React

→ Javascript

* defer attribute in script tag is used to render the js file only after the body is rendered.

* type = "module" in script tag tells that the js file is a module and can be used for imports & exports.

* In react projects, js files aren't linked directly to HTML pages
  ↓
  Infact, React uses build process in the backend
  ↳ It uses "react-scripts"

  2 reasons
  ↳ JSX in js files can't be interpreted directly
  ↳ code is optimise after build process.

* Import & Export
  → Only one 'export default' in a js module.
  ↳ for variables, omit identifier & assignment.
  ex: export default "vijay";
  ↳ for functions & classes
  ex: export default function Comp() {
  }

→ {} for normally exported items } Importing
directly for export default

import {myname} from 'file.js'

import Comp. from 'file.js'

.js can be omitted in react projects because of the build process available

* Null vs Undefined

Null → explicitly assigned by developer (reset value)

Undefined → default when no value assigned

* Array methods

i) findIndex((item) => condition)

ii) map() → used to transform array

iii) filter() → used to filter elements from an array using a => condition

iv) reduce() → used to reduce an array of elements into a single entity

accumulator, current
↓ ↓
updated result. points to current element
after each iteration.

asc function()
↳ returns a value.

→ Destructing in Arrays & Objects.

```
const data = [1, 2];
const x = data[0]
const y = dat[1]
        // Destructuring
const [x, y] = [1, 2];
// x = 1
// y = 2

const user = {
    name: "Max",
    age: 34,
};

const name = user.name;
const age = user.age;
        // Destructuring
const {name, age} = {
    name: "Max"
    age: 34
};
```

Order doesn't matter {name, age} or {age, name}

In object destructuring, the names should be same to the keys of objects.

Used for destructuring

Used to create object

& In arrays we can use different names while destructuring

we can instead use aliasing with colon (:)

```
const {name: userName, age} = {
    name: "Max",
    age: 34
};
```

aliasing

→ It can be used in functions too

```
function showOrder(order) {
    local.setItem('id', order.id);
    local.setItem('currency', order.currency);
}
```

⇓

```
function showOrder({id, currency}) {
    local.setItem('id', id);
    local.setItem('currency', currency);
}
```

Note: Even after destructing, function accepts one parameter only, which is an object, ~~too~~ which then is destructed internally.

Spread operator (...)

It is used on arrays or objects to pull out the values as comma separated values (or) key-value pairs as comma-separated values (in case of objects), which can be used in another array or object.

Ex:

```
user = {
    name: "xxx",
    age: 23
}
Admin = {
    adminStatus: "true",
    ...user
}
```

Passing functions as arguments

→ Do not pass functions with parenthesis as arguments.

↳ If you do so, then function (inner) will execute as soon as the line of code occurs instead of executing when the condition is met.

* We don't need DOM operations through JS

CUZ

React handles that!

Primitive	Reference
↓	↓
Numbers, boolean, string	Objects, Arrays --
↓	↓
direct value is stored in the variable	Address of the object is stored in the variable.

∴ If I do stg like this,

```
const arr = [1, 2]
arr.push(3)
```

I'm just changing the array / pushing an element into the array

not actually changing the address.

React

Events & Event handling on react

↳ We don't need to write big addEventListener code on react.

↳ We just need to write event handling through props for the element we want to handle events.

↳ these props don't handle/pass data, but they handle events by taking event handler function as value.

def clickHandler { // }

Ex: <button onClick={clickHandler}> </button>

↳ starts with 'on' and then the event type

all predefined HTML elements have their respective event handler props

↳ It should just be the function variable without parantheses. (or) Can even define the function there itself.

Interesting case of How React Works

→ If we wanna update the value of title on the browser using a button click.

We can do that using event handling on button, but the value doesn't get updated on browser, though it gets updated on console.

UseState() come on rescue for this matter which basically returns a variable and a function to update that variable.

* useState() can only be defined inside a function component & not outside.
  - It also may not be defined inside another function even though that another function is present in the function component.

* Every time we update the useState() variable using the function of useState(),
  - the component in which the useState() is defined will be reloaded / updated or rendered on browser again.
    - { only that particular component is re-rendered }

* Hence this solves the problem of updating the UI when an event occures, we can just change the state, which results in re-rendering of the component in which the state is part of.

→ It is a react hook. All react hooks start with "use". All react hooks should be called inside component function only.

Ex: const [title, setTitle] = useTitle(props.title);
- title: changes to this variable using this function will lead to the component function to be called again.
- setTitle: initial value

→ State is associated to 'per component instance' basis.

If there is a state defined for a component and there are eventually reusable components created. Each instance of that component has individual / separate state.

Note: There can be multiple states for a component