The distinction between frameworks and libraries is an important one in software development. Both frameworks and libraries provide reusable code to help developers build applications, but they differ in terms of their control over the development process and how they are used.

Key Differences:

1. **Inversion of Control**:
   - **Framework**: In a framework, the control flow of the application is dictated by the framework itself. The framework calls your code and controls when and how certain parts of the application are executed. This is often referred to as the "Hollywood Principle" – "Don't call us, we'll call you."
   - **Library**: When using a library, you are in control of the application flow. You call the library’s functions as needed. The library provides a set of functions or methods that you can invoke.

2. **Structure and Guidelines**:
   - **Framework**: Frameworks often come with a predefined structure and set of guidelines for building applications. They dictate how you should organize your code and typically come with built-in tools and features for common tasks (e.g., routing, state management, templating).
   - **Library**: Libraries are more flexible and do not impose a specific structure on your application. They provide specific functionality that you can integrate into your application as you see fit.

3. **Scope and Functionality**:
   - **Framework**: Frameworks are usually more comprehensive and cover a wide range of functionality needed for building applications. They often include features for handling HTTP requests, middleware, templating, and database integration.
   - **Library**: Libraries tend to be more focused and provide specific functionality (e.g., manipulating the DOM, making HTTP requests, handling dates).

4. **Examples**:
   - **Frameworks**:
     - **JavaScript**: Angular, Vue.js, React (often considered a library but functions as a framework when used with its ecosystem like React Router and Redux), Ember.js, Svelte.
     - **Backend**: Express.js, NestJS, Django (Python), Ruby on Rails (Ruby), Spring (Java).
   - **Libraries**:
     - **JavaScript**: jQuery, Lodash, Moment.js, Axios, D3.js.
     - **Utility Libraries**: NumPy (Python), Pandas (Python), Requests (Python).

Examples in Code:

Using a Library:
When using a library, you control when and how the library functions are called. For example, using Lodash to manipulate arrays:

```javascript
const _ = require('lodash');

const numbers = [1, 2, 3, 4, 5];
const evens = _.filter(numbers, n => n % 2 === 0);

console.log(evens); // Output: [2, 4]
```

Using a Framework:
When using a framework, the framework often sets up the structure of your application and controls the flow. For example, using Express.js to create a web

server:

```javascript
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

In this Express.js example, the framework dictates the structure (routes, middleware, etc.) and calls your code (route handlers) when appropriate.

Summary:

- **Control**: Libraries provide functions that you call, whereas frameworks call your code within their predefined structure.
- **Structure**: Frameworks impose a structure on your code, while libraries are more flexible.
- **Scope**: Frameworks are comprehensive and cover a wide range of tasks; libraries focus on specific functionalities.
- **Examples**: Frameworks include Angular, Vue.js, and Express.js, while libraries include jQuery, Lodash, and Moment.js.

Understanding these differences helps developers choose the right tools for their projects and better manage the development process.