Crack Python interviews

part 2

Author
Arun Arunisto

About the author

Arun Arunisto is a Python developer with over 3 years of experience in the industry. He is a computer enthusiast who learned programming on his own and has since become an expert in Python programming. Arun is passionate about teaching and has helped numerous students to learn Python programming.

Arun's vision is to simplify the learning process of Python programming for everyone. He believes that programming is for everyone and is committed to making programming accessible to all. His teaching style is unique, and he strives to make his classes fun and engaging.

What is django?

- Django is a high-level Python web framework
- Django follows the Model-View-Template (MVT) architectural pattern.
- Provides a robust set of tools and features for building web applications efficiently.

Advantages of django

- **Rapid development**: Django provides many built-in features and libraries that help in faster application development.
- **Scalability**: Django's modular design allows easy scaling of applications.
- **Security**: Django provides several built-in security features to protect against common web application vulnerabilities.
- **ORM (Object-Relational Mapping)**: Django's ORM allows developers to interact with the database using Python objects, making database operations easier and more intuitive.
- **Community and documentation**: Django has a large and active community, which means you can find plenty of resources and support.

django pattern?

- Model-View-Template (MVT): Django follows a variation of the MVC pattern, known as Model-View-Template (MVT).
- Model represents the data structure and business logic.
- View handles the presentation and user interface.
- Template defines the structure and layout of the rendered HTML pages.

Difference between django and flask?

- Django is a full-featured framework, Flask is a microframework.
- Django provides many built-in features and follows a specific project structure
- Flask is more flexible and requires developers to choose the libraries and components they want to use.
- Django includes an ORM and follows the MVT pattern
- Flask does not enforce any specific structure or database layer.
- Django is better suited for larger and complex applications
- Flask is often preferred for smaller and simpler projects.

django ORM?

- Django's ORM (Object-Relational Mapping) is a technique that allows developers to interact with the database using Python objects.
- It eliminates the need to write raw SQL queries and provides a high-level abstraction for performing database operations.
- With the ORM, you can define your data models as Python classes, and Django takes care of mapping those models to database tables and handling the database operations.

django Models?

- Django models are Python classes that represent database tables.
- They define the structure of the data and include fields for storing different types of information.
- Models define relationships between different tables, enforce constraints, and provide methods for querying and manipulating the data.

django Revoke

- In Django, the term "revoke" typically refers to the action of invalidating or canceling an authentication token or session to terminate a user's access or log them out.
- When a token or session is revoked, it becomes invalid, and subsequent attempts to use it for authentication will fail.

django database connection?

- Django supports multiple database backends, such as PostgreSQL, MySQL, SQLite, Oracle, and more. The configuration for the database connection is specified in the Django project's settings file (settings.py).

django adding new column?

1. Modify the model - Open the file containing your Django model (typically located in the models.py file of your app) and add the new field to the corresponding model class. Specify the field type and any additional parameters as needed.
2. Generate a migration - python manage.py makemigrations
3. Apply the migration - python manage.py migrate

django middleware?

- Django middleware is a set of components that process requests and responses.
- Middleware sits between the web server and Django's view system, allowing you to perform operations before and after the view execution.
- It can handle tasks like authentication, session management, caching, and modifying the request/response objects.

django class based views?

- Class-based views are an alternative way to define views in Django using Python classes.
- They provide reusable and structured code by separating different HTTP methods (GET, POST, etc.) into separate methods within the class.
- Class-based views offer more flexibility and code reusability compared to function-based views.

django session framework?

- Django's session framework enables you to store data across multiple requests for a particular user.
- It uses cookies or other server-side storage options to store and retrieve session data. Session data is encrypted and signed to maintain security.
- The session framework is often used to implement user authentication and track user-specific information across requests.

django rest framework?

- Django REST Framework (DRF) is a powerful toolkit for building Web APIs in Django.
- It provides a set of tools and conventions that make it easy to build robust APIs, handle authentication and authorization, serialize and deserialize data, and handle common API patterns like CRUD operations.
- DRF simplifies the process of building APIs, provides extensive documentation, and has a large and active community.

Serializers in django rest?

- Serializers in DRF are used to convert complex data types (such as models or querysets) into Python data types that can be easily rendered into JSON or other content types for API responses.
- They also handle deserialization, allowing parsed data to be converted back into complex types.
- Serializers provide a way to specify which fields and relations should be included in the serialized output and can handle validating input data.

Class based view in django rest?

- In DRF, class-based views (CBVs) are an alternative to function-based views for handling API endpoints.
- CBVs provide a structured way to define views as classes, where each HTTP method (GET, POST, PUT, DELETE, etc.) corresponds to a method within the class.
- CBVs offer code reuse and provide a consistent way to handle different HTTP methods, authentication, permissions, and other common API functionality.