Steps that involved to write Test Script

To write an Appium test in JavaScript (Node.js), we need to choose an Appium-compatible client library. The best-maintained library and the one the Appium team recommends using is [WebdriverIO](https://webdriver.io/).

Step 1: Create a new project directory somewhere on your computer and then initialize a new Node.js project in it:

Step 2: Create a package.json file by running this command (npm init –y).

Step 3: Install the webdriverio package via NPM.

1. Command to install: (npm install --save-dev webdriverio).
2. Command to install globally: (npm list -g webdriverio).
3. Command to Verify: (npm list webdriverio).
4. Command to Update latest: (npm install --save-dev webdriverio@latest).
5. Command to Update Specific: (npm install --save-dev webdriverio@<version>).
6. Command to Uninstall: (npm uninstall --save-dev webdriverio).

Step 4: Create a .js file to write your test script.

Step to run the script to automation testing your app

Step 1: Run the Appium Server with this command (Appium).

Note: Before you run the test, make sure that you have an Appium server running in another terminal session, otherwise you'll get an error about not being able to connect to one.

Step 2: Execute the test script run this command (test\_script.js).

What is package.json?

package.json is a file used by Node.js projects that contains metadata about the project, including its name, version, dependencies, scripts, and other configurations necessary for running and managing the application.

Purpose of this file are:

1. **Project Metadata**: It contains essential information about the project, including its name, version, description, and author.
2. **Dependency Management**: It lists the external packages that your project depends on. This includes both runtime dependencies (libraries needed to run your application) and development dependencies (libraries needed for development and testing but not in production).
3. **Script Definitions**: You can define custom scripts to run various tasks, such as starting the server, running tests, or building the project, making it easier to manage these tasks.
4. **Configuration Settings**: It can also include configuration settings for various tools used in the project (e.g., ESLint, Babel), allowing those tools to read their configuration from the package.json file.
5. **License Information**: It often includes licensing information to inform users or contributors about the terms under which your project is shared.