Python Interview Questions and Answers by Pythonlife

1. What is Python? What are the benefits of using Python?
Ans: Python is a programming language with objects, modules, threads, exceptions and automatic memory management. The benefits of pythons are that it is simple and easy, portable, extensible, build-in data structure and it is an open source.

2. What is PEP 8?
Ans: PEP 8 is a coding convention, a set of recommendation, about how to write your Python code more readable.

3. What is pickling and unpickling?
Ans: Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using dump function, this process is called pickling. While the process of retrieving original Python objects from the stored string representation is called unpickling.

4. How Python is interpreted?
Ans: Python language is an interpreted language. Python program runs directly from the source code. It converts the source code that is written by the programmer into an intermediate language, which is again translated into machine language that has to be executed.

5. How memory is managed in Python?
Ans: Python memory is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have an access to this private heap and interpreter takes care of this Python private heap. The allocation of Python heap space for Python objects is done by Python memory manager. The core API gives access to some tools for the programmer to code. Python also have an inbuilt garbage collector, which recycle all the unused memory and frees the memory and makes it available to the heap space.

6. What are the tools that help to find bugs or perform static analysis?
Ans: PyChecker is a static analysis tool that detects the bugs in Python source code and warns about the style and complexity of the bug. Pylint is another tool that verifies whether the module meets the coding standard.

7. What are Python decorators?
Ans: A Python decorator is a specific change that we make in Python syntax to alter functions easily.

8. What is the difference between list and tuple?
Ans: The difference between list and tuple is that list is mutable while tuple is not. Tuple can be hashed for e.g as a key for dictionaries.

9. How are arguments passed by value or by reference?
Ans: Everything in Python is an object and all variables hold references to the objects. The references values are according to the functions; as a result you cannot change the value of the references. However, you can change the objects if it is mutable.

10. What is Dict and List comprehensions are?
Ans: They are syntax constructions to ease the creation of a Dictionary or List based on existing iterable.

11. What are the built-in type does python provides?
Ans: There are mutable and Immutable types of Pythons built in types Mutable built-in types
List Sets
Dictionaries Immutable built-in types

Strings Tuples Numbers

12. What is namespace in Python?
Ans: In Python, every name introduced has a place where it lives and can be hooked for. This is known as namespace. It is like a box where a variable name is mapped to the object placed. Whenever the variable is searched out, this box will be searched, to get corresponding object.

13. What is lambda in Python?
Ans: It is a single expression anonymous function often used as In-line function.

14. Why lambda forms in python does not have statements?
Ans: A lambda form in python does not have statements as it is used to make new function object and then return them at runtime.

15. What is pass in Python?
Ans: Pass means, no-operation Python statement, or in other words it is a place holder in compound statement, where there should be a blank left and nothing has to be written there.

16. In Python what are iterators?
Ans: In Python, iterators are used to iterate a group of elements, containers like list.

17. What is unit test in Python?
Ans: A unit testing framework in Python is known as unittest. It supports sharing of setups, automation testing, shutdown code for tests, aggregation of tests into collections etc.

18. In Python what is slicing?
Ans: A mechanism to select a range of items from sequence types like list, tuple, strings etc. is known as slicing.

19. What are generators in Python?
Ans: The way of implementing iterators are known as generators. It is a normal function except that it yields expression in the function.

20. What is docstring in Python?
Ans: A Python documentation string is known as docstring, it is a way of documenting Python functions, modules and classes.

21. How can you copy an object in Python?
Ans: To copy an object in Python, you can try copy.copy () or copy.deepcopy() for the general case. You cannot copy all objects but most of them.

22. What is negative index in Python?
Ans: Python sequences can be index in positive and negative numbers. For positive index, 0 is the first index, 1 is the second index and so forth. For negative index, (-1) is the last index and (-2) is the second last index and so forth.

23. How you can convert a number to a string?
Ans: In order to convert a number into a string, use the inbuilt function str(). If you want a octal or hexadecimal representation, use the inbuilt function oct() or hex().

24. What is the difference between Xrange and range?
Ans: Xrange returns the xrange object while range returns the list, and uses the same memory and no matter what the range size is.

25. What is module and package in Python?
Ans: In Python, module is the way to structure program. Each Python program file is a module, which imports other modules like objects and attributes.
The folder of Python program is a package of modules. A package can have modules or subfolders.

26. Mention what are the rules for local and global variables in Python?
Ans: Local variables: If a variable is assigned a new value anywhere within the function’s body, it’s assumed to be local.

Global variables: Those variables that are only referenced inside a function are implicitly global.

27. How can you share global variables across modules?
Ans: To share global variables across modules within a single program, create a special module. Import the config module in all modules of your application. The module will be available as a global variable across modules.

28. Explain how can you make a Python Script executable on Unix?To make a Python Script executable on Unix, you need to do two things,
Ans: Script file’s mode must be executable and
the first line must begin with # (#!/usr/local/bin/python)

29. Explain how to delete a file in Python?
Ans: By using a command os.remove (filename) or os.unlink(filename)

30. Explain how can you generate random numbers in Python?
Ans: To generate random numbers in Python, you need to import command as import random
random.random()
This returns a random floating point number in the range [0,1)

31. Explain how can you access a module written in Python from C?
Ans: You can access a module written in Python from C by following method, Module = =PyImport_ImportModule(“”);

32. Mention the use of // operator in Python?
Ans: It is a Floor Divisionoperator , which is used for dividing two operands with the result as quotient showing only digits before the decimal point. For instance, 10//5 = 2 and 10.0//5.0 = 2.0.

33. Mention five benefits of using Python?
Ans: Python comprises of a huge standard library for most Internet platforms like

Email, HTML, etc.
Python does not require explicit memory management as the interpreter itself allocates the memory to new variables and free them automatically
Provide easy readability due to use of square brackets Easy-to-learn for beginners
Having the built-in data types saves programming time and effort from declaring variables

34. Mention the use of the split function in Python?
Ans: The use of the split function in Python is that it breaks a string into shorter strings using the defined separator. It gives a list of all words present in the string.

35. Explain what is Flask & its benefits?
Ans: Flask is a web micro framework for Python based on “Werkzeug, Jinja 2 and good intentions” BSD licensed. Werkzeug and jingja are two of its dependencies.

Flask is part of the micro-framework. Which means it will have little to no dependencies on external libraries. It makes the framework light while there is little dependency to update and less security bugs.

36. Mention what is the difference between Django, Pyramid, and Flask?
Ans: Flask is a “micro framework” primarily build for a small application with simpler requirements. In flask, you have to use external libraries. Flask is ready to use.
Pyramid are build for larger applications. It provides flexibility and lets the developer use the right tools for their project. The developer can choose the database, URL structure, templating style and more. Pyramid is heavy configurable.

Like Pyramid, Django can also used for larger applications. It includes an ORM.

37. Mention what is Flask-WTF and what are their features?
Ans: Flask-WTF offers simple integration with WTForms. Features include for Flask WTF are
Integration with wtforms Secure form with csrf token Global csrf protection

Internationalization integration Recaptcha supporting
File upload that works with Flask Uploads

38. Explain what is the common way for the Flask script to work?
Ans: The common way for the flask script to work is…
Either it should be the import path for your application Or the path to a Python file

39. Explain how you can access sessions in Flask?
Ans: A session basically allows you to remember information from one request to another. In a flask, it uses a signed cookie so the user can look at the session contents and modify. The user can modify the session if only it has the secret key Flask.secret_key.

40. Is Flask an MVC model and if yes give an example showing MVC pattern for your application?
Ans: Basically, Flask is a minimalistic framework which behaves same as MVC framework. So MVC is a perfect fit for Flask, and the pattern for MVC we will consider for the following example

```
from flask import Flaskapp = Flask(_name_)
@app.route(“/”)

Def hello():

return “Hello World”

app.run(debug = True)
```

In this code your,
Configuration part will be

```
from flask import Flask

app = Flask(_name_)
```

View part will be

```
@app.route(“/”)

Def hello():

return “Hello World”
```

While you model or main part will be
app.run(debug = True)

41. What type of a language is python? Interpreted or Compiled?
Ans: Beginner’s Answer:
Python is an interpreted, interactive, objectoriented programming language.
Expert Answer:
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.

42. What do you mean by python being an “interpreted language”? (Continues from previous question)
Ans: An interpreted languageis a programming languagefor which most of its implementations execute instructions directly, without previously compiling a program
into machinelanguageinstructions. In context of Python, it means that Python program
runs directly from the source code.

43. What is python’s standard way of identifying a block of code?
Ans: Indentation.

44. Please provide an example implementation of a function called “my_func” that returns the square of a given variable “x”. (Continues from previous question)
Ans:

```
defmy_func(x):
returnx**2
```

45. Is python statically typed or dynamically typed?
Ans: Dynamic.
In a statically typed language, the type of variables must be known (and usually declared) at the point at which it is used. Attempting to use it will be an error. In a
dynamically typed language, objects still have a type, but it is determined at runtime.
You are free to bind names (variables) to different objects with a different type. So long
as you only perform operations valid for the type the interpreter doesn't care what type
they actually are.

46. Is python strongly typed or weakly typed language?
Ans: Strong.
In a weakly typed language a compiler / interpreter will sometimes change the type of a variable. For example, in some languages (like JavaScript) you can add strings to numbers 'x' + 3 becomes 'x3'. This can be a problem because if you have made a mistake in your program, instead of raising an exception execution will continue
but your variables now have wrong and unexpected values. In a strongly typed language (like Python) you can't perform operations inappropriate to the type of the
object attempting to add numbers to strings will fail. Problems like these are easier to
diagnose because the exception is raised at the point where the error occurs rather than
at some other, potentially far removed, place.

47. Create a unicode string in python with the string "This is a test string"?
Ans: some_variable=u'Thisisateststring'
Or
some_variable=u"Thisisateststring"

48. What is the python syntax for switch case statements?
Ans: Python doesn't support switchcase statements. You can use ifelse statements for this purpose.

49. What is a lambda statement? Provide an example.
Ans: A lambda statement is used to create new function objects and then return them at
runtime. Example:
my_func=lambdax:x**2
creates a function called my_func that returns the square of the argument passed.

50.What are the rules for local and global variables in Python?
Ans: If a variable is defined outside function then it is implicitly global. If variable is
assigned new value inside the function means it is local. If we want to make it global we

need to explicitly define it as global. Variable referenced inside the function are implicit
global

51.What is the output of the following program?
Ans:

```
#!/usr/bin/python
deffun1(a):
print'a:',a
a=33;
print'locala:',a
a=100
fun1(a)
print'aoutsidefun1:',a
```

Ans. Output:
a:100
locala:33
aoutsidefun1:100

52.What is the output of the following program?
Ans:

```
#!/usr/bin/python
deffun2():
globalb
print’b:’,b
b=33
print’globalb:’,b
b=100
fun2()
print’boutsidefun2',b
```

Ans. Output:

```
b:100
globalb:33
boutsidefun2:33
```

53. What is the output of the following program?
Ans:

```
#!/usr/bin/python

deffoo(x,y):
globala
a=42
x,y=y,x
b=33
b=17
c=100
print(a,b,x,y)
a,b,x,y=1,15,3,4
foo(17,4)
print(a,b,x,y)
```

Ans.Output:

```
4217417
421534
```

54.What is the output of the following program?
Ans:

```
#!/usr/bin/python
deffoo(x=[]):
x.append(1)
returnx
foo()
foo()
```

Output:

```
[1]
[1,1]
```

55. What is the purpose of #!/usr/bin/pythonon the first line in the above code? Is there any advantage?
Ans: By specifying #!/usr/bin/pythonyou specify exactly which interpreter will be used to run the script on a particular system. This is the hardcoded path to the python
interpreter for that particular system. The advantage of this line is that you can use a
specific python version to run your code.

56.What is the output of the following program?
Ans:

```
list=[‘a’,’b’,’c’,’d’,’e’]
printlist[10]
```

Ans. Output:
IndexError.Or Error.

57.What is the output of the following program?
Ans:

```
list=[‘a’,’b’,’c’,’d’,’e’]
printlist[10:]
```

Ans. Output:
[]
Theabovecodewilloutput[],andwillnotresultinanIndexError.
As one would expect, attempting to access a member of a list using an index that exceeds the number of members results in an IndexError.

58. What does this list comprehension do:
Ans:

```
[x**2forxinrange(10)ifx%2==0]
```

Ans. Creates the following list:
[0,4,16,36,64]

59. Do sets, dictionaries and tuples also support comprehensions?
Ans: Sets and dictionaries support it. However tuples are immutable and have generators but not comprehensions.
Set Comprehension:

```
r={xforxinrange(2,101)
ifnotany(x%y==0foryinrange(2,x))}
```

Dictionary Comprehension:

```
{i:jfori,jin{1:’a’,2:’b’}.items()}
```

since
{1:’a’,2:’b’}.items()returnsalistof2-Tuple.iisthefirstelement oftuplejisthesecond.

60.What are some mutable and immutable datatypes/datastructures in python?
Ans:
Mutable Types Immutable Types

Dictionary number
List boolean
string
tuple

61.What are generators in Python?

Ans: Generators are functions that return an iterable collection of items, one at a time, in a set manner. Generators, in general, are used to create iterators with a different approach. They employ the use of yield keyword rather than return to return a generator object.
Let’s try and build a generator for fibonacci numbers –

```
## generate fibonacci numbers upto n
def fib(n):
    p, q = 0, 1
    while(p < n):
        yield p
        p, q = q, p + q

x = fib(10)    # create generator object

## iterating using __next__(), for Python2, use next()
x.__next__()    # output => 0
x.__next__()    # output => 1
x.__next__()    # output => 1
x.__next__()    # output => 2
x.__next__()    # output => 3
x.__next__()    # output => 5
x.__next__()    # output => 8
x.__next__()    # error

## iterating using loop
for i in fib(10):
    print(i)    # output => 0 1 1 2 3 5 8
```

62.What can you use Python generator functions for?
Ans: One of the reasons to use generator is to make the solution clearer for some kind
of solutions.
The other is to treat results one at a time, avoiding building huge lists of results that you
would process separated anyway.

63.When is not a good time to use python generators?
Ans: Use list instead of generator when:
1 You need to access the data multiple times (i.e. cache the results instead of recomputing them)
2 You need random access (or any access other than forward sequential order):
3 You need to join strings (which requires two passes over the data)
4 You are using PyPy which sometimes can’t optimize generator code as much as it can with normal function calls and list manipulations.

64.What’s your preferred text editor?
Ans: Emacs. Any alternate answer leads to instant disqualification of the applicant

65.When should you use generator expressions vs. list comprehensions in Python and vice-versa?
Ans: Iterating over the generator expression or the list comprehension will do the same
thing. However, the list comp will create the entire list in memory first while the generator expression will create the items on the fly, so you are able to use it for very

large (and also infinite!) sequences.

66. What is a negative index in Python?
Ans: Python arrays and list items can be accessed with positive or negative numbers. A
negative Index accesses the elements from the end of the list counting backwards.
Example:
a=[123]
printa[-3]
printa[-2]
Outputs:
1
2

67. What is the difference between range and xrange functions?
Ans: Range returns a list while xrange returns an xrange object which take the same memory no matter of the range size. In the first case you have all items already
generated (this can take a lot of time and memory). In Python 3 however, range is implemented with xrange and you have to explicitly call the list function if you want to
convert it to a list.

68. How can I find methods or attributes of an object in Python?
Ans: Builtin dir() function of Python ,on an instance shows the instance variables as
well as the methods and class attributes defined by the instance’s class and all its base
classes alphabetically. So by any object as argument to dir() we can find all the methods & attributes of the object’s class

69. What is the statement that can be used in Python if a statement is required syntactically but the program requires no action?
Ans:
pass

70. Do you know what is the difference between lists and tuples? Can you give me an example for their usage?
Ans:
First list are mutable while tuples are not, and second tuples can be hashed e.g. to be used as keys for dictionaries. As an example of their usage, tuples are used when
the order of the elements in the sequence matters e.g. a geographic coordinates, “list”
of points in a path or route, or set of actions that should be executed in specific order.
Don’t forget that you can use them a dictionary keys. For everything else use lists

71. What is the function of “self”?
Ans:
“Self” is a variable that represents the instance of the object to itself. In most of
the object oriented programming languages, this is passed to the methods as a hidden
parameter that is defined by an object. But, in python it is passed explicitly. It refers to
separate instance of the variable for individual objects. The variables are

referred as
“self.xxx”.

72. How is memory managed in Python?
Ans:
Memory management in Python involves a private heap containing all
Python objects and data structures. Interpreter takes care of Python heap and
the programmer has no access to it. The allocation of heap space for Python
objects is done by Python memory manager. The core API of Python provides
some tools for the programmer to code reliable and more robust program. Python
also has a builtin garbage collector which recycles all the unused memory.
The gc module defines functions to enable /disable garbage collector:
gc.enable() Enables automatic garbage collection.
gc.disable()-Disables automatic garbage collection

73. What is __init__.py?
Ans:
It is used to import a module in a directory, which is called package import.

74. Print contents of a file ensuring proper error handling?
Ans:
try:
withopen(‘filename’,’r’)asf:
printf.read()
exceptIOError:
print”Nosuchfileexists”

75 How do we share global variables across modules in Python?
Ans:
We can create a config file and store the entire global variable to be
shared across modules in it. By simply importing config, the entire global variable
defined will be available for use in other modules.
For example I want a, b & c to share between modules.
config.py :
a=0
b=0
c=0
module1.py:
importconfig
config.a=1
config.b=2
config.c=3
print”a,b&resp.are:”,config.a,config.b,config.c

output of module1.py will be
123

76. Does Python support Multithreading?
Ans: Yes
Medium

77. How do I get a list of all files (and directories) in a given directory in
Python?
Ans: Following is one possible solution there can be other similar ones:
import os
for dirname,dirnames,filenames in os.walk(‘.’):
#printpathtoallsubdirectoriesfirst.
forsubdirnameindirnames:
printos.path.join(dirname,subdirname)

```
#printpathtoallfilenames.
forfilenameinfilenames:
printos.path.join(dirname,filename)
#Advancedusage:
#editingthe'dirnames'listwillstopos.walk()fromrecursing
intothere.
if'.git'indirnames:
#don'tgointoany.gitdirectories.
dirnames.remove('.git')
```

78. How to append to a string in Python?
Ans: The easiest way is to use the += operator. If the string is a list of character, join()
function can also be used.

79. How to convert a string to lowercase in Python?
Ans: use lower() function.
Example:

```
s='MYSTRING'
prints.lower()
```

80. How to convert a string to lowercase in Python?
Ans: Similar to the above question. use upper() function instead.

81. How to check if string A is substring of string B?
Ans: The easiest way is to use the in operator.

```
>>> 'abc' in 'abcdefg'
True
```

82. Find all occurrences of a substring in Python
Ans: There is no simple builtin string function that does what you're looking for, but
you could use the more powerful regular expressions:

```
>>>[m.start()forminre.finditer('test','testtesttesttest')]
[0,5,10,15]//thesearestartingindicesforthestring
```

83. What is GIL? What does it do?Talk to me about the GIL. How does it impact concurrency in Python? What kinds of applications does it impact more than others?
Ans: Python's GIL is intended to serialize access to interpreter internals from different
threads. On multicore systems, it means that multiple threads can't effectively make
use of multiple cores. (If the GIL didn't lead to this problem, most people wouldn't care
about the GIL it's only being raised as an issue because of the increasing prevalence
of multicore systems.)
Note that Python's GIL is only really an issue for CPython, the reference implementation. Jython and IronPython don't have a GIL. As a Python developer, you don't generally come across the GIL unless you're writing a C extension. C extension
writers need to release the GIL when their extensions do blocking I/O, so that other
threads in the Python process get a chance to run.

84. Print the index of a specific item in a list?
Ans: use the index() function

```
>>>["foo","bar","baz"].index('bar')
1
```

.

85.How do you iterate over a list and pull element indices at the same time?
Ans: You are looking for the enumerate function. It takes each element in a sequence
(like a list) and sticks it’s location right before it. For example:

```
>>>my_list=[‘a’,’b’,’c’]
>>>list(enumerate(my_list))
[(0,’a’),(1,’b’),(2,’c’)]
```

Note that enumerate() returns an object to be iterated over, so wrapping it in list() just
helps us see what enumerate() produces.
An example that directly answers the question is given below

```
my_list=[‘a’,’b’,’c’]
fori,charinenumerate(my_list):
printi,char
```

The output is:

```
0a
1b
2c
```

86. How does Python’s list.sort work at a high level? Is it stable? What’s the runtime?
Ans: In early pythonversions, the sort function implemented a modified version of quicksort. However, it was deemed unstable and as of 2.3 they switched to using an adaptive mergesort algorithm.

87. What does the list comprehension do:
Ans:

```
my_list=[(x,y,z)forxinrange(1,30)foryinrange(x,30)forzin
range(y,30)ifx**2+y**2==z**2]
```

It creates a list of tuples called my_list, where the first 2 elements are the perpendicular sides of right angle triangle and the third value ‘z’ is the hypotenuse.

```
[(3,4,5),(5,12,13),(6,8,10),(7,24,25),(8,15,17),(9,12,15),
(10,24,26),(12,16,20),(15,20,25),(20,21,29)]
```

88. How can we pass optional or keyword parameters from one function to another in Python?
Ans:
Gather the arguments using the * and ** specifiers in the function’s parameter list. This
gives us positional arguments as a tuple and the keyword arguments as a dictionary.
Then we can pass these arguments while calling another function by using * and **:

```
deffun1(a,*tup,**keywordArg):
…
keywordArg[‘width’]=’23.3c’
…
Fun2(a,*tup,**keywordArg)
```

89. Python How do you make a higher order function in Python?
Ans:
A higherorder function accepts one or more functions as input and returns a new function. Sometimes it is required to use function as data To make high order function ,
we need to import functools module The functools.partial() function is used often

for
high order function.

90. What is map?
Ans:
The syntax of map is:
map(aFunction,aSequence)
The first argument is a function to be executed for all the elements of the iterable given
as the second argument. If the function given takes in more than 1 arguments, then many iterables are given.

91. Tell me a very simple solution to print every other element of this list?
Ans:

```
L=[0,10,20,30,40,50,60,70,80,90]
L[::2]
```

92. Are Tuples immutable?
Ans: Yes.

93. Why is not all memory freed when python exits?
Ans: Objects referenced from the global namespaces of Python modules are not always deallocated when Python exits. This may happen if there are circular references. There are also certain bits of memory that are allocated by the C library that
are impossible to free (e.g. a tool like the one Purify will complain about these). Python
is, however, aggressive about cleaning up memory on exit and does try to destroy every
single object. If you want to force Python to delete certain things on de-allocation, you
can use the at exit module to register one or more exit functions to handle those deletions.

94. What is Java implementation of Python popularly know?
Ans: Jython.

95. What is used to create unicode strings in Python?
Ans:
Add u before the string.
u ‘mystring’

96. What is a docstring?
Ans:
docstring is the documentation string for a function. It can be accessed by function_name.__doc__

97. Given the list below remove the repetition of an element.
Ans:
words=[‘one’,’one’,’two’,’three’,’three’,’two’]
A bad solution would be to iterate over the list and checking for copies somehow and
then remove them!

A very good solution would be to use the set type. In a Python set, duplicates are not
allowed.
So, list(set(words)) would remove the duplicates.

98. Print the length of each line in the file ‘file.txt’ not including any whitespaces at the end of the lines?
Ans:
withopen(“filename.txt”,”r”)asf1:
printlen(f1.readline().rstrip())
rstrip() is an inbuilt function which strips the string from the right end of spaces or tabs
(whitespace characters).

99. What is wrong with the code?
Ans:

```
func([1,2,3])#explicitlypassinginalist
func() #usingadefaultemptylist
deffunc(n=[]):
#dosomethingwithn
printn
```

This would result in a NameError. The variable n is local to function func and can’t be accessesd outside. So, printing it won’t be possible.

100. What does the below mean?
Ans:

```
s = a + ‘[‘ + b + ‘:’ + c + ‘]’
```

seems like a string is being concatenated. Nothing much can be said without knowing types of variables a, b, c. Also, if all of the a, b, c are not of type string,
TypeError would be raised. This is because of the string constants (‘[‘ , ‘]’) used in the
statement.
101. What are Python decorators?
Ans:
A Python decorator is a specific change that we make in Python syntax to alter functions easily.

102. What is namespace in Python?
Ans:
In Python, every name introduced has a place where it lives and can be hooked for. This is known as namespace. It is like a box where a variable name is mapped to
the object placed. Whenever the variable is searched out, this box will be searched, to
get corresponding object.

103. Explain the role of repr function.
Ans:
Python can convert any value to a string by making use of two functions repr() or str(). The str() function returns representations of values which are human-readable,
while repr() generates representations which can be read by the interpreter. repr() returns a machinereadable representation of values, suitable for an exec command.
Following code sniipets shows working of repr() & str() :

```
deffun():
y=2333.3
x=str(y)
z=repr(y)
```

```
print”y:”,y
print”str(y):”,x
print”repr(y):”,z
fun()
————-
output
y:2333.3
str(y):2333.3
repr(y):2333.3000000000002
```

104. What is LIST comprehensions features of Python used for?
Ans:
LIST comprehensions features were introduced in Python version 2.0, it creates a new list based on existing list. It maps a list into another list by applying a function to

each of the elements of the existing list. List comprehensions creates lists without using
map() , filter() or lambda form.

105. Explain how to copy an object in Python.?
Ans:
There are two ways in which objects can be copied in python. Shallow copy & Deep copy. Shallow copies duplicate as minute as possible whereas Deep copies duplicate everything. If a is object to be copied then …
copy.copy(a) returns a shallow copy of a.
copy.deepcopy(a) returns a deep copy of a.

106. Describe how to send mail from a Python script?
Ans:
The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine.
A sample email is demonstrated below.

```
import smtplib
SERVER = smtplib.SMTP(‘smtp.server.domain’)
FROM = sender@mail.com
TO = [“user@mail.com”] # must be a list
SUBJECT = “Hello!”
TEXT = “This message was sent with Python’s smtplib.”
# Main message
message = “””
From: Lincoln < sender@mail.com >
To: CarreerRide user@mail.com
Subject: SMTP email msg
This is a test email. Acknowledge the email by responding.
””” % (FROM, “, “.join(TO), SUBJECT, TEXT)
server = smtplib.SMTP(SERVER)
server.sendmail(FROM, TO, message)
server.quit()
```

107. Which of the languages does Python resemble in its class syntax?
Ans: c++.

108. Python How to create a multidimensional list?
Ans: There are two ways in which Multidimensional list can be created:
By direct initializing the list as shown below to create myList below.

```
>>>myList=[[227,122,223],[222,321,192],[21,122,444]]
>>>printmyList[0]
>>>printmyList[1][2]
```

```
_____
Output
[227, 122, 223]
192
The second approach is to create a list of the desired length first and then fill
in each
element with a newly created lists demonstrated below :
>>>list=[0]*3
>>>foriinrange(3):
>>>list[i]=[0]*2
>>>foriinrange(3):
>>>forjinrange(2):
>>>list[i][j]=i+j
>>>printlist
_____
Output
[[0,1],[1,2],[2,3]]
```

109. Explain the disadvantages of python
Ans: Disadvantages of Python are: Python isn't the best for memory intensive tasks. Python is interpreted language & is slow compared to C/C++ or Java.

110. Explain how to make Forms in python.
Ans. As python is scripting language forms processing is done by Python. We need to import cgi module to access form fields using FieldStorage class.
Every instance of class FieldStorage (for 'form') has the following attributes:
form.name: The name of the field, if specified.

form.filename: If an FTP transaction, the clientside filename.
form.value: The value of the field as a string.
form.file: file object from which data can be read.
form.type: The content type, if applicable.
form.type_options: The options of the 'contenttype' line of the HTTP request, returned
as a dictionary.
form.disposition: The field 'contentdisposition'; None if unspecified.
form.disposition_options: The options for 'contentdisposition'.
form.headers: All of the HTTP headers returned as a dictionary.
A code snippet of form handling in python:

```
importcgi
form=cgi.FieldStorage()
ifnot(form.has_key("name")andform.has_key("age")):
print"<H1>Name&AgenotEntered</H1>"
print"FilltheName&Ageaccurately."
return
print"<p>name:",form["name"].value
print"<p>Age:",form["age"].value
```

111. Explain how python is interpreted.
Ans: Python program runs directly from the source code. Each type Python programs are executed code is required. Python converts source code written by the programmer
into intermediate language which is again translated it into the native language machine language that is executed. So Python is an Interpreted language.

112. Explain how to overload constructors (or methods) in Python.?
Ans. _init__ () is a first

113.What is the difference between List & Tuple in Python.?

LIST vs TUPLES

LIST	TUPLES
Lists are mutable i.e they can be edited.	Tuples are immutable (tuples are lists which can't be edited).
Lists are slower than tuples.	Tuples are faster than list.
Syntax: list_1 = [10, 'Chelsea', 20]	Syntax: tup_1 = (10, 'Chelsea' , 20)

114.What are the key features of Python?
Ans:

Python is an interpreted language. That means that, unlike languages like C and its variants, Python does not need to be compiled before it is run. Other interpreted languages include PHP and Ruby.
Python is dynamically typed, this means that you don't need to state the types of variables when you declare them or anything like that. You can do things like x=111 and then x="I'm a string" without error
Python is well suited to object orientated programming in that it allows the definition of classes along with composition and inheritance. Python does not have access specifiers (like C++'s public, private).
In Python, functions are first-class objects. This means that they can be assigned to variables, returned from other functions and passed into functions. Classes are also first class objects
Writing Python code is quick but running it is often slower than compiled languages. Fortunately, Python allows the inclusion of C based extensions so bottlenecks can be optimized away and often are. The numpy package is a good example of this, it's really quite quick because a lot of the number crunching it does isn't actually done by Python
Python finds use in many spheres – web applications, automation, scientific modeling, big data applications and many more. It's also often used as "glue" code to get other languages and components to play nice.

115.How is Python an interpreted language?
Ans: An interpreted language is any programming language which is not in machine level code before runtime. Therefore, Python is an interpreted language.

116.How is memory managed in Python?
Ans:

Memory management in python is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap. The python interpreter takes care of this instead.
The allocation of heap space for Python objects is done by Python's memory manager. The core API gives access to some tools for the programmer to code.
Python also has an inbuilt garbage collector, which recycles all the unused memory and so that it can be made available to the heap space.

117.What is PYTHONPATH?
Ans:It is an environment variable which is used when a module is imported. Whenever a module is imported, PYTHONPATH is also looked up to check for the presence of the imported modules in various directories. The interpreter uses it to determine which module to load.

118. What are python modules? Name some commonly used built-in modules in Python?
Ans:Python modules are files containing Python code. This code can either be functions classes or variables. A Python module is a .py file containing executable

code.

Some of the commonly used built-in modules are:

os
sys
math
random
data time
JSON
119.What are local variables and global variables in Python?
Ans:

Global Variables:

Variables declared outside a function or in global space are called global variables. These variables can be accessed by any function in the program.

Local Variables:

Any variable declared inside a function is known as a local variable. This variable is present in the local space and not in the global space.

Example:

```
1
2
3
4
5
6
a=2
def add():
b=3
c=a+b
print(c)
add()
```
Output: 5

When you try to access the local variable outside the function add(), it will throw an error.

120. Is python case sensitive?
Ans:Yes. Python is a case sensitive language.

121.What is type conversion in Python?
Ans:Type conversion refers to the conversion of one data type iinto another.

int() – converts any data type into integer type

float() – converts any data type into float type

ord() – converts characters into integer

hex() – converts integers to hexadecimal

oct() – converts integer to octal

tuple() – This function is used to convert to a tuple.

set() – This function returns the type after converting to set.

list() – This function is used to convert any data type to a list type.

dict() – This function is used to convert a tuple of order (key,value) into a dictionary.

str() – Used to convert integer into a string.

complex(real,imag) – This functionconverts real numbers to complex(real,imag) number.

122. How to install Python on Windows and set path variable?
Ans:To install Python on Windows, follow the below steps:

Install python from this link: https://www.python.org/downloads/
After this, install it on your PC. Look for the location where PYTHON has been installed on your PC using the following command on your command prompt: cmd python.
Then go to advanced system settings and add a new variable and name it as PYTHON_NAME and paste the copied path.
Look for the path variable, select its value and select ‘edit’.
Add a semicolon towards the end of the value if it’s not present and then type %PYTHON_HOME%

123. Is indentation required in python?
Ans:Indentation is necessary for Python. It specifies a block of code. All code within loops, classes, functions, etc is specified within an indented block. It is usually done using four space characters. If your code is not indented necessarily, it will not execute accurately and will throw errors as well.

124. What is the difference between Python Arrays and lists?
Ans:Arrays and lists, in Python, have the same way of storing data. But, arrays can hold only a single data type elements whereas lists can hold any data type elements.

Example:

```
1
2
3
4
5
import array as arr
My_Array=arr.array('i',[1,2,3,4])
My_list=[1,'abc',1.20]
print(My_Array)
print(My_list)
```

Output:

array(‘i’, [1, 2, 3, 4]) [1, ‘abc’, 1.2]

125. What are functions in Python?
Ans:A function is a block of code which is executed only when it is called. To define a Python function, the def keyword is used.

Example:

```
1
2
3
def Newfunc():
print("Hi, Welcome to Edureka")
Newfunc(); #calling the function
Output: Hi, Welcome to Edureka
```

126.What is __init__?
Ans:__init__ is a method or constructor in Python. This method is automatically called to allocate memory when a new object/ instance of a class is created. All classes have the __init__ method.

Here is an example of how to use it.

```
1
2
3
4
5
6
7
8
9
10
11
class Employee:
def __init__(self, name, age,salary):
self.name = name
self.age = age
self.salary = 20000
E1 = Employee("XYZ", 23, 20000)
# E1 is the instance of class Employee.
#__init__ allocates memory for E1.
print(E1.name)
print(E1.age)
print(E1.salary)
Output:
```

```
XYZ

23

20000
```

127.What is a lambda function?
Ans:An anonymous function is known as a lambda function. This function can have any number of parameters but, can have just one statement.

Example:

```
1
```

2
```
a = lambda x,y : x+y
print(a(5, 6))
```
Output: 11

128. What is self in Python?
Ans:Self is an instance or an object of a class. In Python, this is explicitly included as the first parameter. However, this is not the case in Java where it’s optional. It helps to differentiate between the methods and attributes of a class with local variables.

The self variable in the init method refers to the newly created object while in other methods, it refers to the object whose method was called.

129. How does break, continue and pass work?
Break Allows loop termination when some condition is met and the control is transferred to the next statement.
Continue Allows skipping some part of a loop when some specific condition is met and the control is transferred to the beginning of the loop
Pass Used when you need some block of code syntactically, but you want to skip its execution. This is basically a null operation. Nothing happens when this is executed.
130. What does [::-1} do?
Ans: [::-1] is used to reverse the order of an array or a sequence.
For example:
1
2
3
```
import array as arr
My_Array=arr.array('i',[1,2,3,4,5])
My_Array[::-1]
```
Output: array(‘i’, [5, 4, 3, 2, 1])

[::-1] reprints a reversed copy of ordered data structures such as an array or a list. the original array or list remains unchanged.

131. How can you randomize the items of a list in place in Python?
Ans: Consider the example shown below:

1
2
3
4
```
from random import shuffle
x = ['Keep', 'The', 'Blue', 'Flag', 'Flying', 'High']
shuffle(x)
print(x)
```
The output of the following code is as below.

['Flying', 'Keep', 'Blue', 'High', 'The', 'Flag']
132. What are python iterators?
Ans:Iterators are objects which can be traversed though or iterated upon.

133. How can you generate random numbers in Python?
Ans: Random module is the standard module that is used to generate a random number. The method is defined as:

1

2
import random
random.random
The statement random.random() method return the floating point number that is in the range of [0, 1). The function generates random float numbers. The methods that are used with the random class are the bound methods of the hidden instances. The instances of the Random can be done to show the multi-threading programs that creates a different instance of individual threads. The other random generators that are used in this are:

randrange(a, b): it chooses an integer and define the range in-between [a, b). It returns the elements by selecting it randomly from the range that is specified. It doesn't build a range object.
uniform(a, b): it chooses a floating point number that is defined in the range of [a,b).Iyt returns the floating point number
normalvariate(mean, sdev): it is used for the normal distribution where the mu is a mean and the sdev is a sigma that is used for standard deviation.
The Random class that is used and instantiated creates an independent multiple random number generators.
134. What is the difference between range & xrange?
Ans: For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please. The only difference is that range returns a Python list object and x range returns an xrange object.

This means that xrange doesn't actually generate a static list at run-time like range does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators. That means that if you have a really gigantic range you'd like to generate a list for, say one billion, xrange is the function to use.

This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can to create your array of integers, which can result in a Memory Error and crash your program. It's a memory hungry beast.

135. How do you write comments in python?
Ans:Comments in Python start with a # character. However, alternatively at times, commenting is done using docstrings(strings enclosed within triple quotes).

Example:

```
#Comments in Python start like this
print("Comments in Python start with a #")
```
Output: Comments in Python start with a #

136. What is pickling and unpickling?
Ans: Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using dump function, this process is called pickling. While the process of retrieving original Python objects from the stored string representation is called unpickling.

137. What are the generators in python?
Ans: Functions that return an iterable set of items are called generators.

138. How will you capitalize the first letter of string?
Ans:In Python, the capitalize() method capitalizes the first letter of a string. If the string already consists of a capital letter at the beginning, then, it returns the original string.

139. How will you convert a string to all lowercase?
Ans:To convert a string to lowercase, lower() function can be used.

Example:

```
1
2
stg='ABCD'
print(stg.lower())
Output: abcd
```

140. How to comment multiple lines in python?
Ans: Multi-line comments appear in more than one line. All the lines to be commented are to be prefixed by a #. You can also a very good shortcut method to comment multiple lines. All you need to do is hold the ctrl key and left click in every place wherever you want to include a # character and type a # just once. This will comment all the lines where you introduced your cursor.

141.What are docstrings in Python?
Ans:Docstrings are not actually comments, but, they are documentation strings. These docstrings are within triple quotes. They are not assigned to any variable and therefore, at times, serve the purpose of comments as well.

Example:

```
1
2
3
4
5
6
7
8
"""
Using docstring as a comment.
This code divides 2 numbers
"""
x=8
y=4
z=x/y
print(z)
Output: 2.0
```

141. What is the purpose of is, not and in operators?
Ans:Operators are special functions. They take one or more values and produce a corresponding result.

is: returns true when 2 operands are true (Example: “a” is ‘a’)

not: returns the inverse of the boolean value

in: checks if some element is present in some sequence

142. What is the usage of help() and dir() function in Python?
Ans: Help() and dir() both functions are accessible from the Python interpreter and used for viewing a consolidated dump of built-in functions.

Help() function: The help() function is used to display the documentation string

and also facilitates you to see the help related to modules, keywords, attributes, etc.
Dir() function: The dir() function is used to display the defined symbols.
143. Whenever Python exits, why isn’t all the memory de-allocated?
Ans:

Whenever Python exits, especially those Python modules which are having circular references to other objects or the objects that are referenced from the global namespaces are not always de-allocated or freed.
It is impossible to de-allocate those portions of memory that are reserved by the C library.
On exit, because of having its own efficient clean up mechanism, Python would try to de-allocate/destroy every other object.
143. What is a dictionary in Python?
Ans: The built-in datatypes in Python is called dictionary. It defines one-to-one relationship between keys and values. Dictionaries contain pair of keys and their corresponding values. Dictionaries are indexed by keys.

Let’s take an example:

The following example contains some keys. Country, Capital & PM. Their corresponding values are India, Delhi and Modi respectively.

```
1
dict={'Country':'India','Capital':'Delhi','PM':'Modi'}
1
print dict[Country]
India
1
print dict[Capital]
Delhi
1
print dict[PM]
Modi
```

144. How can the ternary operators be used in python?
Ans: The Ternary operator is the operator that is used to show the conditional statements. This consists of the true or false values with a statement that has to be evaluated for it.

Syntax:

The Ternary operator will be given as:
[on_true] if [expression] else [on_false]x, y = 25, 50big = x if x < y else y

Example:

The expression gets evaluated like if x<y else y, in this case if x<y is true then the value is returned as big=x and if it is incorrect then big=y will be sent as a result.

146. What does this mean: *args, **kwargs? And why would we use it?
Ans: We use *args when we aren’t sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. **kwargs is used when we don’t know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use *bob and **billy but that would not be wise.

147. What does len() do?

Ans:It is used to determine the length of a string, a list, an array, etc.

Example:

```
1
2
stg='ABCD'
len(stg)
```

148. Explain split(), sub(), subn() methods of “re” module in Python.
Ans: To modify the strings, Python’s “re” module is providing 3 methods. They are:

split() – uses a regex pattern to “split” a given string into a list.
sub() – finds all substrings where the regex pattern matches and then replace them with a different string
subn() – it is similar to sub() and also returns the new string along with the no. of replacements.

149. What are negative indexes and why are they used?
Ans: The sequences in Python are indexed and it consists of the positive as well as negative numbers. The numbers that are positive uses ‘0’ that is uses as first index and ‘1’ as the second index and the process goes on like that.

The index for the negative number starts from ‘-1’ that represents the last index in the sequence and ‘-2’ as the penultimate index and the sequence carries forward like the positive number.

The negative index is used to remove any new-line spaces from the string and allow the string to except the last character that is given as S[:-1]. The negative index is also used to show the index to represent the string in correct order.

150. What are Python packages?
Ans: Python packages are namespaces containing multiple modules.
151.How can files be deleted in Python?
Ans: To delete a file in Python, you need to import the OS Module. After that, you need to use the os.remove() function.

Example:

```
1
2
import os
os.remove("xyz.txt")
```

152. What are the built-in types of python?
Ans: Built-in types in Python are as follows –

Integers
Floating-point
Complex numbers
Strings
Boolean
Built-in functions

153. What advantages do NumPy arrays offer over (nested) Python lists?
Ans:

Python’s lists are efficient general-purpose containers. They support (fairly) efficient insertion, deletion, appending, and concatenation, and Python’s list comprehensions make them easy to construct and manipulate.
They have certain limitations: they don’t support “vectorized” operations like elementwise addition and multiplication, and the fact that they can contain objects of differing types mean that Python must store type information for every element,

and must execute type dispatching code when operating on each element.
NumPy is not just more efficient; it is also more convenient. You get a lot of vector and matrix operations for free, which sometimes allow one to avoid unnecessary work. And they are also efficiently implemented.
NumPy array is faster and You get a lot built in with NumPy, FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, etc.
154. How to add values to a python array?
Ans: Elements can be added to an array using the append(), extend() and the insert (i,x) functions.

Example:

```
1
2
3
4
5
6
7
a=arr.array('d', [1.1 , 2.1 ,3.1] )
a.append(3.4)
print(a)
a.extend([4.5,6.3,6.8])
print(a)
a.insert(2,3.8)
print(a)
```

Output:

array(‘d’, [1.1, 2.1, 3.1, 3.4])

array(‘d’, [1.1, 2.1, 3.1, 3.4, 4.5, 6.3, 6.8])

array(‘d’, [1.1, 2.1, 3.8, 3.1, 3.4, 4.5, 6.3, 6.8])

155. How to remove values to a python array?
Ans: Array elements can be removed using pop() or remove() method. The difference between these two functions is that the former returns the deleted value whereas the latter does not.

Example:

```
a=arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
print(a.pop())
print(a.pop(3))
a.remove(1.1)
print(a)
```

Output:

4.6

3.1

array(‘d’, [2.2, 3.8, 3.7, 1.2])

156. Does Python have OOps concepts?
Ans: Python is an object-oriented programming language. This means that any program can be solved in python by creating an object model. However, Python can be treated

as procedural as well as structural language.

157. What is the difference between deep and shallow copy?
Ans: Shallow copy is used when a new instance type gets created and it keeps the values that are copied in the new instance. Shallow copy is used to copy the reference pointers just like it copies the values. These references point to the original objects and the changes made in any member of the class will also affect the original copy of it. Shallow copy allows faster execution of the program and it depends on the size of the data that is used.

Deep copy is used to store the values that are already copied. Deep copy doesn't copy the reference pointers to the objects. It makes the reference to an object and the new object that is pointed by some other object gets stored. The changes made in the original copy won't affect any other copy that uses the object. Deep copy makes execution of the program slower due to making certain copies for each object that is been called.

158. How is Multithreading achieved in Python?
Ans:

Python has a multi-threading package but if you want to multi-thread to speed your code up, then it's usually not a good idea to use it.
Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread.
This happens very quickly so to the human eye it may seem like your threads are executing in parallel, but they are really just taking turns using the same CPU core.
All this GIL passing adds overhead to execution. This means that if you want to make your code run faster then using the threading package often isn't a good idea.
159. What is the process of compilation and linking in python?
Ans: The compiling and linking allows the new extensions to be compiled properly without any error and the linking can be done only when it passes the compiled procedure. If the dynamic loading is used then it depends on the style that is being provided with the system. The python interpreter can be used to provide the dynamic loading of the configuration setup files and will rebuild the interpreter.

The steps that are required in this as:

Create a file with any name and in any language that is supported by the compiler of your system. For example file.c or file.cpp
Place this file in the Modules/ directory of the distribution which is getting used.
Add a line in the file Setup.local that is present in the Modules/ directory.
Run the file using spam file.o
After a successful run of this rebuild the interpreter by using the make command on the top-level directory.
If the file is changed then run rebuildMakefile by using the command as 'make Makefile'.
160. What are Python libraries? Name a few of them.
Ans: Python libraries are a collection of Python packages. Some of the majorly used python libraries are – Numpy, Pandas, Matplotlib, Scikit-learn and many more.

161. What is split used for?
Ans: The split() method is used to separate a given string in Python.

Example:

```
2
a="KausalVikash python"
print(a.split())
```
Output: [‘KausalVikash’, ‘python’]

162. How to import modules in python?
Ans: Modules can be imported using the import keyword. You can import modules in three ways-

Example:

```
1
2
3
import array           #importing using the original module name
import array as arr    # importing using an alias name
from array import *    #imports everything present in the array module
```

163. Explain Inheritance in Python with an example.
Ans: Inheritance allows One class to gain all the members(say attributes and methods) of another class. Inheritance provides code reusability, makes it easier to create and maintain an application. The class from which we are inheriting is called super-class and the class that is inherited is called a derived / child class.

They are different types of inheritance supported by Python:

Single Inheritance – where a derived class acquires the members of a single super class.
Multi-level inheritance – a derived class d1 in inherited from base class base1, and d2 are inherited from base2.
Hierarchical inheritance – from one base class you can inherit any number of child classes
Multiple inheritance – a derived class is inherited from more than one base class.

164. How are classes created in Python?
Ans: Class in Python is created using the class keyword.

Example:

```
1
2
3
4
5
class Employee:
def __init__(self, name):
self.name = name
E1=Employee("abc")
print(E1.name)
```
Output: abc

165. What is monkey patching in Python?
Ans: In Python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

Consider the below example:

```
1
2
3
4
# m.py
class MyClass:
def f(self):
print "f()"
```

We can then run the monkey-patch testing like this:

```
1
2
3
4
5
6
7
import m
def monkey_f(self):
print "monkey_f()"
m.MyClass.f = monkey_f
obj = m.MyClass()
obj.f()
```

The output will be as below:

```
monkey_f()
```

As we can see, we did make some changes in the behavior of f() in MyClass using the function we defined, monkey_f(), outside of the module m.

166. Does python support multiple inheritance?
Ans: Multiple inheritance means that a class can be derived from more than one parent classes. Python does support multiple inheritance, unlike Java.

167. What is Polymorphism in Python?
Ans: Polymorphism means the ability to take multiple forms. So, for instance, if the parent class has a method named ABC then the child class also can have a method with the same name ABC having its own parameters and variables. Python allows polymorphism.

168. Define encapsulation in Python?
Ans: Encapsulation means binding the code and the data together. A Python class in an example of encapsulation.

169. How do you do data abstraction in Python?
Ans: Data Abstraction is providing only the required details and hiding the implementation from the world. It can be achieved in Python by using interfaces and abstract classes.

170.Does python make use of access specifiers?
Ans: Python does not deprive access to an instance variable or function. Python lays down the concept of prefixing the name of the variable, function or method with a single or double underscore to imitate the behavior of protected and private access specifiers.
171. How to create an empty class in Python?
Ans: An empty class is a class that does not have any code defined within its block. It can be created using the pass keyword. However, you can create objects of this class outside the class itself. IN PYTHON THE PASS command does nothing when its executed. it’s a null statement.

For example-
1
2
3
4
5
class a:
    pass
obj=a()
obj.name="xyz"
print("Name = ",obj.name)
Output:

Name = xyz

172.What’s The Process To Get The Home Directory Using ‘~’ In Python?
Ans: You need to import the os module, and then just a single line would do the rest.

import os
print (os.path.expanduser('~'))
Output:

/home/runner

173.How To Find Bugs Or Perform Static Analysis In A Python Application?
Ans:

You can use PyChecker, which is a static analyzer. It identifies the bugs in Python project and also reveals the style and complexity related bugs.
Another tool is Pylint, which checks whether the Python module satisfies the coding standard.
174.When Is The Python Decorator Used?
Ans: Python decorator is a relative change that you do in Python syntax to adjust the functions quickly.

175.Can Python be used for web client and web server side programming? And which one is best suited to Python?
Ans: Python is best suited for web server-side application development due to its vast set of features for creating business logic, database interactions, web server hosting etc.

However, Python can be used as a web client-side application which needs some conversions for a browser to interpret the client side logic. Also, note that Python can be used to create desktop applications which can run as a standalone application such as utilities for test automation.

176. Mention at least 3-4 benefits of using Python over the other scripting languages such as Javascript.
Ans: Enlisted below are some of the benefits of using Python.

Application development is faster and easy.
Extensive support of modules for any kind of application development including data analytics/machine learning/math-intensive applications.
An excellent support community to get your answers.

177.What is the type () in Python?
Ans: The built-in method which decides the types of the variable at the program runtime is known as type() in Python. When a single argument is passed through it, then it returns given object type. When 3 arguments pass through this, then it returns a new object type.

178.What are the key points of Python?
Ans:

Similar to PERL and PHP, Python is processed by the interpreter at runtime. Python supports Object-Oriented style of programming, which encapsulates code within objects.
Derived from other languages, such as ABC, C, C++, Modula-3, SmallTalk, Algol-68, Unix shell, and other scripting languages.
Python is copyrighted, and its source code is available under the GNU General Public License (GPL).
Supports the development of many applications, from text processing to games.
Works for scripting, embedded code and compiled the code.
Detailed

179.How is memory managed in Python?
Ans: Memory is managed by the private heap space. All objects and data structures are located in a private heap, and the programmer has no access to it. Only the interpreter has access. Python memory manager allocates heap space for objects. The programmer is given access to some tools for coding by the core API. The inbuilt garbage collector recycles the unused memory and frees up the memory to make it available for the heap space.

180.What tools can help find bugs or perform the static analysis?
Ans: For performing Static Analysis, PyChecker is a tool that detects the bugs in source code and warns the programmer about the style and complexity. Pylint is another tool that authenticates whether the module meets the coding standard.

181.How Does Python Handle Memory Management?
Ans:

Python uses private heaps to maintain its memory. So the heap holds all the Python objects and the data structures. This area is only accessible to the Python interpreter; programmers can’t use it.
And it’s the Python memory manager that handles the Private heap. It does the required allocation of the memory for Python objects.
Python employs a built-in garbage collector, which salvages all the unused memory and offloads it to the heap space.

182.What Are The Principal Differences Between The Lambda And Def?
Ans:

Lambda Vs. Def.
Def can hold multiple expressions while lambda is a uni-expression function.
Def generates a function and designates a name to call it later. Lambda forms a function object and returns it.

Def can have a return statement. Lambda can’t have return statements.
Lambda supports to get used inside a list and dictionary.

183.Write A Reg Expression That Confirms An Email Id Using The Python Reg Expression Module “Re”?
Ans: Python has a regular expression module “re.”

Check out the “re” expression that can check the email id for .com and .co.in subdomain.

```
import re
print(re.search(r"[0-9a-zA-Z.]+@[a-zA-Z]+\.(com|co\.in)$","micheal.pages@mp.com"))
```

184.What Do You Think Is The Output Of The Following Code Fragment? Is There Any Error In The Code?
Ans:

```
list = ['a', 'b', 'c', 'd', 'e']
print (list[10:])
```

The result of the above lines of code is []. There won’t be any error like an IndexError.

You should know that trying to fetch a member from the list using an index that exceeds the member count (for example, attempting to access list[10] as given in the question) would yield an IndexError. By the way, retrieving only a slice at the starting index that surpasses the no. of items in the list won’t result in an IndexError. It will just return an empty list.

185. Is There A Switch Or Case Statement In Python? If Not Then What Is The Reason For The Same?
Ans: No, Python does not have a Switch statement, but you can write a Switch function and then use it.

186.What Is A Built-In Function That Python Uses To Iterate Over A Number Sequence?
Ans: Range() generates a list of numbers, which is used to iterate over for loops.

```
for i in range(5):
    print(i)
```

The range() function accompanies two sets of parameters.

range(stop)
stop: It is the no. of integers to generate and starts from zero. eg. range(3) == [0, 1, 2].
range([start], stop[, step])
Start: It is the starting no. of the sequence.
Stop: It specifies the upper limit of the sequence.
Step: It is the incrementing factor for generating the sequence.
Points to note:
Only integer arguments are allowed.
Parameters can be positive or negative.
The range() function in Python starts from the zeroth index.

187.What Are The Optional Statements Possible Inside A Try-Except Block In Python?
Ans: There are two optional clauses you can use in the try-except block.

The “else” clause
It is useful if you want to run a piece of code when the try block doesn’t create an exception.
The “finally” clause
It is useful when you want to execute some steps which run, irrespective of whether there occurs an exception or not.

188.What Is A String In Python?
Ans: A string in Python is a sequence of alpha-numeric characters. They are immutable objects. It means that they don’t allow modification once they get assigned a value. Python provides several methods, such as join(), replace(), or split() to alter strings. But none of these change the original object.

189. What Is Slicing In Python?
Ans: Slicing is a string operation for extracting a part of the string, or some part of a list. In Python, a string (say text) begins at index 0, and the nth character stores at position text[n-1]. Python can also perform reverse indexing, i.e., in the backward direction, with the help of negative numbers. In Python, the slice() is also a constructor function which generates a slice object. The result is a set of indices mentioned by range(start, stop, step). The slice() method allows three parameters. 1. start – starting number for the slicing to begin. 2. stop – the number which indicates the end of slicing. 3. step – the value to increment after each index (default = 1).

190. What Is %S In Python?
Ans: Python has support for formatting any value into a string. It may contain quite complex expressions.

One of the common usages is to push values into a string with the %s format specifier. The formatting operation in Python has the comparable syntax as the C function printf() has.

191.What Is The Index In Python?
Ans: An index is an integer data type which denotes a position within an ordered list or a string.

In Python, strings are also lists of characters. We can access them using the index which begins from zero and goes to the length minus one.

For example, in the string “Program,” the indexing happens like this:

Program 0 1 2 3 4 5
192. What Is Docstring In Python?
Ans: A docstring is a unique text that happens to be the first statement in the following Python constructs:

Module, Function, Class, or Method definition.

A docstring gets added to the __doc__ attribute of the string object.

193.What Is A Function In Python Programming?
Ans: A function is an object which represents a block of code and is a reusable entity. It brings modularity to a program and a higher degree of code reusability.

Python has given us many built-in functions such as print() and provides the ability to create user-defined functions.

194. How Many Basic Types Of Functions Are Available In Python?
Ans: Python gives us two basic types of functions.

1. Built-in, and

2. User-defined.

The built-in functions happen to be part of the Python language. Some of these are print(), dir(), len(), and abs() etc.

195. How Do We Write A Function In Python?
Ans: We can create a Python function in the following manner.

Step-1: to begin the function, start writing with the keyword def and then mention the function name.

Step-2: We can now pass the arguments and enclose them using the parentheses. A colon, in the end, marks the end of the function header.

Step-3: After pressing an enter, we can add the desired Python statements for execution.

196. What Is A Function Call Or A Callable Object In Python?
Ans: A function in Python gets treated as a callable object. It can allow some arguments and also return a value or multiple values in the form of a tuple. Apart from the function, Python has other constructs, such as classes or the class instances which fits in the same category.

197. What Is The Return Keyword Used For In Python?
Ans: The purpose of a function is to receive the inputs and return some output.

The return is a Python statement which we can use in a function for sending a value back to its caller.

198. What Is “Call By Value” In Python?
Ans: In call-by-value, the argument whether an expression or a value gets bound to the respective variable in the function.

Python will treat that variable as local in the function-level scope. Any changes made to that variable will remain local and will not reflect outside the function.

199. What Is “Call By Reference” In Python?
Ans: We use both “call-by-reference” and “pass-by-reference” interchangeably. When we pass an argument by reference, then it is available as an implicit reference to the function, rather than a simple copy. In such a case, any modification to the argument will also be visible to the caller.

This scheme also has the advantage of bringing more time and space efficiency because it leaves the need for creating local copies.

On the contrary, the disadvantage could be that a variable can get changed accidentally during a function call. Hence, the programmers need to handle in the code to avoid such uncertainty.

200. What Is The Return Value Of The Trunc() Function?

Ans: The Python trunc() function performs a mathematical operation to remove the decimal values from a particular expression and provides an integer value as its output.

201. Is It Mandatory For A Python Function To Return A Value?

Ans: It is not at all necessary for a function to return any value. However, if needed, we can use None as a return value.

202. What Does The Continue Do In Python?

Ans: The continue is a jump statement in Python which moves the control to execute the next iteration in a loop leaving all the remaining instructions in the block unexecuted.

The continue statement is applicable for both the “while” and “for” loops.

203. What Is The Purpose Of Id() Function In Python?

Ans: The id() is one of the built-in functions in Python.

Signature: id(object)

It accepts one parameter and returns a unique identifier associated with the input object.

204. What Does The *Args Do In Python?

Ans: We use *args as a parameter in the function header. It gives us the ability to pass N (variable) number of arguments.

Please note that this type of argument syntax doesn’t allow passing a named argument to the function.

Example of using the *args:

```
# Python code to demonstrate
# *args for dynamic arguments
def fn(*argList):
    for argx in argList:
        print (argx)

fn('I', 'am', 'Learning', 'Python')
```

The output:

```
I
am
Learning
Python
```

205.Does Python Have A Main() Method?

Ans: The main() is the entry point function which happens to be called first in most programming languages.

Since Python is interpreter-based, so it sequentially executes the lines of the code one-by-one.

Python also does have a Main() method. But it gets executed whenever we run our Python script either by directly clicking it or starts it from the command line.

We can also override the Python default main() function using the Python if statement. Please see the below code.

```
print("Welcome")
```

```
print("__name__ contains: ", __name__)
def main():
    print("Testing the main function")
if __name__ == '__main__':
    main()
```
The output:

```
Welcome
__name__ contains:  __main__
Testing the main function
```
206. What Does The __ Name __ Do In Python?
Ans: The __name__ is a unique variable. Since Python doesn't expose the main() function, so when its interpreter gets to run the script, it first executes the code which is at level 0 indentation.

To see whether the main() gets called, we can use the __name__ variable in an if clause compares with the value "__main__."

207. What Is The Purpose Of "End" In Python?
Ans: Python's print() function always prints a newline in the end. The print() function accepts an optional parameter known as the 'end.' Its value is '\n' by default. We can change the end character in a print statement with the value of our choice using this parameter.

```
# Example: Print a  instead of the new line in the end.
print("Let's learn" , end = ' ')
print("Python")

# Printing a dot in the end.
print("Learn to code from techbeamers" , end = '.')
print("com", end = ' ')
```
The output is:

```
Let's learn Python
Learn to code from techbeamers.com
```
208. When Should You Use The "Break" In Python?
Ans: Python provides a break statement to exit from a loop. Whenever the break hits in the code, the control of the program immediately exits from the body of the loop.

The break statement in a nested loop causes the control to exit from the inner iterative block.

209.What Is The Difference Between Pass And Continue In Python?
Ans: The continue statement makes the loop to resume from the next iteration.

On the contrary, the pass statement instructs to do nothing, and the remainder of the code executes as usual.

210. What Does The Len() Function Do In Python?
Ans: In Python, the len() is a primary string function. It determines the length of an input string.

```
>>> some_string = 'techbeamers'
>>> len(some_string)
11
```
211. What Does The Chr() Function Do In Python?

Ans: The chr() function got re-added in Python 3.2. In version 3.0, it got removed.

It returns the string denoting a character whose Unicode code point is an integer.

For example, the chr(122) returns the string ‘z’ whereas the chr(1212) returns the string ‘ҽ’.

212. What Does The Ord() Function Do In Python?
Ans: The ord(char) in Python takes a string of size one and returns an integer denoting the Unicode code format of the character in case of a Unicode type object, or the value of the byte if the argument is of 8-bit string type.

```
>>> ord("z")
122
```

213. What Is Rstrip() In Python?
Ans: Python provides the rstrip() method which duplicates the string but leaves out the whitespace characters from the end.

The rstrip() escapes the characters from the right end based on the argument value, i.e., a string mentioning the group of characters to get excluded.

The signature of the rstrip() is:

```
str.rstrip([char sequence/pre>
#Example
test_str = 'Programming    '
# The trailing whitespaces are excluded
print(test_str.rstrip())
```

214.What Is Whitespace In Python?
Ans: Whitespace represents the characters that we use for spacing and separation.

They possess an “empty” representation. In Python, it could be a tab or space.

215. What Is Isalpha() In Python?
Ans: Python provides this built-in isalpha() function for the string handling purpose.

It returns True if all characters in the string are of alphabet type, else it returns False.

216. How Do You Use The Split() Function In Python?
Python’s split() function works on strings to cut a large piece into smaller chunks, or sub-strings. We can specify a separator to start splitting, or it uses the space as one by default.

```
#Example
str = 'pdf csv json'
print(str.split(" "))
print(str.split())
```

The output:

```
['pdf', 'csv', 'json']
['pdf', 'csv', 'json']
```

217. What Does The Join Method Do In Python?
Ans: Python provides the join() method which works on strings, lists, and tuples. It combines them and returns a united value.

218. What Does The Title() Method Do In Python?
Ans: Python provides the title() method to convert the first letter in each word to capital format while the rest turns to Lowercase.

```
#Example
str = 'lEaRn pYtHoN'
print(str.title())
```
The output:

```
Learn Python
```
Now, check out some general purpose Python interview questions.

219. What Makes The CPython Different From Python?
Ans: CPython has its core developed in C. The prefix ‘C’ represents this fact. It runs an interpreter loop used for translating the Python-ish code to C language.

220. Which Package Is The Fastest Form Of Python?
Ans: PyPy provides maximum compatibility while utilizing CPython implementation for improving its performance.

The tests confirmed that PyPy is nearly five times faster than the CPython. It currently supports Python 2.7.

221. What Is GIL In Python Language?
Ans: Python supports GIL (the global interpreter lock) which is a mutex used to secure access to Python objects, synchronizing multiple threads from running the Python bytecodes at the same time.

222. How Is Python Thread Safe?
Ans: Python ensures safe access to threads. It uses the GIL mutex to set synchronization. If a thread loses the GIL lock at any time, then you have to make the code thread-safe.

For example, many of the Python operations execute as atomic such as calling the sort() method on a list.

223. How Does Python Manage The Memory?
Ans: Python implements a heap manager internally which holds all of its objects and data structures.

This heap manager does the allocation/de-allocation of heap space for objects.

224.What Is The Set Object In Python?
Ans: Sets are unordered collection objects in Python. They store unique and immutable objects. Python has its implementation derived from mathematics.

225. What Is The Use Of The Dictionary In Python?
Ans: A dictionary has a group of objects (the keys) map to another group of objects (the values). A Python dictionary represents a mapping of unique Keys to Values.

They are mutable and hence will not change. The values associated with the keys can be of any Python types.

226. Is Python List A Linked List?
Ans: A Python list is a variable-length array which is different from C-style linked lists.

Internally, it has a contiguous array for referencing to other objects and stores a pointer to the array variable and its length in the list head structure.

Here are some Python interview questions on classes and objects

227.What Is Class In Python?
Ans: Python supports object-oriented programming and provides almost all OOP features to use in programs.

A Python class is a blueprint for creating the objects. It defines member variables and gets their behavior associated with them.

We can make it by using the keyword “class.” An object gets created from the constructor. This object represents the instance of the class.

In Python, we generate classes and instances in the following way.

```
>>>class Human:  # Create the class
...     pass
>>>man = Human()  # Create the instance
>>>print(man)
<__main__.Human object at 0x0000000003559E10>
```

228. What Are Attributes And Methods In A Python Class?
Ans: A class is useless if it has not defined any functionality. We can do so by adding attributes. They work as containers for data and functions. We can add an attribute directly specifying inside the class body.

```
>>> class Human:
...     profession = "programmer" # specify the attribute 'profession' of the class
>>> man = Human()
>>> print(man.profession)
programmer
```

After we added the attributes, we can go on to define the functions. Generally, we call them methods. In the method signature, we always have to provide the first argument with a self-keyword.

```
>>> class Human:
    profession = "programmer"
    def set_profession(self, new_profession):
        self.profession = new_profession
>>> man = Human()
>>> man.set_profession("Manager")
>>> print(man.profession)
Manager
```

229. How To Assign Values For The Class Attributes At Runtime?
Ans: We can specify the values for the attributes at runtime. We need to add an init method and pass input to object constructor. See the following example demonstrating this.

```
>>> class Human:
    def __init__(self, profession):
        self.profession = profession
    def set_profession(self, new_profession):
        self.profession = new_profession

>>> man = Human("Manager")
>>> print(man.profession)
```

Manager

230.What Is Inheritance In Python Programming?

Ans: Inheritance is an OOP mechanism which allows an object to access its parent class features. It carries forward the base class functionality to the child.

Python Interview Questions - Inheritance

We do it intentionally to abstract away the similar code in different classes.

The common code rests with the base class, and the child class objects can access it via inheritance. Check out the below example.

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def set_processor(self, new_processor):
        processor = new_processor

class Desktop(PC): # Derived class
    os = "Mac OS High Sierra" # Personalized attribute
    ram = "32 GB"

class Laptop(PC): # Derived class
    os = "Windows 10 Pro 64" # Personalized attribute
    ram = "16 GB"

desk = Desktop()
print(desk.processor, desk.os, desk.ram)

lap = Laptop()
print(lap.processor, lap.os, lap.ram)
```

The output:

```
Xeon Mac OS High Sierra 32 GB
Xeon Windows 10 Pro 64 16 GB
```

231.What Is Composition In Python?

Ans: The composition is also a type of inheritance in Python. It intends to inherit from the base class but a little differently, i.e., by using an instance variable of the base class acting as a member of the derived class.

See the below diagram.

Python Interview Questions - Composition

To demonstrate composition, we need to instantiate other objects in the class and then make use of those instances.

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def __init__(self, processor, ram):
        self.processor = processor
        self.ram = ram

    def set_processor(self, new_processor):
        processor = new_processor

    def get_PC(self):
        return "%s cpu & %s ram" % (self.processor, self.ram)
```

```
class Tablet():
    make = "Intel"
    def __init__(self, processor, ram, make):
        self.PC = PC(processor, ram) # Composition
        self.make = make

    def get_Tablet(self):
        return "Tablet with %s CPU & %s ram by %s" % (self.PC.processor,
self.PC.ram, self.make)

if __name__ == "__main__":
    tab = Tablet("i7", "16 GB", "Intel")
    print(tab.get_Tablet())
```
The output is:

```
Tablet with i7 CPU & 16 GB ram by Intel
```

232.What Are Errors And Exceptions In Python Programs?
Ans: Errors are coding issues in a program which may cause it to exit abnormally.

On the contrary, exceptions happen due to the occurrence of an external event which interrupts the normal flow of the program.

233. How Do You Handle Exceptions With Try/Except/Finally In Python?
Ans: Python lay down Try, Except, Finally constructs to handle errors as well as Exceptions. We enclose the unsafe code indented under the try block. And we can keep our fall-back code inside the except block. Any instructions intended for execution last should come under the finally block.

```
try:
    print("Executing code in the try block")
    print(exception)
except:
    print("Entering in the except block")
finally:
    print("Reached to the final block")
```
The output is:

```
Executing code in the try block
Entering in the except block
Reached to the final block
```
234. How Do You Raise Exceptions For A Predefined Condition In Python?
Ans: We can raise an exception based on some condition.

For example, if we want the user to enter only odd numbers, else will raise an exception.

```
# Example - Raise an exception
while True:
    try:
        value = int(input("Enter an odd number- "))
        if value%2 == 0:
            raise ValueError("Exited due to invalid input!!!")
        else:
            print("Value entered is : %s" % value)
    except ValueError as ex:
```

```
        print(ex)
        break
```
The output is:

```
Enter an odd number- 2
Exited due to invalid input!!!
Enter an odd number- 1
Value entered is : 1
Enter an odd number-
```
235. What Are Python Iterators?
Ans: Iterators in Python are array-like objects which allow moving on the next element. We use them in traversing a loop, for example, in a “for” loop.

Python library has a no. of iterators. For example, a list is also an iterator and we can start a for loop over it.

236. What Is The Difference Between An Iterator And Iterable?
Ans: The collection type like a list, tuple, dictionary, and set are all iterable objects whereas they are also iterable containers which return an iterator while traversing.

Here are some advanced-level Python interview questions.

237. What Are Python Generators?
Ans: A Generator is a kind of function which lets us specify a function that acts like an iterator and hence can get used in a “for” loop.

In a generator function, the yield keyword substitutes the return statement.

```
# Simple Python function
def fn():
    return "Simple Python function."

# Python Generator function
def generate():
    yield "Python Generator function."

print(next(generate()))
```
The output is:

```
Python Generator function.
```

238.What Are Closures In Python?
Ans: Python closures are function objects returned by another function. We use them to eliminate code redundancy.

In the example below, we’ve written a simple closure for multiplying numbers.

```
def multiply_number(num):
    def product(number):
        'product() here is a closure'
        return num * number
    return product

num_2 = multiply_number(2)
print(num_2(11))
print(num_2(24))
```

```
num_6 = multiply_number(6)
print(num_6(1))
```
The output is:

```
22
48
6
```
239. What Are Decorators In Python?
Ans: Python decorator gives us the ability to add new behavior to the given objects dynamically. In the example below, we've written a simple example to display a message pre and post the execution of a function.

```
def decorator_sample(func):
    def decorator_hook(*args, **kwargs):
        print("Before the function call")
        result = func(*args, **kwargs)
        print("After the function call")
        return result
    return decorator_hook

@decorator_sample
def product(x, y):
    "Function to multiply two numbers."
    return x * y

print(product(3, 3))
```
The output is:

```
Before the function call
After the function call
9
```
240. How Do You Create A Dictionary In Python?
Ans: Let's take the example of building site statistics. For this, we first need to break up the key-value pairs using a colon(":"). The keys should be of an immutable type, i.e., so we'll use the data-types which don't allow changes at runtime. We'll choose from an int, string, or tuple.

However, we can take values of any kind. For distinguishing the data pairs, we can use a comma(",") and keep the whole stuff inside curly braces({…}).

```
 >>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> type(site_stats)
<class 'dict'>
>>> print(site_stats)
{'type': 'organic', 'site': 'tecbeamers.com', 'traffic': 10000}
```
241. How Do You Read From A Dictionary In Python?
Ans:To fetch data from a dictionary, we can directly access using the keys. We can enclose a "key" using brackets […] after mentioning the variable name corresponding to the dictionary.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> print(site_stats["traffic"])
```
We can even call the get method to fetch the values from a dict. It also let us set a default value. If the key is missing, then the KeyError would occur.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> print(site_stats.get('site'))
tecbeamers.com
```
242. How Do You Traverse Through A Dictionary Object In Python?

Ans: We can use the “for” and “in” loop for traversing the dictionary object.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> for k, v in site_stats.items():
    print("The key is: %s" % k)
    print("The value is: %s" % v)
    print("++++++++++++++++++++++++")
```

The output is:

```
The key is: type
The value is: organic
++++++++++++++++++++++++
The key is: site
The value is: tecbeamers.com
++++++++++++++++++++++++
The key is: traffic
The value is: 10000
++++++++++++++++++++++++
```

243. How Do You Add Elements To A Dictionary In Python?

Ans: We can add elements by modifying the dictionary with a fresh key and then set the value to it.

```
>>> # Setup a blank dictionary
>>> site_stats = {}
>>> site_stats['site'] = 'google.com'
>>> site_stats['traffic'] = 10000000000
>>> site_stats['type'] = 'Referral'
>>> print(site_stats)
{'type': 'Referral', 'site': 'google.com', 'traffic': 10000000000}
```

We can even join two dictionaries to get a bigger dictionary with the help of the update() method.

```
>>> site_stats['site'] = 'google.co.in'
>>> print(site_stats)
{'site': 'google.co.in'}
>>> site_stats_new = {'traffic': 1000000, "type": "social media"}
>>> site_stats.update(site_stats_new)
>>> print(site_stats)
{'type': 'social media', 'site': 'google.co.in', 'traffic': 1000000}
```

244. How Do You Delete Elements Of A Dictionary In Python?

Ans: We can delete a key in a dictionary by using the del() method.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> del site_stats["type"]
>>> print(site_stats)
{'site': 'google.co.in', 'traffic': 1000000}
```

Another method, we can use is the pop() function. It accepts the key as the parameter. Also, a second parameter, we can pass a default value if the key doesn’t exist.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> print(site_stats.pop("type", None))
organic
>>> print(site_stats)
{'site': 'tecbeamers.com', 'traffic': 10000}
```

245. How Do You Check The Presence Of A Key In A Dictionary?

Ans: We can use Python’s “in” operator to test the presence of a key inside a dict object.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type": "organic"}
>>> 'site' in site_stats
True
>>> 'traffic' in site_stats
True
>>> "type" in site_stats
True
```
Earlier, Python also provided the has_key() method which got deprecated.

246. What Is The Syntax For List Comprehension In Python?
Ans: The signature for the list comprehension is as follows:

```
[ expression(var) for var in iterable ]
```
For example, the below code will return all the numbers from 10 to 20 and store them in a list.

```
>>> alist = [var for var in range(10, 20)]
>>> print(alist)
```
247. What Is The Syntax For Dictionary Comprehension In Python?
A dictionary has the same syntax as was for the list comprehension but the difference is that it uses curly braces:

```
{ aKey, itsValue for aKey in iterable }
```
For example, the below code will return all the numbers 10 to 20 as the keys and will store the respective squares of those numbers as the values.

```
>>> adict = {var:var**2 for var in range(10, 20)}
>>> print(adict)
```
248. What Is The Syntax For Generator Expression In Python?
Ans: The syntax for generator expression matches with the list comprehension, but the difference is that it uses parenthesis:

```
( expression(var) for var in iterable )
```
For example, the below code will create a generator object that generates the values from 10 to 20 upon using it.

```
>>> (var for var in range(10, 20))
 at 0x0000000003668728>
>>> list((var for var in range(10, 20)))
```
Now, see more Python interview questions for practice.

249. How Do You Write A Conditional Expression In Python?
Ans: We can utilize the following single statement as a conditional expression.
default_statment if Condition else another_statement

```
>>> no_of_days = 366
>>> is_leap_year = "Yes" if no_of_days == 366 else "No"
>>> print(is_leap_year)
Yes
```
250. What Do You Know About The Python Enumerate?
Ans: While using the iterators, sometimes we might have a use case to store the count of iterations. Python gets this task quite easy for us by giving a built-in method known as the enumerate().

The enumerate() function attaches a counter variable to an iterable and returns it as the “enumerated” object.

We can use this object directly in the “for” loops or transform it into a list of tuples by calling the list() method. It has the following signature:

```
enumerate(iterable, to_begin=0)
Arguments:
iterable: array type object which enables iteration
to_begin: the base index for the counter is to get started, its default value is 0
# Example - enumerate function
alist = ["apple","mango", "orange"]
astr = "banana"

# Let's set the enumerate objects
list_obj = enumerate(alist)
str_obj = enumerate(astr)

print("list_obj type:", type(list_obj))
print("str_obj type:", type(str_obj))

print(list(enumerate(alist)) )
# Move the starting index to two from zero
print(list(enumerate(astr, 2)))
The output is:

list_obj type: <class 'enumerate'>
str_obj type: <class 'enumerate'>
[(0, 'apple'), (1, 'mango'), (2, 'orange')]
[(2, 'b'), (3, 'a'), (4, 'n'), (5, 'a'), (6, 'n'), (7, 'a')]
251. What Is The Use Of Globals() Function In Python?
Ans: The globals() function in Python returns the current global symbol table as a
dictionary object.

Python maintains a symbol table to keep all necessary information about a program.
This info includes the names of variables, methods, and classes used by the
program.

All the information in this table remains in the global scope of the program and
Python allows us to retrieve it using the globals() method.

Signature: globals()

Arguments: None
# Example: globals() function
x = 9
def fn():
    y = 3
    z = y + x
    # Calling the globals() method
    z = globals()['x'] = z
    return z

# Test Code
ret = fn()
print(ret)
The output is:

12
252. Why Do You Use The Zip() Method In Python?
Ans: The zip method lets us map the corresponding index of multiple containers so
that we can use them using as a single unit.

Signature:
```

```
 zip(*iterators)
Arguments:
 Python iterables or collections (e.g., list, string, etc.)
Returns:
 A single iterator object with combined mapped values
# Example: zip() function

emp = [ "tom", "john", "jerry", "jake" ]
age = [ 32, 28, 33, 44 ]
dept = [ 'HR', 'Accounts', 'R&D', 'IT' ]

# call zip() to map values
out = zip(emp, age, dept)

# convert all values for printing them as set
out = set(out)

# Displaying the final values
print ("The output of zip() is : ",end="")
print (out)
The output is:

The output of zip() is : {('jerry', 33, 'R&D'), ('jake', 44, 'IT'), ('john', 28,
'Accounts'), ('tom', 32, 'HR')}
253. What Are Class Or Static Variables In Python Programming?
Ans: In Python, all the objects share common class or static variables.

But the instance or non-static variables are altogether different for different
objects.

The programming languages like C++ and Java need to use the static keyword to make
a variable as the class variable. However, Python has a unique way to declare a
static variable.

All names initialized with a value in the class declaration becomes the class
variables. And those which get assigned values in the class methods becomes the
instance variables.

# Example
class Test:
    aclass = 'programming' # A class variable
    def __init__(self, ainst):
        self.ainst = ainst # An instance variable

# Objects of CSStudent class
test1 = Test(1)
test2 = Test(2)

print(test1.aclass)
print(test2.aclass)
print(test1.ainst)
print(test2.ainst)

# A class variable is also accessible using the class name
print(Test.aclass)
The output is:

programming
programming
```

1
2
programming
Let’s now answer some advanced-level Python interview questions.

254. How Does The Ternary Operator Work In Python?
Ans: The ternary operator is an alternative for the conditional statements. It combines true or false values with a statement that you need to test.

The syntax would look like the one given below.

[onTrue] if [Condition] else [onFalse]

```
x, y = 35, 75
smaller = x if x < y else y
print(smaller)
```

255. What Does The “Self” Keyword Do?
Ans: The self is a Python keyword which represents a variable that holds the instance of an object.

In almost, all the object-oriented languages, it is passed to the methods as a hidden parameter.

256. What Are The Different Methods To Copy An Object In Python?
Ans: There are two ways to copy objects in Python.

copy.copy() function
It makes a copy of the file from source to destination.
It’ll return a shallow copy of the parameter.
copy.deepcopy() function
It also produces the copy of an object from the source to destination.
It’ll return a deep copy of the parameter that you can pass to the function.
257: What Is The Purpose Of Docstrings In Python?
Ans: In Python, the docstring is what we call as the docstrings. It sets a process of recording Python functions, modules, and classes.

258. Which Python Function Will You Use To Convert A Number To A String?
Ans: For converting a number into a string, you can use the built-in function str(). If you want an octal or hexadecimal representation, use the inbuilt function oct() or hex().

259. How Do You Debug A Program In Python? Is It Possible To Step Through The Python Code?
Ans: Yes, we can use the Python debugger (pdb) to debug any Python program. And if we start a program using pdb, then it let us even step through the code.

260. List Down Some Of The PDB Commands For Debugging Python Programs?
Ans: Here are a few PDB commands to start debugging Python code.

Add breakpoint (b)
Resume execution (c)
Step by step debugging (s)
Move to the next line (n)
List source code (l)
Print an expression (p)
261. What Is The Command To Debug A Python Program?
Ans: The following command helps run a Python program in debug mode.

```
$ python -m pdb python-script.py
```

262. How Do You Monitor The Code Flow Of A Program In Python?
Ans: In Python, we can use the sys module’s settrace() method to setup trace hooks and monitor the functions inside a program.

You need to define a trace callback method and pass it to the settrace() function. The callback should specify three arguments as shown below.

```
import sys

def trace_calls(frame, event, arg):
    # The 'call' event occurs before a function gets executed.
    if event != 'call':
        return
    # Next, inspect the frame data and print information.
    print 'Function name=%s, line num=%s' % (frame.f_code.co_name, frame.f_lineno)
    return

def demo2():
    print 'in demo2()'

def demo1():
    print 'in demo1()'
    demo2()

sys.settrace(trace_calls)
demo1()
```

263. How long can an identifier be in Python?
Ans: According to the official Python documentation, an identifier can be of any length. However, PEP 8 suggests that you should limit all lines to a maximum of 79 characters. Also, PEP 20 says ‘readability counts’. So, a very long identifier will violate PEP-8 and PEP-20.

Apart from that, there are certain rules we must follow to name one:

According to the official Python documentation, an identifier can be of any length. However, PEP 8 suggests that you should limit all lines to a maximum of 79 characters. Also, PEP 20 says ‘readability counts’. So, a very long identifier will violate PEP-8 and PEP-20.

Apart from that, there are certain rules we must follow to name one:

It can only begin with an underscore or a character from A-Z or a-z.
The rest of it can contain anything from the following: A-Z/a-z/_/0-9.
Python is case-sensitive, as we discussed in the previous question.
Keywords cannot be used as identifiers. Python has the following keywords:

```
and   def   False import      not   True
as    del   finally     in    or    try
assert      elif  for   is    pass  while
break else  from  lambda      print with
class except      global      None  raise yield
continue    exec  if    nonlocal    return
```

264. How would you convert a string into lowercase?
Ans: We use the lower() method for this.

```
>>> ‘AyuShi’.lower()
‘ayushi’
```

To convert it into uppercase, then, we use upper().

```
>>> ‘AyuShi’.upper()
‘AYUSHI’
```

Also, to check if a string is in all uppercase or all lowercase, we use the methods isupper() and islower().

```
>>> ‘AyuShi’.isupper()
False

>>> ‘AYUSHI’.isupper()
True

>>> ‘ayushi’.islower()
True

>>> ‘@yu$hi’.islower()
True

>>> ‘@YU$HI’.isupper()
True
```

So, characters like @ and $ will suffice for both cases

Also, istitle() will tell us if a string is in title case.

```
>>> ‘The Corpse Bride’.istitle()
True
```

265. 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.

```
>>> def func(*args):
pass
>>>
```

Similarly, the break statement breaks out of a loop.

```
>>> for i in range(7):
if i==3: break
print(i)
1

2
```

Finally, the continue statement skips to the next iteration.

```
>>> for i in range(7):
if i==3: continue
```

```
print(i)
1

2

4

5

6
```

266. Explain help() and dir() functions in Python?
Ans:

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

```
>>> import copy
>>> 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).

```
>>> dir(copy.copy)
['__annotations__', '__call__', '__class__', '__closure__', '__code__',
'__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__get__', '__getattribute__', '__globals__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__',
'__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__']
```

267. How do you get a list of all the keys in a dictionary?
Ans:For this, we use the function keys().

```
>>> mydict={'a':1,'b':2,'c':3,'e':5}
>>> mydict.keys()
dict_keys(['a', 'b', 'c', 'e'])
```

268. How will you check if all characters in a string are alphanumeric?
Ans: For this, we use the method isalnum().

269. With Python, how do you find out which directory you are currently in?
Ans: To find this, we use the function/method getcwd(). We import it from the module os.

```
>>> import os
>>> os.getcwd()
```

‘C:\\Users\\lifei\\AppData\\Local\\Programs\\Python\\Python36-32’

```
>>> type(os.getcwd)
<class ‘builtin_function_or_method’>
```

We can also change the current working directory with chdir().

```
>>> os.chdir(‘C:\\Users\\lifei\\Desktop’)
>>> os.getcwd()
‘C:\\Users\\lifei\\Desktop’
```

270. How do you insert an object at a given index in Python?
Ans: Let’s build a list first.

```
>>> a=[1,2,4]
```

Now, we use the method insert. The first argument is the index at which to insert, the second is the value to insert.

```
>>> a.insert(2,3)
>>> a
[1, 2, 3, 4]
```

271. how do you reverse a list?
Ans: Using the reverse() method.

```
>>> a.reverse()
>>> a
[4, 3, 2, 1]
```

You can also do it via slicing from right to left:

```
>>> a[::-1]
>>> a
[1, 2, 3, 4]
```

This gives us the original list because we already reversed it once. However, this does not modify the original list to reverse it.

272. How does a function return values?
Ans: A function uses the ‘return’ keyword to return a value. Take a look:

```
>>> def add(a,b):
return a+b
```

273. How would you define a block in Python?
Ans: For any kind of statements, we possibly need to define a block of code under them. However, Python does not support curly braces. This means we must end such statements with colons and then indent the blocks under those with the same amount.

```
>>> if 3>1:
print(“Hello”)
print(“Goodbye”)
```

Hello

Goodbye

274. Will the do-while loop work if you don’t end it with a semicolon?
Ans: Trick question! Python does not support an intrinsic do-while loop. Secondly, to terminate do-while loops is a necessity for languages like C++.

275. In one line, show us how you’ll get the max alphabetical character from a string.?
Ans: For this, we’ll simply use the max function.

```
>>> max(‘flyiNg’)
‘y’
```

The following are the ASCII values for all the letters of this string-

f- 102

l- 108

y- 121

i- 105

N- 78

g- 103

By this logic, try to explain the following line of code-

```
>>> max(‘fly{}iNg’)
‘}’
```

(Bonus: } – 125)

276. What is Python good for?
Ans: Python is a jack of many trades, check out Applications of Python to find out more.

Meanwhile, we’ll say we can use it for:

Web and Internet Development
Desktop GUI
Scientific and Numeric Applications
Software Development Applications
Applications in Education
Applications in Business
Database Access
Network Programming
Games, 3D Graphics

Other Python Applications

277. Can you name ten built-in functions in Python and explain each in brief?
Ans: Ten Built-in Functions, you say? Okay, here you go.

complex()- Creates a complex number.

```
>>> complex(3.5,4)
(3.5+4j)
```

eval()- Parses a string as an expression.

```
>>> eval('print(max(22,22.0)-min(2,3))')
20
```

filter()- Filters in items for which the condition is true.

```
>>> list(filter(lambda x:x%2==0,[1,2,0,False]))
[2, 0, False]
```

format()- Lets us format a string.

```
>>> print("a={0} but b={1}".format(a,b))
a=2 but b=3
```

hash()- Returns the hash value of an object.

```
>>> hash(3.7)
644245917
```

hex()- Converts an integer to a hexadecimal.

```
>>> hex(14)
'0xe'
```

input()- Reads and returns a line of string.

```
>>> input('Enter a number')
Enter a number7

'7'
```

len()- Returns the length of an object.

```
>>> len('Ayushi')
6
```

locals()- Returns a dictionary of the current local symbol table.

```
>>> locals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>, 'a': 2, 'b': 3}
```

open()- Opens a file.

```
>>> file=open('tabs.txt')
```

278. How will you convert a list into a string?
Ans: We will use the join() method for this.

```
>>> nums=[‘one’,‘two’,‘three’,‘four’,‘five’,‘six’,‘seven’]
>>> s=‘ ‘.join(nums)
>>> s
o/p= ‘one two three four five six seven’
```

279. How will you remove a duplicate element from a list?
Ans: We can turn it into a set to do that.

```
>>> list=[1,2,1,3,4,2]
>>> set(list)
{1, 2, 3, 4}
```

280. Can you explain the life cycle of a thread?
Ans:

python scripting interview questions

To create a thread, we create a class that we make override the run method of the thread class. Then, we instantiate it.
A thread that we just created is in the new state. When we make a call to start() on it, it forwards the threads for scheduling. These are in the ready state.
When execution begins, the thread is in the running state.
Calls to methods like sleep() and join() make a thread wait. Such a thread is in the waiting/blocked state.
When a thread is done waiting or executing, other waiting threads are sent for scheduling.
A running thread that is done executing terminates and is in the dead state.

281. Explain the //, %, and ** operators in Python?
Ans: The // operator performs floor division. It will return the integer part of the result on division.

```
>>> 7//2
3
```

Normal division would return 3.5 here.

Similarly, ** performs exponentiation. a**b returns the value of a raised to the power b.

```
>>> 2**10
1024
```

Finally, % is for modulus. This gives us the value left after the highest achievable division.

```
>>> 13%7
6

>>> 3.5%1.5
0.5
```

282. What are membership operators?
Ans: With the operators ‘in’ and ‘not in’, we can confirm if a value is a member in another.

```
>>> ‘me’ in ‘disappointment’
True

>>> ‘us’ not in ‘disappointment’
True
```

283. Explain identity operators in Python?
Ans: The operators ‘is’ and ‘is not’ tell us if two values have the same identity.

```
>>> 10 is ’10’
False

>>> True is not False
True
```

284. Finally, tell us about bitwise operators in Python?

Ans:

python interview questions for freshers

These operate on values bit by bit.

AND (&) This performs & on each bit pair.

```
>>> 0b110 & 0b010
2
```

OR (|) This performs | on each bit pair.

```
>>> 3|2
3
```

XOR (^) This performs an exclusive-OR operation on each bit pair.

```
>>> 3^2
1
```

Binary One’s Complement (~) This returns the one’s complement of a value.

```
>>> ~2
-3
```

Binary Left-Shift (<<) This shifts the bits to the left by the specified amount.

```
>>> 1<<2
4
```

Here, 001 was shifted to the left by two places to get 100, which is binary for 4.

Binary Right-Shift (>>)

```
>>> 4>>2
1
```

285. What data types does Python support?
Ans: Python provides us with five kinds of data types:

Numbers – Numbers use to hold numerical values.

```
>>> a=7.0
>>>
```

Strings – A string is a sequence of characters. We declare it using single or double quotes.

```
>>> title=“Ayushi’s Book”
```

Lists – A list is an ordered collection of values, and we declare it using square brackets.

```
>>> colors=[‘red’,‘green’,‘blue’]
>>> type(colors)
<class ‘list’>
```

Tuples – A tuple, like a list, is an ordered collection of values. The difference. However, is that a tuple is immutable. This means that we cannot change a value in it.

```
>>> name=(‘Ayushi’,‘Sharma’)
>>> name[0]=‘Avery’
Traceback (most recent call last):

File “<pyshell#129>”, line 1, in <module>

name[0]=’Avery’

TypeError: ‘tuple’ object does not support item assignment
```

Dictionary – A dictionary is a data structure that holds key-value pairs. We declare it using curly braces.

```
>>> squares={1:1,2:4,3:9,4:16,5:25}
>>> type(squares)
<class ‘dict’>

>>> type({})
<class ‘dict’>
```

We can also use a dictionary comprehension:

```
>>> squares={x:x**2 for x in range(1,6)}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

286. How would you convert a string into an int in Python?
Ans: If a string contains only numerical characters, you can convert it into an integer using the int() function.

```
>>> int('227')
227
```

Let's check the types:

```
>>> type('227')
<class 'str'>

>>> type(int('227'))
<class 'int'>
```

287. How do you take input in Python?
Ans: For taking input from the user, we have the function input(). In Python 2, we had another function raw_input().

The input() function takes, as an argument, the text to be displayed for the task:

```
>>> a=input('Enter a number')
Enter a number7
```

But if you have paid attention, you know that it takes input in the form of a string.

```
>>> type(a)
<class 'str'>
```

Multiplying this by 2 gives us this:

```
>>> a*=2
>>> a
'77'
```

So, what if we need to work on an integer instead?

We use the int() function for this.

```
>>> a=int(input('Enter a number'))
Enter a number7
```

Now when we multiply it by 2, we get this:

```
>>> a*=2
>>> a
14
```

288. What is a function?
Ans: When we want to execute a sequence of statements, we can give it a name. Let's define a function to take two numbers and return the greater number.

```
>>> def greater(a,b):
return a is a>b else b
```

```
>>> greater(3,3.5)
3.5
```

289. What is recursion?
Ans: When a function makes a call to itself, it is termed recursion. But then, in order for it to avoid forming an infinite loop, we must have a base condition.

Let’s take an example.

```
>>> def facto(n):
if n==1: return 1
return n*facto(n-1)
>>> facto(4)
24
```

290. What do you know about relational operators in Python?
Ans:

Top python interview questions with answers

Relational operators compare values.

Less than (<) If the value on the left is lesser, it returns True.

```
>>> ‘hi’<‘Hi’
False
```
Greater than (>) If the value on the left is greater, it returns True.

```
>>> 1.1+2.2>3.3
True
```

This is because of the flawed floating-point arithmetic in Python, due to hardware dependencies.

Less than or equal to (<=) If the value on the left is lesser than or equal to, it returns True.

```
>>> 3.0<=3
True
```

Greater than or equal to (>=) If the value on the left is greater than or equal to, it returns True.

```
>>> True>=False
True
```

Equal to (==) If the two values are equal, it returns True.

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

True

Not equal to (!=) If the two values are unequal, it returns True.

```
>>> True!=0.1
True

>>> False!=0.1
True
```

291. What are assignment operators in Python?
Ans:

python coding interview questions

We can combine all arithmetic operators with the assignment symbol.

```
>>> a=7
>>> a+=1
>>> a
8

>>> a-=1
>>> a
7

>>> a*=2
>>> a
14

>>> a/=2
>>> a
7.0

>>> a**=2
>>> a
49.0

>>> a//=3
>>> a
16.0

>>> a%=4
>>> a
0.0
```

292. Explain logical operators in Python.?
Ans: We have three logical operators- and, or, not.

```
>>> False and True
False
```

```
>>> 7<7 or True
True

>>> not 2==2
False
```

293. What does the function zip() do?
Ans: One of the less common functions with beginners, zip() returns an iterator of tuples.

```
>>> 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.

```
>>> list(zip((‘a’,‘b’,‘c’),(1,2,3)))
[(‘a’, 1), (‘b’, 2), (‘c’, 3)]
```

294. How can you declare multiple assignments in one statement?
Ans: There are two ways to do this:

First –

```
>>> a,b,c=3,4,5 #This assigns 3, 4, and 5 to a, b, and c respectively
```

Second –

```
>>> a=b=c=3 #This assigns 3 to a, b, and c
```

295. If you are ever stuck in an infinite loop, how will you break out of it?
Ans: For this, we press Ctrl+C. This interrupts the execution. Let’s create an infinite loop to demonstrate this.

```
>>> def counterfunc(n):
while(n==7):print(n)
>>> counterfunc(7)
7

7

7

7

7

7

7
```

```
7

7

7

7

7

7

7

7

7

7

Traceback (most recent call last):

File “<pyshell#332>”, line 1, in <module>

counterfunc(7)

File “<pyshell#331>”, line 2, in counterfunc

while(n==7):print(n)

KeyboardInterrupt
```

296. How is a .pyc file different from a .py file?
Ans: 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).

297. How many types of objects does Python support?
Ans: Immutable objects- Those which do not let us modify their contents. Examples of these will be tuples, booleans, strings, integers, floats, and complexes. Iterations on such objects are faster.

```
>>> tuple=(1,2,4)
>>> tuple
(1, 2, 4)

>>> 2+4j
(2+4j)
```

Mutable objects – Those that let you modify their contents. Examples of these are lists, sets, and dicts. Iterations on such objects are slower.

```
>>> [2,4,9]
[2, 4, 9]

>>> dict1={1:1,2:2}
>>> dict1
{1: 1, 2: 2}
```

While two equal immutable objects' reference variables share the same address, it is possible to create two mutable objects with the same content.

298. When is the else part of a try-except block executed?
Ans: In an if-else block, the else part is executed when the condition in the if-statement is False. But with a try-except block, the else part executes only if no exception is raised in the try part.

299. Explain join() and split() in Python?
Ans:

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

```
>>> ‘,’.join(‘12345’)
‘1,2,3,4,5’
```

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

```
>>> ‘1,2,3,4,5’.split(‘,’)
[‘1’, ‘2’, ‘3’, ‘4’, ‘5’]
```

300. Explain a few methods to implement Functionally Oriented Programming in Python?
Ans:

Sometimes, when we want to iterate over a list, a few methods come in handy.

a. filter()

Filter lets us filter in some values based on conditional logic.

```
>>> list(filter(lambda x:x>5,range(8)))
[6, 7]
```

b. map()

Map applies a function to every element in an iterable.

```
>>> list(map(lambda x:x**2,range(8)))
[0, 1, 4, 9, 16, 25, 36, 49]
```

c. reduce()

Reduce repeatedly reduces a sequence pair-wise until we reach a single value.

```
>>> from functools import reduce
>>> reduce(lambda x,y:x-y,[1,2,3,4,5])
-13
```

300. Explain a few methods to implement Functionally Oriented Programming in Python?
Ans:

Sometimes, when we want to iterate over a list, a few methods come in handy.

a. filter()

Filter lets us filter in some values based on conditional logic.

```
>>> list(filter(lambda x:x>5,range(8)))
[6, 7]
```

b. map()

Map applies a function to every element in an iterable.

```
>>> list(map(lambda x:x**2,range(8)))
[0, 1, 4, 9, 16, 25, 36, 49]
```

c. reduce()

Reduce repeatedly reduces a sequence pair-wise until we reach a single value.

```
>>> from functools import reduce
>>> reduce(lambda x,y:x-y,[1,2,3,4,5])
-13
```

301. Is del the same as remove()? What are they?
Ans:

del and remove() are methods on lists/ ways to eliminate elements.

```
>>> list=[3,4,5,6,7]
>>> del list[3]
>>> list
[3, 4, 5, 7]

>>> list.remove(5)
>>> list
[3, 4, 7]
```

While del lets us delete an element at a certain index, remove() lets us remove an element by its value.

302. Explain a few methods to implement Functionally Oriented Programming in Python?
Ans: Sometimes, when we want to iterate over a list, a few methods come in handy.

a. filter()

Filter lets us filter in some values based on conditional logic.

```
>>> list(filter(lambda x:x>5,range(8)))
[6, 7]
```

b. map()

Map applies a function to every element in an iterable.

```
>>> list(map(lambda x:x**2,range(8)))
[0, 1, 4, 9, 16, 25, 36, 49]
```

c. reduce()

Reduce repeatedly reduces a sequence pair-wise until we reach a single value.

```
>>> from functools import reduce
>>> reduce(lambda x,y:x-y,[1,2,3,4,5])
-13
```

304. How do you open a file for writing?
Ans: Let’s create a text file on our Desktop and call it tabs.txt. To open it to be able to write to it, use the following line of code-

```
>>> file=open(‘tabs.txt’,‘w’)
```

This opens the file in writing mode. You should close it once you’re done.

```
>>> file.close()
```

305. Difference between the append() and extend() methods of a list.
Ans: The 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.

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

This is how append() works:

```
>>> list1.append(4)
>>> list1
[1, 2, 3, 4]
```

And this is how extend() works:

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

306. What are the different file-processing modes with Python?
We have the following modes-

read-only – ‘r’
write-only – ‘w’

read-write – ‘rw’
append – ‘a’
We can open a text file with the option ‘t’. So to open a text file to read it, we can use the mode ‘rt’. Similarly, for binary files, we use ‘b’.

307. What does the map() function do?
Ans: map() executes the function we pass to it as the first argument; it does so on all elements of the iterable in the second argument. Let’s take an example, shall we?

```
>>> for i in map(lambda i:i**3, (2,3,7)):
print(i)
8
27
343
```

This gives us the cubes of the values 2, 3, and 7.

308. Is there a way to remove the last object from a list?
Yes, there is. Try running the following piece of code-

```
>>> list=[1,2,3,4,5
>>> list.pop(–1)
5

>>> list
[1, 2, 3, 4]
```

309. How will you convert an integer to a Unicode character?
Ans: This is simple. All we need is the chr(x) built-in function. See how.

```
>>> chr(52)
‘4’

>>> chr(49)
‘1’

>>> chr(67)
‘C’
```

310. So does recursion cause any trouble?

Ans: Sure does:

Needs more function calls.
Each function call stores a state variable to the program stack- consumes memory, can cause memory overflow.
Calling a function consumes time.

311. What good is recursion?
Ans: With recursion, we observe the following:

Need to put in less efforts.
Smaller code than that by loops.
Easier-to-understand code.

312. Can you remove the whitespaces from the string “aaa bbb ccc ddd eee”?
Ans: I can think of three ways to do this.

Using join-

```
>>> s=‘aaa bbb ccc ddd eee’
>>> s1=”.join(s.split())
>>> s1
‘aaabbbcccdddeee’
```

Using a list comprehension–

```
>>> s=‘aaa bbb ccc ddd eee’
>>> s1=str(”.join(([i for i in s if i!=‘ ‘])))
>>> s1
‘aaabbbcccdddeee’
```

Using replace()-

```
>>> s=‘aaa bbb ccc ddd eee’
>>> s1 = s.replace(‘ ‘,”)
>>> s1
‘aaabbbcccdddeee’
```

313. How do you get the current working directory using Python?

Ans: Working on software with Python, you may need to read and write files from various directories. To find out which directory we're presently working under, we can borrow the getcwd() method from the os module.

```
>>> import os
>>> os.getcwd()
'C:\\Users\\Raj\\AppData\\Local\\Programs\\Python\\Python37-32'
```

314. What are the file-related modules we have in Python?
Ans: We have the following libraries and modules that let us manipulate text and binary files on our file systems-

os
os.path
shutil

315. Explain the uses of the modules sqlite3, ctypes, pickle, traceback, and itertools.
sqlite3- Helps with handling databases of type SQLite
ctypes- Lets create and manipulate C data types in Python
pickle- Lets put any data structure to external files
traceback- Allows extraction, formatting, and printing of stack traces
itertools– Supports working with permutations, combinations, and other useful iterables.

316. How will you print the contents of a file?

```
>>> try:
with open('tabs.txt','r') as f:
print(f.read())
except IOError:
print("File not found")
```

317. What is Virtualenv in Python?
Ans: virtualenv is a tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need. It can be used standalone, in place of Pipenv.

Install virtualenv via pip: $ pip install virtualenv.

318. What is the function of “self”?
Ans: Self is a variable that represents the instance of the object to itself. In most of the object oriented programming language, this is passed to the methods as a hidden parameters that is defined by an object. But, in python, it is declared and passed explicitly. It is the first argument that gets created in the instance of the class A and the parameters to the methods are passed automatically. It refers to separate instance of the variable for individual objects.

Let’s say you have a class ClassA which contains a method methodA defined as:

def methodA(self, arg1, arg2): #do something
and ObjectA is an instance of this class.

Now when ObjectA.methodA(arg1, arg2) is called, python internally converts it for you as:

ClassA.methodA(ObjectA, arg1, arg2)
The self variable refers to the object itself.

319. What does the Python nonlocal statement do (in Python 3.0 and later)?
Ans: In short, it lets you assign values to a variable in an outer (but non-global) scope.

The nonlocal statement causes the listed identifiers to refer to previously bound variables in the nearest enclosing scope excluding globals.

For example the counter generator can be rewritten to use this so that it looks more like the idioms of languages with closures.

```
def make_counter():
    count = 0
    def counter():
        nonlocal count
        count += 1
        return count
    return counter
```

320. What are the wheels and eggs? What is the difference?
Ans:

Wheel and Egg are both packaging formats that aim to support the use case of needing an install artifact that doesn’t require building or compilation, which can be costly in testing and production workflows.

The Egg format was introduced by setuptools in 2004, whereas the Wheel format was introduced by PEP 427 in 2012.

Wheel is currently considered the standard for built and binary packaging for

Python.

Here’s a breakdown of the important differences between Wheel and Egg.

Wheel has an official PEP. Egg did not.
Wheel is a distribution format, i.e a packaging format. 1 Egg was both a distribution format and a runtime installation format (if left zipped), and was designed to be importable.
Wheel archives do not include .pyc files. Therefore, when the distribution only contains Python files (i.e. no compiled extensions), and is compatible with Python 2 and 3, it’s possible for a wheel to be “universal”, similar to an sdist.
Wheel uses PEP376-compliant .dist-info directories. Egg used .egg-info.
Wheel has a richer file naming convention. A single wheel archive can indicate its compatibility with a number of Python language versions and implementations, ABIs, and system architectures.
Wheel is versioned. Every wheel file contains the version of the wheel specification and the implementation that packaged it.
Wheel is internally organized by sysconfig path type, therefore making it easier to convert to other formats.

321. What is webpack?
Ans: Webpack is a build tool that puts all of your assets, including Javascript, images, fonts, and CSS, in a dependency graph. Webpack lets you use require() in your source code to point to local files, like images, and decide how they’re processed in your final Javascript bundle, like replacing the path with a URL pointing to a CDN.

322. Name some benefits of using webpack
Ans: Webpack and static assets in a dependency graph offers many benefits. Here’s a few:

Dead asset elimination. This is killer, especially for CSS rules. You only build the images and CSS into your dist/ folder that your application actually needs.
Easier code splitting. For example, because you know that your file Homepage.js only requires specific CSS files, Webpack could easily build a homepage.css file to greatly reduce initial file size.
You control how assets are processed. If an image is below a certain size, you could base64 encode it directly into your Javascript for fewer HTTP requests. If a JSON file is too big, you can load it from a URL. You can require('./style.less') and it’s automaticaly parsed by Less into vanilla CSS.
Stable production deploys. You can’t accidentally deploy code with images missing, or outdated styles.
Webpack will slow you down at the start, but give you great speed benefits when used correctly. You get hot page reloading. True CSS management. CDN cache busting because Webpack automatically changes file names to hashes of the file contents, etc.
Webpack is the main build tool adopted by the React community.

323. Name some plugins you think are very important and helpful?
Ans:

CommonsChunkPlugin – creates a separate file (known as a chunk), consisting of common modules shared between multiple entry points.
DefinePlugin – allows you to create global constants which can be configured at compile time.
HtmlWebpackPlugin – simplifies creation of HTML files to serve your webpack bundles.
ExtractTextWebpackPlugin – Extract text from a bundle, or bundles, into a separate file.
CompressionWebpackPlugin – Prepare compressed versions of assets to serve them with Content-Encoding.

324. Webpack gives us a dependency graph. What does that mean?
Ans: Any time one file depends on another, webpack treats this as a dependency. This allows webpack to take non-code assets, such as images or web fonts, and also provide them as dependencies for your application.

Webpack lets you use require() on local “static assets”:

When webpack processes your application, it starts from a list of modules defined on the command line or in its config file. Starting from these entry points, webpack recursively builds a dependency graph that includes every module your application needs, then packages all of those modules into a small number of bundles – often, just one – to be loaded by the browser.

The require('logo.png') source code never actually gets executed in the browser (nor in Node.js). Webpack builds a new Javascript file, replacing require() calls with valid Javascript code, such as URLs. The bundled file is what’s executed by Node or the browser.

325. What are metaclasses in Python?
Ans: A metaclass is the class of a class. A class defines how an instance of the class (i.e. an object) behaves while a metaclass defines how a class behaves. A class is an instance of a metaclass. You can call it a ‘class factory’.

326. How to make a chain of function decorators?
Ans: How can I make two decorators in Python that would do the following?

```
@makebold
@makeitalic
def say():
   return "Hello"
```
which should return:

```
"<b><i>Hello</i></b>"
```
Answer:
Consider:

```
from functools import wraps

def makebold(fn):
    @wraps(fn)
    def wrapped(*args, **kwargs):
        return "<b>" + fn(*args, **kwargs) + "</b>"
    return wrapped

def makeitalic(fn):
    @wraps(fn)
    def wrapped(*args, **kwargs):
        return "<i>" + fn(*args, **kwargs) + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello world"

@makebold
@makeitalic
def log(s):
    return s

print hello()        # returns "<b><i>hello world</i></b>"
print hello.__name__ # with functools.wraps() this returns "hello"
print log('hello')   # returns "<b><i>hello</i></b>"
```

327. What is the difference between @staticmethod and @classmethod?
Ans: A staticmethod is a method that knows nothing about the class or instance it was called on. It just gets the arguments that were passed, no implicit first argument. It’s definition is immutable via inheritance.

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```
A classmethod, on the other hand, is a method that gets passed the class it was called on, or the class of the instance it was called on, as first argument. Its definition follows Sub class, not Parent class, via inheritance.

```
class C:
   @classmethod
```

```
   def f(cls, arg1, arg2, ...): ...
```
If your method accesses other variables/methods in your class then use @classmethod.

328. What's the difference between a Python module and a Python package?
Ans: Any Python file is a module, its name being the file's base name without the .py extension.

```
import my_module
```
A package is a collection of Python modules: while a module is a single Python file, a package is a directory of Python modules containing an additional init.py file, to distinguish a package from a directory that just happens to contain a bunch of Python scripts. Packages can be nested to any depth, provided that the corresponding directories contain their own init.py file.

Packages are modules too. They are just packaged up differently; they are formed by the combination of a directory plus init.py file. They are modules that can contain other modules.

```
from my_package.timing.danger.internets import function_of_love
```

329. What is GIL?
Ans: Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread. This happens very quickly so to the human eye it may seem like your threads are executing in parallel, but they are really just taking turns using the same CPU core. All this GIL passing adds overhead to execution.

330. Is it a good idea to use multi-thread to speed your Python code?
Ans: Python doesn't allow multi-threading in the truest sense of the word. It has a multi-threading package but if you want to multi-thread to speed your code up, then it's usually not a good idea to use it.

Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread. This happens very quickly so to the human eye it may seem like your threads are executing in parallel, but they are really just taking turns using the same CPU core. All this GIL passing adds overhead to execution.

331. How do I write a function with output parameters (call by reference)?
Ans: In Python arguments are passed by assignment. When you call a function with a parameter, a new reference is created that refers to the object passed in. This is separate from the reference that was used in the function call, so there's no way to update that reference and make it refer to a new object.

If you pass a mutable object into a method, the method gets a reference to that same object and you can mutate it to your heart’s delight, but if you rebind the reference in the method (like b = b + 1), the outer scope will know nothing about it, and after you’re done, the outer reference will still point at the original object.

So to achieve the desired effect your best choice is to return a tuple containing the multiple results:

```
def func2(a, b):
    a = 'new-value'        # a and b are local names
    b = b + 1              # assigned to new objects
    return a, b            # return new values

x, y = 'old-value', 99
x, y = func2(x, y)
print(x, y)
```

332. Whenever you exit Python, is all memory de-allocated?
Ans: 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.

333. What is the purpose of the single underscore “_” variable in Python?
Ans: has 4 main conventional uses in Python:

To hold the result of the last executed expression(/statement) in an interactive interpreter session. This precedent was set by the standard CPython interpreter, and other interpreters have followed suit
For translation lookup in i18n (see the gettext documentation for example), as in code like: raise forms.ValidationError(_("Please enter a correct username"))
As a general purpose “throwaway” variable name to indicate that part of a function result is being deliberately ignored (Conceptually, it is being discarded.), as in code like: label, has_label, _ = text.partition(':').
As part of a function definition (using either def or lambda), where the signature is fixed (e.g. by a callback or parent class API), but this particular function implementation doesn’t need all of the parameters, as in code like: callback = lambda _: True

334. How is set() implemented internally?
I’ve seen people say that set objects in python have O(1) membership-checking. How

are they implemented internally to allow this? What sort of data structure does it use? What other implications does that implementation have?

Ans:
Indeed, CPython’s sets are implemented as something like dictionaries with dummy values (the keys being the members of the set), with some optimization(s) that exploit this lack of values.

So basically a set uses a hashtable as its underlying data structure. This explains the O(1) membership checking, since looking up an item in a hashtable is an O(1) operation, on average.

Also, it worth to mention when people say sets have O(1) membership-checking, they are talking about the average case. In the worst case (when all hashed values collide) membership-checking is O(n).

335. What is MRO in Python? How does it work?
Ans: Method Resolution Order (MRO) it denotes the way a programming language resolves a method or attribute. Python supports classes inheriting from other classes. The class being inherited is called the Parent or Superclass, while the class that inherits is called the Child or Subclass.

In Python,** method resolution order** defines the order in which the base classes are searched when executing a method. First, the method or attribute is searched within a class and then it follows the order we specified while inheriting. This order is also called Linearization of a class and set of rules are called MRO (Method Resolution Order). While inheriting from another class, the interpreter needs a way to resolve the methods that are being called via an instance. Thus we need the method resolution order.

Python resolves method and attribute lookups using the C3 linearisation of the class and its parents. The C3 linearisation is neither depth-first nor breadth-first in complex multiple inheritance hierarchies.

336. What is the difference between old style and new style classes in Python?
Ans: Declaration-wise:

New-style classes inherit from object, or from another new-style class.

```
class NewStyleClass(object):
    pass

class AnotherNewStyleClass(NewStyleClass):
    pass
```

Old-style classes don’t.

```
class OldStyleClass():
    pass
```
Python 3 Note:

Python 3 doesn’t support old style classes, so either form noted above results in a new-style class.

Also, MRO (Method Resolution Order) changed:

Classic classes do a depth first search from left to right. Stop on first match. They do not have the mro attribute.
New-style classes MRO is more complicated to synthesize in a single English sentence. One of its properties is that a Base class is only searched for once all its Derived classes have been. They have the mro attribute which shows the search order.
Some other notes:

New style class objects cannot be raised unless derived from Exception.
Old style classes are still marginally faster for attribute lookup.

337. Why Python (CPython and others) uses the GIL?
Ans: In CPython, the global interpreter lock, or GIL, is a mutex that prevents multiple native threads from executing Python bytecodes at once. This lock is necessary mainly because CPython’s memory management is not thread-safe.

Python has a GIL as opposed to fine-grained locking for several reasons:

It is faster in the single-threaded case.
It is faster in the multi-threaded case for i/o bound programs.
It is faster in the multi-threaded case for cpu-bound programs that do their compute-intensive work in C libraries.
It makes C extensions easier to write: there will be no switch of Python threads except where you allow it to happen (i.e. between the Py_BEGIN_ALLOW_THREADS and Py_END_ALLOW_THREADS macros).
It makes wrapping C libraries easier. You don’t have to worry about thread-safety. If the library is not thread-safe, you simply keep the GIL locked while you call it

338. How are arguments passed by value or by reference in python?
Ans:

Pass by value: Copy of the actual object is passed. Changing the value of the copy of the object will not change the value of the original object.
Pass by reference: Reference to the actual object is passed. Changing the value of the new object will change the value of the original object.
In Python, arguments are passed by reference, i.e., reference to the actual object is passed.

```
def appendNumber(arr):
    arr.append(4)

arr = [1, 2, 3]

print(arr)  #Output: => [1, 2, 3]
appendNumber(arr)
print(arr)  #Output: => [1, 2, 3, 4]
```

339. What is a boolean in Python?
Ans: Boolean is one of the built-in data types in Python, it mainly contains two values, and they are true and false.

Python bool() is the method used to convert a value to a boolean value.

```
1
Syntax for bool() method: bool([a])
```

340. What is Python String format and Python String replace?
Ans: Python String Format: The String format() method in Python is mainly used to format the given string into an accurate output or result.

Syntax for String format() method:

```
1
template.format(p0, p1, ..., k0=v0, k1=v1, ...)
```

Python String Replace: This method is mainly used to return a copy of the string in which all the occurrence of the substring is replaced by another substring.

Syntax for String replace() method:

```
1
str.replace(old, new [, count])
```

341. Name some of the built-in modules in Python?
Ans: The built-in modules in Python are:

sys module
OS module

random module
collection module
JSON
Math module

342. How do we convert the string to lowercase?
Ans: lower() function is used to convert string to lowercase.

Example:

```
1
2
str = 'XYZ'
print(str.lower())
```

Output:

```
1
xyz
```

343. How to remove values from a Python array?
Ans: The elements can be removed from a Python array using remove() or pop() function. The difference between pop() and remove() will be explained in the below example.

Example:

```
1
2
3
4
5
x = arr.array('d',  [ 1.0, 2.2, 3.4, 4.8, 5.2, 6.6, 7.3])
print(x.pop())
print(x.pop(3))
x.remove(1.0)
print(a)
```

Output:

```
1
2
3
7.3
4.8
array(‘d’, [2.2, 3.4, 5.2, 6.6])
```

344. What is Try Block?
A block which is preceded by the try keyword is known as a try block

Syntax:

```
1
2
3
try{
   //statements that may cause an exception
}
```

345. How can we access a module written in Python from C?
Ans: We can access the module written in Python from C by using the following method.

```
1
Module == PyImport_ImportModule("<modulename>");
```

346. Write a program to count the number of capital letters in a file?
Ans:

```
1
2
3
4
5
6
with open(SOME_LARGE_FILE) as countletter:
count = 0
text = countletter.read()
for character in text:
if character.isupper():
count += 1
```

347. Write a program to display the Fibonacci sequence in Python?
Ans:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
# Displaying Fibonacci sequence
n = 10
# first two terms
n0 = 0
n1 = 1
#Count
x = 0
# check if the number of terms is valid
if n <= 0:
   print("Enter positive integer")
elif n == 1:
   print("Numbers in Fibonacci sequence upto",n,":")
   print(n0)
else:
   print("Numbers in Fibonacci sequence upto",n,":")
   while x < n:
       print(n0,end=', ')
       nth = n0 + n1
       n0 = n1
       n1 = nth
       x += 1
Output:

1
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

348. Write a program in Python to produce Star triangle?
Ans: The code to produce star triangle is as follows:

```
1
2
3
4
```

```
def pyfun(r):
for a in range(r):
print(' '*(r-x-1)+'*'*(2*x+1))
pyfun(9)
```

Output:

```
1
2
3
4
5
6
7
8
9
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
```

349. Write a program to check whether the given number is prime or not?

Ans: The code to check prime number is as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
# program to check the number is prime or not
n1 = 409
# num1 = int(input("Enter any one number: "))
# prime number is greater than 1
if n1 > 1:
# check the following factors
for x is in range of(2,num1):
if (n1 % x) == 0:
print(n1,"is not a prime number")
print(x,"times",n1//x,"is",num)
break
```

```
else:
print(n1,"is a prime number")
# if input number is smaller than
# or equal to the value 1, then it is not prime number
else:
print(n1,"is not a prime number")
```
Output:

```
1
409 is a prime number
```

350. Write Python code to check the given sequence is a palindrome or not?
Ans:

```
1
2
3
4
5
6
7
8
9
10
11
# Python code to check a given sequence
#  is palindrome or not
my_string1 = 'MOM'
My_string1 = my_string1.casefold()
# reverse the given string
rev_string1  = reversed(my_string1)
# check whether the string is equal to the reverse of it or not
if list(my_string1) == list(rev_string1):
print("It is a palindrome")
else:
print("It is not a palindrome")
```
Output:

```
1
it is a palindrome
```

351. Write Python code to sort a numerical dataset?
Ans: The code to sort a numerical dataset is as follows:

```
1
```

```
2
3
4
list = [ "13", "16", "1", "5" , "8"]
list = [int(x) for x in the list]
list.sort()
print(list)
Output:

1
1, 5, 8, 13, 16
```

Top 50 Python Interview Questions

CREATE BY - ATUL KUMAR (LINKEDIN)

1. How will you improve the performance of a program in Python?
2. What are the benefits of using Python?
3. How will you specify source code encoding in a Python source file?
4. What is the use of PEP 8 in Python?
5. What is Pickling in Python?
6. How does memory management work in Python?
7. How will you perform Static Analysis on a Python Script?
8. What is the difference between a Tuple and List in Python?
9. What is a Python Decorator?
10. How are arguments passed in a Python method? By value or by reference?
11. What is the difference between List and Dictionary data types in Python?
12. What are the different built-in data types available in Python?
13. What is a Namespace in Python?
14. How will you concatenate multiple strings together in Python?
15. What is the use of Pass statement in Python?
16. What is the use of Slicing in Python?
17. What is the difference between Docstring in Python and Javadoc in Java?
18. How do you perform unit testing for Python code?
19. What is the difference between an Iterator and Iterable in Python?
20. What is the use of Generator in Python?

21. What is the significance of functions that start and end with _ symbol in Python?

22. What is the difference between xrange and range in Python?

23. What is lambda expression in Python?

24. How will you copy an object in Python?

25. What are the main benefits of using Python?

26. What is a metaclass in Python?

27. What is the use of frozenset in Python?

28. What is Python Flask?

29. What is None in Python?

30. What is the use of zip() function in Python?

31. What is the use of // operator in Python?

32. What is a Module in Python?

33. How can we create a dictionary with ordered set of keys in Python?

34. Python is an Object Oriented programming language or a functional programming language?

35. How can we retrieve data from a MySQL database in a Python script?

36. What is the difference between append() and extend() functions of a list in Python?

37. How will you handle an error condition in Python code?

38. What is the difference between split() and slicing in Python?

39. How will you check in Python, if a class is subclass of another class?

40. How will you debug a piece of code in Python?

41. How do you profile a Python script?

42. What is the difference between ‘is’ and ‘==’ in Python?

43. How will you share variables across modules in Python?

44. How can we do Functional programming in Python?

45. What is the improvement in enumerate() function of Python?

46. How will you execute a Python script in Unix?

47. What are the popular Python libraries used in Data analysis?

48. What is the output of following code in Python?

49. What is the output of following code in Python?

50. If you have data with name of customers and their location, which data type will you use to store it in Python?

ACKNOWLEDGMENTS

We thank our readers who constantly send feedback and reviews to motivate us in creating these useful books with the latest information!

INTRODUCTION

This book contains basic to expert level Python interview questions that an interviewer asks. Each question is accompanied with an answer so that you can prepare for job interview in short time.

We have compiled this list after attending dozens of technical interviews in top-notch companies like- Google, Facebook, Netflix, Amazon etc.

Often, these questions and concepts are used in our daily programming work. But these are most helpful when an Interviewer is trying to test your deep knowledge of Python.

The difficulty rating on these Questions varies from a Fresher level software programmer to a Senior software programmer.

Once you go through them in the first pass, mark the questions that you could not answer by yourself. Then, in second pass go through only the difficult questions.

After going through this book 2-3 times, you will be well prepared to face a technical interview on Python for an experienced programmer.

Python Interview Questions

1. How will you improve the performance of a program in Python?

There are many ways to improve the performance of a Python program. Some of these are as follows:

i. **Data Structure**: We have to select the right data structure for our purpose in a Python program.

ii. **Standard Library**: Wherever possible, we should use methods from standard library. Methods implemented in standard library have much better performance than user implementation.

iii. **Abstraction**: At times, a lot of abstraction and indirection can cause slow performance of a program. We should remove the redundant abstraction in code.

iv. **Algorithm**: Use of right algorithm can make a big difference in a program. We have to find and select the suitable algorithm to solve our problem with high performance.

2. What are the benefits of using Python?

Python is strong that even Google uses it. Some of the benefits of using Python are as follows:

i. **Efficient**: Python is very efficient in memory management. For a large data set like Big Data, it is much easier to program in Python.

ii. **Faster**: Though Python code is interpreted, still Python has very fast performance.

iii. **Wide usage**: Python is widely used among different organizations for different projects. Due to this wide usage, there are thousands of add-ons available for use with Python.

iv. **Easy to learn**: Python is quite easy to learn. This is the biggest benefit of using Python. Complex tasks can be very easily implemented in Python.

3. How will you specify source code encoding in a Python source file?

By default, every source code file in Python is in UTF-8 encoding. But we can also specify our own encoding for source files. This can be done by adding following line after #! line in the source file.

-*- coding: encoding -*-

In the above line we can replace encoding with the encoding that we want to use.

4. What is the use of PEP 8 in Python?

PEP 8 is a style guide for Python code. This document provides the coding conventions for writing code in Python. Coding conventions are about indentation, formatting, tabs, maximum line length, imports organization, line spacing etc. We use PEP 8 to bring consistency in our code. We consistency it is easier for other developers to read the code.

5. What is Pickling in Python?

Pickling is a process by which a Python object hierarchy can be converted into a byte stream. The reverse operation of Pickling is Unpickling.

Python has a module named pickle. This module has the implementation of a powerful algorithm for serialization and de-serialization of Python object structure.

Some people also call Pickling as Serialization or Marshalling.

With Serialization we can transfer Python objects over the network. It is also used in persisting the state of a Python object. We can write it to a file or a database.

6. How does memory management work in Python?

There is a private heap space in Python that contains all the Python objects and data structures. In CPython there is a memory manager responsible for managing the heap space.

There are different components in Python memory manager that handle segmentation, sharing, caching, memory pre-allocation etc.

Python memory manager also takes care of garbage collection by using Reference counting algorithm.

7. How will you perform Static Analysis on a Python Script?

We can use Static Analysis tool called PyChecker for this purpose. PyChecker can detect errors in Python code.

PyChecker also gives warnings for any style issues.

Some other tools to find bugs in Python code are pylint and pyflakes.

8. What is the difference between a Tuple and List in Python?

In Python, Tuple and List are built-in data structures.

Some of the differences between Tuple and List are as follows:

I. **Syntax**: A Tuple is enclosed in parentheses:
E.g. myTuple = (10, 20, “apple”);
A List is enclosed in brackets:
E.g. myList = [10, 20, 30];

II. **Mutable**: Tuple is an immutable data structure. Whereas, a List is a mutable data structure.

III. **Size**: A Tuple takes much lesser space than a List in Python.

IV. **Performance**: Tuple is faster than a List in Python. So it gives us good performance.

V. **Use case**: Since Tuple is immutable, we can use it in cases like Dictionary creation. Whereas, a List is preferred in the use case where data can alter.

9. What is a Python Decorator?

A Python Decorator is a mechanism to wrap a Python function and modify its behavior by adding more functionality to it. We can use @ symbol to call a Python Decorator function.

10. How are arguments passed in a Python method? By value or by reference?

Every argument in a Python method is an Object. All the variables in Python have reference to an Object. Therefore arguments in Python method are passed by Reference.

Since some of the objects passed as reference are mutable, we can change those objects in a method. But for an Immutable object like String, any change done within a method is not reflected outside.

11. What is the difference between List and Dictionary data types in Python?

Main differences between List and Dictionary data types in Python are as follows:

I. **Syntax**: In a List we store objects in a sequence. In a Dictionary we store objects in key-value pairs.

II. **Reference**: In List we access objects by index number. It starts from 0 index. In a Dictionary we access objects by key specified at the time of Dictionary creation.

III. **Ordering**: In a List objects are stored in an ordered sequence. In a Dictionary objects are not stored in an ordered sequence.

IV. **Hashing**: In a Dictionary, keys have to be hashable. In a List there is no need for hashing.

12. What are the different built-in data types available in Python?

Some of the built-in data types available in Python are as follows:

Numeric types: These are the data types used to represent numbers in Python.

int: It is used for Integers

long: It is used for very large integers of non-limited length.

float: It is used for decimal numbers.

complex: This one is for representing complex numbers

Sequence types: These data types are used to represent sequence of characters or objects.

str: This is similar to String in Java. It can represent a sequence of characters.

bytes: This is a sequence of integers in the range of 0-255.

byte array: like bytes, but mutable (see below); only available in Python 3.x

list: This is a sequence of objects.

tuple: This is a sequence of immutable objects.

Sets: These are unordered collections.

set: This is a collection of unique objects.

frozen set: This is a collection of unique immutable objects.

Mappings: This is similar to a Map in Java.

dict: This is also called hashmap. It has key value pair to store information by using hashing.

13. What is a Namespace in Python?

A Namespace in Python is a mapping between a name and an object. It is currently implemented as Python dictionary.

E.g. the set of built-in exception names, the set of built-in names, local names in a function

At different moments in Python, different Namespaces are created. Each Namespace in Python can have a different lifetime.

For the list of built-in names, Namespace is created when Python interpreter starts.

When Python interpreter reads the definition of a module, it creates global namespace for that module.

When Python interpreter calls a function, it creates local namespace for that function.

14. How will you concatenate multiple strings together in Python?

We can use following ways to concatenate multiple string together in Python:

I. **use + operator:**

E.g.
```
>>> fname="John"
>>> lname="Ray"
>>> print fname+lname
JohnRay
```

II. **use join function:**

E.g.
```
>>> ''.join(['John','Ray'])
'JohnRay'
```

15. What is the use of Pass statement in Python?

The use of Pass statement is to do nothing. It is just a placeholder for a statement that is required for syntax purpose. It does not execute any code or command.

Some of the use cases for pass statement are as follows:

I. **Syntax purpose:**

>>> while True:

... pass # Wait till user input is received

II. **Minimal Class:** It can be used for creating minimal classes:

>>> class MyMinimalClass:

... pass

III. **Place-holder for TODO work:**

We can also use it as a placeholder for TODO work on a function or code that needs to be implemented at a later point of time.

>>> def initialization():

... pass # TODO

16. What is the use of Slicing in Python?

We can use Slicing in Python to get a substring from a String.

The syntax of Slicing is very convenient to use.

E.g. In following example we are getting a substring out of the name John.

```
>>> name="John"
>>> name[1:3]
'oh'
```

In Slicing we can give two indices in the String to create a Substring. If we do not give first index, then it defaults to 0.

E.g.

```
>>> name="John"
>>> name[:2]
'Jo'
```

If we do not give second index, then it defaults to the size of the String.

```
>>> name="John"
>>> name[3:]
'n'
```

17. What is the difference between Docstring in Python and Javadoc in Java?

A Docstring in Python is a string used for adding comments or summarizing a piece of code in Python.

The main difference between Javadoc and Docstring is that docstring is available during runtime as well. Whereas, Javadoc is removed from the Bytecode and it is not present in .class file.

We can even use Docstring comments at run time as an interactive help manual.

In Python, we have to specify docstring as the first statement of a code object, just after the def or class statement.

The docstring for a code object can be accessed from the '__doc__' attribute of that object.

18. How do you perform unit testing for Python code?

We can use the unit testing modules **unittest** or **unittest2** to create and run unit tests for Python code.

We can even do automation of tests with these modules. Some of the main components of unittest are as follows:

I. **Test fixture**: We use test fixture to create preparation methods required to run a test. It can even perform post-test cleanup.

II. **Test case**: This is main unit test that we run on a piece of code. We can use **Testcase** base class to create new test cases.

III. **Test suite**: We can aggregate our unit test cases in a Test suite.

IV. **Test runner**: We use test runner to execute unit tests and produce reports of the test run.

19. What is the difference between an Iterator and Iterable in Python?

An Iterable is an object that can be iterated by an Iterator.

In Python, Iterator object provides _iter_() and next() methods.

In Python, an Iterable object has _iter_ function that returns an Iterator object.

When we work on a map or a for loop in Python, we can use next() method to get an Iterable item from the Iterator.

20. What is the use of Generator in Python?

We can use Generator to create Iterators in Python. A Generator is written like a regular function. It can make use yield statement to return data during the function call. In this way we can write complex logic that works as an Iterator.

A Generator is more compact than an Iterator due to the fact that _iter_() and next() functions are automatically created in a Generator.

Also within a Generator code, local variables and execution state are saved between multiple calls. Therefore, there is no need to add extra variables like self.index etc to keep track of iteration.

Generator also increases the readability of the code written in Python. It is a very simple implementation of an Iterator.

21. What is the significance of functions that start and end with _ symbol in Python?

Python provides many built-in functions that are surrounded by _ symbol at the start and end of the function name. As per Python documentation, double _ symbol is used for reserved names of functions.

These are also known as System-defined names.

Some of the important functions are:

Object._new_

Object._init_

Object._del_

22. What is the difference between xrange and range in Python?

In Python, we use range(0,10) to create a list in memory for 10 numbers.

Python provides another function xrange() that is similar to range() but xrange() returns a sequence object instead of list object. In xrange() all the values are not stored simultaneously in memory. It is a lazy loading based function.

But as per Python documentation, the benefit of xrange() over range() is very minimal in regular scenarios.

As of version 3.1, xrange is deprecated.

23. What is lambda expression in Python?

A lambda expression in Python is used for creating an anonymous function.

Wherever we need a function, we can also use a lambda expression.

We have to use lambda keyword for creating a lambda expression. Syntax of lambda function is as follows:

lambda argumentList: expression

E.g. lambda a,b: a+b

The above mentioned lambda expression takes two arguments and returns their sum.

We can use lambda expression to return a function.

A lambda expression can be used to pass a function as an argument in another function.

24. How will you copy an object in Python?

In Python we have two options to copy an object. It is similar to cloning an object in Java.

I. **Shallow Copy**: To create a shallow copy we call copy.copy(x). In a shallow copy, Python creates a new compound object based on the original object. And it tries to put references from the original object into copy object.

II. **Deep Copy**: To create a deep copy, we call copy.deepcopy(x). In a deep copy, Python creates a new object and recursively creates and inserts copies of the objects from original object into copy object. In a deep copy, we may face the issue of recursive loop due to infinite recursion.

25. What are the main benefits of using Python?

Some of the main benefits of using Python are as follows:

I. **Easy to learn**: Python is simple language. It is easy to learn for a new programmer.

II. **Large library**: There is a large library for utilities in Python that can be used for different kinds of applications.

III. **Readability**: Python has a variety of statements and expressions that are quite readable and very explicit in their use. It increases the readability of overall code.

IV. **Memory management**: In Python, memory management is built into the Interpreter. So a developer does not have to spend effort on managing memory among objects.

V. **Complex built-in Data types**: Python has built-in Complex data types like list, set, dict etc. These data types give very good performance as well as save time in coding new features.

26. What is a metaclass in Python?

A metaclass in Python is also known as class of a class. A class defines the behavior of an instance. A metaclass defines the behavior of a class.

One of the most common metaclass in Python is type. We can subclass type to create our own metaclass.

We can use metaclass as a class-factory to create different types of classes.

27. What is the use of frozenset in Python?

A frozenset is a collection of unique values in Python. In addition to all the properties of set, a frozenset is immutable and hashable.

Once we have set the values in a frozenset, we cannot change. So we cannot use and update methods from set on frozenset.

Being hashable, we can use the objects in frozenset as keys in a Dictionary.

28. What is Python Flask?

Python Flask is a micro-framework based on Python to develop a web application.

It is a very simple application framework that has many extensions to build an enterprise level application.

Flask does not provide a data abstraction layer or form validation by default. We can use external libraries on top of Flask to perform such tasks.

29. What is None in Python?

None is a reserved keyword used in Python for null objects. It is neither a null value nor a null pointer. It is an actual object in Python. But there is only one instance of None in a Python environment.

We can use None as a default argument in a function.

During comparison we have to use “is” operator instead of “==” for None.

30. What is the use of zip() function in Python?

In Python, we have a built-in function zip() that can be used to aggregate all the Iterable objects of an Iterator.

We can use it to aggregate Iterable objects from two iterators as well.

E.g.

```
list_1 = ['a', 'b', 'c']

list_2 = ['1', '2', '3']

for a, b in zip(list_1, list_2):

    print a, b
```

Output:

a1

b2

c3

By using zip() function we can divide our input data from different sources into fixed number of sets.

31. What is the use of // operator in Python?

Python provides // operator to perform floor division of a number by another. The result of // operator is a whole number (without decimal part) quotient that we get by dividing left number with right number.

It can also be used floordiv(a,b).

E.g.

10// 4 = 2

-10//4 = -3

32. What is a Module in Python?

A Module is a script written in Python with import statements, classes, functions etc. We can use a module in another Python script by importing it or by giving the complete namespace.

With Modules, we can divide the functionality of our application in smaller chunks that can be easily managed.

33. How can we create a dictionary with ordered set of keys in Python?

In a normal dictionary in Python, there is no order maintained between keys. To solve this problem, we can use OrderDict class in Python. This class is available for use since version 2.7.

It is similar to a dictionary in Python, but it maintains the insertion order of keys in the dictionary collection.

34. Python is an Object Oriented programming language or a functional programming language?

Python uses most of the Object Oriented programming concepts. But we can also do functional programming in Python. As per the opinion of experts, Python is a multi-paradigm programming language.

We can do functional, procedural, object-oriented and imperative programming with the help of Python.

35. How can we retrieve data from a MySQL database in a Python script?

To retrieve data from a database we have to make use of the module available for that database. For MySQL database, we import MySQLdb module in our Python script.

We have to first connect to a specific database by passing URL, username, password and the name of database.

Once we establish the connection, we can open a cursor with cursor() function. On an open cursor, we can run fetch() function to execute queries and retrieve data from the database tables.

36. What is the difference between append() and extend() functions of a list in Python?

In Python, we get a built-in sequence called list. We can call standard functions like append() and extend() on a list.

We call append() method to add an item to the end of a list.

We call extend() method to add another list to the end of a list.

In append() we have to add items one by one. But in extend() multiple items from another list can be added at the same time.

37. How will you handle an error condition in Python code?

We can implement exception handling to handle error conditions in Python code. If we are expecting an error condition that we cannot handle, we can raise an error with appropriate message.

E.g.

>>> if student_score < 0: raise ValueError(“Score can not be negative”)

If we do not want to stop the program, we can just catch the error condition, print a message and continue with our program.

E.g. In following code snippet we are catching the error and continuing with the default value of age.

```
#!/usr/bin/python
try:
   age=18+'duration'
except:
   print("duration has to be a number")
age=18
print(age)
```

38. What is the difference between split() and slicing in Python?

Both split() function and slicing work on a String object. By using split() function, we can get the list of words from a String.

E.g. 'a b c '.split() returns [‘a’, ‘b’, ‘c’]

Slicing is a way of getting substring from a String. It returns another String.

E.g. >>> 'a b c'[2:3] returns b

39. How will you check in Python, if a class is subclass of another class?

Python provides a useful method issubclass(a,b) to check whether class a is a subclass of b.

E.g. int is not a subclass of long
>>> issubclass(int,long)
False

bool is a subclass of int

>>> issubclass(bool,int)
True

40. How will you debug a piece of code in Python?

In Python, we can use the debugger pdb for debugging the code. To start debugging we have to enter following lines on the top of a Python script.

```
import pdb

pdb.set_trace()
```

After adding these lines, our code runs in debug mode. Now we can use commands like breakpoint, step through, step into etc for debugging.

41. How do you profile a Python script?

Python provides a profiler called cProfile that can be used for profiling Python code.

We can call it from our code as well as from the interpreter.

It gives use the number of function calls as well as the total time taken to run the script.

We can even write the profile results to a file instead of standard out.

42. What is the difference between ‘is’ and ‘==’ in Python?

We use ‘is’ to check an object against its identity.

We use ‘==’ to check equality of two objects.

E.g.

>>> lst = [10,20, 20]

>>> lst == lst[:]

True

>>> lst is lst[:]

False

43. How will you share variables across modules in Python?

We can create a common module with variables that we want to share.

This common module can be imported in all the modules in which we want to share the variables.

In this way, all the shared variables will be in one module and available for sharing with any new module as well.

44. How can we do Functional programming in Python?

In Functional Programming, we decompose a program into functions. These functions take input and after processing give an output. The function does not maintain any state.

Python provides built-in functions that can be used for Functional programming. Some of these functions are:

I. Map()
II. reduce()
III. filter()

Event iterators and generators can be used for Functional programming in Python.

45. What is the improvement in enumerate() function of Python?

In Python, enumerate() function is an improvement over regular iteration. The enumerate() function returns an iterator that gives (0, item[0]).

E.g.

```
>>> thelist=['a','b']
>>> for i,j in enumerate(thelist):
... print i,j
...
0 a
1 b
```

46. How will you execute a Python script in Unix?

To execute a Python script in Unix, we need to have Python executor in Unix environment.

In addition to that we have to add following line as the first line in a Python script file.

#!/usr/local/bin/python

This will tell Unix to use Python interpreter to execute the script.

47. What are the popular Python libraries used in Data analysis?

Some of the popular libraries of Python used for Data analysis are:

I. **Pandas**: Powerful Python Data Analysis Toolkit
II. **SciKit**: This is a machine learning library in Python.
III. **Seaborn**: This is a statistical data visualization library in Python.
IV. **SciPy**: This is an open source system for science, mathematics and engineering implemented in Python.

48. What is the output of following code in Python?

>>> thelist=['a','b']

>>> print thelist[3:]

Ans: The output of this code is following:

[]

Even though the list has only 2 elements, the call to thelist with index 3 does not give any index error.

49. What is the output of following code in Python?

>>>name=’John Smith’

>>>print name[:5] + name[5:]

Ans: Output of this will be

John Smith

This is an example of Slicing. Since we are slicing at the same index, the first name[:5] gives the substring name upto 5^{th} location excluding 5^{th} location. The name[5:] gives the rest of the substring of name from the 5^{th} location. So we get the full name as output.

50. If you have data with name of customers and their location, which data type will you use to store it in Python?

In Python, we can use dict data type to store key value pairs. In this example, customer name can be the key and their location can be the value in a dict data type.

Dictionary is an efficient way to store data that can be looked up based on a key.

PYTHON LIFE

Python | Django Interview Questions and Answers

1. What is Django?
Ans: Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source..

2. What does Django mean?
Ans: Django is named after Django Reinhardt, a gypsy jazz guitarist from the 1930s to early 1950s who is known as one of the best guitarists of all time.

3. Which architectural pattern does Django Follow?
Ans: Django follows Model-View Template (MVT) architectural pattern.

4. Explain the architecture of Django?
Ans: Django is based on MVT architecture. It contains the following layers:

Models: It describes the database schema and data structure.

Views: The view layer is a user interface. It controls what a user sees, the view retrieves data from appropriate models and execute any calculation made to the data and pass it to the template.

Templates: It determines how the user sees it. It describes how the data received from the views should be changed or formatted for display on the page.

Controller: Controller is the heart of the system. It handles requests and responses, setting up database connections and loading add-ons. It specifies the Django framework and URL parsing.

5. Which foundation manages Django web framework?
Ans: Django web framework is managed and maintained by an independent and non-profit organization named Django Software Foundation (DSF).

6. Is Django stable?
Ans: Yes, Django is quite stable. Many companies like Disqus, Instagram, Pinterest, and Mozilla have been using Django for many years.

7. What are the features available in Django web framework?
Ans: Features available in Django web framework are:

Admin Interface (CRUD)
Templating
Form handling
Internationalization
Session, user management, role-based permissions
Object-relational mapping (ORM)
Testing Framework
Fantastic Documentation

8. What are the advantages of using Django for web development?
Ans: It facilitates you to divide code modules into logical groups to make it flexible to change.
It provides auto-generated web admin to make website administration easy.
It provides pre-packaged API for common user tasks.
It provides template system to define HTML template for your web page to avoid code duplication.

It enables you to define what URL is for a given function.
It enables you to separate business logic from the HTML.

9. How to create a project in Django?
Ans: To start a project in Django, use the command $django-admin.py and then use the following command:

Project

init.py

manage.py

settings.py

urls.py

10. What are the inheritance styles in Django?
Ans: There are three possible inheritance styles in Django:

1. Abstract base classes: This style is used when you only want parent?s class to hold information that you don’t want to type out for each child model.

2. Multi-table Inheritance: This style is used if you are sub-classing an existing model and need each model to have its own database table.

3. Proxy models: This style is used, if you only want to modify the Python level behavior of the model, without changing the model’s fields.

11. How can you set up the database in Djanago?
Ans: To set up a database in Django, you can use the command edit mysite/setting.py , it is a normal python module with module level representing Django settings.

By default, Django uses SQLite database. It is easy for Django users because it doesn’t require any other type of installation. In the case of other database you have to the following keys in the DATABASE ‘default’ item to match your database connection settings.

Engines: you can change database by using ‘django.db.backends.sqlite3’ , ‘django.db.backeneds.mysql’, ‘django.db.backends.postgresql_psycopg2’, ‘django.db.backends.oracle’ and so on

Name: The name of your database. In the case if you are using SQLite as your database, in that case database will be a file on your computer, Name should be a full absolute path, including file name of that file.

Note: You have to add setting likes setting like Password, Host, User, etc. in your database, if you are not choosing SQLite as your database.

12. What does the Django templates contain?
Ans: A template is a simple text file. It can create any text-based format like XML, CSV, HTML, etc. A template contains variables that get replaced with values when the template is evaluated and tags (%tag%) that controls the logic of the template.

13. Is Django a content management system (CMS)?
Ans: No, Django is not a CMS. Instead, it is a Web framework and a programming tool that makes you able to build websites.

14. What is the use of session framework in Django?
Ans: The session framework facilitates you to store and retrieve arbitrary data on a per-site visitor basis. It stores data on the server side and abstracts the receiving and sending of cookies. Session can be implemented through a piece of middleware.

15. How can you set up static files in Django?
Ans: There are three main things required to set up static files in Django:

1. Set STATIC_ROOT in settings.py

2. run manage.py collectsatic

3. set up a Static Files entry on the PythonAnywhere web tab

16. How to use file based sessions?
Ans: You have to set the SESSION_ENGINE settings to "django.contrib.sessions.backends.file" to use file based session.

17. What is some typical usage of middlewares in Django?
Ans: Some usage of middlewares in Django is:

Session management,
Use authentication
Cross-site request forgery protection
Content Gzipping, etc.

18. What does of Django field class types do?
Ans: The Django field class types specify:

The database column type.
The default HTML widget to avail while rendering a form field.
The minimal validation requirements used in Django admin.
Automatic generated forms.

19. What is the command to start Django's built-in development server?
A. manage.py runserver
B. manage.py –start
C. manage.py run
D. manage.py startserver –dev
E. manage.py –run
Ans: A

20. Given a model named 'User' that contains a DateTime field named 'last_login', how do you query for users that have never logged in?
A. User.objects.filter(last_login=Null)
B. User.objects.filter(last_login__null=True)
C. User.objects.filter(last_login__isnull=False)
D. User.objects.filter(last_login__isnull=True)
E. User.objects.filter(last_login=Never)
Ans: D

21. What does the Django command `manage.py shell` do?
A. Starts a command line in whatever $SHELL your environment uses.
B. Starts a Django command prompt with your Python environment pre-loaded.
C. Starts a Python command prompt with your Django environment pre-loaded.
D. Loads a special Pythonic version of the Bash shell.
E. Loads a Python command prompt you can use to sync your database schema remotely.
Ans: C

22. Assuming you've imported the proper Django model file, how do you add a 'User' model to the Django admin?
A. admin.register(Users)
B. admin.site(self, User)
C. user.site.register(Admin)
D. users.site.register(Admin)
E. admin.site.register(User)
Ans: E

23. What is the Django command to start a new app named 'users' in an existing project?
A. manage.py –newapp users
B. manage.py newapp users
C. manage.py –startapp users
D. manage.py startapp users
E. manage.py start users
Ans: D

24. What does a urls.py file do in Django?
A. This file contains site deployment data such as server names and ports.
B. It contains a site map of Django-approved URLs.
C. It contains URL matching patterns and their corresponding view methods.
D. You run this file when you get obscure 404 Not Found errors in your server logs.
E. This file provides an up to date list of how-to URLs for learning Django more easily.
Ans: C

25. What is the command to run Django's development server on port 8080 on IP address 12.34.56.78?
A. manage.py –run 12.34.56.78 8080
B. manage.py –dev 12.34.56.78:8080
C. manage.py runserver 12.34.56.78:8000
D. manage.py run 12.34.56.78:8080
E. manage.py runserver 12.34.56.78:8080
Ans: E

26. Django is written using what programming language?
A. PHP
B. Ruby
C. Javascript
D. Java
E. Python
Ans: E

27. After you make a new 'app' in your existing Django project, how do you get Django to notice it?
A. No additional action is required, Django notices new apps automatically.
B. Run the `manage.py validate` command, and then start a new shell.
C. Run the `manage.py syncdb` command.
D. In settings.py, add the app to the PROJECT_APPS variable.

E. In settings.py, add the new app to the INSTALLED_APPS variable.
Ans: E
28. What is the purpose of settings.py?
A. To configure settings for the Django project
B. To configure settings for an app
C. To set the date and time on the server
D. To sync the database schema
Ans: A

29. How do you define a ‘name’ field in a Django model with a maximum length of 255 characters?
A. name = models.CharField(max_len=255)
B. model.CharField(max_length=255)
C. name = models.CharField(max_length=255)
D. model = CharField(max_length=255)
E. name = model.StringField(max_length=auto)
Ans: C

30. What is the definition of a good Django app?
A. A good Django app provides a small, specific piece of functionality that can be used in any number of Django projects.
B. A good Django app is a fully functioning website that has 100% test coverage.
C. A good Django app is highly customized and cannot be used in multiple projects.
Ans: A

31. What is the most easiest, fastest, and most stable deployment choice in most cases with Django?
A. FastCGI
B. mod_wsgi
C. SCGI
D. AJP
Ans: B

32. How do you exclude a specific field from a ModelForm?
A. Create a new Form, don’t use a ModelForm
B. Use the exclude parameter in the Meta class in your form
C. Set the field to hidden
D. You can not do this
Ans: B

33. Assuming you have a Django model named ‘User’, how do you define a foreign key field for this model in another model?
A. model = new ForeignKey(User)
B. user = models.IntegerKey(User)
C. user = models.ForeignKey(User)
D. models.ForeignKey(self, User)
Ans: C

34. What preferred method do you add to a Django model to get a better string representation of the model in the Django admin?
A. __unicode__
B. to_s(self)
C. __translate__
D. __utf_8__
Ans: A

36. What is Model Form used for?
A. To model an input form for a template
B. To specify rules for correct form when writing Django code

C. To define a form based on an existing model
Ans: C

37. What happens if MyObject.objects.get() is called with parameters that do not match an existing item in the database?
A. The Http404 exception is raised.
B. The DatabaseError exception is raised.
C. The MyObject.DoesNotExist exception is raised.
D. The object is created and returned.
Ans: C

38. A set of helpful applications to use within your Django projects is included in the official distribution. This module is called what?
A. django.extras
B. django.helpers
C. django.utilities
D. django.ponies
E. django.contrib
Ans: E

39. What is the correct syntax for including a class based view in a URLconf?
A. (r'^pattern/$', YourView.as_view()),
B. (r'^pattern/$', YourView.init()),
C. (r'^pattern/$', YourView),
D. (r'^pattern/$', YourView()),
Ans: A

40. What is the command to start a new Django project called 'myproject'?
A. django-admin.py startproject myproject
B. django-admin.py –start myproject
C. django.py startproject myproject
D. django.py –new myproject
E. django.py new myproject
Ans: A

41. How to make django timezone-aware?
A. In settings.py: USE_L10N=True
B. in views.py, import timezone
C. in views.py, import tz
D. in urls.py, import timezone
E. In settings.py: USE_TZ=True
Ans: E

42. In Django how would you retrieve all the 'User' records from a given database?
A. User.objects.all()
B. Users.objects.all()
C. User.all_records()
D. User.object.all()
E. User.objects
Ans: A

43. How can you define additional behavior and characteristics of a Django class?
A. def setUp():
B. class Meta:
C. class __init__:
D. def Meta():
E. def __init__():
Ans: B

44. What is the Django shortcut method to more easily render an html response?
A. render_to_html
B. render_to_response
C. response_render
D. render
Ans: B

45. What does the Django command `manage.py validate` do?
A. Checks for errors in your views.
B. Checks for errors in your templates.
C. Checks for errors in your controllers.
D. Checks for errors in your models.
E. Checks for errors in your settings.py file.
Ans: D

46. What is the correct way to include django's admin urls? from django.contrib import admin') from django.conf.urls import patterns, include, url urlpatterns = patterns(", _____)
A. url(r'^admin/', admin.as_view(), name='admin),
B. url(r'^admin/', include(admin)),
C. url(r'^admin/', include(admin.site.urls)),
D. url(r'^admin/', admin.urls),
E. admin.autodiscover()
Ans: C

47. Where is pre_save signal in Django
A. from django.db.models import pre_save
B. from django.db.models.signals import pre_save
C. There is no pre_save signal
D. from django.db.models.signal import pre_save
Ans: B

48. Given the Python data: mydata = [[0, 'Fred'], [1, 'Wilma']] How do you access the data in a Django template?
A. {% for d in mydata %}

{% d.1 %}

{% endfor %}
B. {% for d in mydata -%}

{{ d.1 }}

{% end -%}
C. {% for d in mydata %}

{{ d.1 }}

{% endfor %}
D. {{ for d in mydata }}

{{ d[1] }}

{{ endfor }}
E. {% mydata.each |d| %}

{{ d.2 }}

{% end %}

Ans: C

49. What is the purpose of the STATIC_ROOT setting?
A. Defines the URL prefix where static files will be served from .
B. Defines the location where all static files will be copied by the 'collectstatic' management command, to be served by the production webserver.
C. A project's static assets should be stored here to be served by the development server.
D. Defines the location for serving user uploaded files.
Ans: B

50. How to create a DateTimeField named created and filled in only on the creation with the current time?
A. created = models.CreationTimeField()
B. created = models.DateTimeField(default=datetime.datetime.now())
C. created = models.DateTimeField(auto_now_add=True, auto_now=True)
D. created = models.DateTimeField(auto_now=True)
E. created = models.DateTimeField(auto_now_add=True)
Ans: E

51. What is the difference between a project and an app in Django?
Ans: 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.

52. What is Django Admin Interface?
Ans: 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.

53. Explain Django's Request/Response Cycle.
Ans: 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.

54. What is a model in Django?
Ans: 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.

55. What are migrations in Django?
Ans: A migration in Django is a Python file that contains changes we make to our models so that they can be converted into a database schema in our DBMS. So, instead of manually making changes to our database schema by writing queries in our DBMS shell, we can just make changes to our models. Then, we can use Django to generate migrations from those model changes and run those migrations to make changes to our database schema.

56. What are views in Django?
Ans: 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.

57. What is the use of the include function in the urls.py file in Django?
Ans: 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.

58. Why is Django called a loosely coupled framework?
Ans: 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.

59.Mention the architecture of Django architecture?
Ans: Django architecture consists of

Models: It describes your database schema and your data structure
Views: It controls what a user sees, the view retrieves data from appropriate models and execute any calculation made to the data and pass it to the template
Templates: It determines how the user sees it. It describes how the data received from the views should be changed or formatted for display on the page
Controller: The Django framework and URL parsing

60. Why Django should be used for web-development?
Ans:

It allows you to divide code modules into logical groups to make it flexible to change
To ease the website administration, it provides auto-generated web admin
It provides pre-packaged API for common user tasks
It gives you template system to define HTML template for your web page to avoid code duplication
It enables you to define what URL be for a given function
It enables you to separate business logic from the HTML
Everything is in python

61. Explain how you can create a project in Django?
Ans: To start a project in Django, you use command $ django-admin.py and then use the command
Project
init.py
manage.py
settings.py
urls.py

62. Explain how you can set up the Database in Django?
Ans: You can use the command edit mysite/setting.py , it is a normal python module with module level representing Django settings.
Django uses SQLite by default; it is easy for Django users as such it won't require any other type of installation. In the case your database choice is different that

you have to the following keys in the DATABASE ‘default’ item to match your database connection settings

Engines: you can change database by using ‘django.db.backends.sqlite3’ , ‘django.db.backeneds.mysql’, ‘django.db.backends.postgresql_psycopg2’, ‘django.db.backends.oracle’ and so on
Name: The name of your database. In the case if you are using SQLite as your database, in that case database will be a file on your computer, Name should be a full absolute path, including file name of that file.
If you are not choosing SQLite as your database then setting like Password, Host, User, etc. must be added.

63. Give an example how you can write a VIEW in Django?
Ans: Views are Django functions that take a request and return a response. To write a view in Django we take a simple example of “Guru99_home” which uses the template Guru99_home.html and uses the date-time module to tell us what the time is whenever the page is refreshed. The file we required to edit is called view.py, and it will be inside mysite/myapp/

Copy the below code into it and save the file

```
       from datatime import datetime
      from django.shortcuts import render
     def home (request):
return render(request, ‘Guru99_home.html’, {‘right_now’: datetime.utcnow()})
```

Once you have determined the VIEW, you can uncomment this line in urls.py

```
# url ( r ‘^$’ , ‘mysite.myapp.views.home’ , name ‘Guru99’),
```

The last step will reload your web app so that the changes are noticed by the web server.

64. Explain how you can setup static files in Django?
Ans: There are three main things required to set up static files in Django

Set STATIC_ROOT in settings.py
run manage.py collectsatic
set up a Static Files entry on the PythonAnywhere web tab

65. Mention what does the Django templates consists of?
Ans: The template is a simple text file. It can create any text-based format like XML, CSV, HTML, etc. A template contains variables that get replaced with values when the template is evaluated and tags (% tag %) that controls the logic of the template.

66. Explain the use of session framework in Django?
Ans: In Django, the session framework enables you to store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the receiving and sending of cookies. Session can be implemented through a piece of middleware.

67. Explain how you can use file based sessions?
Ans: To use file based session you have to set the SESSION_ENGINE settings to “django.contrib.sessions.backends.file”

68. Explain the migration in Django and how you can do in SQL?
Ans: Migration in Django is to make changes to your models like deleting a model, adding a field, etc. into your database schema. There are several commands you use to interact with migrations.

Migrate
Makemigrations
Sqlmigrate
To do the migration in SQL, you have to print the SQL statement for resetting sequences for a given app name.
django-admin.py sqlsequencreset
Use this command to generate SQL that will fix cases where a sequence is out sync with its automatically incremented field data.

69. Mention what command line can be used to load data into Django?
Ans: To load data into Django you have to use the command line Django-admin.py loaddata. The command line will searches the data and loads the contents of the named fixtures into the database.

70. Explain what does django-admin.py makemessages command is used for?
Ans: This command line executes over the entire source tree of the current directory and abstracts all the strings marked for translation. It makes a message file in the locale directory.

71. List out the inheritance styles in Django?
Ans: In Django, there is three possible inheritance styles

Abstract base classes: This style is used when you only wants parent’s class to hold information that you don’t want to type out for each child model
Multi-table Inheritance: This style is used If you are sub-classing an existing model and need each model to have its own database table
Proxy models: You can use this model, If you only want to modify the Python level behavior of the model, without changing the model’s fields
72. Mention what does the Django field class types?
Ans: Field class types determines

The database column type
The default HTML widget to avail while rendering a form field
The minimal validation requirements used in Django admin and in automatically generated forms
73. What constitutes Django templates ?
Ans: Template can create formats like XML,HTML and CSV(which are text based formats). In general terms template is a simple text file. It is made up of variables that will later be replaced by values after the template is evaluated and has tags which will control template’s logic.

74. List some typical usage of middlewares in Django.
Ans: Some of the typical usage of middlewares in Django are: Session management, user authentication, cross-site request forgery protection, content Gzipping, etc.

75. How do you use views in Django?
Ans: Views will take request to return response. Let’s write a view in Django : “example” using template example.html , using the date-time module to tell us exact time of reloading the page. Let’s edit a file called view.py, and it will be inside randomsite/randomapp/
To do this save and copy following into a file:
Default

from datatime import datetime

```
from django.shortcuts import render

def home (request):

return render(request, ‘Guru99_home.html’, {‘right_now’: datetime.utcnow()})
```
Default

You have to determine the VIEW first, and then uncomment this line located in file urls.py

```
# url ( r ‘^$’ , ‘randomsite.randomapp.views.home’ , name ‘example’),
```

76. How do you make a Django app that is test driven and will display Fibonacci’s sequence?
This will reload the site making changes obvious.

Ans: Keep in mind that it should take an index number and output the sequence. Additionally, there should be a page that shows the most recent generated sequences.

Following is one of the solution for generating fibonacci series:
Default

```
def fib(n):

“Complexity: O(log(n))”

if n <= 0:

return 0

i = n – 1

(a, b) = (1, 0)

(c, d) = (0, 1)

while i > 0:

if i % 2:

(a, b) = (d * b + c * a,  d * (b + a) + c * b)

(c, d) = (c * c + d * d, d * (2 * c + d))

i = i / 2

return a + b
```
Default

Below is a model that would keep track of latest numbers:

```
from django.db import models

class Fibonacci(models.Model):

parameter = models.IntegerField(primary_key=True)
```

```
result = models.CharField(max_length=200)

time = models.DateTimeField()
```
DefaultFor view, you can simply use the following code:

```
from models import Fibonacci

def index(request):

result = None

if request.method==”POST”:

try:

n=int(request.POST.get(‘n’))

except:

return Http404

try:

result = Fibonacci.objects.get(pk=n)

result.time = datetime.now()

except DoesNotExist:

result = str(fib(n))

result = Fibonacci(n, result, datetime.now())

result.save()

return direct_to_template(request, ‘base.html’, {‘result’:result.result})
```
You could use models to get last ‘n’ entities.

77.What makes up Django architecture?
Ans: Django runs on MVC architecture. Following are the components that make up django architecture:

Models: Models elaborate back-end stuffs like database schema.(relationships)
Views: Views control what is to be shown to end-user.
Templates: Templates deal with formatting of view.
Controller: Takes entire control of Models.A MVC framework can be compared to a Cable TV with remote. A Television set is View(that interacts with end user), cable provider is model(that works in back-end) and Controller is remote that controls which channel to select and display it through view.
78. What does session framework do in django framework ?
Ans: Session framework in django will store data on server side and interact with end-users. Session is generally used with a middle-ware. It also helps in receiving and sending cookies for authentication of a user.

79.Can you create singleton object in python?If yes, how do you do it?
Ans: Yes, you can create singleton object. Here’s how you do it :

Default

12
3

4

5

class Singleton(object):def __new__(cls,*args,**kwargs):
if not hasattr(cls,’_inst’):

cls._inst = super(Singleton,cls).__new__(cls,*args,**kwargs)

return cls._inst

80. Mention caching strategies that you know in Django!
Ans: Few caching strategies that are available in Django are as follows:

File sytem caching
In-memory caching
Using Memcached
Database caching
81. What are inheritance type in Django?
Ans: There are 3 inheritance types in Django

Abstract base classes
Multi-table Inheritance
Proxy models

82. What do you think are limitation of Django Object relation mapping(ORM) ?
Ans: If the data is complex and consists of multiple joins using the SQL will be clearer.

If Performance is a concern for your, ORM aren’t your choice. Genrally. Object-relation-mapping are considered good option to construct an optimized query, SQL has an upper hand when compared to ORM.

83. How to Start Django project with ‘Hello World!’? Just say hello world in django project.
Ans: There are 7 steps ahead to start Django project.

Step 1: Create project in terminal/shell

f2finterview:~$ django-admin.py startproject sampleproject

Step 2: Create application

f2finterview:~$ cd sampleproject/

```
f2finterview:~/sampleproject$ python manage.py startapp sampleapp
```

Step 3: Make template directory and index.html file

```
f2finterview:~/sampleproject$ mkdir templates
f2finterview:~/sampleproject$ cd templates/
f2finterview:~/sampleproject/templates$ touch index.html
```

Step 4: Configure initial configuration in settings.py

Add PROJECT_PATH and PROJECT_NAME

```
import os
PROJECT_PATH = os.path.dirname(os.path.abspath(__file__))
PROJECT_NAME = 'sampleproject'
```

Add Template directories path

```
TEMPLATE_DIRS = (
os.path.join(PROJECT_PATH, 'templates'),
)
```

Add Your app to INSTALLED_APPS

```
INSTALLED_APPS = (
'sampleapp',
)
```

Step 5: Urls configuration in urls.py

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
url(r'^$', 'sampleproject.sampleapp.views.index', name='index'),
)
```

Step 6: Add index method in views.py

```
from django.shortcuts import render_to_response, get_object_or_404
from django.template import RequestContext
def index(request):
welcome_msg = 'Hello World'
return
render_to_response('index.html',locals(),context_instance=RequestContext(request))
```

Step7: Add welcome_msg in index.html

```
<!DOCTYPE html>

<html>

<body>

<h1>My First Heading For Say…</h1>

<p>{{welcome_msg}}</p>

</body>

</html>
```

84. How to login with email instead of username in Django?
Ans: Use bellow sample method to login with email or username.

```
from django.conf import settings
from django.contrib.auth import authenticate, login, REDIRECT_FIELD_NAME
from django.shortcuts import render_to_response
from django.contrib.sites.models import Site
from django.template import Context, RequestContext
from django.views.decorators.cache import never_cache
from django.views.decorators.csrf import csrf_protect
@csrf_protect
@never_cache
def
signin(request,redirect_field_name=REDIRECT_FIELD_NAME,authentication_form=LoginFor
m):
redirect_to = request.REQUEST.get(redirect_field_name, settings.LOGIN_REDIRECT_URL)
form = authentication_form()
current_site = Site.objects.get_current()
if request.method == “POST”:
pDict =request.POST.copy()
form = authentication_form(data=request.POST)
if form.is_valid():
username = form.cleaned_data[‘username’]
password = form.cleaned_data[‘password’]
try:
user = User.objects.get(email=username)
username = user.username
except User.DoesNotExist:
username = username
user = authenticate(username=username, password=password)
# Log the user in.
login(request, user)
return HttpResponseRedirect(redirect_to)
else:
form = authentication_form()
request.session.set_test_cookie()
if Site._meta.installed:
current_site = Site.objects.get_current()
else:
current_site = RequestSite(request)
return render_to_response(‘login.html’,locals(),
```

context_instance=RequestContext(request))

85. How Django processes a request?
Ans: When a user requests a page from your Django-powered site, this is the algorithm the system follows to determine which Python code to execute:
Django determines the root URLconf module to use. Ordinarily, this is the value of the ROOT_URLCONF setting, but if the incoming HttpRequest object has an attribute called urlconf (set by middleware request processing), its value will be used in place of the ROOT_URLCONF setting.
Django loads that Python module and looks for the variable urlpatterns. This should be a Python list, in the format returned by the function django.conf.urls.patterns()
Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL.

Once one of the regexes matches, Django imports and calls the given view, which is a simple Python function (or a class based view). The view gets passed an HttpRequest as its first argument and any values captured in the regex as remaining arguments.

If no regex matches, or if an exception is raised during any point in this process, Django invokes an appropriate error-handling view.

86. How to filter latest record by date in Django?
Ans: Messages(models.Model):
message_from = models.ForeignKey(User,related_name=”%(class)s_from”)
message_to = models.ForeignKey(User,related_name=”%(class)s_to”)
message=models.CharField(max_length=140,help_text=”Your message”)
created_on = models.DateTimeField(auto_now_add=True)
class Meta:
db_table = ‘messages’

Query:messages = Messages.objects.filter(message_to = user).order_by(‘-created_on’)[0]

Output:

message_from | message_to | message | created_on

———|———-|———-|———-

Stephen | Anto | Hi, How are you? | 2012-10-09 14:27:48

87.How to filter data from Django models using python datetime?
Ans: Assume Bellow model for storing messages with timelines
class Message(models.Model):
from = models.ForeignKey(User,related_name = “%(class)s_from”)
to = models.ForeignKey(User, related_name = “%(class)s_to”)
msg = models.CharField(max_length=255)
rating = models.IntegerField(blank=’True’,default=0)
created_on = models.DateTimeField(auto_now_add=True)
updated_on = models.DateTimeField(auto_now=True)
Filter messages with specified Date and Time
today = date.today().strftime(‘%Y-%m-%d’)

```
yesterday = date.today() – timedelta(days=1)
yesterday = yesterday.strftime(‘%Y-%m-%d’)

this_month = date.today().strftime(‘%m’)
last_month = date.today() – timedelta(days=32)
last_month = last_month.strftime(‘%m’)
this_year = date.today().strftime(‘%Y’)

last_year = date.today() – timedelta(days=367)
last_year = last_year.strftime(‘%Y’)

today_msgs = Message.objects.filter(created_on__gte=today).count()
yesterday_msgs = Message.objects.filter(created_on__gte=yesterday).count()
this_month_msgs =
Message.objects.filter(created_on__month=this_month,created_on__year=this_year).cou
nt()
last_month_msgs =
Message.objects.filter(created_on__month=last_month,created_on__year=this_year).cou
nt()
this_year_msgs = Message.objects.filter(created_on__year=this_year).count()
last_year_msgs = Message.objects.filter(created_on__year=last_year).count()
```

88. What does Django mean?
Ans: Django is named after Django Reinhardt, a gypsy jazz guitarist from the 1930s to early 1950s who is known as one of the best guitarists of all time.

89. Which architectural pattern does Django Follow?
Ans: Django follows Model-View Controller (MVC) architectural pattern.

90. Is Django a high level web framework or low level framework?
Ans: Django is a high level Python’s web framework which was designed for rapid development and clean realistic design.

91. How would you pronounce Django?
Ans: Django is pronounced JANG-oh. Here D is silent.

92. How does Django work?
Ans: Django can be broken into many components:

Models.py file: This file defines your data model by extending your single line of code into full database tables and add a pre-built administration section to manage content.

Urls.py file: It uses regular expression to capture URL patterns for processing.

Views.py file: It is the main part of Django. The actual processing happens in view.
When a visitor lands on Django page, first Django checks the URLs pattern you have created and uses information to retrieve the view. After that view processes the

request, querying your database if necessary, and passes the requested information to template.
After that the template renders the data in a layout you have created and displays the page.

93. Which foundation manages Django web framework?
Ans: Django web framework is managed and maintained by an independent and non-profit organization named Django Software Foundation (DSF).

94. Is Django stable?
Ans: Yes, Django is quite stable. Many companies like Disqus, Instagram, Pinterest, and Mozilla have been using Django for many years.

95. What are the features available in Django web framework?
Ans: Features available in Django web framework are:

Admin Interface (CRUD)
Templating
Form handling
Internationalization
Session, user management, role-based permissions
Object-relational mapping (ORM)
Testing Framework
Fantastic Documentation

96. What are the advantages of using Django for web development?
Ans:

It facilitates you to divide code modules into logical groups to make it flexible to change.
It provides auto-generated web admin to make website administration easy.
It provides pre-packaged API for common user tasks.
It provides template system to define HTML template for your web page to avoid code duplication.
It enables you to define what URL is for a given function.
It enables you to separate business logic from the HTML.
97. How to create a project in Django?
Ans: To start a project in Django, use the command $django-admin.py and then use the following command:
Project
init.py
manage.py
settings.py
urls.py

98. What are the inheritance styles in Django?
Ans: There are three possible inheritance styles in Django:

1) Abstract base classes: This style is used when you only want parent’s class to hold information that you don’t want to type out for each child model.

2) Multi-table Inheritance: This style is used if you are sub-classing an existing model and need each model to have its own database table.

3) Proxy models: This style is used, if you only want to modify the Python level behavior of the model, without changing the model's fields.

99. How can you set up the database in Djanago?
Ans: A: To set up a database in Django, you can use the command edit mysite/setting.py , it is a normal python module with module level representing Django settings.
By default, Django uses SQLite database. It is easy for Django users because it doesn't require any other type of installation. In the case of other database you have to the following keys in the DATABASE 'default' item to match your database connection settings.

Engines: you can change database by using 'django.db.backends.sqlite3' , 'django.db.backeneds.mysql', 'django.db.backends.postgresql_psycopg2', 'django.db.backends.oracle' and so on

Name: The name of your database. In the case if you are using SQLite as your database, in that case database will be a file on your computer, Name should be a full absolute path, including file name of that file.
Note: You have to add setting likes setting like Password, Host, User, etc. in your database, if you are not choosing SQLite as your database.

100. What does the Django templates contain?
Ans: A template is a simple text file. It can create any text-based format like XML, CSV, HTML, etc. A template contains variables that get replaced with values when the template is evaluated and tags (%tag%) that controls the logic of the template.

101. Is Django a content management system (CMS)?
Ans: No, Django is not a CMS. Instead, it is a Web framework and a programming tool that makes you able to build websites.

102.What is the use of session framework in Django?
Ans: The session framework facilitates you to store and retrieve arbitrary data on a per-site visitor basis. It stores data on the server side and abstracts the receiving and sending of cookies. Session can be implemented through a piece of middleware.

103. Explain Django Admin Interface?
Ans. The Django Admin interface is predefined interface made to fulfill the need of web developers as they won't need to make another admin panel which is time-consuming and expensive.

Django Admin is application imported from django.contrib packages. It is operated by the organization itself and thus doesn't need the extensive frontend.

Admin interface of Django has its own user authentication and most of the general features. It also offers lots of advanced features like authorization access, managing different models, CMS (Content Management System), etc.

104. Explain Django.
Ans. Django is web application framework which is a free and open source. Django is written in Python. It is a server-side web framework that provides rapid development of secure and maintainable websites.

105. What does Django mean?
Ans. Django Reinhardt, was a gypsy jazz guitarist from the 1930s to early 1950s who

is known as one of the best guitarists of all time. The name was given Django after this person.

106. Which architectural pattern does Django follow?
Ans. Django follows Model-View-Template (MVT) architectural pattern.

The graph below shows the MVT based control flow.

107. Explain Django architecture.
Ans. Django follows MVT (Model View Template) pattern. It is slightly different from MVC.

Model: It is the data access layer. It contains everything about the data, i.e., how to access it, how to validate it, its behaviors and the relationships between the data.

Let’s see an example. We are creating Employee model who has two fields first_name and last_name.

```
from django.db import models

class Employee(models.Model):

first_name = models.CharField(max_length=30)

last_name = models.CharField(max_length=30)
```

View: It is the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template. It is like a bridge between the model and the template.

```
import datetime

# Create your views here.

from django.http import HttpResponse

def index(request):

now = datetime.datetime.now()

html = “&lt;html&gt;&lt;body><h3>Now time is %s.</h3>&lt;/body&gt;&lt;/html>” % now

return HttpResponse(html)    # rendering the template in HttpResponse
```

Template: It is a presentation layer. This layer contains presentation-related decisions, i.e., how something should be displayed on a Web page or other type of document.

For the configuration of the templates, we have to provide some entries in settings.py file.

```
TEMPLATES = [

{
```

```
‘BACKEND’: ‘django.template.backends.django.DjangoTemplates’,

‘DIRS’: [os.path.join(BASE_DIR,’templates’)],

‘APP_DIRS’: True,

‘OPTIONS’: {

‘context_processors’: [

‘django.template.context_processors.debug’,

‘django.template.context_processors.request’,

‘django.contrib.auth.context_processors.auth’,

‘django.contrib.messages.context_processors.messages’,

],

},

},

]
```

108. Explain the working of Django?
Ans. Django can be broken into following components:

Models.py : Models.py file will define your data model by extending your single line of code into full database tables and add a pre-built administration section to manage content.

Urls.py : Uses a regular expression to capture URL patterns for processing.

Views.py : This is main part of Django. The presentation logic is defined in this.

When a visitor visits Django page, first Django checks the URLs pattern you have created and use this information to retrieve the view. Then it is the responsibility of view to processes the request, querying your database if necessary, and passes the requested information to a template.

Then template renders the data in a layout you have created and displayed the page.

109. Name the foundation that manages the Django web framework?
Ans. Django is managed and maintained by an independent and non-profit organization named . Goal of foundation is to promote, support, and advance the Django Web framework.

110. Comment about Django’s stability?
Ans. Django is quite stable web development framework.There are many companies like Disqus, Instagram, Pinterest, and Mozilla that are using Django for many years.

111. Specify the features available in Django web framework?
Ans. Features available in Django web framework are:

Admin Interface (CRUD)

Templating

Form handling

Internationalization

A Session, user management, role-based permissions

Object-relational mapping (ORM)

Testing Framework

Fantastic Documentation

112. Explain the advantages of Django?
Ans. Advantages of Django:

Web development framework Django is a Python’s framework which is easy to learn.

It is clear and readable.

It is versatile.

It is fast to write.

No loopholes in design.

It is secure.

It is scalable.

It is versatile.

113. What are the disadvantages of Django?
Ans. Following is the list of disadvantages of Django:

Django’ modules are bulky.

It is completely based on Django ORM.

Components are deployed together.

You must know the full system to work with it.

114. What are the inheritance styles in Django?
Ans. There are three possible inheritance styles in Django:

1) Abstract base class: In this only parent’s class to hold information that you don’t want to type out for each child model then this style is used.

2) Multi-table Inheritance: This inheritance style is used if you are sub-classing an existing model and need each model to have its database table.

3) Proxy models: Proxy models is used, if you only want to modify the Python level behavior of the model, without changing the model’s fields.

115. Is Django a CMS i.e. content management system?
Ans. No, Django is not a CMS. But, it is a Web framework and a programming tool that makes you able to build websites.

116. Can you set up static files in Django? How?
Ans. Yes we can. We need to set three main things to set up static files in Django:

1) Set STATIC_ROOT in settings.py

2) run manage.py collect static

3) Static Files entry on the PythonAnywhere web tab

117. What is some typical usage of middlewares in Django?
Ans. Some usage of middlewares in Django is:

Session management,

Use authentication

Cross-site request forgery protection

Content Gzipping

118. What is the use of Django field class type?
Ans. Django field class type specifies:

The database column type.

Default HTML widget used to avail while rendering a form field.

The minimal validation requirements used in Django admin.

Automatic generated forms.

119. Explain the use of Django-admin.py and manage.py?
Ans. admin.py: This is Django’s command line utility for administrative tasks.

Manage.py: This file is created automatically in each Django project. It is a thin wrapper around the Django-admin.py. It has the following usage:

It puts your project’s package on sys.path.

DJANGO_SETTING_MODULE is the environment variable used to points to your project’s setting.py file.

120. What are the signals in Django?
Ans. Signals in Django are pieces of code which contain information about what is happening. A dispatcher is used to sending the signals and listen for those signals.

121. What are the two important parameters in signals?
Ans. Two important parameters in signals are:

Receiver: It specifies the callback function which connected to the signal.

Sender: It specifies a particular sender from where a signal is received.

122. How to handle URLs in Django?
In order to handle URL in Django, django.urls module is used by the Django framework.

Given below is the urls.py file of the project, lets see how it looks:

```
// urls.py

from django.contrib import admin

from django.urls import path

urlpatterns = [

path(‘admin/’, admin.site.urls),

]
```

We can see that, Django has already mentioned a URL here for the admin. Function path takes the first argument as a route of string or regex type.

Argument view is a view function which is used to return a response (template) to the user.

Module django.urls contains various functions, path(route,view,kwargs,name) is one of those which is used to map the URL and call the specified view.

123. What is Django Session?
Ans. In Django session is a mechanism to store information on the server side during the interaction with the web application. Session stores in the database and also allows file-based and cache based sessions, by default,

124 Explainthe role of Cookie in Django?
Ans. Cookie is nothing but a small piece of information which is stored in the client browser. Cookies are used to store user’s data in a file permanently (or for

the specified time). There is an expiry date and time for each cookie and removes automatically when gets expire. There are built-in methods to set and fetch cookie provided by Django.

set_cookie() method is used to set a cookie and get() method is used to get the cookie.

request.COOKIES[‘key’] is an array which canbe used to get cookie values.

```
from django.shortcuts import render

from django.http import HttpResponse

def setcookie(request):

response = HttpResponse(“Cookie Set”)

response.set_cookie(‘java-tutorial’, ‘javatpoint.com’)

return response

def getcookie(request):

tutorial  = request.COOKIES[‘java-tutorial’]

return HttpResponse(“java tutorials @: “+  tutorial);
```

125. 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

126. How do you check for the version of Django installed on your system?
Ans:

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:

```
1
2
importdjango
print(django.get_version())
```

127. What is the usage of middlewares in Django?
Ans: Middlewares are used go to modify the request i.e. HttpRequest object which is sent to the view, to modify the HttpResponse object returned from the view and to perform an operation before the view executes.

128. What are the roles of receiver and sender in signals?
The receiver is the callback function which will be connected to a signal
The sender specifies a particular sender to receive signals from
129. What does Django templates contain ?
Ans: Django templates contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

130. How to create super user in django ?
To create a super user,,

Create project using the django-admin startproject command.
Move into the project location and run python manage.py makemigrations && python manage.py migrate && python manage.py createsuperuser
131. How to create simple application in django ?
To create a simple application use the command django-admin startproject followed by the application’s name.

132. What is ORM ? Advantages of ORM ?
ORM (Object-relational mapping) is a programming technique for converting data between incompatible type systems using object-oriented programming languages.

Advantages include:

Concurrency support
Cache management
133. How to create a model in django ?
Add the model object in the models.py file, updated settings for the newly created app by adding it to the INSTALLED_APPS section in settings.py, make migrations, and verify the database schema.

134. What is migration in django ?
Migrations are a way of propagating changes made in the model into the database schema (adding a field, deleting a model, etc.)

135. How to do migrations in django ?
To do migrations , create or update a model and in the app directory, run the command ./manage.py makemigrations <app name> && ./manage.py migrate <app name>

136. How to clear cache in django ?
To clear cache, run the clear() method from django.core.cache in a python script.

137. What is Rest API ?
A REST API is an application program interface that uses HTTP requests to GET, PUT, POST and DELETE data.

138. How to Create APIs in Django ?
Create a project directory, create python virtual environment, and activate it, install Django and djangorestframework using the pip install command. In the same

project directory, create project using the command django-admin.py startproject api. Start the app. Add the rest_framework and the Djano app to INSTALLED_APPS to settings. Open the api/urls.py and add urls for the Django app. We can then create models and make migrations, create serializers, and finally wiring up the views.

139. What is DRF of Django Rest Frame work ?
Django Rest Framework (DRF) is a powerful module for building web APIs. It’s very easy to build model-backed APIs that have authentication policies and are browsable.

140. How to Fetch data from apis using Django ?
We use the Fetch API and SessionAuthentication by adding it to the settings.py file on the server and on the client, include the getCookie method. Finally, use the fetch method to call your endpoint.

141. How to update the data from apis ?
We update data by sending PUT requests. Add a new path in the data model urlpatterns from which the update will be sent to. We then add an update method to the serializer that will do the update.

142. What is Authentication ?
Authentication is the process or action of verifying the identity of a user or process.

143. Types of Authentication in REST API ?
Token based authentication and Session based authentication.

144. What is token based authentication system ?
A token based authentication system is a security system that authenticates the users who attempt to log in to a server, a network, or some other secure system, using a security token provided by the server

145. Can i use django apis in mobile application development ?
Yes
146. Explain Mixins in Django ?
A mixin is a special kind of multiple inheritance. There are two main situations where mixins are used: to provide a lot of optional features for a class and to use one particular feature in a lot of different classes

147. Different types caching strategies in django ?
Different types of caching strategies include Filesystem caching, in-memory caching, using memcached and database caching.

148. How a request is process in Django ?
When the user makes a request of your application, a WSGI handler is instantiated, which:

imports your settings.py file and Django’s exception classes.
loads all the middleware classes it finds in the MIDDLEWARE_CLASSES or MIDDLEWARES(depending on Django version) tuple located in settings.py
builds four lists of methods which handle processing of request, view, response, and exception.
loops through the request methods, running them in order
resolves the requested URL
loops through each of the view processing methods
calls the view function (usually rendering a template)
processes any exception methods
loops through each of the response methods, (from the inside out, reverse order from request middlewares)

finally builds a return value and calls the callback function to the web server
149. When to use iterator in Django ORM ?
The iterator is used when processing results that take up a large amount of available memory (lots of small objects or fewer large objects).

150. What are signals in Django ?

Signals allow certain senders to notify a set of receivers that some action has taken place. They're especially useful when many pieces of code may be interested in the same events.

151. How to implement social login authentication in Django ?
Run the development server to make sure all is in order. The install python-social-auth using the pip install command. Update settings.py to include/register the library in the project Update the database by making migrations. Update the Project's urlpatterns in urls.py to include the main auth URLs. Create a new app https://apps.twitter.com/app/new and make sure to use the callback url http://127.0.0.1:8000/complete/twitter. In the project directory, add a config.py file and grab the consumer key and consumer secret and add them to the config file. Finally add urls to the config file to specify the login and redirect urls. Do a sanity check and add friendly views.

152. Where to store static files in django ?
Static files are stored in the folder called static in the Django app.

 The Questions
General
153. Explain the use of session framework in Django? ↑
In Django, the session framework enables you to store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the receiving and sending of cookies. Session can be implemented through a piece of middleware.

154. List out the inheritance styles in Django? ↑
In Django, there is three possible inheritance styles

Abstract base classes: This style is used when you only wants parent's class to hold information that you don't want to type out for each child model
Multi-table Inheritance: This style is used If you are sub-classing an existing model and need each model to have its own database table
Proxy models: You can use this model, If you only want to modify the Python level behavior of the model, without changing the model's fields
155. What is Django? ↑
Django is a web development framework that was developed in a fast-paced newsroom. It is a free and open-source framework that was named after Django Reinhardt who was a jazz guitarist from the 1930s. Django is maintained by a non-profit organization called the Django Software Foundation. The main goal of Django is to enable Web Development quickly and with ease.

156. 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:

python manage.py migrate (migrate command looks at the INSTALLED_APPS settings and creates database tables accordingly)
python manage.py makemigrations (tells Django you have created/ changed your models)
python manage.py sqlmigrate (sqlmigrate takes the migration names and returns their SQL)

157. What are Models? ↑
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.

158. What are views? ↑
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.

159. What are templates? ↑
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.

160. 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.

161. 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 rendertoresponse() 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.

162. 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.

163. What are the signals in Django? ↑
Signals are pieces of code which contain information about what is happening. A dispatcher is used to sending the signals and listen for those signals.

164. What is the role of Cookie in Django? ↑
A cookie is a small piece of information which 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 removes automatically when gets expire. Django provides built-in methods to set and fetch cookie.

The set_cookie() method is used to set a cookie and get() method is used to get the cookie.

165. Django is an MVC based framework, how does this framework implement MVC? ↑
Django is based on MTV architecture which is a variant of MVC architecture. MVC is an acronym for Model, View, and Controller. There are different parts of a website so that they can develop and execute in different machines to achieve faster and more responsive websites. Django implements MTV architecture by having 3 different components and they are all handled by Django itself.

Models are the part which is models.py file in a Django application, which defines the data structure of the particular application.

View are the mediators between models and templates, they receive the data from the Model and make it a dictionary and return the same as a response to a request to the Template.

The Template is the component with which user interacts, and it generates both statically and dynamically in the Django server.

That’s how the Django implements 3 components and work in coordination with each other.

166. How is Django’s code reusability feature different from other frameworks? ↑
Django framework offers more code-reusability then other frameworks out there. As Django Project is a collection of different applications like login application, signup application. These applications can be just copied from one directory to another with some tweaks to settings.py file and you won’t need to write new signup application from scratch.

That’s why Django is a rapid development framework and this level of code reusability is not there in other frameworks.

167. What happens when a typical Django website gets a request? ↑
When a user enters a URL in the browser the same request is received by the Django Server. The server then looks for the match of the requested URL in its URL-config and if the URL matches, it returns the corresponding view function. It will then request the data from the Model of that application, if any data is required and pass it to the corresponding template which is then rendered in the browser, otherwise, a 404 error is returned.

168. Why is Django called loosely coupled framework? ↑
Django is called a loosely coupled framework because of the MTV architecture it’s based on. Django’s architecture is a variant of MVC architecture and MTV is useful because it completely separates server code from the client’s machine.

Django’s Models and Views are present on the client machine and only templates return to the client, which are essentially HTML, CSS code and contains the required data from the models.

These components are totally different from each other and therefore, front-end developers and backend developers can work simultaneously on the project as these two parts changing will have little to no effect on each other when changed.

Therefore, Django is called a loosely coupled framework.

169. Explain the importance of settings.py file and what data/ settings it contains. ↑
When Django server starts, it first looks for settings.py. As the name settings, it is the main settings file of your web application. Everything inside your Django project like databases, backend engines, middlewares, installed applications, main URL configurations, static file addresses, templating engines, allowed hosts and servers and security key stores in this file as a list or dictionary.

So, when your Django server starts it executes settings.py file and then loads particular engines and databases so that when a request is given it can serve the same quickly.

170. Why does Django use regular expressions to define URLs? Is it necessary to use them? ↑
Django uses a very powerful format for storing URLs, that is regular expressions. RegEx or regular expression is the format for sophisticated string searching algorithms. It makes the searching process faster. Although it’s not necessary to use RegEx when defining URLs.

They can be defined as normal string also, Django server should still be able to match them, but when you need to pass some data from the user via URL, then RegEx is used. The RegEx also makes much cleaner URLs then other formats.

171. Django is too monolithic. Explain this statement. ↑
Django framework is too monolithic, that is true to some extent. Django is MTV architecture based framework and since Django is the controller of the architecture, it requires some rules that the developer should follow so that the framework can find and execute appropriate files at the right time. Therefore, Django is one of the frameworks where file structure is as important as its architecture. In Django, you get great customisability with the implementations. There is just one condition that you cannot change the file names, the pre-defined lists and variable names.

You can create new ones but you can’t change the pre-defined variables for which people say that they always have to follow a certain pattern while working on Django.

Django’s file structure is one of the most logical workflows. The monolithic behavior is actually helping the developers to easily understand the project. Even, when the company changes, the project layout remains the same. Therefore, the developer would take less time to understand every aspect, will be able to perform more work productively.

172. Why permanent redirecting is not a good option? ↑
Permanent redirecting is not good an option because the browser caches the response generated by the permanent redirect. This is the difference between permanent and temporary redirect. It causes all sorts of issues when you change that redirect to something different.

Since the browser has cached the redirect before, this time it won’t look on the server for the changed redirection and will load the previously saved redirect. So, even though the developer might have redirected the user to a different page, it will still load the same page. It is browser/ client-side operation, therefore, the user can’t even do anything about the same.

Because of this reason, permanent redirecting is not a good option as informing the

users to clear their internal caching data is not good for any website.

173. Explain user authentication in Django? ↑
Django comes with a built-in user authentication system, which handles objects like users, groups, user-permissions and some cookie-based user sessions. Django's User authentication not only authenticates (verifying the user identity) the user but also authorizes him (determines what permissions user have).

The system consists and operates on these objects: - users - Permissions - Groups - Password Hashing System - Forms Validation - A pluggable backend system

There are many third-party web applications which we can use in place of the default system as they provide much more control over user authentication with more features.

174. 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.

175. What are custom validation rules in form data? ↑
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.

176. What is the difference between authentication and authorization? ↑
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.

177. What is Pagination? ↑
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.

178 Explain the use of migrate command in Django? ↑
In Django, migrations are used to propagate changes made to the models. The migrate command is basically used to apply or unapply migrations changes made to the models. This command basically synchronizes the current set of models and migrations with the database state. You can use this command with or without parameters. In case you do not specify any parameter, all apps will have all their migrations running.

179. What are the roles of receiver and sender in signals? ↑
The roles of receiver and sender in signals are: - Receiver: It specifies the callback function which will be connected to the signal. - Sender: It specifies a particular sender to receive a signal from.

180. What is CSRF? ↑
CSRF – Cross Site Request Forgery. Csrf tokens could also be sent to a client by an attacker due to session fixation or other vulnerabilities or guessed via a brute-force attack, rendered on a malicious page that generates thousands of failed requests.

181. What is CRUD? ↑
The most common task in web application development is to write create, read, update and delete functionality (CRUD) for each table. It refers to the set of common operations that are used in web applications to interact with data from the database. It provides a CRUD interface that allows users to create, read, update or delete data in the application database.

Django helps us with its simplified implementation for CRUD operations using Function-Based views and class-based views:

Function- based views are simple to implement and easy to read but they are hard to customize or extend the functionality. Code reuse is not allowed and so there is repetitiveness.

Class-based views - In no time CRUD operations can be implemented using CBVs. As the model evolves changes would be reflected automatically in CBVs. CBVs are easily

extendable and allow code reuse. Django has built-in generic CBVs which makes it easy to use.

182. Can you tell us something about the Django admin interface? ↑
Django is actually of preloaded interface designed to fulfill the requirement of web developers. Basically, it indicates any requirements for the web developer to make other admin because the whole process is time-consuming and costly. Django admin interface supports user authentication and follows most of the included features. The application Django admin is imported from the Django.contrib package. This imported application is also expected to get control by the corresponding organization hence it does not require an additional front end.

The Django admin interface provides a number of advanced features like-

Authorization access

Managing multiple models

Content management system

183. How is the reusability feature of Django different from the rest of the frameworks? ↑
Django offers maximum code reusability to the developers as compared to other frameworks. This framework is also a collection of various applications including login application or signup application. With the help of this framework, a large number of applications can directly be copied from one directory to the next following some of the settings.py files. With this framework, developers can easily work over the application without writing the new sign up application. This is the reason which supports the rapid development framework in Django and there are no other compatible frameworks supporting this level of code reusability.

184. How does Django's admin interface support customization? ↑
Django admin interface easily supports the customization and developer can download various other third-party applications and install them in a completely different view. Also, if the developer wants overall control over their admin then they can make their own admin application. The admin site can also be customized for example changing the properties of admin.site.object. this admin interface also supports the changes in modifications in models and to apply them in Django admin for specific applications. The Django admin interface supports customization exact from the lowest level and the developer can also create new admin interfaces.

185. How are RESTful APIs beneficial for developers? ↑
RESTful APIs are extremely beneficial for web developers to build web applications as they consume the lowest bandwidth and their design in such a way to communicate well with the internet mainly like PUT, POST, GET, etc.

186. 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.

187. Explain how you can use file based sessions? ↑
To use file based session you have to set the SESSION_ENGINE settings to “django.contrib.sessions.backends.file”

188. What is the typical usage of middlewares in Django? ↑
Some usage of middlewares in Django is:

Session management
Use authentication
Cross-site request forgery protection
Content Gzipping

189. What is DRF of Django Rest Frame work? ↑
Django Rest Framework (DRF) is a powerful module for building web APIs. It’s very easy to build model-backed APIs that have authentication policies and are browsable.

190. What is token based authentication system? ↑
A token based authentication system is a security system that authenticates the users who attempt to log in to a server, a network, or some other secure system, using a security token provided by the server

191. How to implement social login authentication in Django? ↑
Run the development server to make sure all is in order. The install python-social-auth using the pip install command. Update settings.py to include/register the library in the project Update the database by making migrations.

Update the Project’s urlpatterns in urls.py to include the main auth URLs. Create a new app https://apps.twitter.com/app/new and make sure to use the callback url http://127.0.0.1:8000/complete/twitter. In the project directory, add a config.py file and grab the consumer key and consumer secret and add them to the config file.

Finally add urls to the config file to specify the login and redirect urls. Do a sanity check and add friendly views.

192. Mention the differences between Django, Pyramid and Flask. ↑
Flask is a “microframework” primarily build for a small application with simpler requirements. In flask, you have to use external libraries. Flask is ready to use. Pyramid is built for larger applications. It provides flexibility and lets the developer use the right tools for their project. The developer can choose the database, URL structure, templating style and more. Pyramid is heavy configurable. Django can also used for larger applications just like Pyramid. It includes an ORM.

41. Explain the use of decorators. ↑
Decorators in Python are used to modify or inject code in functions or classes. Using decorators, you can wrap a class or function method call so that a piece of code can be executed before or after the execution of the original code. Decorators can be used to check for permissions, modify or track the arguments passed to a method, logging the calls to a specific method, etc.

Advanced

193. How would you compare Node.js and Django? ↑
It heavily depends on the priorities you set in your project. The most common criteria are: database type (e.g. Django for a relational one), security (Django offers great security), rapid development (also Django). Other criteria may include: better performance (Node.js is more fitting), creating features from the ground up (also Node.js) or better client-side processing (Node.js).

A distinctive caveat of Node.js lies in its asynchronous elements: they require the developer to be ever so vigilant because code errors may not reveal themselves until late in production.

194. How can we use model inheritance? ↑
Django’s object-oriented models can be easily mapped to database table structures, creating inheritance properties that they will be sharing. That way, the inheriting model will not interfere with the base one. It is common for a web project to include multiple models; to showcase inheritance, we can use a base model (which we can name “content” and have it store general description values related to it:

title, description, dates of creation and/or modification, etc.) together with another model (that we can name “audio”). The “audio” model will inherit the properties of the base model “content”, while also utilizing its own: source, link, embed code.

Having created the base model, we let Django map it to a table named content — and makes it inherit from model.Model. We then create the “audio” model but make it inherit from “content” instead of model.Model. The convenience of this method comes from the fact that Django manages the inheritance systems autonomously, creating two tables: content and audio respectively. This relation can also be accessed via SQL methods.

195. How can we optimize a Django project’s performance? ↑
Although it depends on the databases and models used in the project, there are some methods that always prove to be effective:

Analyzing the runtime of functions via the line_profiler module. Sometimes we may notice that the code runs slowly — and we need to examine our functions line-by-line. Using the IPython debugger with the line_profiler module, we can see exactly how much time each function takes to execute — and then we optimize the code, if needed.
SQL logging. To dissect a function suspected of low performance even further, we can utilize SQL logging, getting a list of every SQL query that gets executed. It is important to limit this type of logging to a single function — otherwise, the printout would simply flood us with information.
196. What is Unicode, what is UTF-8 and how do they relate? ↑
Unicode is an international encoding standard that works with different languages and scripts. It consists of letters, digits or symbols representing characters from across the world. UTF-8 is a type of encoding, a way of storing the code points of Unicode in a byte form, so you can send Unicode strings over the network or store them in files.

197. How would you scale an existing application when starting a new project? ↑
I see performance and scaling as two separate things. Performance is how fast a user is served and scaling refers to the number of users that can be served by an app at the same time. Usually, time is best spent developing during the early stages of a project. When scale does become an issue, usually business is good and there are sufficient funds to optimize the application.

198. Are there situations where you wouldn’t use Python/Django? ↑
Sure. For example, if a project involves some kind of reasoning it might be better to use Prolog and have Python interface with it. Of course, I would be mindful about
 adding more complexity to the stack by introducing a new language.
199. What is Django Admin Interface?

Ans. Django Admin is the preloaded interface made to fulfill the need of web developers as they won’t need to make another admin panel which is time-consuming and expensive.

Django Admin is application imported from django.contrib packages.

It is meant to be operated by the organization itself and therefore doesn’t need the extensive frontend.

Django’s Admin interface has its own user authentication and most of the general features.

It also offers lots of advanced features like authorization access, managing

different models, CMS (Content Management System), etc.

200. How is Django's code reusability feature different from other frameworks?

Ans. Django framework offers more code-reusability than other frameworks out there.

As Django Project is a collection of different applications like login application, signup application.

These applications can be just copied from one directory to another with some tweaks to settings.py file and you won’t need to write new signup application from scratch.

That as why Django is a rapid development framework and this level of code reusability is not there in other frameworks.

SQL NOTES

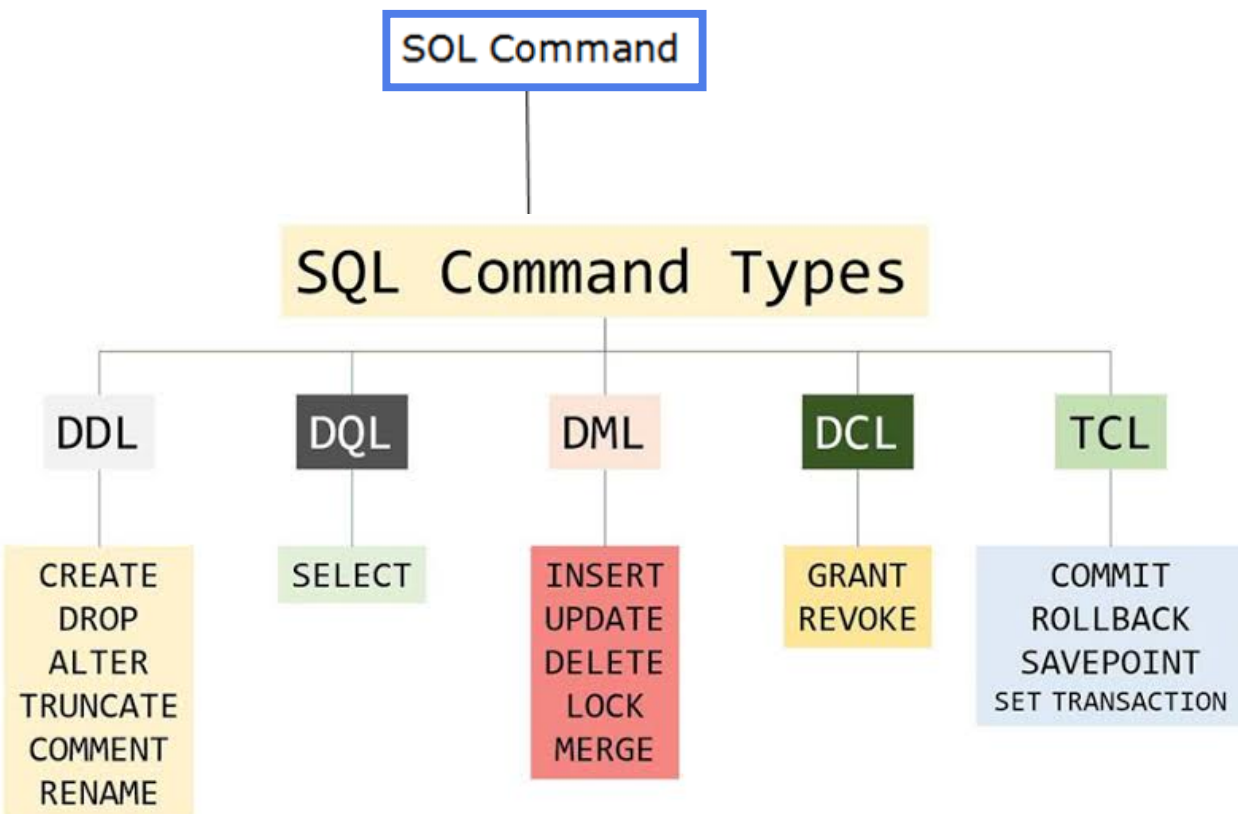
SQL Commands

CREATE BY - ATUL KUMAR (LINKEDIN)

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

```
1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

Example:

```
1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
```

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
1. DROP TABLE ;
```

Example

```
1. DROP TABLE EMPLOYEE;
```

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

```
1. ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify existing column in the table:

```
1. ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

EXAMPLE

```
1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);
```

Or

```
1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, .... valueN);
```

For example:

```
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
1. UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

For example:

```
1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
1. DELETE FROM table_name [WHERE condition];
```

For example:

```
1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

```
1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

b. Revoke: It is used to take back permissions from the user.

Example

```
1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
1. COMMIT;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
1. ROLLBACK;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
1.  SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;
```

For example:

```
1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;
```

- **CREATE TABLE**

The SQL **CREATE TABLE** statement is used to create a new table.

Syntax

The basic syntax of the CREATE TABLE statement is as follows –

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY( one or more columns )
);
```

```
CREATE TABLE Employees_details(
ID int,
Name varchar(20),
Address varchar(20)
);
```

Ex:

```
CREATE TABLE CUSTOMERS(
  ID   INT         NOT NULL,
  NAME VARCHAR (20)    NOT NULL,
  AGE  INT         NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

- **DROP TABLE**

The SQL **DROP TABLE** statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

NOTE – You should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.

Syntax

The basic syntax of this DROP TABLE statement is as follows –

```
DROP TABLE table_name;
```

- **INSERT INTO**

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows –

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example

The following statements would create six records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in the CUSTOMERS table as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

- **The Select Query**

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax:

The basic syntax of the SELECT statement is as follows –

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

Example:

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
```

```
| 3 | kaushik  |  2000.00 |
| 4 | Chaitali |  6500.00 |
| 5 | Hardik   |  8500.00 |
| 6 | Komal    |  4500.00 |
| 7 | Muffy    | 10000.00 |
+----+----------+----------+
```

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the result as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
| 2 | Khilan   |  25 | Delhi     |  1500.00 |
| 3 | kaushik  |  23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

- **WHERE Clause**

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc. The following examples would make this concept clear.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
| 2 | Khilan   |  25 | Delhi     |  1500.00 |
| 3 | kaushik  |  23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
```

```
+----+----------+-----+-----------+----------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes (''). Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

- **AND and OR Conjunctive Operators**

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

- The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

This would produce the following result −

```
+----+-------+----------+
| ID | NAME  | SALARY   |
+----+-------+----------+
|  6 | Komal |  4500.00 |
|  7 | Muffy | 10000.00 |
+----+-------+----------+
```

- The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
```

```
| 6 | Komal    | 22 | MP       | 4500.00 |
| 7 | Muffy    | 24 | Indore   | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block hasa query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
| 3 | kaushik  |  2000.00 |
| 4 | Chaitali |  6500.00 |
| 5 | Hardik   |  8500.00 |
| 6 | Komal    |  4500.00 |
| 7 | Muffy    | 10000.00 |
```

- **UPDATE Query**

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows −

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali | 25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 6 | Komal    | 22 | MP        |  4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following code block.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records −

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  1 | Ramesh   |  32 | Pune    | 1000.00 |
|  2 | Khilan   |  25 | Pune    | 1000.00 |
|  3 | kaushik  |  23 | Pune    | 1000.00 |
|  4 | Chaitali |  25 | Pune    | 1000.00 |
|  5 | Hardik   |  27 | Pune    | 1000.00 |
|  6 | Komal    |  22 | Pune    | 1000.00 |
|  7 | Muffy    |  24 | Pune    | 1000.00 |
+----+----------+-----+---------+---------+
```

- **DELETE Query**

The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows −

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
```

```
| 6 | Komal    | 22 | MP       |  4500.00 |
| 7 | Muffy    | 24 | Indore   | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code has a query, which will DELETE a customer, whose ID is 6.

```
SQL> DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows −

```
SQL> DELETE FROM CUSTOMERS;
```

Now, the CUSTOMERS table would not have any record.

- **LIKE Clause**

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

Syntax

The basic syntax of % and _ is as follows −

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'
```

or

```
SELECT FROM table_name
WHERE column LIKE '%XXXX%'
```

or

```
SELECT FROM table_name
WHERE column LIKE 'XXXX_'
```

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '_' operators −

Sr.No.	Statement & Description
1	**WHERE SALARY LIKE '200%'** Finds any values that start with 200.
2	**WHERE SALARY LIKE '%200%'** Finds any values that have 200 in any position.
3	**WHERE SALARY LIKE '_00%'** Finds any values that have 00 in the second and third positions.
4	**WHERE SALARY LIKE '2_%_%'** Finds any values that start with 2 and are at least 3 characters in length.
5	**WHERE SALARY LIKE '%2'** Finds any values that end with 2.
6	**WHERE SALARY LIKE '_2%3'** Finds any values that have a 2 in the second position and end with a 3.
7	**WHERE SALARY LIKE '2___3'** Finds any values in a five-digit number that start with 2 and end with 3.

Let us take a real example, consider the CUSTOMERS table having the records as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
```

```
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 6 | Komal    | 22 | MP        |  4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

```
SQL> SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

- **WILDCARD Operator**

The SQL LIKE operator, which is used to compare a value to similar values using the wildcard operators.

SQL supports two wildcard operators in conjunction with the LIKE operator which are explained in detail in the following table.

Sr.No.	Wildcard & Description
1	**The percent sign (%)** Matches one or more characters. **Note** − MS Access uses the asterisk (*) wildcard character instead of the percent sign (%) wildcard character.
2	**The underscore (_)** Matches one character. **Note** − MS Access uses a question mark (?) instead of the underscore (_) to match any one character.

The percent sign represents zero, one or multiple characters. The underscore represents a single number or a character. These symbols can be used in combinations.

Syntax

The basic syntax of a '%' and a '_' operator is as follows.

```
SELECT * FROM table_name
WHERE column LIKE 'XXXX%'
```

or

```
SELECT * FROM table_name
```

WHERE column LIKE '%XXXX%'

or

SELECT * FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT * FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT * FROM table_name
WHERE column LIKE '_XXXX_'

You can combine N number of conditions using the AND or the OR operators. Here, XXXX could be any numeric or string value.

Example

The following table has a number of examples showing the WHERE part having different LIKE clauses with '%' and '_' operators.

Sr.No.	Statement & Description
1	**WHERE SALARY LIKE '200%'** Finds any values that start with 200.
2	**WHERE SALARY LIKE '%200%'** Finds any values that have 200 in any position.
3	**WHERE SALARY LIKE '_00%'** Finds any values that have 00 in the second and third positions.
4	**WHERE SALARY LIKE '2_%_%'** Finds any values that start with 2 and are at least 3 characters in length.
5	**WHERE SALARY LIKE '%2'** Finds any values that end with 2.
6	**WHERE SALARY LIKE '_2%3'** Finds any values that have a 2 in the second position and end with a 3.

7	**WHERE SALARY LIKE '2___3'** Finds any values in a five-digit number that start with 2 and end with 3.

Let us take a real example, consider the CUSTOMERS table having the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block is an example, which would display all the records from the CUSTOMERS table where the SALARY starts with 200.

```
SQL> SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

This would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

- **TOP, LIMIT or ROWNUM Clause**

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

Note − All the databases do not support the TOP clause. For example MySQL supports the **LIMIT** clause to fetch limited number of records while Oracle uses the **ROWNUM** command to fetch a limited number of records.

Syntax

The basic syntax of the TOP clause with a SELECT statement would be as follows.

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE [condition]
```

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
```

```
| 3 | kaushik  |  23 | Kota     |  2000.00 |
| 4 | Chaitali |  25 | Mumbai   |  6500.00 |
| 5 | Hardik   |  27 | Bhopal   |  8500.00 |
| 6 | Komal    |  22 | MP       |  4500.00 |
| 7 | Muffy    |  24 | Indore   | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following query is an example on the SQL server, which would fetch the top 3 records from the CUSTOMERS table.

```
SQL> SELECT TOP 3 * FROM CUSTOMERS;
```

This would produce the following result −

```
+----+---------+-----+-----------+---------+
| ID | NAME    | AGE | ADDRESS   | SALARY  |
+----+---------+-----+-----------+---------+
|  1 | Ramesh  |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan  |  25 | Delhi     | 1500.00 |
|  3 | kaushik |  23 | Kota      | 2000.00 |
+----+---------+-----+-----------+---------+
```

If you are using MySQL server, then here is an equivalent example −

```
SQL> SELECT * FROM CUSTOMERS
LIMIT 3;
```

This would produce the following result −

```
+----+---------+-----+-----------+---------+
| ID | NAME    | AGE | ADDRESS   | SALARY  |
+----+---------+-----+-----------+---------+
|  1 | Ramesh  |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan  |  25 | Delhi     | 1500.00 |
|  3 | kaushik |  23 | Kota      | 2000.00 |
+----+---------+-----+-----------+---------+
```

If you are using an Oracle server, then the following code block has an equivalent example.

```
SQL> SELECT * FROM CUSTOMERS
WHERE ROWNUM <= 3;
```

This would produce the following result −

```
+----+---------+-----+-----------+---------+
| ID | NAME    | AGE | ADDRESS   | SALARY  |
+----+---------+-----+-----------+---------+
|  1 | Ramesh  |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan  |  25 | Delhi     | 1500.00 |
|  3 | kaushik |  23 | Kota      | 2000.00 |
+----+---------+-----+-----------+---------+
```

- **ORDER BY Clause**

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

Syntax

The basic syntax of the ORDER BY clause is as follows −

```
SELECT column-list
```

FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort that column should be in the column-list.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY −

```
SQL> SELECT * FROM CUSTOMERS
  ORDER BY NAME, SALARY;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

The following code block has an example, which would sort the result in the descending order by NAME.

```
SQL> SELECT * FROM CUSTOMERS
  ORDER BY NAME DESC;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
+----+----------+-----+-----------+----------+
```

- **Group By Clause**

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example

Consider the CUSTOMERS table is having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
  GROUP BY NAME;
```

This would produce the following result –

```
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

Now, let us look at a table where the CUSTOMERS table has the following records with duplicate names –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
```

```
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Ramesh   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | kaushik  | 25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 6 | Komal    | 22 | MP        |  4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now again, if you want to know the total amount of salary on each customer, then the GROUP BY query would be as follows −

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
  GROUP BY NAME;
```

This would produce the following result −

```
+---------+-------------+
| NAME    | SUM(SALARY) |
+---------+-------------+
| Hardik  |    8500.00 |
| kaushik |    8500.00 |
| Komal   |    4500.00 |
| Muffy   |   10000.00 |
| Ramesh  |    3500.00 |
+---------+-------------+
```

- **Distinct Keyword**

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only those unique records instead of fetching duplicate records.

Syntax

The basic syntax of DISTINCT keyword to eliminate the duplicate records is as follows −

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

First, let us see how the following SELECT query returns the duplicate salary records.

```
SQL> SELECT SALARY FROM CUSTOMERS
  ORDER BY SALARY;
```

This would produce the following result, where the salary (2000) is coming twice which is a duplicate record from the original table.

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

Now, let us use the DISTINCT keyword with the above SELECT query and then see the result.

```
SQL> SELECT DISTINCT SALARY FROM CUSTOMERS
  ORDER BY SALARY;
```

This would produce the following result where we do not have any duplicate entry.

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

- **Alias query**

You can rename a table or a column temporarily by giving another name known as **Alias**. The use of table aliases is to rename a table in a specific SQL statement. The renaming is a temporary change and the actual table name does not change in the database. The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

Syntax

The basic syntax of a **table** alias is as follows.

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

The basic syntax of a **column** alias is as follows.

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, the following code block shows the usage of a **table alias**.

```
SQL> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
  FROM CUSTOMERS AS C, ORDERS AS O
  WHERE  C.ID = O.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Following is the usage of a **column alias**.

```
SQL> SELECT  ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME
  FROM CUSTOMERS
  WHERE SALARY IS NOT NULL;
```

This would produce the following result.

```
+-------------+---------------+
| CUSTOMER_ID | CUSTOMER_NAME |
+-------------+---------------+
|           1 | Ramesh        |
|           2 | Khilan        |
|           3 | kaushik       |
|           4 | Chaitali      |
|           5 | Hardik        |
|           6 | Komal         |
```

```
|        7 | Muffy       |
+----------+---------------+
```

- **UNIONS CLAUSE**

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order

But they need not have to be in the same length.

Syntax

The basic syntax of a **UNION** clause is as follows −

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
```

```
| 100 | 2009-10-08 00:00:00 |       3 |  1500 |
| 101 | 2009-11-20 00:00:00 |       2 |  1560 |
| 103 | 2008-05-20 00:00:00 |       4 |  2060 |
+-----+---------------------+---------+-------+
```

Now, let us join these two tables in our SELECT statement as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION
  SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result −

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
+------+----------+--------+---------------------+
```

- The UNION ALL Clause

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.

Syntax

The basic syntax of the **UNION ALL** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables,

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 – ORDERS table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows –

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION ALL
  SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

There are two other clauses (i.e., operators), which are like the UNION clause.

- SQL INTERSECT Clause – This is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.

- SQL EXCEPT Clause – This combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

- **Using Joins**

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables –

Table 1 – CUSTOMERS Table

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 – ORDERS Table

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT ID, NAME, AGE, AMOUNT
  FROM CUSTOMERS, ORDERS
  WHERE  CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

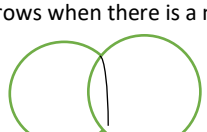
This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

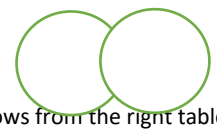
Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL –

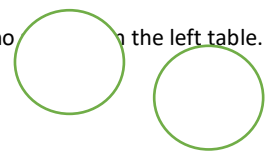
- INNER JOIN – returns rows when there is a match in both tables.



- LEFT JOIN – returns all rows from the left table, even if there are no matches in the right table.



- RIGHT JOIN – returns all rows from the right table, even if there are no matches in the left table.



- FULL JOIN – returns rows when there is a match in one of the tables.

- SELF JOIN – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

- CARTESIAN JOIN – returns the Cartesian product of the sets of records from the two or more joined tables.

1. INNER JOIN

- The most important and frequently used of the joins is the **INNER JOIN**. They are also referred to as an **EQUIJOIN**.
- The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.
- Syntax
- The basic syntax of the **INNER JOIN** is as follows.
- SELECT table1.column1, table1.column2,table2.column2...
- FROM table1
- INNER JOIN table2
- ON table1.common_field = table2.common_field;
- Example
- Consider the following two tables.
- **Table 1** – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali | 25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 6 | Komal    | 22 | MP        |  4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
```

```
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
| OID | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables using the INNER JOIN as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   INNER JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```

2. LEFT JOIN

The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax

The basic syntax of a **LEFT JOIN** is as follows.

```
SELECT table1.column1, table2.column2...
FROM table1
LEFT JOIN table2
ON table1.common_field = table2.common_field;
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables,

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 – Orders Table is as follows.

```
+-----+---------------------+-------------+--------+
| OID | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables using the LEFT JOIN as follows.

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```

3.RIGHT JOIN

The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax

The basic syntax of a **RIGHT JOIN** is as follow.

```
SELECT table1.column1, table2.column2...
```

```
FROM table1
RIGHT JOIN table2
ON table1.common_field = table2.common_field;
```

Example

Consider the following two tables,

Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 – ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables using the RIGHT JOIN as follows.

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

4.FULL JOIN

The SQL **FULL JOIN** combines the results of both left and right outer joins.

The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

Syntax

The basic syntax of a **FULL JOIN** is as follows −

```
SELECT table1.column1, table2.column2...
FROM table1
FULL JOIN table2
ON table1.common_field = table2.common_field;
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables using FULL JOIN as follows.

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  FULL JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result −

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
```

```
|   2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|   4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

If your Database does not support FULL JOIN (MySQL does not support FULL JOIN), then you can use **UNION ALL** clause to combine these two JOINS as shown below.

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION ALL
  SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
```

5.SELF JOINS

The SQL **SELF JOIN** is used to join a table to itself as if the table were two tables; temporarily renaming at least one table in the SQL statement.

Syntax

The basic syntax of SELF JOIN is as follows −

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_field = b.common_field;
```

Here, the WHERE clause could be any given expression based on your requirement.

Example

Consider the following table.

CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, let us join this table using SELF JOIN as follows −

```
SQL> SELECT  a.ID, b.NAME, a.SALARY
  FROM CUSTOMERS a, CUSTOMERS b
  WHERE a.SALARY < b.SALARY;
```

This would produce the following result −

```
+----+----------+---------+
```

```
| ID | NAME     | SALARY  |
+----+----------+---------+
|  2 | Ramesh   | 1500.00 |
|  2 | kaushik  | 1500.00 |
|  1 | Chaitali | 2000.00 |
|  2 | Chaitali | 1500.00 |
|  3 | Chaitali | 2000.00 |
|  6 | Chaitali | 4500.00 |
|  1 | Hardik   | 2000.00 |
|  2 | Hardik   | 1500.00 |
|  3 | Hardik   | 2000.00 |
|  4 | Hardik   | 6500.00 |
|  6 | Hardik   | 4500.00 |
|  1 | Komal    | 2000.00 |
|  2 | Komal    | 1500.00 |
|  3 | Komal    | 2000.00 |
|  1 | Muffy    | 2000.00 |
|  2 | Muffy    | 1500.00 |
|  3 | Muffy    | 2000.00 |
|  4 | Muffy    | 6500.00 |
|  5 | Muffy    | 8500.00 |
|  6 | Muffy    | 4500.00 |
+----+----------+---------+
```

6. CARTESIAN JOIN

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

Syntax

The basic syntax of the **CARTESIAN JOIN** or the **CROSS JOIN** is as follows −

```
SELECT table1.column1, table2.column2...
FROM  table1, table2 [, table3 ]
```

Example

Consider the following two tables.

Table 1 − CUSTOMERS table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2: ORDERS Table is as follows −

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables using CARTESIAN JOIN as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS, ORDERS;
```

This would produce the following result −

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   3000 | 2009-10-08 00:00:00 |
|  1 | Ramesh   |   1500 | 2009-10-08 00:00:00 |
|  1 | Ramesh   |   1560 | 2009-11-20 00:00:00 |
|  1 | Ramesh   |   2060 | 2008-05-20 00:00:00 |
|  2 | Khilan   |   3000 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  2 | Khilan   |   2060 | 2008-05-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   2060 | 2008-05-20 00:00:00 |
|  4 | Chaitali |   3000 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   3000 | 2009-10-08 00:00:00 |
|  5 | Hardik   |   1500 | 2009-10-08 00:00:00 |
|  5 | Hardik   |   1560 | 2009-11-20 00:00:00 |
|  5 | Hardik   |   2060 | 2008-05-20 00:00:00 |
|  6 | Komal    |   3000 | 2009-10-08 00:00:00 |
|  6 | Komal    |   1500 | 2009-10-08 00:00:00 |
|  6 | Komal    |   1560 | 2009-11-20 00:00:00 |
|  6 | Komal    |   2060 | 2008-05-20 00:00:00 |
|  7 | Muffy    |   3000 | 2009-10-08 00:00:00 |
|  7 | Muffy    |   1500 | 2009-10-08 00:00:00 |
|  7 | Muffy    |   1560 | 2009-11-20 00:00:00 |
|  7 | Muffy    |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```

- **Constraints**

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL. These constraints have already been discussed in SQL - RDBMS Concepts chapter, but it's worth to revise them at this point.

- NOT NULL Constraint – Ensures that a column cannot have NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX – Used to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

1. **NOT NULL :** By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.

A NULL is not the same as no data, rather, it represents unknown data.

Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs –

```
CREATE TABLE CUSTOMERS(
  ID   INT           NOT NULL,
  NAME VARCHAR (20)    NOT NULL,
  AGE  INT           NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column in Oracle and MySQL, you would write a query like the one that is shown in the following code block.

```
ALTER TABLE CUSTOMERS
  MODIFY SALARY  DECIMAL (18, 2) NOT NULL;
```

2. **DEFAULT CONSTRAINT:** The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Example

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
  ID   INT           NOT NULL,
  NAME VARCHAR (20)    NOT NULL,
  AGE  INT           NOT NULL,
  ADDRESS  CHAR (25) ,
```

```
  SALARY   DECIMAL (18, 2) DEFAULT 5000.00,
  PRIMARY KEY (ID)
);
```

If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like the one which is shown in the code block below.

ALTER TABLE CUSTOMERS

```
MODIFY SALARY  DECIMAL (18, 2) DEFAULT 5000.00;
```

Drop Default Constraint

To drop a DEFAULT constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS
  ALTER COLUMN SALARY DROP DEFAULT;
```

3. **UNIQUE CONSTRAINT:** The UNIQUE Constraint prevents two records from having identical values in a column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having an identical age.

Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

```
CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL UNIQUE,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the AGE column. You would write a statement like the query that is given in the code block below.

```
ALTER TABLE CUSTOMERS
  MODIFY AGE INT NOT NULL UNIQUE;
```

You can also use the following syntax, which supports naming the constraint in multiple columns as well.

```
ALTER TABLE CUSTOMERS
  ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS
  DROP CONSTRAINT myUniqueConstraint;
```

If you are using MySQL, then you can use the following syntax −

```
ALTER TABLE CUSTOMERS
  DROP INDEX myUniqueConstraint;
```

4. **PRIMARY KEY:** A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Note − You would use these concepts while creating database tables.

Create Primary Key

Here is the syntax to define the ID attribute as a primary key in a CUSTOMERS table.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

To create a PRIMARY KEY constraint on the "ID" column when the CUSTOMERS table already exists, use the following SQL syntax −

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

NOTE − If you use the ALTER TABLE statement to add a primary key, the primary key column(s) should have already been declared to not contain NULL values (when the table was first created).

For defining a PRIMARY KEY constraint on multiple columns, use the SQL syntax given below.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID, NAME)
);
```

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax.

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

Delete Primary Key

You can clear the primary key constraints from the table with the syntax given below.

```
ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;
```

5.FOREIGN KEY: A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.

A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Example

Consider the structure of the following two tables.

CUSTOMERS table

```
CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

ORDERS table

```
CREATE TABLE ORDERS (
  ID        INT       NOT NULL,
  DATE       DATETIME,
  CUSTOMER_ID INT references CUSTOMERS(ID),
  AMOUNT     double,
  PRIMARY KEY (ID)
);
```

If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
  ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL syntax.

```
ALTER TABLE ORDERS
  DROP FOREIGN KEY;
```

6.CHECK CONSTRAINT: The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.

Example

For example, the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

```
CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL CHECK (AGE >= 18),
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like the one given below.

```
ALTER TABLE CUSTOMERS
```

```
MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );
```

You can also use the following syntax, which supports naming the constraint in multiple columns as well −

```
ALTER TABLE CUSTOMERS
  ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);
```

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL syntax. This syntax does not work with MySQL.

```
ALTER TABLE CUSTOMERS
  DROP CONSTRAINT myCheckConstraint;
```

Dropping Constraints

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command.

```
ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;
```

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command.

```
ALTER TABLE EMPLOYEES DROP PRIMARY KEY;
```

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint and then enable it later.

Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of the data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in **Referential Integrity (RI)**. These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints which are mentioned above.

- **NULL Values**

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax

The basic syntax of **NULL** while creating a table.

```
SQL> CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
```

```
   AGE  INT           NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is the one that has been left blank during the record creation.

Example

The NULL value can cause problems when selecting data. However, because when comparing an unknown value to any other value, the result is always unknown and not included in the results. You must use the **IS NULL** or **IS NOT NULL** operators to check for a NULL value.

Consider the following CUSTOMERS table having the records as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

Now, following is the usage of the **IS NOT NULL**operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
  FROM CUSTOMERS
  WHERE SALARY IS NOT NULL;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
+----+----------+-----+-----------+----------+
```

Now, following is the usage of the **IS NULL** operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
  FROM CUSTOMERS
  WHERE SALARY IS NULL;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  6 | Komal    |  22 | MP        |          |
```

```
| 7 | Muffy   | 24 | Indore  |          |
+----+----------+-----+-----------+----------+
```

- **ALTER TABLE Command**

The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table. You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

Syntax

The basic syntax of an ALTER TABLE command to add a **New Column** in an existing table is as follows.

```
ALTER TABLE table_name ADD column_name datatype;
```

The basic syntax of an ALTER TABLE command to **DROP COLUMN** in an existing table is as follows.

```
ALTER TABLE table_name DROP COLUMN column_name;
```

The basic syntax of an ALTER TABLE command to change the **DATA TYPE** of a column in a table is as follows.

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

The basic syntax of an ALTER TABLE command to add a **NOT NULL** constraint to a column in a table is as follows.

```
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;
```

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

The basic syntax of an ALTER TABLE command to **ADD CHECK CONSTRAINT** to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
```

The basic syntax of an ALTER TABLE command to **ADD PRIMARY KEY** constraint to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

The basic syntax of an ALTER TABLE command to **DROP CONSTRAINT** from a table is as follows.

```
ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;
```

If you're using MySQL, the code is as follows −

```
ALTER TABLE table_name
DROP INDEX MyUniqueConstraint;
```

The basic syntax of an ALTER TABLE command to **DROP PRIMARY KEY** constraint from a table is as follows.

```
ALTER TABLE table_name
DROP CONSTRAINT MyPrimaryKey;
```

If you're using MySQL, the code is as follows −

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example to ADD a **New Column** to an existing table −

```
ALTER TABLE CUSTOMERS ADD SEX char(1);
```

Now, the CUSTOMERS table is changed and following would be output from the SELECT statement.

```
+----+---------+-----+-----------+----------+------+
| ID | NAME    | AGE | ADDRESS   | SALARY   | SEX  |
+----+---------+-----+-----------+----------+------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 | NULL |
|  2 | Ramesh  |  25 | Delhi     |  1500.00 | NULL |
|  3 | kaushik |  23 | Kota      |  2000.00 | NULL |
|  4 | kaushik |  25 | Mumbai    |  6500.00 | NULL |
|  5 | Hardik  |  27 | Bhopal    |  8500.00 | NULL |
|  6 | Komal   |  22 | MP        |  4500.00 | NULL |
|  7 | Muffy   |  24 | Indore    | 10000.00 | NULL |
+----+---------+-----+-----------+----------+------+
```

Following is the example to DROP sex column from the existing table.

```
ALTER TABLE CUSTOMERS DROP SEX;
```

Now, the CUSTOMERS table is changed and following would be the output from the SELECT statement.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Ramesh   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | kaushik  |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

- **TRUNCATE TABLE**

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax

The basic syntax of a **TRUNCATE TABLE** command is as follows.

```
TRUNCATE TABLE  table_name;
```

Example

Consider a CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example of a Truncate command.

```
SQL > TRUNCATE TABLE CUSTOMERS;
```

Now, the CUSTOMERS table is truncated and the output from SELECT statement will be as shown in the code block below −

```
SQL> SELECT * FROM CUSTOMERS;
Empty set (0.00 sec)
```

- **GROUP BY** clause

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example

Consider the CUSTOMERS table is having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
```

```
| 5 | Hardik  | 27 | Bhopal   | 8500.00 |
| 6 | Komal   | 22 | MP       | 4500.00 |
| 7 | Muffy   | 24 | Indore   | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
  GROUP BY NAME;
```

This would produce the following result −

```
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

Now, let us look at a table where the CUSTOMERS table has the following records with duplicate names −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh  | 32 | Ahmedabad | 2000.00 |
| 2 | Ramesh  | 25 | Delhi     | 1500.00 |
| 3 | kaushik | 23 | Kota      | 2000.00 |
| 4 | kaushik | 25 | Mumbai    | 6500.00 |
| 5 | Hardik  | 27 | Bhopal    | 8500.00 |
| 6 | Komal   | 22 | MP        | 4500.00 |
| 7 | Muffy   | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now again, if you want to know the total amount of salary on each customer, then the GROUP BY query would be as follows −

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
  GROUP BY NAME;
```

This would produce the following result −

```
+---------+-------------+
| NAME    | SUM(SALARY) |
+---------+-------------+
| Hardik  |     8500.00 |
| kaushik |     8500.00 |
| Komal   |     4500.00 |
| Muffy   |    10000.00 |
| Ramesh  |     3500.00 |
+---------+-------------+
```

- **Using VIEWS**

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
```

```
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View

A view can be updated under certain conditions which are given below −

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW
  SET AGE = 35
  WHERE name = 'Ramesh';
```

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
```

```
| ID | NAME    | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 35 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali | 25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 6 | Komal    | 22 | MP        |  4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
  WHERE age = 22;
```

This would ultimately delete a row from the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME    | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 35 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali | 25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

DROP VIEW view_name;

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

Queries Using aggregate functions :-

1. **2nd highest Salary :**

   Syntax:

   SELECT MAX(SALARY)

   FROM TABLENAME

   WHERE SALARY NOT IN (SELECT MAX(SALARY) FROM TABLENAME);

2. **2ND minimum Salary**

   Syntax:

   SELECT MIN(SALARY)

   FROM TABLENAME

   WHERE SALARY NOT IN (SELECT MIN(SALARY) FROM TABLENAME);

3. **Nth Highest Salary:**

   Syntax:

   SELECT MIN(SALARY)
   FROM TABLENAME
   WHERE SALARY IN (SELECT TOP Nth SALARY FROM TABLENAME
   ORDER BY SALARY DESC);

4. **Nth Minimum Salary :**
   Syntax:

SELECT MAX (SALARY)
FROM TABLENAME
WHERE SALARY IN (SELECT TOP Nth SALARY FROM TABLENAME
ORDER BY SALARY);

IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Or

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

For ex: The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

Ex 2: The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Ex: The following SQL statement selects all products with a price between 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni:

Ex:

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Union and Union ALL

UNION

The UNION command is used to select related information from two tables, which is like a JOIN command. However, when using UNION command, all the selected columns need to be of the same data type. With UNION, only distinct values are selected.

UNION ALL

UNION ALL command is equal to UNION command, except that UNION ALL selects all the values.

The difference between Union and Union all is that Union all will not eliminate duplicate rows, instead it just pulls all the rows from all the tables fitting your query specifics and combines them into a table.

A UNION statement effectively does a SELECT DISTINCT on the results set. If you know that all the records returned are unique from your union, use UNION ALL instead, it gives faster results.

Example

Table 1 : First,Second,Third,Fourth,Fifth
Table 2 : First,Second,Fifth,Sixth

Result Set

UNION: First,Second,Third,Fourth,Fifth,Sixth (This will remove duplicate values)
UNION ALL: First,First,Second,Second,Third,Fourth,Fifth,Fifth,Sixth,Sixth (This will repeat values)

The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

COUNT() Example

The following SQL statement finds the number of products:

```
SELECT COUNT(ProductID)
FROM Products;
```

The AVG() function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

AVG() Example

The following SQL statement finds the average price of all products:

```
SELECT AVG(Price)
FROM Products;
```

The SUM() function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

Min Example

The following SQL statement finds the price of the cheapest product:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

MAX() Example

The following SQL statement finds the price of the most expensive product:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

The FIRST() Function

The FIRST() function returns the first value of the selected column.

SQL FIRST() Syntax

```
SELECT FIRST(column_name) FROM table_name;
```

SQL FIRST() Example

The following SQL statement selects the first value of the "CustomerName" column from the "Customers" table:

```
SELECT FIRST(CustomerName) AS FirstCustomer FROM Customers; LAST
```

The LAST() Function

The LAST() function returns the last value of the selected column.

SQL LAST() Syntax

```
SELECT LAST(column_name) FROM table_name;
```

SQL LAST() Example

The following SQL statement selects the last value of the "CustomerName" column from the "Customers" table:

```
SELECT LAST(CustomerName) AS LastCustomer FROM Customers;
```

The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Having:- used to select records which satisfy the given condition and also it is used with aggregate function.

SQL HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

SQL Views

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

VIEW: CREATES A VIRTUAL TABLE BASED ON RESULT SET OF SQL STATEMENT.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

We can query the view above as follows:

```
SELECT * FROM [Brazil Customers];
```

TABLE 1 TID, TNAME, TCITY, TID - PK

TABLE 2 TDATE, T2ID2, T2NAME, TID TID-FK, T2ID2 - PK