### Explain what is Django.

### Django is an open-source web framework for creating Python-based web applications. As there is a huge demand of Python based web applications, the demand for Django Developers is also high.

1. **Django architecture**

Django architecture consists of

**Models:** It describes database schema and data structure.

**Views:** It controls what a user sees, the view retrieves data from appropriate models

and executes 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

### What is [****Decorators****](https://www.geeksforgeeks.org/function-decorators-in-python-set-1-introduction/) ?

### In Python, a decorator is a function that takes another function as an argument and adds functionality or augments the function without changing it. Django, as a Python web framework, comes with a large number of built-in decorators. These built-in decorators are used when decorating function-based views.

### Syntax:

### -----------

@my\_decorator

def say\_hello():

print("Hello!")

### Mention what are the features available in Django?

Features available in Django are

* Admin Interface (CRUD)
* Templating
* Form handlingx
* Internationalization
* Session, user management, role-based permissions
* Object-relational mapping (ORM)
* Testing Framework
* Fantastic Documentation

### How you can create a project in Django

django-admin startproject ProjectName

python3 manage.py startapp ProjectApp

1. **Explain the migration in Django and how you can do in SQL.**

Migration in Django is to make changes to models like deleting a model, adding a field, etc. into database schema. There are several commands you use to interact with migrations.

* Migrate
* Makemigrations
* Sqlmigrate

To do the migration in [SQL](https://www.guru99.com/sql-server-questions.html), you have to print the SQL statement for resetting sequences for a given app name.

**django-admin.py sqlsequencereset**

Use this command to generate SQL that will fix cases where a sequence is out sync with its automatically incremented field data.

### 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.

### 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”

### ****What is Sessions Framework in Django?****

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

### ****What is a cookie in Django?****

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

### ****Explain Django’s Request / Response Cycle.****

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

### ****12. 9 ?****

### Django ORM is a database abstraction API using which we can interact with its database models i.e., perform actions like add, delete, modify and query objects. e.g., SQLite.

### ****13. What is a middleware in Django ?****

### Middleware is a layer in Django’s Request / Response processing pipeline. Each middleware is responsible for performing some specific functions on the request and/or response, such as caching, Gzipping, etc.

### What is the use of Middleware in Django?

### Middleware in Django is a lightweight plugin that processes during request and response execution. It performs functions like security, CSRF protection, session, authentication, etc. Django supports various built-in middleware.

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

### ****16. What is a context in Django?****

### A context in Django is a [dictionary](https://intellipaat.com/blog/tutorial/python-tutorial/python-dictionary/), in which keys represent variable names and values represent their values. This dictionary (context) is passed to the template which then uses the variables to output the dynamic content.

### 17. ****What is a Meta Class in Django?****

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

### ****What is serialization in Django?****

### Serialization is the process of converting Django models into other formats such as XML, JSON, etc.

### 19. 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 want 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
* **Concrete Base Classes:**

### 20. What is the usage of Django-admin.py and manage.py

**Django-admin.py:**  It is a Django's command line utility for administrative tasks.

**Manage.py:** It is an automatically created file 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.
* It sets the DJANGO\_SETTING\_MODULE environment variable to points to your project's setting.py file.

### 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.

### 22. What are the two important parameters in signals?

Two important parameters in signals are:

**Sender:** It specifies a particular sender from where a signal is received.

**Receiver:** It specifies the callback function which connected to the signal.

**23. What’s the difference between configuration a project and an app in Django?**

### A project is a collection of files and apps whereas

### the app is a web application which is written to perform business logic.

### 24. What is Trigger?

## **It is a special type of stored procedure that is invoked automatically in response to an event**. Each trigger is associated with a table, which is activated on any DML statement such as **INSERT, UPDATE**, or **DELETE**.

## **25. Explain the use of ‘migrate’ command in Django ?**

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. We can use this command with or without parameters. In case you do not specify any parameter, all apps will have all their migrations running.

# **26. Difference between DELETE, TRUNCATE and DROP commands in SQL**

**Syntax of a SQL DELETE Statement:**

// Without where clause, removed all records **DELETE FROM Table\_Name**

// With where clause, removed specific records **DELETE FROM Table\_name WHERE Condition**

**Syntax of a SQL TRUNCATE Statement**

// Removed all records **TRUNCATE TABLE table\_name**

// Drop Database table **DROP TABLE table\_name**

**28. What is AJAX**

AJAX stands for **Asynchronous JavaScript and XML**. It is a group of related technologies used to display data asynchronously. In other words, it sends and retrieves data without reloading the web page

## **29.** **What is Python Module**

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

### ****Syntax of Python Import****

### ****----------------------------------------****

import module

**List:**

Sys, os, Re, Time, Random, Threading, Subprocess, Signal, Glob, sqlite3, Logging

## **30**. **What are the various files that are created when you create a Django Project?**

## **Explain briefly?**

**manage.py** -> is the command-line utility of your Django project and controls the Django project on the server.

**\_\_init\_\_ .py** -> file is an empty file that makes the python interpreter understand that the directory consisting of settings.py is a module/ package.

**settings.py** -> file includes information on all the apps installed in the project.

**urls.py** -> Contains the URL’s for the current project

**wsgi.py** - > This is an entry-point for the web servers to serve the project you have created.

**asgi.py ->** An entry-point for ASGI-compatible web servers to serve your project.

**31. How to combine data frames**

1. pd.concat(): Concatenates DataFrames along a specified axis (rows or columns). It can stack DataFrames vertically or horizontally.

1. import pandas as pd

# Example DataFrames

2. df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})

3. df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

# Concatenating vertically (along rows)

4. result\_vertical = pd.concat([df1, df2], ignore\_index=True)

# Concatenating horizontally (along columns)

result\_horizontal = pd.concat([df1, df2], axis=1)

print(result\_vertical)

print(result\_horizontal)

### 32. What is \*args, \*\*kwargs with example

### \*args: This is used to pass a variable number of non-keyword arguments to a function. It collects additional positional arguments as a tuple.

### Example using args:

### ----------------------

### def my\_function(\*args):

### for arg in args:

### print(arg)

### my\_function(1, 2, 3) # Output: 1 2 3

### \*\*kwargs: This is used to pass a variable number of keyword arguments to a function. It collects additional keyword arguments as a dictionary.

### Example using args:

### ----------------------

### def my\_function(\*\*kwargs):

### for key, value in kwargs.items():

### print(key, value)

### my\_function(name="Alice", age=25) # Output: name Alice, age 25

**34.** **What is difference between mutable and immutable objects**

|  |  |
| --- | --- |
| Mutable | Immutable |
| Mutable objects can be changed after creation | Immutable objects can’t be changed after creation |
| Generally, provides a method to add or remove elements | it does not provide any method to add,remove or change the element |
| Mutable slower to access | Immutable quicker to access |
| Changing mutable objects is easy and efficient | Immutable Expensive or can't be changed |
| Eg: Dictionary, Sets, Lists  Eg: Dictionary  country\_capitals = {  "United States": "Washington D.C.",  "Italy": "Rome",  "England": "London"  }  Eg: Sets:  my\_set = {1, 2, 3}  print(my\_set) # Output: {1, 2, 3}  Eg: List: ages = [19, 26, 23] | Eg: Int, float, str, tuple, bool, frozenset, bytes |

### 35. Difference Between List and Tuple

|  |  |
| --- | --- |
| **List** | **Tuple** |
| List is a Group of comma separated values within Square Brackets and Square Brackets are mandatory.  Eg: i = [10,20,30,40] | Tuple is a Group of comma separated Values with parentheses and Parenthesis are optional  Ex: t= (10,20,30,40)  Ex: t =10,20,30,30 |
| List are mutable  i.e they can be edited. | Tuples are immutable (tuples are lists which can’t be edited) |
| The implication of iterations is Time-consuming | The Implication of iterations is comparatively Faster |
| The List is better for performing operations, such as insertion and deletion. | Tuple data type is appropriate for accessing the elements |
| Lists Consume more memory | Tuple consumes less memory |
| Lists have several built-in methods | Tuple does not have many built-in methods |
| The unexpected changes and errors are more likely to occur. | In tuple, it is hard to take place |

### 36. Example for list and tuple

### List Example:

### A list is an ordered, mutable collection of elements. It allows duplicates and can store items of different data types.

### # Creating a list:

### fruits = ['apple', 'banana', 'cherry']

### # Accessing elements:

### print(fruits[0]) # Output: 'apple'

### # Modifying elements

### fruits[1] = 'orange'

### print(fruits) # Output: ['apple', 'orange', 'cherry']

### # Adding elements

### fruits.append('grape')

### print(fruits) # Output: ['apple', 'orange', 'cherry', 'grape']

### # Removing elements

### fruits.remove('cherry')

### print(fruits) # Output: ['apple', 'orange', 'grape']

### Tuple Example:

### # Creating a tuple

### coordinates = (3, 4)

### # Accessing elements

### x = coordinates[0]

### y = coordinates[1]

### print(x, y) # Output: 3 4

### # Unpacking tuple

### x, y = coordinates

### print(x, y) # Output: 3 4

### # Tuples can also be used for multiple assignment

### point = (5, 6)

### x, y = point

### print(x, y) # Output: 5 6

# 38. **Python Data Structures**

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Characteristics** | **Example Usage** |
| List | Ordered, mutable, allows duplicates | my\_list = [1, 2, 3, 4] |
| Tuple | Ordered, immutable, allows duplicates | my\_tuple = (1, 2, 3) |
| Set | Unordered, mutable, does not allow duplicates | my\_set = {1, 2, 3} |
| Dictionary | Unordered key-value pairs, mutable | my\_dict = {'name': 'Alice', 'age': 25} |
| String | Sequence of characters, immutable | my\_string = "Hello, World!" |
| Stack | Last-in-first-out (LIFO), implemented using lists | Push, pop operations |
| Queue | First-in-first-out (FIFO), implemented using collections | Enqueue, dequeue operations |
| Linked List | Collection of elements with links, dynamic size | LinkedList class or libraries like llist |
| Stack (using LinkedList) | LIFO using linked list implementation | Stack class or libraries like llist |
| Queue (using LinkedList) | FIFO using linked list implementation | Queue class or libraries like llist |
| Hash Table | Key-value storage with fast access times | Implementations include dict and custom classes |
| Heap | Binary tree structure with heap property | Priority queues, heapq module |

39. **What is Python Lambda ?**

A Lambda function is small anonymous function.

A Lambda function can take any number of arguments but can only have one expression.

**Syntax**: **lambda arguments: expression**

Example: x = lambda a : a + 10  
 print(x(5))

## **40. Why Use Lambda Functions?**

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

def myfunc(n):  
  return lambda a : a \* n

### Example

def myfunc(n):  
  return lambda a : a \* n  
mydoubler = myfunc(2)  
print(mydoubler(11))

## **40. How to install virtual environment**

Pip install virtualenv

vietualenv myenv

\myenv\Scripts\activate

**41. Why use virtual environment ?**

Virtualenv is used **to manage Python packages for different projects**. Using virtualenv allows you to avoid installing Python packages globally which could break system tools or other projects.

**42. What is difference between GET and POST Method.**

|  |  |
| --- | --- |
| **GET** | **POST** |
| 1) In case of Get request, only **a limited amount of data**can be sent because data is sent in header. | In case of post request, **large amounts of data**can be sent because data is sent in body. |
| 2) Get request is **not secured**because data is exposed in URL bar. | Post request is **secured**because data is not exposed in URL bar. |
| 3) Get request **can be bookmarked.** | Post request **cannot be bookmarked.** |
| 4) Get request is **more efficient.** | Post request is **less efficient**. |

43. **Second highest salary:**

Select \* from employee where salary=(select Max(salary) from employee);

### SELECT DISTINCT Salary FROM Employee ORDER BY Salary DESC LIMIT 1 OFFSET 1;

### 44. Explain Q objects in Django ORM

### When writing complex queries, Q objects are employed because filter () functions only allow you to 'AND' the conditions; whereas Q objects allow you to 'OR' the conditions.

45. **What is difference between COOKIE and SESSION Method.**

|  |  |
| --- | --- |
| **COOKIE** | **SESSION** |
| 1) Cookie stored in browser as text file format. | Session are stored in server side. |
| 2) Cookie stored Limited Amount of Data. | Session stored unlimited Amount of data |
| 3) Cookie Less Secure | Session More Secure |
| 4) Cookies can only store string | Session can store objects |

# 46. **Second Largest Number in Python**

data = [2, 5, 2, 4, 6, 8, 3, 6, 6, 9, 7]

sorted\_data = sorted(data)

print("Second largest number:", sorted\_data[-2])

# print("First largest number:", sorted\_data[-1])

**47. csv file library**  
 **import** csv or import pandas as pd

**48. Difference between SQL and NOSQL**

|  |  |  |
| --- | --- | --- |
|  | **SQL** | **NoSQL** |
| 1 | SQL databases are categorized as Relational Database Management System (RDBMS). | NoSQL databases are categorized as Non-relational or distributed database system |
| 2. | SQL databases have fixed or static or predefined schema. | NoSQL databases have dynamic schema. |
| 3. | SQL databases are vertically scalable. | NoSQL databases are horizontally scalable. |
| 4. | SQL databases display data in form of tables so it is known as table-based database. | NoSQL databases display data as collection of key-value pair, documents, graph databases or wide-column stores. |
| 5. | SQL databases use a powerful language "Structured Query Language" to define and manipulate the data. | In NoSQL databases, collection of documents are used to query the data. It is also called unstructured query language. It varies from database to database. |
| 6. | SQL databases are best suited for complex queries. | NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries. |
| 7. | SQL databases are not best suited for hierarchical data storage. | NoSQL databases are best suited for hierarchical data storage. |
| 8. | MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc. are the example of SQL database. | MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc. are the example of nosql database |

49. **Comparison between range() and xrange()**

|  |  |
| --- | --- |
| **range() function** | **xrange() function** |
| The range() function is used in Python-3. | The xrange() function is used in Python-2. |
| The range() function is faster. | The xrange() function is slower. |
| The range() function returns a sequence (list) consisting of number that are immutables. | The xrange() function returns a generator object of xrange type. |
| The range() function is less memory optimized. | The xrange() function is more memory optimized. |
| The range() function supports arithmetic operations as it returns a list of integers. | The xrange() function does not support arithmetic operations as it returns a generator object. |
| The range() function is supported by Python-2. | The xrange() function is not supported by Python-3. |
| The syntax of range() function is:  range(start, stop, steps). | The syntax of xrange() function is:  xrange(start, stop, steps). |

**50. What is difference between Pickling and unpickling**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Pickling** | **Unpickling** |
| Definition | Converting Python objects into a byte stream. | Converting a byte stream back into Python objects. |
| Purpose | Serialization for storage or transmission. | Deserialization to recreate Python objects. |
| Result | Byte stream (binary data) | Original Python objects |
| Security Concerns | Risky if unpickling untrusted data, as it can execute arbitrary code. | Potential security risks from unpickling untrusted data. |
| Example | Serialize data into a file or database. | Deserialize data from files or databases. |

51. **How convert a list to a tuple ?**

my\_list=['list1','list2','list3','list4','list5','list6']

my\_tuple=tuple(my\_list)

print(my\_tuple) o/p: ('list1', 'list2', 'list3', 'list4', 'list5', 'list6')

print(type(my\_tuple)) o/p: <class 'tuple'>

52. **How to get dictionary values and in list ?**

data = {'manoja': 'java', 'tripura': 'python', 'manoj': 'statistics', 'manoji': 'cpp'}

# get list of keys

list(data.keys()) o/p: ['manoja', 'tripura', 'manoj', 'manoji']

# get list of values

list(data.values()) o/p: ['java', 'python', 'statistics', 'cpp']

**53. Python Data Types**

Numeric Types: int, float

Boolean Type: bool

Text Type : str

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Binary Types: bytes, bytearray, memoryview

NoneType: None Type;

55. s = "put"

s[0] = "c"

print(s)

o/p: TypeError: 'str' object does not support item assignment

56. s = "put"

print(s)

s = s + "cat"

print(s)

s.upper()

print(s)

o/p: put, putcat,putcat

57. l = [1, 2, 3, 4, 5]

print(l[::-1])

o/p: [5, 4, 3, 2, 1]

58. d = {"a":10, "b":20, "c":10, "a":100}

print(d)

o/p : {"a":10, "b":20, "c":10}

59. p = [1, 2, 3, 4]

q = p

q[0] = 100

print(p)

o/p: [100, 2, 3, 4]

60. Remove duplicate elements from list:   
 my\_list = [1, 2, 1, 3, 2, 3, 1, 4, 5]

my\_final\_list = set(my\_list)

print(list(my\_final\_list))

61. d = {"a":10, "b":20, "c":100}

print(d["x"])

o/p: KeyError: 'x'

62. **Difference between Class method and Static method in Python**

|  |  |  |
| --- | --- | --- |
| **Class Method** | **Static Method** | **Instance Method** |
| Method bound to the class | Method bound to the class | Method bound to an instance |
| `cls` (class itself) | None (no special parameter) | `self` (instance itself) |
| Class-level operations | Utility functions | Access and modify instance-specific data |
| `@classmethod` decorator | `@staticmethod` decorator | Regular method within a class |

63. How can you find common elements and uncommon elements in two lists using Python?

To find common elements,

convert the lists to sets and use the intersection method.

To find uncommon elements,

convert the lists to sets and use set operations like symmetric\_difference.

example:

-----------

list1 = [1, 2, 3, 4, 5]

list2 = [3, 4, 5, 6, 7]

# Find common elements

common\_elements = set(list1).intersection(list2)

print("Common elements:", common\_elements)

# Find uncommon elements

uncommon\_elements = set(list1).symmetric\_difference(list2)

print("Uncommon elements:", uncommon\_elements)

63. **Diff b/w Local & Global scope**

Ans:

|  |  |
| --- | --- |
| **Local Scope** | **Global Scope** |
| Variables defined within a specific block of code, typically a function. | Variables defined outside of any function or block |
| Limited to the specific block where they are defined. | Accessible from anywhere in the program, including within functions. |
| Created when the block is entered and destroyed when the block is exited. | Persist throughout the program's execution, from program start to end. |
| Local variables take precedence over global variables with the same name within the local scope. | Global variables can be accessed within local scopes, but local variables take precedence if they share the same name. |
| python def my\_function(): local\_var = 10 | (None, as global variables are typically defined outside  functions.) |
| python global\_var = 20 | python def my\_function(): global global\_var global\_var = 30 |

## 64. **Difference chart between Authentication and Authorization**

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| Authentication is the process of identifying a user to provide access to a system. | Authorization is the process of giving permission to access the resources. |
| In this, the user or client and server are verified. | In this, it is verified that if the user is allowed through the defined policies and rules. |
| It is usually performed before the authorization. | It is usually done once the user is successfully authenticated. |
| It requires the login details of the user, such as user name & password, etc. | It requires the user's privilege or security level. |
| Data is provided through the Token Ids. | Data is provided through the access tokens. |
| **Example:** Entering Login details is necessary for the employees to authenticate themselves to access the organizational emails or software. | **Example:** After employees successfully authenticate themselves, they can access and work on certain functions only as per their roles and profiles. |
| Authentication credentials can be partially changed by the user as per the requirement. | Authorization permissions cannot be changed by the user. The permissions are given to a user by the owner/manager of the system, and he can only change it. |

65**. Comparisons between Stateless and Stateful Protocol:**

| **Stateless Protocol** | **Stateful Protocol** |
| --- | --- |
| Stateless Protocol does not require the server to retain the server information or session details. | Stateful Protocol require server to save the status and session information. |
| In Stateless Protocol, there is no tight dependency between [server](https://www.geeksforgeeks.org/introduction-of-server/) and [client](https://www.geeksforgeeks.org/client-server-model/). | In Stateful protocol, there is tight dependency between server and client |
| The Stateless protocol design simplify the server design. | The Stateful protocol design makes the design of server very complex and heavy. |
| Stateless Protocols works better at the time of crash because there is no state that must be restored, a failed server can simply restart after a crash. | Stateful Protocol does not work better at the time of crash because stateful server have to keep the information of the status and session details of the internal states. |
| Stateless Protocols handle the transaction very fastly. | Stateful Protocols handle the transaction very slowly. |
| Stateless Protocols are easy to implement in Internet. | Stateful protocols are logically heavy to implement in Internet. |
| Scaling architecture is relatively easier. | It is difficult and complex to scale architecture. |
| The requests are not dependent on the server side and are self contained. | The requests are always dependent on the server side. |
| To process different information at a time , different servers can be used. | To process every request , the same server must be utilized. |
| Example of Stateless are [UDP](https://www.geeksforgeeks.org/user-datagram-protocol-udp/) , [DNS](https://www.geeksforgeeks.org/domain-name-system-dns-in-application-layer/) , [HTTP](https://www.geeksforgeeks.org/http-full-form/) , etc. | Example of Stateful are [FTP](https://www.geeksforgeeks.org/file-transfer-protocol-ftp/) , [Telnet](https://www.geeksforgeeks.org/introduction-to-telnet/) , etc. |

### 66. Python map function

### Syntax: map(function, iterable, ...)

### function: The function to be applied to each item in the iterable(s).

### iterable: One or more iterables containing the elements you want to apply the function to.

### Ex: data = [1,2,3,4,5,6]

### data = list(map(lambda x:x\*\*2,data))

### Print(data)

### 67. Python Program to Print the Fibonacci sequence ?

### Ans:

### def fibonacci(n):

### fib\_sequence = [0, 1]

### while len(fib\_sequence) < n:

### fib\_sequence.append(fib\_sequence[-1] + fib\_sequence[-2])

### return fib\_sequence[:n]

### result=fibonacci(10)

### print(result)

Fibonacci sequence:

0

1

1

2

3

5

8

### 68. How memory management worked in API?

### Memory management in APIs involves handling memory resources efficiently on both the client and server sides. Client-side memory management includes allocation, deallocation, and resource cleanup, while server-side management focuses on optimizing memory usage to handle concurrent requests. Techniques such as garbage collection, buffering, and streaming may be employed, and efficient error handling is crucial. Overall, it ensures responsible use of memory during API interactions.

**69) How to work lambda function in AWS**

1)Lambda runs instances of your function in containers.

2)Each Lambda function has its own container.

3)AWS Lambda packages the function into a new container and executes it on a multi-tenant cluster of machines.

4)Before execution, each function's container is allocated necessary RAM and CPU capacity.

5)Lambda minimizes cold starts by loading the code, creating a micro VM, and optimizing for reduced latency.

6)Once the micro VM is created, the Lambda function runs smoothly without latency.

7)This captures the key points about the containerized and serverless nature of AWS Lambda.

70) Difference between break, continue and pass?

Ans:

-------

Break: Exist the loop prematurely.

Continue: Skips the rest of the loop body for the current iteration.

Pass: No-operation statement, serves as a placeholder.