1. What is a RESTful API?

A **RESTful API** (Representational State Transfer API) is an architectural style for designing networked applications. It uses HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on resources, typically represented as JSON or XML data. REST APIs follow stateless communication and leverage standard HTTP methods such as GET, POST, PUT, DELETE, and PATCH.

---

2. Explain the concept of API specification.

An **API specification** is a document or standard that defines how an API should behave, including its endpoints, request parameters, response formats, and authentication mechanisms. Common API specification formats include **OpenAPI (Swagger)** and **RAML**, which help in documenting and designing APIs consistently.

---

3. What is Flask, and why is it popular for building APIs?

Flask is a **lightweight, micro-framework** for Python that is widely used for building APIs and web applications. It is popular because:

- It is **simple and flexible** to use.
- It provides **built-in routing** and support for HTTP methods.
- It supports **extensions** like Flask-SQLAlchemy, Flask-RESTful, and Flask-JWT for advanced functionalities.
- It has a **minimalist design**, allowing developers to customize and extend features as needed.

---

4. What is routing in Flask?

Routing in Flask refers to the process of mapping a **URL** to a specific **function** in the application. This allows the app to respond to different URLs by executing corresponding logic. Routing is defined using the `@app.route()` decorator.

Example:

python
CopyEdit

```
from flask import Flask
```

```
app = Flask(__name__)

@app.route("/")
def home():
    return "Welcome to Flask!"

if __name__ == "__main__":
    app.run(debug=True)
```

---

5. How do you create a simple Flask application?

To create a basic Flask application:

1. Install Flask: `pip install flask`

2. Write the application:

python
CopyEdit
```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"

if __name__ == "__main__":
    app.run(debug=True)
```

3. Run the script and access the app at `http://127.0.0.1:5000/`.

---

6. What are HTTP methods used in RESTful APIs?

Common HTTP methods in REST APIs:

- **GET** – Retrieve data.
- **POST** – Create a new resource.
- **PUT** – Update an existing resource.
- **DELETE** – Remove a resource.
- **PATCH** – Partially update a resource.

---

7. What is the purpose of the `@app.route()` decorator in Flask?

The `@app.route()` decorator in Flask maps a URL **endpoint** to a specific **view function**. It tells Flask which function to execute when a specific route is accessed.

Example:

python
CopyEdit

```
@app.route('/hello')
def hello():
    return "Hello, World!"
```

Accessing `/hello` in the browser will return `"Hello, World!"`.

---

8. What is the difference between GET and POST HTTP methods?

Feature	GET	POST
Purpose	Retrieve data	Send data to the server
Visibility	Parameters are visible in the URL	Parameters are in the request body
Security	Less secure	More secure for sensitive data
Idempotent	Yes (repeated requests have no side effects)	No (repeated requests may create multiple records)

Example:

- `GET /users?id=123` retrieves user with ID 123.

- `POST /users` with JSON `{ "name": "John" }` creates a new user.

---

9. How do you handle errors in Flask APIs?

Flask provides error handling using `@app.errorhandler()`.

Example:

python
CopyEdit

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.errorhandler(404)
def not_found(error):
    return jsonify({"error": "Not found"}), 404
```

This returns a JSON response when a **404 error** occurs.

---

10. How do you connect Flask to a SQL database?

Flask uses **Flask-SQLAlchemy** to connect to databases.

Example connection to SQLite:

python
CopyEdit

```
from flask_sqlalchemy import SQLAlchemy
from flask import Flask

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
```

---

11. What is the role of Flask-SQLAlchemy?

Flask-SQLAlchemy is an ORM (Object-Relational Mapping) extension for Flask that allows developers to interact with databases using **Python objects** instead of raw SQL queries.

---

12. What are Flask blueprints, and how are they useful?

Blueprints allow structuring Flask applications into **modular components**, making large applications easier to manage.

Example:

python
CopyEdit

```
from flask import Blueprint

user_bp = Blueprint('user', __name__)

@user_bp.route('/profile')
def profile():
    return "User Profile Page"
```

Blueprints help in organizing routes and functionalities.

---

13. What is the purpose of Flask's request object?

Flask's `request` object provides access to **HTTP request data**, such as form inputs, JSON payloads, headers, and query parameters.

Example:

python
CopyEdit

```
from flask import request

@app.route('/data', methods=['POST'])
def get_data():
    json_data = request.json
    return f"Received: {json_data}"
```

---

14. How do you create a RESTful API endpoint using Flask?

Example:

python

CopyEdit

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/api/data', methods=['GET'])
def get_data():
    return jsonify({"message": "Hello, API!"})

if __name__ == "__main__":
    app.run(debug=True)
```

---

15. What is the purpose of Flask's `jsonify()` function?

The `jsonify()` function converts Python dictionaries into JSON responses.

Example:

python
CopyEdit

```
from flask import jsonify

@app.route('/info')
def info():
    return jsonify({"name": "Flask", "version": "1.1.2"})
```

---

16. Explain Flask's `url_for()` function.

The `url_for()` function generates URLs dynamically instead of hardcoding them.

Example:

python
CopyEdit

```
from flask import url_for

@app.route('/home')
def home():
    return "Home Page"

print(url_for('home'))  # Outputs '/home'
```

---

17. How does Flask handle static files (CSS, JavaScript, etc.)?

Flask serves static files from the **static/** directory.

Example:

html
CopyEdit

```
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

---

18. What is an API specification, and how does it help in building a Flask API?

An API specification **defines endpoints, methods, request formats, and responses**. It helps ensure **consistency** and enables **auto-generation of documentation** using tools like Swagger.

---

19. What are HTTP status codes, and why are they important in a Flask API?

HTTP status codes indicate the result of an API request.

- **200 OK** – Success
- **201 Created** – Resource created
- **400 Bad Request** – Invalid input
- **404 Not Found** – Resource not found
- **500 Internal Server Error** – Server issue

---

20. How do you handle POST requests in Flask?

Example:

python

CopyEdit

```
@app.route('/submit', methods=['POST'])
def submit():
    data = request.json
    return jsonify({"received": data})
```

---

21. How would you secure a Flask API?

- Use **JWT (JSON Web Tokens)** for authentication.
- **Rate limiting** to prevent abuse.
- Enable **CORS protection**.
- **Validate input** to prevent injection attacks.

---

22. What is the significance of the Flask-RESTful extension?

Flask-RESTful simplifies building REST APIs with built-in support for request parsing, response formatting, and error handling.

---

23. What is the role of Flask's session object?

The `session` object stores user-specific data **across requests** using cookies.

Example:

python
CopyEdit

```
from flask import session

session['username'] = 'admin'
```

4o