1. **What is Flask, and how does it differ from other web frameworks?**
   Flask is a micro web framework for Python that is designed to be lightweight, flexible, and easy to use. Unlike other full-stack frameworks like Django, Flask provides just the essentials for building web applications, allowing developers to add only the components they need. This minimalist approach gives developers more control over their applications' architecture and dependencies, making Flask ideal for small to medium-sized projects or when specific requirements call for a lightweight solution. Flask follows the WSGI (Web Server Gateway Interface) specification, making it compatible with a wide range of web servers and deployment options.

2. **Describe the basic structure of a Flask application.**
   A basic Flask application typically consists of a Python script serving as the main application file, a templates directory for HTML templates, and a static directory for storing static files like CSS, JavaScript, and images. The main application file defines routes, which are URL endpoints that map to specific functions in the application. These functions handle HTTP requests and return responses, often by rendering templates or returning JSON data.

3. **How do you install Flask and set up a Flask project?**
   To install Flask, you can use pip, Python's package manager, by running **pip install Flask**. Once Flask is installed, you can set up a Flask project by creating a directory for your project, creating a virtual environment to isolate dependencies, installing Flask within the virtual environment, and then creating the necessary files and directories for your application structure.

4. **Explain the concept of routing in Flask and how it maps URLs to Python functions.**
   Routing in Flask refers to the process of associating URLs with specific functions in the application. This is achieved using the **@app.route()** decorator, where **@app** refers to the Flask application instance, and **route()** specifies the URL pattern for the associated function. When a request is made to a URL matching the specified pattern, Flask invokes the corresponding function, passing any parameters extracted from the URL.

5. **What is a template in Flask, and how is it used to generate dynamic HTML content?**
   A template in Flask is an HTML file that contains placeholders for dynamic content. These placeholders are typically filled in with data from the application when the template is rendered. Flask uses the Jinja template engine to parse templates and substitute variables, expressions, and control structures with actual values, resulting in dynamic HTML content tailored to each request.

6. **Describe how to pass variables from Flask routes to templates for rendering.**
   Variables can be passed from Flask routes to templates using the **render_template()** function, which is provided by Flask. This function takes the name of the template file and any additional variables to be passed to the template as keyword arguments. Inside the template, these variables can be accessed using Jinja syntax, such as **{{ variable_name }}**, allowing for dynamic content rendering.

7. **How do you retrieve form data submitted by users in a Flask application?** Form data submitted by users in a Flask application can be retrieved using the **request** object, which is provided by Flask and represents the current HTTP request. Form fields are accessed through the **request.form** dictionary, where the keys are the names of the form fields and the values are the submitted data.

8. **What are Jinja templates, and what advantages do they offer over traditional HTML?**
   Jinja templates are a templating engine used by Flask for generating dynamic content in HTML files. Jinja templates allow for the inclusion of variables, expressions, control structures, and template inheritance, making it easier to create dynamic and reusable HTML content. Compared to traditional HTML, Jinja templates offer the advantage of dynamic content generation, improved code organization, and better separation of concerns between presentation and application logic.

9. **Explain the process of fetching values from templates in Flask and performing arithmetic calculations.**
   Values from templates in Flask can be fetched using Jinja syntax, such as **{{ variable_name }}**, where **variable_name** is the name of the variable passed from the Flask route. These values can then be used in arithmetic calculations directly within the template using Jinja's expression syntax, which supports basic arithmetic operators like **+**, **-**, *****, and /.

10. **Discuss some best practices for organizing and structuring a Flask project to maintain scalability and readability.**
    Best practices for organizing and structuring a Flask project include modularizing the application using blueprints to separate concerns, structuring the project directory with clear naming conventions and logical organization, using separate files for routes, models, forms, and templates, employing Flask extensions for common functionalities like authentication and database access, and adhering to design patterns like MVC (Model-View-Controller) to maintain scalability, readability, and code maintainability as the project grows. Additionally, documenting the code and following PEP 8 style guidelines can enhance collaboration and code quality.