React – A javascript library for building User interfaces which run on **Browser.**

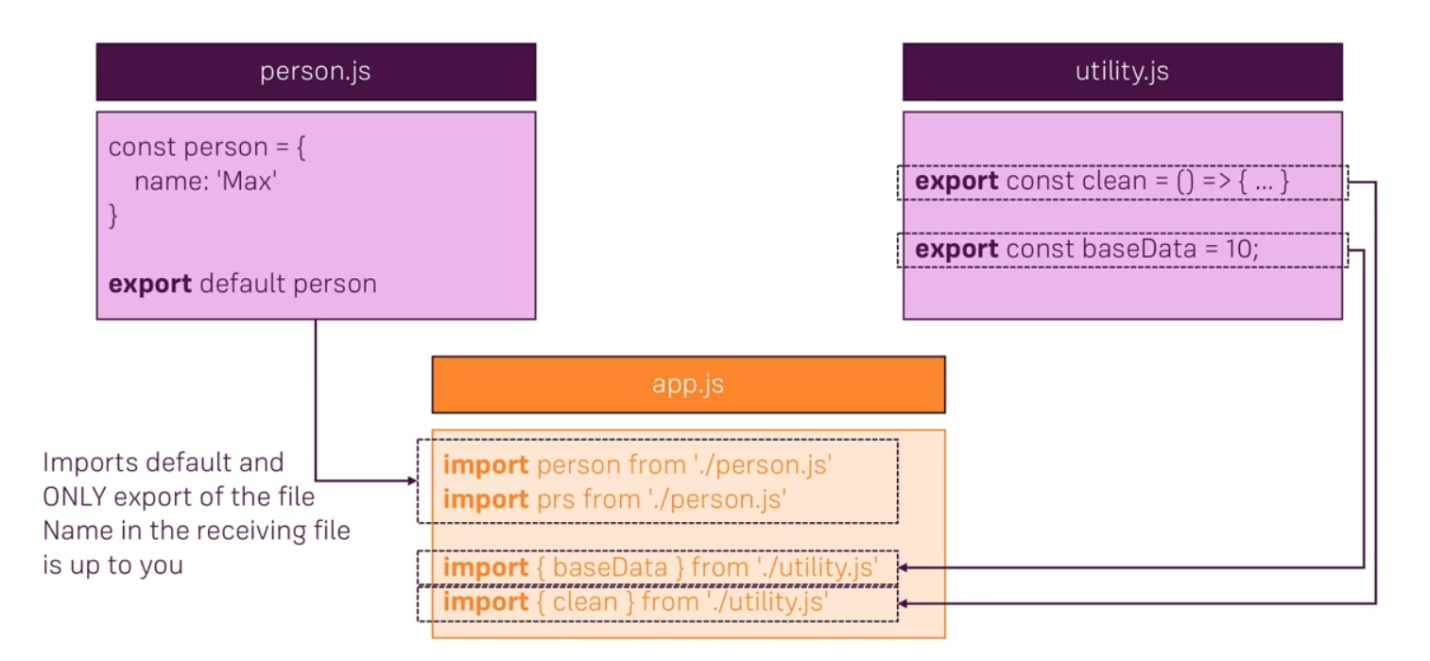
**Uses Components , so code updation is easy, code reusability is enhanced and total manageability also shoots up.**

**PreRequisites:**

1. **JavaScript**
2. **Let** (for variables) **and const**(for constant variables)
3. **Arrow functions:**

Const functionName=() =>{ … }

1. **Import and Export:**



Default export – Use any name of your choice while importing.

Named export – Use the exact name for importing back or can use an alias like:

Import {clean as cln} from ‘./utility.js’.

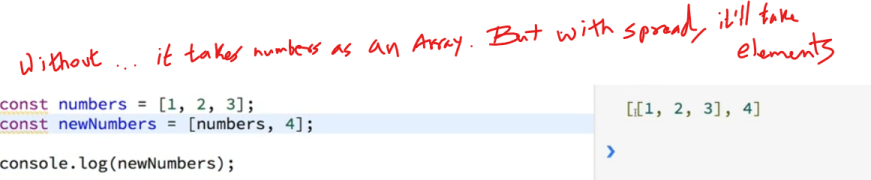
1. **Classes** : contains variables and functions, can have constructor and supports inheritance.
2. **Spread and Rest operator :**

**Spread:**  use of ‘…’ in props and objects for splitting array props or spreading/copying stuff for properties or objects . eg

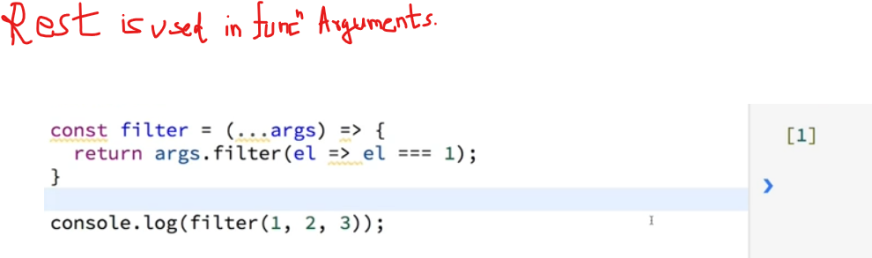
**const NewArray = […oldArray,5,6,8]**

**and const NewObj= {…oldObject, name=”Ram”}**

now if there isn’t a property called “name” in oldObject then it will be added. Otherwise it will be updated.



**Rest : use of “…” in functions**



1. **Destructuring** : copying specific contents of an array or object.



1. **References**:

Primitive data types actually copy the contents but when we try to use = for array or objects, they doesn’t copy the contents instead they just point it to the same memory location.

For true copying, we have to use spread operator .

1. **Inbuilt functions like Array functions** ( map, filter



Installing React :

npm install create-react-app –g

Making a new Component (first go to the respective folder):

create-react-app one

Starting the server:

npm start inside the folder

**File Structure:**

index.html has div with id “root”

index.js file renderes App.js into the #root div.

App.js sends html by returning JSX via class which extends component.

Other components can be made and imported in the App.js which can use the components as tags.

**Components:**

* Make new folder and put js file in it
* Import react and make a function which returns some JSX and export it with a name which can be imported in App.js eg:
* import React from 'react';
* const person =(props) =>{
* return (
* // <p> This is a person and that person is {Math.floor(Math.random()\*30)}  </p>
* <p>This is {props.name}, who is {props.age} years old.</p>
* )
* };
* export default person;
* “props” is basically the collection of properties sent to this file by App.js which looks like this:
* import React, { Component } from 'react';
* import Person from './Person/Person.js' ;
* import './App.css';
* class App extends Component {
* render() {
* return (
* <div className="App">
* <h1>This is React App</h1>
* <p>This part is main app section</p>
* <Person name ="Max" age ="12">"Yo I am an engineer"</Person>
* <Person name ="Mark" age ="23">"I love Badminton"</Person>
* <Person name ="Marshall" age ="32">"I am done with hobbies"</Person>
* </div>
* );
* }
* }
* export default App;
* To use the content written between the opening and closing tags, we can use {props.children}
* Using State to keep values inside component:
* class App extends Component {
* state = {
* persons:[
* { name:'Max', age:12 },
* { name :'Mark', age:23 },
* { name :'Marshall', age:32}
* ]
* }
* render() {
* return (
* <div className="App">
* <h1>This is React App</h1>
* <p>This part is main app section</p>
* <Person name ={this.state.persons[0].name} age ={this.state.persons[0].age}>"Yo I am an engineer"</Person>
* <Person name ={this.state.persons[1].name} age ={this.state.persons[1].age}>"I love Badminton"</Person>
* <Person name ={this.state.persons[2].name} age ={this.state.persons[2].age}>"I am done with hobbies"</Person>
* </div>
* );
* }
* }
* Jsx
* JSX actually uses it’s own tags, although they define <div> as <div> internally, they do have differences like className instead of class and onClick instead of the usual onclick.
* We can change state values by using this.setstate(), it basically updates the contents which are being subjected to change.
* For calling a function on occurrence of an event, we shouldn’t use parenthesis, coz if we use parenthesis, the function will be called while compilation itself.
* <button onClick={this.switchName}>Switch Name</button>
  + To send in parameters we can use something like this:
* <button onClick={this.switchName.bind(this,"New Name!")}>Switch Name</button>
  + Or we can use : (using arrow function allows us to use parenthesis and avoids execution of the function during compilation)
* <button onClick={() => this.switchName("New Name!")}>Switch Name</button>
* We can also pass functions from components as props……
* <Person name ={this.state.persons[0].name} age ={this.state.persons[0].age} click={this.switchName}>"Yo I am an engineer"</Person>
* And using it in presentation file like:
* <p onClick = {props.click}  >This is {props.name}, who is {props.age} years old.</p>

**React Hooks:**

let you use state and other React features without writing a class.

Key concept – use functions instead of classes, and it also reduces the hassle of writing functions for eg, setState is eased into useState ie. Use of below istead of the one above…. :

 const App = props =>{

    const [personsState, setPersonsState] = useState({

    persons:[

      { name:'Max', age:12 },

      { name :'Mark', age:23 },

      { name :'Marshall', age:32}

    ]

  });

    const switchName=() =>{

    setPersonsState({

      persons:[

        { name:'Max 2.0', age:12 },

      { name :'Mark 2.0', age:23 },

      { name :'Marshall 2.0', age:32}

      ]

    })

    personsState.persons[0].name = "Max 2.0";

  }

    return (

      <div className="App">

        <h1>This is React App</h1>

        <p>This part is main app section</p>

        <button onClick={switchName}>Switch Name</button>

        <Person name ={personsState.persons[0].name} age ={personsState.persons[0].age}>"Yo I am an engineer"</Person>

        <Person name ={personsState.persons[1].name} age ={personsState.persons[1].age}>"I love Badminton"</Person>

        <Person name ={personsState.persons[2].name} age ={personsState.persons[2].age}>"I am done with hobbies"</Person>

      </div>

    );

  }

The issue with hooks is that the update function doesn’t merge the updated part with the previously defined function, instead it replaces it.

Two Way binding:

    <input type="text" onChange={props.changed } value={props.name} ></input>

props has ‘changed’ as a property defined in App.js

<Person name ={this.state.persons[0].name} age ={this.state.persons[0].age} changed={this.nameChange}></Person>

nameChange is a function, which focuses on changing the value of the first element of array according to the typed content which is being sent in for of an event triggered by onchange :

nameChange=(event)=>{

    this.setState(

      {

        persons:[

          {name : event.target.value , age:29 },

          { name :'Mark 2.0', age:23 },

          { name :'Marshall 2.0', age:32}

        ]

      }

    )

  }