A **store** is a container that holds and manages application global state.

An **action** is a plain JS object that represents an intention to change the store's state. These objects must have a *type* property with a user-defined string value that describes the action being taken. Can also accept a payload property.

```
const addItem = { type: 'shopping/addItem', payload: 'Chocolate Cake'}
```

A **reducer** is a plain JS function that accepts the store's current state and an action in order to return the new state. They calculate the new state based on the action it receives.

```
const shoppingReducer = (state = [], action) => {
	switch(action.type) {
		case "shopping/addItem":
			return [...state, action.payload];
		default:
			return state;
	}
}
```

One-way Data flow: STORE -> VIEW -> ACTIONS -> STORE

Creating a Redux Store
createStore() creates and returns a Redux store object. It takes as an argument a reducer function which is called every time an action is dispatched.

Useful functions:

store.getState()

store.dispatch(action) -> The only wait to trigger a state change. Upon receiving the action object, the store's reducer function will be called with the current value of getState() and the action object

store.subscribe(listener) -> Adds a callback function to a list of callbacks maintained by the store. When the store's state changes, all of the listener callbacks are executed.

combineReducers()

```
const rootReducer = combineReducers({
	todos: todosReducer,
	filter: filterReducer
})
```