cloud repository where developers can commit changes and share them with other teammates.



3. What is a distributed VCS?

- These are the systems that don't rely on a central server to store a project file and all its versions.
- In Distributed VCS, every contributor can get a local copy or "clone" of the main repository.
- As you can see in the above diagram, every programmer can maintain a local repository which is actually the copy or clone of the central repository which is present on their hard drive. They can commit and update their local repository without any hassles.
- With an operation called "pull", they can update their local repositories with new data from the central server and "pull" operation affects changes to the main repository from their local repository.



4. What is the difference between Git and Github?

Git is a version control system of distributed nature that is used to track changes in source code during software development. It aids in coordinating work among programmers, but it can be used to track changes in any set of files. The main objectives of Git are speed, data integrity, and support for distributed, non-linear workflows.

GitHub is a Git repository hosting service, plus it adds many of its own features. GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, basic task management tools for every project.

5. What are the benefits of using Version Control System?

- With the Version Control System(VCS), all the team members are allowed to work freely on any file at any time. VCS gives you the flexibility to merge all the changes into a common version.
- All the previous versions and variants are neatly packed up inside the VCS. You can request any version at any time as per your requirement and you'll have a snapshot of the complete project right at hand.
- Whenever you save a new version of your project, your VCS requires you to provide a short description of the changes that you have made. Additionally, you can see what changes are made in the file's content. This helps you to know what changes have been made in the project and by whom.
- A distributed VCS like Git allows all the team members to have a complete history of the project so if there is a breakdown in the central server you can use any of your teammate's local Git

repository.

6. What language is used in Git?

Instead of just telling the name of the language, you need to tell the reason for using it as well. I will suggest you to answer this by saying:

Git uses 'C' language. GIT is fast, and 'C' language makes this possible by reducing the overhead of run times associated with high-level languages.

7. Mention the various Git repository hosting functions.

- Github, Gitlab, Bitbucket, SourceForge, GitEnterprise

8. What is a commit message?

The command that is used to write a commit message is "**git commit -a**".
Now explain about -a flag by saying -a on the command line instructs git to commit the new content of all tracked files that have been modified. Also, mention you can use "**git add <file>**" before git commit -a if new files need to be committed for the first time.

9. How can you fix a broken commit?

In order to fix any broken commit, use the command "git commit --amend". When you run this command, you can fix the broken commit message in the editor.

10. What is a repository in Git?

Repository in Git is a place where Git stores all the files. Git can store the files either on the local repository or on the remote repository.

11. How can you create a repository in Git?

This is probably the most frequently asked question and the answer to this is really simple.

To create a repository, create a directory for the project if it does not exist, then run the command "**git init**". By running this command .git directory will be created in the project directory.

12. What is 'bare repository' in Git?

A "bare" repository in Git contains information about the version control and no working files (no tree) and it doesn't contain the special .git sub-directory. Instead, it contains all the contents of the .git sub-directory directly in the main directory itself, whereas the working directory consists of:

1. A .git subdirectory with all the Git related revision history of your repository.
2. A working tree, or checked out copies of your project files.

13. What is a 'conflict' in git?

Git can handle on its own most merges by using its automatic merging features. There arises a conflict when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts are most likely to happen when working in a team environment.

14. How is git instaweb used?

'git instaweb' is used to automatically direct a web browser and run a webserver with an interface into your local repository.

15. What is git is-tree?

'git is-tree' represents a tree object including the mode and the name of each item and the SHA-1 value of the blob or the tree.

16. Name a few Git commands and explain their usage.

Below are some basic Git commands:

Command	Function
git rm [file]	deletes the file from your working directory and stages the deletion.
git log	list the version history for the current branch.
git show [commit]	shows the metadata and content changes of the specified commit.
git tag [commitID]	used to give tags to the specified commit.
git checkout [branch name] git checkout -b [branch name]	used to switch from one branch to another. creates a new branch and also switches to it.

Intermediate level Questions

17. How to resolve a conflict in Git?

The following steps will resolve conflict in Git-

1. Identify the files that have caused the conflict.
2. Make the necessary changes in the files so that conflict does not arise again.
3. Add these files by the command git add.
4. Finally to commit the changed file using the command git commit

18. In Git how do you revert a commit that has already been pushed and made public?

There can be two approaches to tackle this question and make sure that you include both because any of the below options can be used depending on the situation:

- Remove or fix the bad file in a new commit and then push it to the remote repository. This is the most obvious way to fix an error. Once you have made necessary changes to the file, then commit it to the remote repository using the command: git commit -m "commit message"
- Also, you can create a new commit that undoes all changes that were made in the bad commit. To do this use the command

git revert <name of bad commit>

19. What is SubGit?

SubGit is a tool for SVN to Git migration. It can create a writable Git mirror of a local or remote Subversion repository and use both Subversion and Git as long as you like.

Now you can also include some advantages like you can do a fast one-time import from Subversion to Git or use SubGit within Atlassian Bitbucket Server. We can use SubGit to create a bi-directional Git-SVN mirror of an existing Subversion repository. You can push to Git or commit to Subversion as per your

convenience. Synchronization will be done by SubGit.

20. What is the difference between git pull and git fetch?

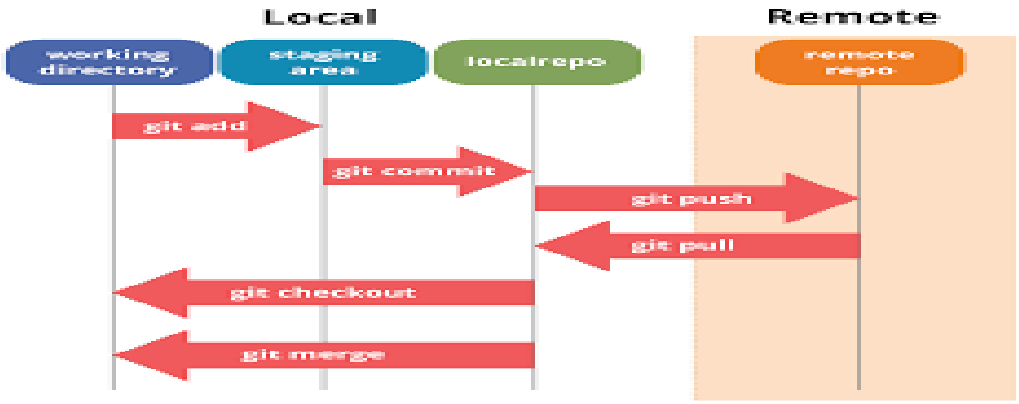
Git pull command pulls new changes or commits from a particular branch from your central repository and updates your target branch in your local repository.

Git fetch is also used for the same purpose but it works in a slightly different way. When you perform a git fetch, it pulls all new commits from the desired branch and stores it in a new branch in your local repository. If you want to reflect these changes in your target branch, git fetch must be followed with a git merge. Your target branch will only be updated after merging the target branch and fetched branch. Just to make it easy for you, remember the equation below:

Git pull = git fetch + git merge

21. What is 'staging area' or 'index' in Git?

That before completing the commits, it can be formatted and reviewed in an intermediate area known as 'Staging Area' or 'Index'. From the diagram it is evident that every change is first verified in the staging area I have termed it as "stage file" and then that change is committed to the repository.



22. What work is restored when the deleted branch is recovered?

The files which were stashed and saved in the stash index list will be recovered back. Any untracked files will be lost. Also, it is a good idea to always stage and commit your work or stash them.
If you want to fetch the log references of a particular branch or tag then run the command – "git reflog <ref_name>".

23. What is git stash?

Often, when you've been working on part of your project, things are in a messy state and you want to switch branches for some time to work on something else. The problem is, you don't want to do a commit of half-done work just so you can get back to this point later. The answer to this issue is Git stash.

Stashing takes your working directory that is, your modified tracked files and staged changes and saves it on a stack of unfinished changes that you can reapply at any time.

24. What is the function of 'git stash apply'?

If you want to continue working where you had left your work then 'git stash apply' command is used to bring back the saved changes onto your current working directory.

25. What is the difference between the 'git diff 'and 'git status'?

'git diff ' depicts the changes between commits, commit and working tree, etc. whereas 'git status' shows you the difference between the working directory and the index, it is helpful in understanding a git more comprehensively. 'git diff' is similar to 'git status', the only difference is that it shows the differences between various commits and also between the working directory and index.

26. What is the difference between 'git remote' and 'git clone'?

'git remote add' creates an entry in your git config that specifies a name for a particular URL whereas 'git clone' creates a new git repository by copying an existing one located at the URL

27. What is git stash drop?

Git 'stash drop' command is used to remove the stashed item. It will remove the last added stash item by default, and it can also remove a specific item if you include it as an argument.

Now give an example.

If you want to remove a particular stash item from the list of stashed items you can use the below commands:

git stash list: It will display the list of stashed items like:
stash@{0}: WIP on master: 049d078 added the index file
stash@{1}: WIP on master: c264051 Revert "added file_size"
stash@{2}: WIP on master: 21d80a5 added number to log

If you want to remove an item named stash@{0} use command **git stash drop stash@{0}**.

28. How do you find a list of files that have changed in a particular commit?

For this answer instead of just telling the command, explain what exactly this command will do.

To get a list file that has changed in a particular commit use the below command:

git diff-tree -r {hash}

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only.

You can also include the below-mentioned point, although it is totally optional but will help in impressing the interviewer.

The output will also include some extra information, which can be easily suppressed by including two flags:

git diff-tree –no-commit-id --name-only -r {hash}

Here –no-commit-id will suppress the commit hashes from appearing in the output, and –name-only will only print the file names, instead of their paths.

29. What is the function of 'git config'?

Git uses your username to associate commits with an identity. The git config command can be used to change your Git configuration, including your username.

Now explain with an example.

Suppose you want to give a username and email id to associate a commit with an identity so that you can know who has made a particular commit. For that I will use:

git config –global user.name "Your Name": This command will add a username.
git config –global user.email "Your E-mail Address": This command will add an email id.

30. What does a commit object contain?

Commit object contains the following components, you should mention all the three points presented below:

- A set of files, representing the state of a project at a given point of time
- Reference to parent commit objects
- An SHA-1 name, a 40 character string that uniquely identifies the commit object

31. Describe the branching strategies you have used.

- **Feature branching** – A feature branch model keeps all of the changes for a particular feature inside of a branch. When the feature is fully tested and validated by automated tests, the branch is then merged into master.
- **Task branching** – In this model, each task is implemented on its own branch with the task key included in the branch name. It is easy to see which code implements which task, just look for the task key in the branch name.
- **Release branching** – Once the develop branch has acquired enough features for a release, you can clone that branch to form a Release branch. Creating this branch starts the next release cycle, so no new features can be added after this point, only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it is ready to ship, the release gets merged into master and tagged with a version number. In addition, it should be merged back into the develop branch, which may have progressed since the release was initiated.
- In the end tell them that branching strategies vary from one organization to another so I know basic branching operations like delete, merge, checking out a branch, etc.

32. Explain the advantages of forking workflow

- There is a fundamental difference between the forking workflow and other popular git workflows. Rather than using a single server-side to act as the "central" codebase, it gives every developer their own server-side repository. The Forking Workflow is commonly seen in public open-source projects.
- A crucial advantage of the Forking Workflow is that contributions can be integrated without even needing everybody to push to a single central repository that leads to clean project history. Developers can push to their own server-side repositories, but only the project maintainer can push to the official repository.
- If developers are ready to publish a local commit, then they push the commit to their own public repository and not the official one. After this, they go for a pull request with the main repository that lets the project maintainer know an update is ready to be integrated.

33. How will you know in Git if a branch has already been merged into master?

The answer is pretty direct.

To know if a branch has been merged into master or not you can use the below commands:

git branch –merged – It lists the branches that have been merged into the current branch.
git branch –no-merged – It lists the branches that have not been merged.

.

34. Why is it desirable to create an additional commit rather than amending an existing commit?

There are a couple of reasons for this –

1. The amend operation destroys the state that was previously saved in a commit. If there is just the commit message being changed then that's not a problem. But if the contents are being amended then chances of eliminating something important remains more.
2. Abusing "git commit- amend" can result in the growth of a small commit and acquire unrelated changes.

35. What does 'hooks' comprise of in Git?

This directory consists of shell scripts that are activated if you run the corresponding Git commands. For example, git will try to execute the post-commit script after you have run a commit.

36. In Git, how would you return a commit that has just been pushed and made open?

One or more commits can be reverted through the use of git revert. This command, in a true sense, creates a new commit with patches that cancel out the changes introduced in specific commits. If in case the commit that needs to be reverted has already been published or changing the repository history is not an option then in such cases, git revert can be used to revert commits. If you run the following command then it will revert the last two commits:

git revert HEAD~2..HEAD

Next

Alternatively, there is always an option to check out the state of a particular commit from the past and commit it anew.

37. How to remove a file from git without removing it from your file system?

One has to be careful during a git add, else you may end up adding files that you didn't want to commit. However, git rm will remove it from both your staging area (index), as well as your file system (working tree), which may not be what you want.

Instead, use git reset:
git reset filename # or

echo filename >> .gitingore # add it to .gitignore to avoid re-adding it

This means that git reset <paths> is exactly the opposite of git add <paths>.

38. Can you explain the Gitflow workflow?

To record the history of the project, Gitflow workflow employs two parallel long-running branches – master and develop:

- Master – this branch is always ready to be released on LIVE, with everything fully tested and approved (production-ready).
- Hotfix – these branches are used to quickly patch production releases. These branches are a lot like release branches and feature branches except they're based on master instead of develop.
- Develop – this is the branch to which all feature branches are merged and where all tests are performed. Only when everything's been thoroughly checked and fixed it can be merged to the master.

- Feature – each new feature should reside in its own branch, which can be pushed to the develop branch as their parent one.

39. Tell me the difference between HEAD, working tree and index, in Git.

- The working tree/working directory/workspace is the directory tree of (source) files that you are able to see and edit.
- The index/staging area is a single, large, binary file in <baseOfRepo>/.git/index, which lists all files in the current branch, their SHA-1 checksums, timestamps, and the file name – it is not another directory which contains a copy of files in it.
- HEAD is used to refer to the last commit in the currently checked-out branch.

40. What is Git fork? What is the difference between fork, branch, and clone?

- A fork is a copy of a repository. Normally you fork a repository so that you are able to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.
- git cloning means pointing to an existing repository and make a copy of that repository in a new directory, at some other location. The original repository can be located on the local file system or on remote machine accessible supported protocols. The git clone command is used to create a copy of an existing Git repository.
- In very simple words, git branches are individual projects within a git repository. Different branches within a repository can have completely different files and folders, or it could have everything the same except for some lines of code in a file.

41. What are the different ways you can refer to a commit?

- In Git each commit has a unique hash. These hashes are used to identify the corresponding commits in various scenarios, for example, while trying to checkout a particular state of the code using the git checkout {hash} command.
- Along with this, Git maintains a number of aliases to certain commits, known as refs. Also, every tag that is created in the repository effectively becomes a ref and that is exactly why you can use tags instead of committing hashes in various git commands. Git also maintains a number of special aliases that are changed based on the state of the repository, such as HEAD, FETCH_HEAD, MERGE_HEAD, etc.
- In Git, commits are allowed to be referred to as relative to one another. In the case of merge commits, where the commit has two parents, ^ can be used to select one of the two parents, for example, HEAD^2 can be used to follow the second parent.
- And finally, refspecs are used to map local and remote branches together. However, these can also be used to refer to commits that reside on remote branches allowing one to control and manipulate them from a local git environment.

42. What is the difference between rebasing and merge in Git?

- In Git, the rebase command is used to integrate changes from one branch into another. It is an alternative to the "merge" command. The difference between rebasing and merge is that rebase rewrites the commit history in order to produce a straight, linear succession of commits.
- Merging is Git's way of putting a forked history back together again. The git merge command

helps you take the independent lines of development created by git branch and integrate them into a single branch.

43. Explain the difference between reverting and resetting.

- Git reset is a powerful command that is used to undo local changes to the state of a Git repository. Git reset operates on "The Three Trees of Git" which are, Commit History (HEAD), the Staging Index, and the Working Directory.
- Revert command in Git creates a new commit that undoes the changes from the previous commit. This command adds a new history to the project. It does not modify the existing history.

44. What is git cherry-pick?

The command git cherry-pick is normally used to introduce particular commits from one branch within a repository onto a different branch. Another common use is to forward- or back-port commits from a maintenance branch to a development branch. This is in contrast with other ways such as merge and rebase which normally apply many commits onto another branch.

Consider:

git cherry-pick <commit-hash>

45. How do you find a list of files that have changed in a particular commit?

git diff-tree -r {hash}

Given the commit hash, this will list all the files that were changed or added in that commit. The *-r* flag makes the command list individual files, rather than collapsing them into root directory names only.

The output will also include some extra information, which can be easily suppressed by including a couple of flags:

git diff-tree --no-commit-id --name-only -r {hash}

Here *–no-commit-id* will suppress the commit hashes from appearing in the output, and *–name-only* will only print the file names, instead of their paths.

<u>Advanced level Questions</u>

46. How do you squash the last N commits into a single commit?

There are two options to squash the last N commits into a single commit include both of the below-mentioned options in your answer

If you want to write the new commit message from scratch use the following command
git reset –soft HEAD~N &&git commit

If you want to start editing the new commit message with a concatenation of the existing commit messages then you need to extract those messages and pass them to Git commit for that I will use
git reset –soft HEAD~N &&git commit –edit -m"$(git log –format=%B –reverse .HEAD@{N})"

47. What is Git bisect? How can you use it to determine the source of a (regression) bug?

- Git bisect is used to find the commit that introduced a bug by using binary search. The command for Git bisect is
  git bisect <subcommand> <options>
- Now since you have mentioned the command above explain to them what this command will do.

- This command uses a binary search algorithm to find which commit in your project's history introduced a bug. You use it by first telling it a "bad" commit that is known to contain the bug, and a "good" commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is "good" or "bad". It continues narrowing down the range until it finds the exact commit that introduced the change.

48. How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?

I will suggest you to first give a small introduction to sanity checking.

Sanity or smoke test determines whether it is possible and reasonable to continue testing.

Now explain how to achieve this.

This can be done with a simple script related to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script, one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository.

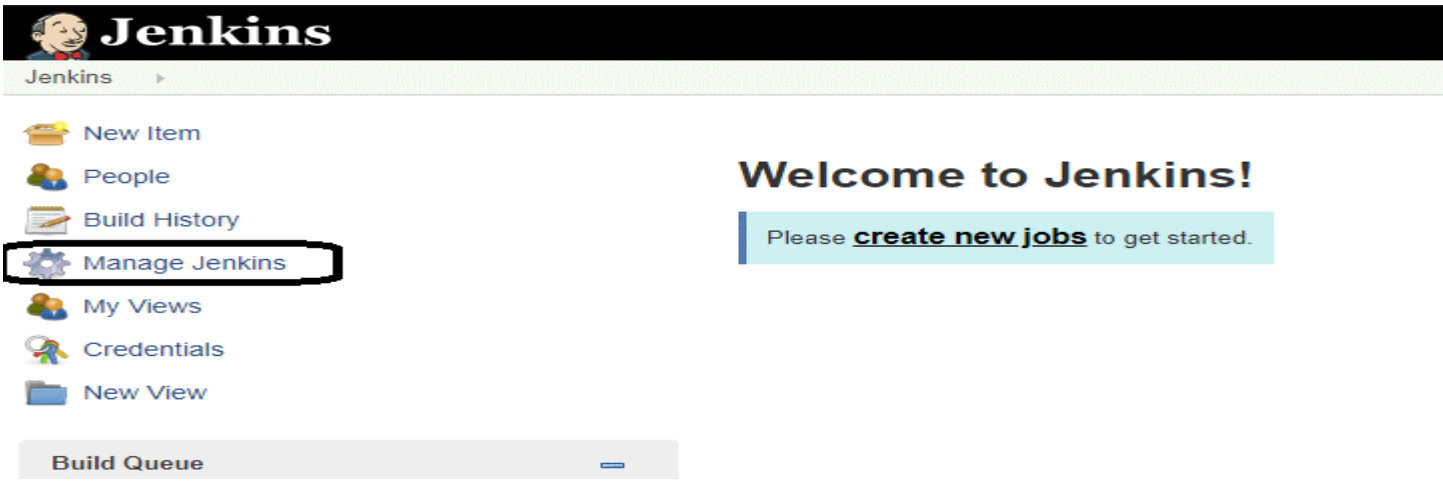
Finally, give an example, you can refer the below script:

```
#!/bin/sh
files=$(git diff –cached –name-only –diff-filter=ACM | grep '.go$')
if [ -z files ]; then
exit 0
fi
unfmtd=$(gofmt -l $files)
if [ -z unfmtd ]; then
exit 0
fi
echo "Some .go files are not fmt'd"
exit 1
```

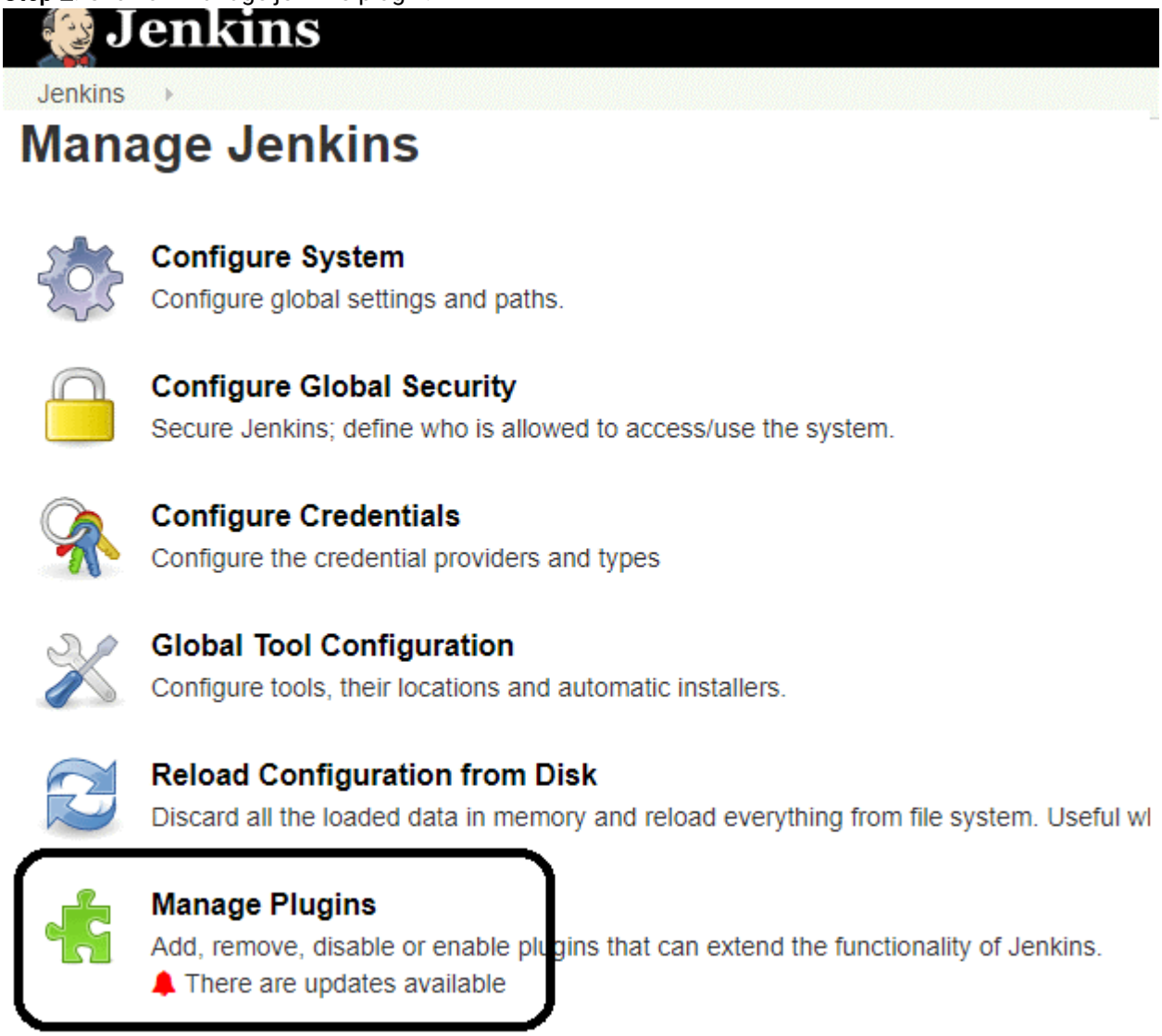
This script checks to see if any .go file that is about to be committed needs to be passed through the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

49. How do you integrate Git with Jenkins?

Step 1. Click on the manage jenkins button on your jenkins dashboard.



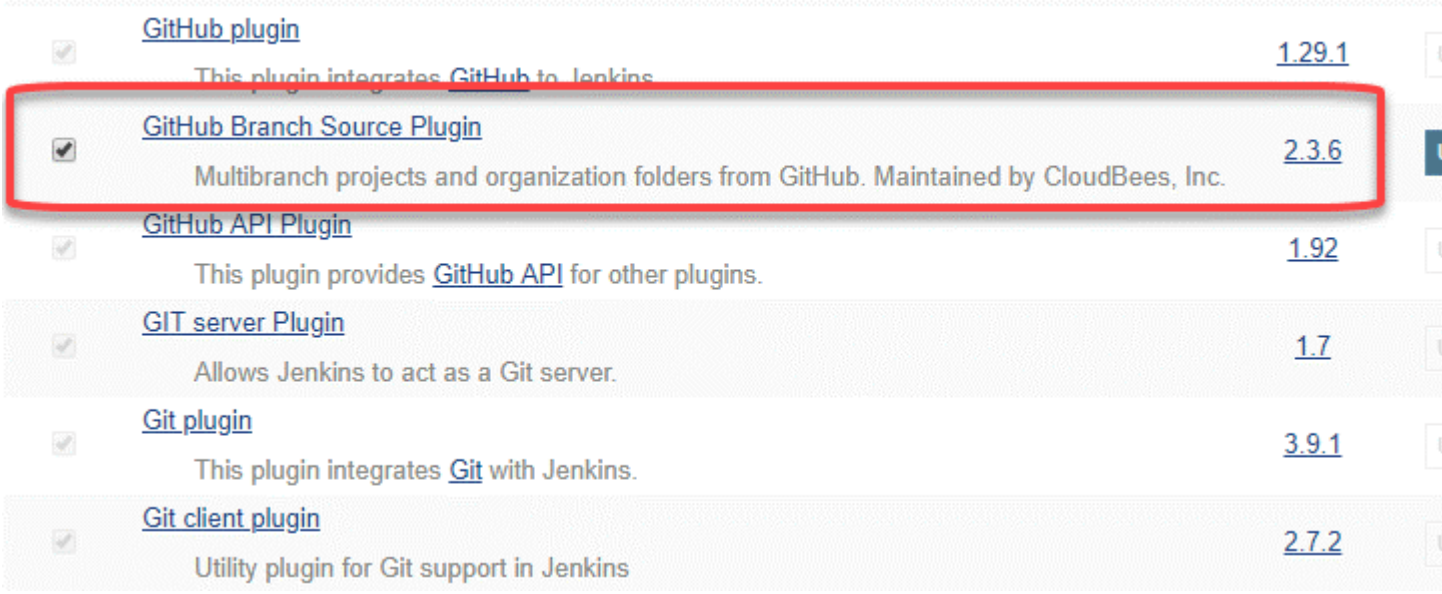
Step 2. Click on manage jenkins plugin.



Step 3: In the Plugins Page

1. Select the GIT Plugin
2. Click on **Install without restart.** The plugin will take a few moments to finish downloading depending on your internet connection, and will be installed automatically.
3. You can also select the option **Download now and Install after restart** In which plugin is installed after restart
4. You will be shown a "No updates available" message if you already have the Git plugin installed.

Step 4: Once the plugins have been installed, go to **Manage Jenkins** on your Jenkins dashboard. You will see your plugins listed among the rest.



50. What is git reflog?

The 'reflog' command keeps a **track of every single change made in the references** (branches or tags) of a repository and keeps a log history of the branches and tags that were either created locally or checked out. Reference logs such as the commit snapshot of when the branch was created or cloned, checked-out, renamed, or any commits made on the branch are maintained by Git and listed by the 'reflog' command.

Note: *The branch will be recoverable from your working directory only if the branch ever existed in your local repository i.e. the branch was either created locally or checked-out from a remote repository in your local repository for Git to store its reference history logs.*

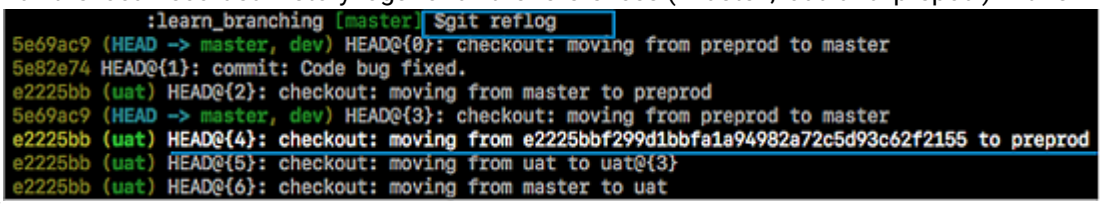
This command must be executed in the repository that had the lost branch. If you consider the remote repository situation, then you have to execute the reflog command on the developer's machine who had the branch.

command: *git reflog*

51. How to recover a deleted branch using git reflog?

Step 1: **History logs of all the references**

Get a list of all the local recorded history logs for all the references (‘master’, ‘uat’ and ‘prepod’) in this repository.

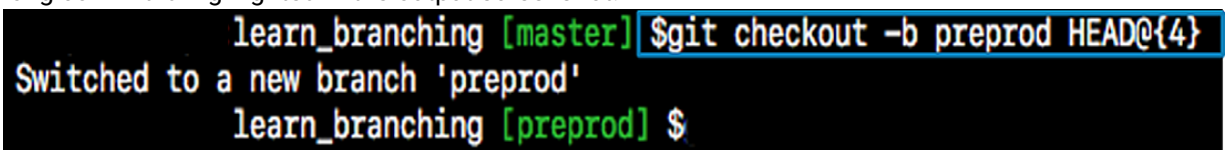


Step 2: Identify the history stamp

As you can see from the above snapshot, the highlighted commit id: e2225bb along with the HEAD pointer index:4 is the one when ‘preprod’ branch was created from the current HEAD pointer pointing to your latest work.

Step 3: **Recover**

If you want to recover back the ‘preprod‘ branch then use the command ‘git checkout’ passing the HEAD pointer reference with the index id – 4. This is the pointer reference when ‘preprod’ branch was created long commit id highlighted in the output screenshot.



1. What is RDBMS ? explain Features of RDBMS.

RDBMS is based on relational model. A relational database that stores data in a structured format, using rows and columns.

Features:

a) Facilitates primary key which helps in unique identification of the rows.

b) Index creation for retrieving data at higher speed

c) Facilitates common column to be shared amid two or more table

d) Multi user accessibility.

2. Explain E-R Model?

ER model is based on real world. Which is made up of entities and related objects. Entities are illustrated in database by a set of attributes.

3. Explain three levels of data abstraction?

a) Physical level: This is the lowest level of abstraction and it describes how data is stored.

b) Logical level: the next level of abstraction is logical. It describes what types of data is stored in database and what is relationship between these data.

c) View level: The highest level of abstraction and it's describes the only entire database.

4. Define Primary key, foreign key, candidate key, super key

Primary key: The primary key is the key that doesn't allow duplicate values and null values. Only one primary key per table is allowed.

Foreign key: foreign key allows the values present in the reference column only. It allows duplicate or null values. It can reference a column of unique/primary key.

Candidate key: A candidate key is minimum super key, there is no proper subgroup of candidate key attributes can a super key.

Super key: A super key is a set of attributes of a relational schema on which all attributes of the schema are partially dependent. No two rows can have same value of super key attributes

5. Explain difference between primary key and foreign key?

Primary key	Foreign key
1. Values cannot be null	1. Value can be Null
2. Duplicated values not allowed	2. Duplicate values are allowed.
3. Primary key can only be one	3. There can be multiple foreign keys

6. What do you mean by ER model and object oriented model?

An Entity-Relationship Model is primarily used to represent the real scenarios as entity. ER model clearly defines the objects and the relations between entities.

An object oriented model is mainly used to represent the real scenarios as objects. The objects with similar functions are grouped and linked to the other different purpose in the model. The data can be

reused in various missions.

7. What do you mean by cardinality and its type?

In the context of the data models. Cardinality means the relationship between two tables. The connection can be of four types.

I. One to one: one row of the first table partners with one row of the second table.

II. One to many: one row of the first table partners with more than one rows of the second table.

III. Many to one: More than one rows of the first table partners with one row of the second table.

IV. Many to many: More than one rows of the first table partners with more than one rows of the second table.

8. Explain constraint in SQL?

It is rule used to limit the type of data that can be insert into table, to maintain integrity and accuracy of the data inside the table.

It can be divided into two types:

I. Column level constraints

II. Table level constraints

Common constraints are NOT NULL, DEFAULT, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, INDEX

9. Explain Aggregate functions are available in SQL?

Aggregate function performs a calculation on a set of values, and it returns a single value as output. It ignores Null values when it performs calculation except for Count function.

Avg(), count(), sum(), min(), max()

10. How to find 3rd highest salary of an employee from the employee table in SQL?

SELECT salary FROM Employee ORDER BY salary DESC LIMIT N-1, 1

SELECT first_name, salary FROM employee e1 WHERE 3-1 = (SELECT COUNT(DISTINCT salary) FROM employee e2 WHERE e2.salary > e1.salary)

N maximum salary:

SELECT MIN(EmpSalary)

FROM Salary

WHERE EmpSalary IN(SELECT TOP N EmpSalary FROM Salary ORDER BY EmpSalary DESC)

11. How to find 2nd highest salary of an employee from the employee table in SQL?

SELECT salary FROM Employee ORDER BY salary DESC LIMIT N-1, 1;

SELECT max(salary) from employee where salary < (select max(salary) from employee);

SELECT max(salary) from employee where salary not in (select max(salary) from employee)

12. What are joins in SQL?

Joins combine columns from one or additional tables during relational database. A sql join could be a means that for combining columns from one (self join) or additional tables by using values common to every.

INNER, LEFT, RIGHT, FULL

13. How would you migrate an application from a database to another, for example from MySQL to PostgreSQL? If you had to manage that project, which issues would you expect to face?

pgLoader is an open-source database migration tool that aims to simplify the process of migrating to PostgreSQL. It supports migrations from several file types and RBDMSs — including MySQL and SQLite — to PostgreSQL.

Open Source Database Migration Tools. - Apache NiFi, Flyway, Pentaho

Cloud-Based Database Migration Tools - Alooma, Matillion, Snaplogic

14. ACID is an acronym that refers to Atomicity, Consistency, Isolation and Durability, 4 properties guaranteed by a database transaction in most database engines. What do you know about this topic? ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

Atomicity – This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

Consistency – The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Durability – The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

Isolation – In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

15. How would you manage database schema migrations? That is, how would you automate changes to database schema, as the application evolves, version after version?

Database schema migration essentially involves migration tables, indexes, and views in a database. Relational databases are similar in terms of how their data is organized in tables and indexes, but they are different in terms of additional extensions to these tables and indexes that are designed to improve performance and facilitate development. Most migration tools can convert the database schema relatively quickly and accurately. Target-specific database schemas can also be generated from modelling tools such as Erwin.

Django Unit Testing Process

Each django unit test comprises of 3 methods. 1. setUp, 2. test_<test_name> and 3.tearDown. When we run the test using command python manage.py test test runner will create a empty test database with required migrations. It will look for the tests in the files whose name starts with test and finds the all tests. The test runner will take the each test case and runs it. Whenever the test is running first it will call “setUp” method which is used to create basic requirements to run the test. After, it will run the test and then the method “tearDown” will be called. “tearDown” is used to destroy the data created by the “setUp” method. Django unit testing has a automated mechanism to remove it. But, in some cases we need to do it manually.

What is Design Patterns and why anyone should use them?

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**. A **design pattern** isn't a finished **design** that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations

Python provides support for the list of design patterns that are mentioned below. These design patterns will be used throughout this tutorial –

Model View Controller Pattern, Singleton pattern, Factory pattern, Builder Pattern
Prototype Pattern, Facade Pattern, Command Pattern, Adapter Pattern, Prototype Pattern, Decorator Pattern, Proxy Pattern, Chain of Responsibility Pattern, Observer Pattern, State Pattern, Strategy Pattern
Template Pattern, Flyweight Pattern, Abstract Factory Pattern, Object Oriented Pattern

What are the differences between continuous integration, continuous delivery, and continuous deployment? What are the differences between continuous integration, continuous delivery, and continuous deployment?

- Developers practicing **continuous integration** merge their changes back to the main branch as often as possible. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.
- **Continuous delivery** is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.
- **Continuous deployment** goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

What is the difference between JOIN and UNION? What is the difference between JOIN and UNION?

SQL JOIN allows us to “lookup” records on other table based on the given conditions between two tables.
UNION operation allows us to add 2 similar data sets to create resulting data set that contains all the data from the source data sets. Union does not require any condition for joining.

How to debug in Django, the good way?
How will you debug the code?