Flask Interview Questions With Answers

Q #1) What is Flask?

Answer: Flask is a web development framework created in Python language. This Framework is based on the robust foundation of Jinja2 templates engine and Werkzeug comprehensive WSGI web application library.

Flask is created by Armin Ronacher and is developed as a part of the Pallets Projects, which is a collection of Python web development libraries such as Flask, Click, ItsDangerous, Jinja, MarkupSafe, and Werkzeug.

Q #2) Is the Flask framework open source?

Answer: Yes, the Flask framework is open-source. The source code of the Flask framework is available here . It is released under the BSD-3 Clause “New” or “Revised” License.

Q #3) How to get the development version of the Flask framework?

Answer: The development version of the Flask framework can be obtained using the below-mentioned commands.

```
git clone https://github.com/pallets/flask
cd flask && python3 setup.py develop
```

Q #4) How to add the mailing feature in the Flask Application?

Answer: To send emails, we need to install the Flask-Mail flask extension using the below-given command.

```
pip install Flask-Mail
```

Once installed, then we need to use Flask Config API to configure MAIL-SERVER, MAIL_PORT, MAIL_USERNAME, MAIL_PASSWORD, etc. Then we need to import Message Class, instantiate it and form a message object before sending the email by using the mail.send() method.

An example is shown below.

```
 from flask_mail import Mail, Message
 from flask import Flask

 app = Flask(__name__)
 mail = Mail(app)

 @app.route(“/mail”)
 def email():
     msg = Message( “Hello Message”, sender=”admin@test.com”, recipients=[“to@test.com”])
    mail.send(msg)
```

Q #5) What is WSGI?

Answer: WSGI stands for the Web Server Gateway Interface. It is a Python standard defined in PEP 3333. WSGI is pronounced as “Whiskey.” It is a specification that describes how a web server communicates with a web application.

Q #6) Who created Flask?

Answer: Armin Ronacher created the Flask framework. Flask was born out of April fool Joke in 2011.

Q #7) Why do we use Flask?

Answer: Flask is used to create Web applications using Python programming language. Flask is a microframework that is also used for quick prototyping web and networking based applications.

Q #8) How to install Flask on Linux?

Answer: On Linux, Flask can be installed using Python's package manager, pip.

Use the below command to install Flask.

```
pip install Flask
```

Q #9) What is the default host port and port of Flask?

Answer: Flask default host is a localhost (127.0.0.1), and the default port is 5000.

Q #10) How to change default host and port in Flask?

Answer: Flask default host and port can be changed by passing the values to host and port parameters while calling run method on the app.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
    return "Hello, World!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Q #11) Which Flask extension can be used to create an Ajax application?

Answer: We can use Flask-Sijax to create an Ajax application. Flask-Sijax is an extension that uses Python/JQuery. It is available on PyPI and can be installed using pip.

Sijax stands for Simple Ajax. Once configured and initialized, it enables the use of @flask_sijax decorator, which we can use for making Ajax aware of the views in a Flask Application.

Q #12) How to use the Flask commands?

Answer: As a result of the Flask installation, we also get access to a command-line application called Flask. There are various commands that we can use.

Use Flask –help on the command line to see all the options. Default commands are routes, run, and shell. This utility provides commands from Flask, extensions, and the application.

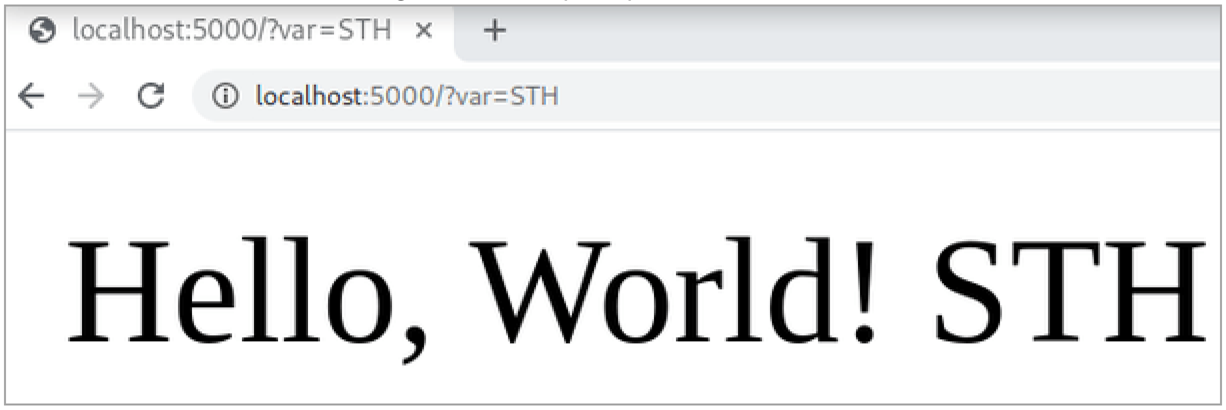
Q #13) How to get query String in Flask?

Answer: We can get the argument's value using the request object in Flask.

An example is shown below.

```
from flask import Flask
from flask import request

app = Flask(__name__)

@app.route("/")
def index():
val = request.args.get("var")

return "Hello, World! {}".format(val)

if __name__=="__main__":
app.run(host="0.0.0.0", port=8080)
```

When we use the browser to navigate with a request parameter, then we see the below result.



Hello, World! STH

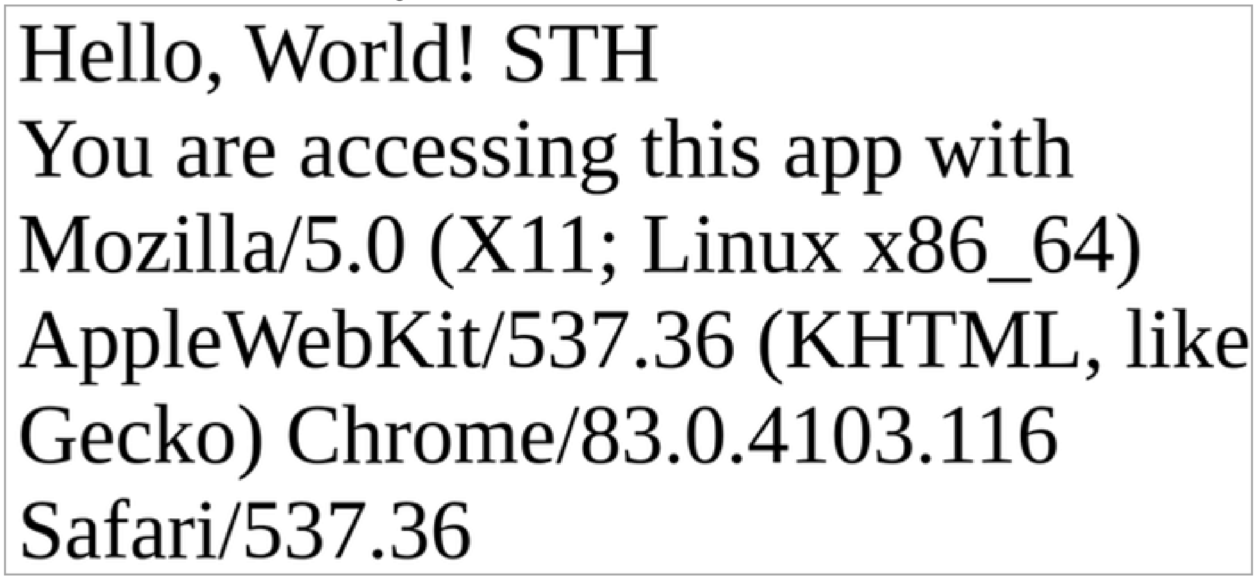
Q #14) How to get the user agent in Flask?

Answer: We can use the request object to get the User-Agent in Flask.

Use the below-mentioned code for the same.

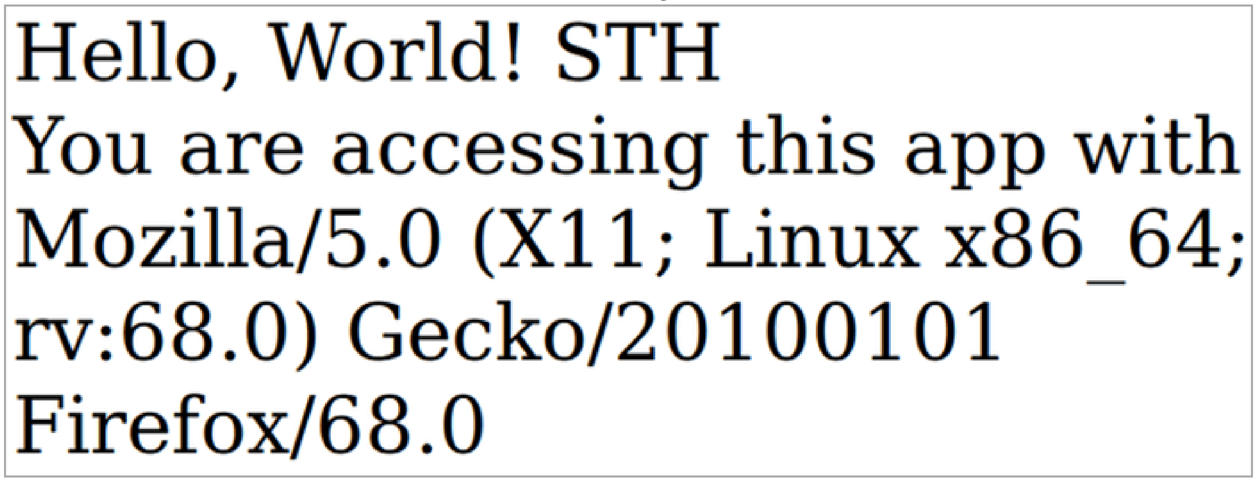
```
from flask import Flask
from flask import request

app = Flask(__name__)

@app.route("/")
def index():
    val = request.args.get("var")
    user_agent = request.headers.get('User-Agent')

    response = """
    &lt;p&gt;
    Hello, World! {}
    &lt;br/&gt;
    You are accessing this app with {}
    &lt;/p&gt;
    """.format(val, user_agent)
return response
if __name__=="__main__":
    app.run(host="0.0.0.0", port=8080)
```

Once you run this code and navigate to the required URL using the Chrome browser, you will see the result, as shown in the below image.



```
Hello, World! STH
You are accessing this app with
Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/83.0.4103.116
Safari/537.36
```

The result in Firefox will look as shown in the below image.



```
Hello, World! STH
You are accessing this app with
Mozilla/5.0 (X11; Linux x86_64;
rv:68.0) Gecko/20100101
Firefox/68.0
```

Q #15) How to use url_for in the Flask application?

Answer: Flask's url_for function helps in creating dynamic routes. We can make use of url_for in the Flask templates. We can call the view function with parameters and values to generate URLs.

For example, pass a function and its arguments, as shown below.

```
<a href="{{ url_for('get_post_id', post_id=post.id}}">{{post.title}}<a>
```

View function for handling variables in routes.

```
@app.route("/blog/post/<string:post_id>")
def get_post_id(post_id):
return post_id
```

Q #16) How to create an Admin interface in Flask?
Answer: We can create an Admin interface in Flask using the Flask-Admin extension. It helps in grouping individual views together in classes. We can use the Flask-Appbuilder extension too. Flask-Appbuilder already comes with an Admin interface.
Q #17) How to integrate Twitter or Similar API with the Flask Application?
Answer: To integrate with Flask, we can make use of a Flask extension called Flask-Social. It not only helps in authenticating users from Twitter but also other social platforms or accounts such as Facebook and Google. We need to use Flask-Social along with Flask-Security.
We need to install individual API libraries in Python and also need to get consumer and secret keys by registering the Flask application on the external account providers.
Q #18) Why is Flask called a Microframework?
Answer: Flask is called a microframework because Flask only provides core features such as request, routing, and blueprints. For other features, such as Caching, ORM, forms, etc., we need to make use of Flask-Extensions.
Q #19) What are the benefits of using the Flask framework?
Answer: Some benefits of using the Flask framework are:

- It has an inbuilt development server.
- It has vast third-party extensions.
- It has a tiny API and can be quickly learned by a web developer.
- It is WSGI compliant.
- It supports Unicode.

Q #20) Is the SQLite database built-in Flask?
Answer: SQLite is in-built with Python. To use the database in Flask, we need not install any additional Flask-Extension. Inside the view, we can import SQLite and write SQL queries for interacting with the database.
However, Flask developers generally make use of Flask-SQLAlchemy, which eliminates the need to write complex SQL queries and is an ORM to interact with the SQLite database.
Q #21) What do you mean by template engines in the Flask framework?
Answer: A template is a file that contains two types of data, i.e., static and dynamic. Dynamic data in a template is populated during run time. Flask makes use of Jinja2 template engine to let developers create HTML templates with placeholders for dynamic data.
These placeholders can be filled during run time by using Flask's render_template method with required parameters and values.
Q #22) What do you mean by Thread local object in Flask?
Answer: In Flask, thread-safety has been provided out of the box. We can use objects such as current_app, g, and request without worrying about problems related to locking and concurrency.
Moreover, we need not pass objects from methods to methods, and these objects are generally available within a valid request context.
This attribute of Flask makes it a bit unique and provides a lot of convenience to the Flask developers while keeping Flask application thread-safe.
Q #23) What is the difference between Django and Flask? Why should one choose Flask?
Answer: Django is also a web development framework created in the Python programming language. It is a full-featured web application framework with a lot of features that are built into it, such as an Admin backend, and an ORM with migration capability. It is a little bit older and more mature.
Flask is better for quick development use cases and is perfect for prototyping. Django has inspired even some Flask extensions that are written. Flask is more suitable for developing lightweight web applications that do not require a large codebase. It is apt for developing microservices or serverless applications.
Flask is easy to learn and has fewer API's when compared to Django. As the industry is following the trends towards microservices served as part of containers, it is excellent to keep Flask in your web development toolkit.

Q #24) Describe the features of Forms extension for Flask.

Answer: Forms in Flask can be implemented by using an extension called Flask-WTF. Flask-WTF is created by integrating Flask with WTForms. WTForms is a python-based form rendering and validation library. It supports data validation, internationalization, and CSRF protection.
Flask-WTF also provides reCAPTCHA support along with file uploads when tied with Flask-Uploads. You also can handle JavaScript requests, and customize the error response.

Q #25) How to use a session in Flask?

Answer: Whenever we want to save some data between requests, then we make use of session objects in Flask. We can set and get data from the session object, as shown below.

```
fromflask import Flask, session

app = Flask(__name__)

@app.route('/use_session')
def use_session()
    if 'song' not in session:
        session['songs'] = {'title': 'Tapestry', 'singer': 'Bruno Major'}

    return session.get('songs')

@app.route('/delete_session')
def delete_session()
    session.pop('song', None)
    return "removed song from session"
```

Q #26) What is the g object in Flask? How does it differ from the session object?

Answer: Flask's g object is used as a global namespace for holding any data during the application context. g object is not appropriate for storing the data between requests. The letter g, in a sense, stands for global.
In situations, when you need to keep global variables during an application context, then rather than creating your global variable, it is best to use the g object as each request in Flask has a separate g object. Flask's g object saves us from accidental modifications of self-defined global variables.

Q #27) What is the application context in Flask?

Answer: The application context in Flask relates to the idea of a complete request/response cycle. It keeps a track of application-level data during a request or a CLI command. We make use of g and current_app proxies to achieve the same.
There are situations when it is difficult to directly import the Flask app, such as in the case of a Flask extension or a Blueprint. Moreover, introducing applications may raise the problem of circular imports. Flask pushes the application context with each request. Therefore, during a request, functions have access to g and current_app to overcome the problem highlighted above.

Q #28) In what ways can you connect to a database in Flask?

Answer: Flask works with most of the RDBMSs, such as PostgreSQL, SQLite, and MySQL. However, to connect with databases, we must make use of the Flask-SQLAlchemy extension.
It makes database interaction and management easy during development without the need to write raw SQL queries. Moreover, raw SQL queries are prone to SQL injection attacks. For working with No-SQL data stores such as MongoDB, we can make use of the Flask-MongoEngine extension.

Q #29) How to create a RESTful application in Flask?

Answer: A RESTful application can be created in Flask with the help of many extensions.

Some proven Flask extensions are listed below.

1. Flask-API
2. Flask-RESTful
3. Flask-RESTX
4. Connexion

However, we need to evaluate these extensions and see which one is more appropriate based on our project requirements and constraints.

Q #30) How to debug a Flask Application?

Answer: Flask comes with a development server, and the development server has a Debug Mode. The Debug mode can be set to true when we call the run method of the Flask Application object.

<u>Given below is an example.</u>

```
from flask import Flask
app = Flask(__name__)
app.run(host='127.0.0.1', debug=True)
```

However, we need to disable the debug mode before deploying the application on production to avoid full stack trace display in the browser. Such a stack trace can reveal a lot of essential details and is prone to exploitation by bad actors.

Further, we can make use of the Flask-DebugToolbar extension for easy debugging in the browser. We can also make use of Python's pdb module and the debugging statement import pdb;pdb.set_trace() to support the debugging process.

Q #31) What type of Applications can we create with Flask?

Answer: With Flask, we can create almost all types of web applications. We can create Single Page Application, RESTful API based Applications, SAS applications, Small to medium size websites, static websites, Microservices, and serverless apps.

Flask is so versatile and flexible as it can be integrated with other technologies very quickly to achieve the same. **<u>For example,</u>** Flask can be combined with the NodeJS serverless, AWS lambda, and similar other third party services to build new-age systems.