React.js

→ build user interface

→ react.js components can be thought of as components of burger each of which can be changed.

each component can update itself independantly without affecting rest of website

→ Note

ReactDOM.render(What to show, where to show, callback);

eg

```
var React = require("react")
var ReactDOM = require
            ("react-dom")

ReactDOM.render(<div>
                  <h1></h1>
                  <ul>
                    <li></li>
                    :
                  </ul>
                </div>,
      document.getElementById
              ("root"));
```

new

```
import React from "react";
import ReactDOM from
            "react-dom";
```

→ NextGen JS with JSX $\xrightarrow{\text{Babel}}$ Old vanilla JS

```
→ ReactDOM.render(
      <div>
        <h1> Hello {name}! </h1>
        <p> Your lucky number
             is {num} </p>
      </div>,
      document.getElementById
                 ("root")
  );
```

{ —— }
↳ only expressions no statement

→ In JS, attributes only in camelCase.

→ in 'index.html' in script tag change "text/javascript" to 'text/JSX'

→ eg

className → to specify on which class css applied

contentEditable → makes a content editable

spellCheck → turns off auto spell checker

→ css style in React

style={{color: "red"}}

js element ↳ to be enterpreted as css

```
const s = {color: "red",
           fontSize: "20px",
           border: "1px solid
                    black",
          };
ReactDOM.render(
    <h1 style={s}>
        Hello world </h1>,
    document.getElementBy
                Id("root")
  );
```

→ React.js components

```
function Heading(){
   return(<h1> .. </h1>);
}

function List(){
   return( <ul>
             <li> ... </li>
             <li>   </li>
           </ul>);
}
ReactDOM.render( <div>
                   <Heading/>
                   <List/>
                 </div>,
      document.getElementById
                     ("root")
  );
```

→ Heading.jsx

```
import React from "react"
function Heading(){
    return <h1> </h1>;
}
export default Heading;
```

→ List.jsx

```
import React from "react";
function List(){
    return ( <ul>
                <li></li>
                <li> </li>
             </ul> );
}
export default List;
```

→ React props.

```
import React from "react";
import ReactDOM from
            "react-dom";
function Card(props) {
    return(
        <div className="mystyle" >
            <h2>{props.name}</h2>
            <img src={props.img}/>
            <p>{props.phone}</p>
            <p> {props.mail} </p>
        </div>
    );
}
ReactDOM.render(
    <div>
        <h1> My contacts </h1>
        <Card name=" " img=" "
              phone=" " mail="" />
    </div>,
    document.getElementById("root"),
);
```

→ Import / Export Modules

→ there can be only one default export

→ there can be multiple non default exports

<u>Note</u>

```
import pi, {doublepi, triplepi}
        from "./math.js";
import * as pi from "./math.js";

    pi.default // default export.
    pi.douplepi()
    pi.triplepi()

export default pi;
export {doublepi, triplepi };
```

→ Mapping Components

when we map through array each of the component created on the fly must have a unique key component
↳ keyword that cannot be proped

1) numbers = [3, 56, 2, 48, 5]
map

```
const newNumbers =
    numbers.map(function(x){
        return x * 2; });
```

2) filter

```
const newNums = numbers.filter
        (function(x) {
        return x>10;
        });
```

3)

```
var num = numbers.reduce(
        function (sum, no) {
        return sum+n;
        });
```

4)

```
var n = numbers.find(
        function(num) {
            return num>10;
        });
```

5)

```
var n = numbers.findIndex(
        function(num) {
            return num>10;
        });
```

→ Arrow function

```
var nums = [3, 56, 6, 94, 2]
const newNums = nums.map((x) => {
    return x*x;
});
```

if we have a single expression,

```
const newNums = nums.map (x => x*x);
```

→ Ternery Operator.

```
↳ if (condition){ }
  else { }
      ↓
condition ? true-condⁿ : false-condⁿ
      ↓
condition && true-condition
```

↓ will be executed only if 'condition' is true

→ Hookes in JS.

syntax.

```
const [state, setstate] =
                useState(init state);
```

→ Destructing in JS.

```
const value = [32, 56, 48];
~~value~~
console.log(value[0]) → error.
const [red, green, blue] =
                [32, 56, 48]
console.log(red);
```

→ Event handling in JS.

```
import React from "react";
function App (){
  function handleChange(event){
      console.log (event.target.value);
  }
  return (
    <div className = "container")
     <h1>Hello </h1>
     <input onChange =
             {handleChange}
        type = "text"
        placeholder = "whats your name?" />
  </div> );}
```

→ Spread Operator.

```
const citrus = ["lime", "lemon",
                "orange"]
const fruits = ["Apple",
                "Banana",
                "Coconut"
                ...citrus]
```

spread operata

posⁿ of spread operator will decide pos of inserted array elements