Python final list of questions

**🧾 Restaurant Management System – Viva Questions & Answers**

**📌 Flask & app.py**

**Q1: What is Flask?**  
A micro web framework in Python used for developing web applications. It's lightweight and ideal for small to medium projects.

**Q2: How many routes are defined in your Flask app?**  
There are 7 main routes: /, /order, /review, /admin, /admin/add\_item, /admin/update, /admin/delete.

**Q3: What happens at the / route?**  
It displays the homepage with menu items (buffet by default), the a la carte menu, and customer reviews.

**Q4: How is form data retrieved in Flask?**  
Using request.form['fieldname'] or safely with request.form.get('fieldname').

**Q5: Why is redirect(url\_for(...)) used?**  
To redirect users to another route after an action like placing an order or updating a menu item.

**⚙️ restaurant.py (Backend Classes)**

**Q6: What is the role of restaurant.py?**  
It contains backend logic, managing the database connection, menu, orders, and reviews through separate classes.

**Q7: Explain the DatabaseConnector class.**  
It connects to MySQL and creates necessary tables if they don’t exist.

**Q8: What does MenuManager do?**  
Manages fetching, adding, updating, and deleting menu items from buffet\_menu and a\_la\_carte\_menu.

**Q9: What does OrderManager do?**  
Places orders by calculating the total cost and inserting order data into the orders table.

**Q10: What is the purpose of ReviewManager?**  
Handles inserting and retrieving customer reviews from the reviews table.

**🌐 index.html (Frontend for Users)**

**Q11: What is index.html?**  
It’s the main user interface. Users can view menus, order items, and submit reviews.

**Q12: How does the menu display work?**  
Menu items are looped and inserted dynamically using Jinja templating with {{ item[1] }} etc.

**Q13: How are orders placed from the UI?**  
Each item has a form with quantity input and a submit button posting to /order.

**Q14: Where is the admin panel link?**  
At the bottom of the page — <a href="{{ url\_for('admin') }}">Go to Admin Panel</a>.

**🛠️ admin.html (Admin Panel)**

**Q15: What is admin.html used for?**  
To allow the admin to securely add, update, or delete menu items.

**Q16: How does password verification work for admin?**  
A password is entered and validated in app.py using regex.

**Q17: What regex is used?**  
^(?=.\*[A-Z])(?=.\*[a-z])(?=.\*\d).{8,}$ – Ensures 1 uppercase, 1 lowercase, 1 number, minimum 8 characters.

**Q18: What admin functionalities are available?**  
Add new menu item, update item price, delete item from menu.

**Q19: How are form submissions handled?**  
Each form submits via POST to its dedicated route: /admin/add\_item, /admin/update, /admin/delete.

**🔁 General Python, OOP, DB, and Regex Concepts**

**Q20: What is a class in Python?**  
A blueprint for creating objects that encapsulate data and functions together.

**Q21: What is an object?**  
An instance of a class with access to its methods and attributes.

**Q22: What is a method?**  
A function defined inside a class, operating on object data.

**Q23: What is polymorphism?**  
The ability to define the same method name in different classes with different behaviors.

**Q24: What is inheritance?**  
It allows a class (child) to inherit properties and methods from another class (parent).

**Q25: What is abstraction?**  
Hiding complex logic inside methods and exposing only necessary interfaces.

**Q26: What is encapsulation?**  
Keeping data and methods bundled together and limiting external access.

**Q27: What are exceptions?**  
Errors during runtime that can be caught and handled to avoid crashes.

**Q28: What is exception handling?**  
Using try, except, finally blocks to manage errors gracefully.

**Q29: What is a custom exception?**  
A user-defined exception created by subclassing the Exception class.

**Q30: What is a database?**  
An organized collection of data, managed by a DBMS like MySQL.

**Q31: What is MySQL?**  
A relational database management system used for structured data.

**Q32: What is a regular expression?**  
A pattern used to match or validate strings.

**Q33: How did you use regex?**  
To validate the admin password for security using a regex pattern.

**Q34: What are CRUD operations?**  
Create, Read, Update, Delete — basic database functions.

**Q35: What is Jinja2 templating?**  
Flask’s templating engine for rendering HTML with dynamic data.

**🔣 Inheritance and OOP in This Project**

**Q36: What type of inheritance is used?**  
Single inheritance. Each class stands alone, without multi-level or multiple inheritance.

**Q37: Is inheritance mandatory?**  
No. Composition and modularity can also be used depending on the design.

**Q38: Why didn’t you use multiple inheritance?**  
To avoid complexity. Each class has a clear and separate responsibility.

**Q39: Explain constructor chaining.**  
Calling a parent’s constructor using super(). Not used here, but possible in Python.

**🎨 HTML Styling and UI Elements**

**Q40: What fonts did you use?**  
Arial, sans-serif for clean and modern appearance.

**Q41: What background colors were used?**

* #EBC2D5: Light pink (main background)
* #A42153: Dark pink (headers, buttons)
* #C5558E: Mid-pink (highlighted sections)
* #62021E: Dark red (text)

**Q42: Why use inline styles instead of external CSS?**  
For simplicity in a small project. In larger apps, external CSS is better.

**Q43: How are reviews and orders displayed?**  
Using HTML <table> tags with styled headers and rows.

**Q44: How did you ensure responsiveness?**  
With the <meta name="viewport"> tag and fluid-width elements.

**Q45: What form elements were used?**  
<input>, <select>, <textarea>, <button> for different fields.

**Q46: How is navigation between pages done?**  
Using <a href="{{ url\_for('...') }}"> to link between Flask routes.

Based on your project and presentation content, here’s a **script** you can use to confidently explain the **Buffet and A-La-Carte Menu** part of your Restaurant Management System during your presentation:

### 🎤 **Script: Explaining Buffet & A-La-Carte Menu (3–4 mins)**

"Good [morning/afternoon], everyone. I’ll now walk you through one of the core features of our Restaurant Management System — the Buffet and A-La-Carte menu management.

### 🍽️ Buffet and A-La-Carte – What’s the Difference?

In our system, we support **two types of menu styles**:

1. **Buffet Menu** – This is a pre-set meal option where dishes are grouped under sections like Starters, Main Course, Desserts, etc.
2. **A-La-Carte Menu** – Here, customers can individually pick and choose items from the menu and pay per dish.

We wanted to implement both because many real restaurants use this combination – buffet for dine-in and a-la-carte for delivery or small orders.

### 💻 How It's Built in the System

From a technical point of view, these menus are stored in two different **MySQL tables**:

* buffet\_menu → Stores item name, section, and price.
* a\_la\_carte\_menu → Stores just the item name and price.

### 👨‍🍳 On the Admin Side

In our **Admin Panel** (admin.html), the admin can:

* **Add**, **Update**, or **Delete** items from either menu.
* Select the category — Buffet or A-La-Carte.
* For Buffet items, an additional section like “Main Course” or “Dessert” is also added.
* We’ve even added **password protection with regex validation** to secure admin operations.

### 📄 Code-wise

In our app.py file:

* The route /admin/add\_item handles admin menu additions.
* If the category selected is **buffet**, we insert into the buffet\_menu table including its section.
* For **a-la-carte**, we only need the item name and price.

We’ve handled exceptions like invalid price formats and unauthorized access, and used regular expressions to validate strong admin passwords.

### 🖥️ On the User Side

On the **homepage** (index.html), both menus are visible:

* The **Buffet Menu** is displayed with Category, Section, and Item.
* The **A-La-Carte Menu** is shown in a separate section below with item names and prices, where users can choose quantity and place an order directly.

Both menus are dynamically fetched from the database using the MenuManager class in restaurant.py.

### 🧠 OOP Concepts Applied

We used **Encapsulation** in the MenuManager class to separate logic for fetching buffet and a-la-carte menus.

* We used **Polymorphism** while updating or deleting items, as both menu types are handled with the same method signature but different logic internally.

### ✅ Why This Matters

This dual menu system ensures flexibility:

* Buffet for fixed price meals.
* A-La-Carte for personalized orders.

It reflects how real restaurants operate and improves customer experience by offering choice and clarity.

### 🙏 Closing

That’s how we structured and managed the **Buffet and A-La-Carte menus** in our Restaurant Management System. Thank you!"

Would you like me to help you make flashcards, a quick recap sheet, or even rehearse this with you in a Q&A style?