A function is a process which takes inputs called arguments, and produces some output known as return value. A function is called pure if it abides by the following rules −

* A function returns the same result for same arguments.
* Its evaluation has no side effects, i.e., it does not alter input data.
* No mutation of local & global variables.
* It does not depend on the external state like a global variable.

Let us take the example of a function which returns two times of the value passed as an input to the function. In general, it is written as, f(x) => x\*2. If a function is called with an argument value 2, then the output would be 4, f(2) => 4.

Let us write the definition of the function in JavaScript as shown below −

const double = x => x\*2; // es6 arrow function

console.log(double(2)); // 4

**Here, double is a pure function.**

As per the three principles in Redux, changes must be made by a pure function, i.e., reducer in Redux. Now, a question arises as to why a reducer must be a pure function.

Suppose, you want to dispatch an action whose type is **'ADD\_TO\_CART\_SUCCESS'** to add an item to your shopping cart application by clicking add to cart button.

Let us assume the reducer is adding an item to your cart as given below −

const initialState = {

isAddedToCart: false;

}

const addToCartReducer = (state = initialState, action) => { //es6 arrow function

switch (action.type) {

case 'ADD\_TO\_CART\_SUCCESS' :

state.isAddedToCart = !state.isAddedToCart; //original object altered

return state;

default:

return state;

}

}

export default addToCartReducer ;

Let us suppose, **isAddedToCart** is a property on state object that allows you to decide when to disable ‘add to cart’ button for the item by returning a Boolean value **‘true or false’**. This prevents user to add same product multiple times. Now, instead of returning a new object, we are mutating isAddedToCart prop on the state like above. Now if we try to add an item to cart, nothing happens. Add to cart button will not get disabled.

The reason for this behaviour is as follows −

Redux compares old and new objects by the memory location of both the objects. It expects a new object from reducer if any change has happened. And it also expects to get the old object back if no change occurs. In this case, it is the same. Due to this reason, Redux assumes that nothing has happened.

So, it is necessary for a reducer to be a pure function in Redux. The following is a way to write it without mutation −

const initialState = {

isAddedToCart: false;

}

const addToCartReducer = (state = initialState, action) => { //es6 arrow function

switch (action.type) {

case 'ADD\_TO\_CART\_SUCCESS' :

return {

...state,

isAddedToCart: !state.isAddedToCart

}

default:

return state;

}

}

export default addToCartReducer;