1.Explain redux to a 5 year old (ELI5) ?  
A. Let’s consider an event you’re likely familiar with — going to the bank to withdraw cash. Even if you don’t do this often, you’re likely aware of what the process looks like.  
You wake up one morning, and head to the bank as quickly as possible. While going to the bank there’s just one intention / action you’ve got in mind: to WITHDRAW\_MONEY.  
When you get into the bank, you then go straight to the Cashier to make your request known.  
Wait, you went to the Cashier?  
Why didn’t you just go into the bank vault to get your money?After all, it’s your hard earned money  
Well, like you already know, things don’t work that way. Yes, the bank has money in the vault, but you have to talk to the Cashier to help you follow a due process for withdrawing your own money.  
The Cashier, from their computer, then enters some commands and delivers your cash to you. Easy-peasy  
Now, how does Redux fit into this story?  
  
 The Bank Vault is to the bank what the Redux Store is to Redux.  
Well, within your application, you don’t spend money. Instead, the state of your application is like the money you spend. The entire user interface of your application is a function of your state.

Just like the bank vault keeps your money safe in the bank, the state of your application is kept safe by something called a store. So, the store keeps your “money” or state intact  
The Redux Store can be likened to the Bank Vault. It holds the state of your application — and keeps it safe.  
This leads to the first Redux principle #1:  
Have a single source of truth: The state of your whole application is stored in an object tree within a single Redux store.  
In simple terms, with Redux, it is is advisable to store your application state in a single object managed by the Redux store. It’s like having one vaultas opposed to littering money everywhere along the bank hall  
In the same way you follow a due process to withdraw your own money from the bank, Redux also accounts for a due process to change/update the state of your application.  
Now, this leads to Redux principle #2.  
State is read-only  
In Redux terms, the money you spend is your state. So, your reducer knows what to do, and it always returns your new state.  
And this leads to the last Redux principle #3:  
To specify how the state tree is transformed by actions, you write pure reducers.

2. What is immutability?  
A.*Immutability is the property whose state cannot be changed once it is created.  
For example all the primitive data-types are immutable i.e; its value cannot be changed once they are created. Where as for non-primitive data-types like object and array , their value and property can be modified. They are mutable. There are few inbuilt functions which helps us to make objects and array immutable.*

*3.What does Object.freeze() do?  
A.The Object.freeze() method freezes an object. A frozen object can no longer be changed; freezing an object prevents new properties from being added to it, existing properties from being removed, prevents changing the enumerability, configurability, or writability of existing properties, and prevents the values of existing properties from being changed. In addition, freezing an object also prevents its prototype from being changed. freeze() returns the same object that was passed in.  
Syntax : Object.freeze(obj)*

*4.****Why is immutability required by Redux?  
A. Both Redux and React-Redux employ*** [***shallow equality checking***](https://redux.js.org/faq/immutable-data#how-do-shallow-and-deep-equality-checking-differ)***. In particular:***

* + ***Redux's combineReducers utility*** [***shallowly checks for reference changes***](https://redux.js.org/faq/immutable-data#how-does-redux-use-shallow-equality-checking) ***caused by the reducers that it calls.***
  + ***React-Redux's connect method generates components that*** [***shallowly check reference changes to the root state***](https://redux.js.org/faq/immutable-data#how-does-react-redux-use-shallow-equality-checking)***, and the return values from the mapStateToProps function to see if the wrapped components actually need to re-render. Such*** [***shallow checking requires immutability***](https://redux.js.org/faq/immutable-data#why-will-shallow-equality-checking-not-work-with-mutable-objects) ***to function correctly.***
* ***Immutable data management ultimately makes data handling safer.***
* ***Time-travel debugging requires that reducers be pure functions with no side effects, so that you can correctly jump between different states.***

***5.How does Redux use shallow equality checking?  
A. Redux's use of shallow equality checking requires immutability if any connected components are to be updated correctly.   
Shallow equality checking (or reference equality) simply checks that two different variables reference the same object.  
Redux uses shallow equality checking in its combineReducers function to return either a new mutated copy of the root state object, or, if no mutations have been made, the current root state object.  
How does combineReducers use shallow equality checking?***[***​***](https://redux.js.org/faq/immutable-data#how-does-combinereducers-use-shallow-equality-checking)

***The*** [***suggested structure***](https://redux.js.org/faq/reducers#reducers-share-state) ***for a Redux store is to split the state object into multiple "slices" or "domains" by key, and provide a separate reducer function to manage each individual data slice.***

***combineReducers makes working with this style of structure easier by taking a reducers argument that’s defined as a hash table comprising a set of key/value pairs, where each key is the name of a state slice, and the corresponding value is the reducer function that will act on it.***

***So, for example, if your state shape is { todos, counter }, the call to combineReducers would be:***

***combineReducers({ todos: myTodosReducer, counter: myCounterReducer })***

***where:***

* ***the keys todos and counter each refer to a separate state slice;***
* ***the values myTodosReducer and myCounterReducer are reducer functions, with each acting on the state slice identified by the respective key.***

***combineReducers iterates through each of these key/value pairs. For each iteration, it:***

* ***creates a reference to the current state slice referred to by each key;***
* ***calls the appropriate reducer and passes it the slice;***
* ***creates a reference to the possibly-mutated state slice that's returned by the reducer.***

***As it continues through the iterations, combineReducers will construct a new state object with the state slices returned from each reducer. This new state object may or may not be different from the current state object. It is here that combineReducers uses shallow equality checking to determine whether the state has changed.***

***Specifically, at each stage of the iteration, combineReducers performs a shallow equality check on the current state slice and the state slice returned from the reducer. If the reducer returns a new object, the shallow equality check will fail, and combineReducers will set a hasChanged flag to true.***

***After the iterations have completed, combineReducers will check the state of the hasChanged flag. If it’s true, the newly-constructed state object will be returned. If it’s false, the current state object is returned.***

***6. How well does Redux “scale” in terms of performance and architecture?  
A. While there's no single definitive answer to this, most of the time this should not be a concern in either case.***

***The work done by Redux generally falls into a few areas: processing actions in middleware and reducers (including object duplication for immutable updates), notifying subscribers after actions are dispatched, and updating UI components based on the state changes. While it's certainly possible for each of these to become a performance concern in sufficiently complex situations, there's nothing inherently slow or inefficient about how Redux is implemented. In fact, React Redux in particular is heavily optimized to cut down on unnecessary re-renders, and React-Redux v5 shows noticeable improvements over earlier versions.***

***Redux may not be as efficient out of the box when compared to other libraries. For maximum rendering performance in a React application, state should be stored in a normalized shape, many individual components should be connected to the store instead of just a few, and connected list components should pass item IDs to their connected child list items (allowing the list items to look up their own data by ID). This minimizes the overall amount of rendering to be done. Use of memoized selector functions is also an important performance consideration.***

***As for architecture, anecdotal evidence is that Redux works well for varying project and team sizes. Redux is currently used by hundreds of companies and thousands of developers, with several hundred thousand monthly installations from NPM. One developer reported:***

***7. How does Redux compare to the React Context API?  
A.***

| ***Context API*** | ***Redux*** |
| --- | --- |
| ***Built-in tool that ships with React*** | ***Additional installation Required, driving up the final bundle size*** |
| ***Requires minimal Setup*** | ***Requires extensive setup to integrate it with a React Application*** |
| ***Specifically designed for static data, that is not often refreshed or updated*** | ***Works like a charm with both static and dynamic data*** |
| ***Adding new contexts requires creation from scratch*** | ***Easily extendible due to the ease of adding new data/actions after the initial setup*** |
| ***Debugging can be hard in highly nested React Component Structure even with Dev Tool*** | ***Incredibly powerful Redux Dev Tools to ease debugging*** |
| ***UI logic and State Management Logic are in the same component*** | ***Better code organization with separate UI logic and State Management Logic*** |

***8. What is webpack ?  
A. At its core, webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a*** [***dependency graph***](https://webpack.js.org/concepts/dependency-graph/) ***from one or more entry points and then combines every module your project needs into one or more bundles, which are static assets to serve your content from.***

***9. What is treeshaking ?  
A.Tree shaking is a term commonly used within a JavaScript context to describe the removal of dead code.***

***It relies on the*** [***import***](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import) ***and*** [***export***](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export) ***statements in ES2015 to detect if code modules are exported and imported for use between JavaScript files.***

***In modern JavaScript applications, we use module bundlers (e.g.,*** [***webpack***](https://webpack.js.org/) ***or*** [***Rollup***](https://github.com/rollup/rollup)***) to automatically remove dead code when bundling multiple JavaScript files into single files. This is important for preparing code that is production ready, for example with clean structures and minimal file size.***

***10. What are some features of using webpack ?  
A. Entry :***

***Webpack creates a graph of all of your application's dependencies. The starting point of this graph is known as an entry point. The entry point tells Webpack where to start and follows the graph of dependencies to know what to bundle. You can think of your application's entry point as the contextual root or the first file to kick off your app.***

***In Webpack we define entry points using the entry property in our Webpack configuration object.  
Output :***

***Once you've bundled all of your assets together, we still need to tell Webpack where to bundle our application. The webpack output property describes to Webpack how to treat bundled code.***

## ***Loaders :***

***The goal is to have all of the assets in your project to be Webpack's concern and not the browser's. (This doesn't mean that they all have to be bundled together). Webpack treats*** [***every file (.css, .html, .scss, .jpg, etc.) as a module***](https://webpack.js.org/concepts/modules)***. However, Webpack only understands JavaScript.***

***Loaders in webpack\_transform these files into modules\_as they are added to your dependency graph.***

## ***Plugins :***

***Since Loaders only execute transforms on a per-file basis, plugins are most commonly used (but not limited to) performing actions and custom functionality on "compilations" or "chunks" of your bundled modules*** [***(and so much more)***](https://webpack.js.org/concepts/plugins)***. The Webpack Plugin system is*** [***extremely powerful and customizable***](https://webpack.js.org/api/plugins)***.***

***In order to use a plugin, you just need to require() it and add it to the plugins array. Most plugins are customizable via options. Since you can use a plugin multiple times in a config for different purposes, you need to create an instance of it by calling it with new.***