Datatypes,
functions = default parameters,
REST parameters = … in the function parameter list
Callback functions = We pass the function A to another function(called)
The called function will call the callback functionA
foreach , map

class syntax of JS = we can have only one constructor
get and set properties = to ensure property hiding
Controlled access
We can extends but we must call super in the constructor

JSON objects = Stringified JSOn and JSON object
{obj1:val,obj2:val2 } = JSON object
Str = JSON.stringify(obj)
Obj = JSON.parse(Str)

DESTRUCTURING Syntax
let { uname,dept} = emp //uname and dept are local variables
That get the uname and dept property values

Let [v, func] = arr

Spread operator = creating copy of the object
Let obj = {…oldone}

While using spread operator - it will create copy primitive properties
and copy of reference of object or array properties
We can think of explicitly deep copying

Client side JS = DOM updation at run time
= document<=====> DOM
= find a TAG using document , we can modify it - add TAGS
Remove TAGS , change attributes of the TAGS

REACT = SPA vs MPA

Component based architecture = each component describes a DOM fragment
At component level the updation = rerendering decision is taken

Function component
Class component

These components return the JSX = html like code , names of the attrubutes changes
className instead of class
style={ obj } //all the css properties are written in PASCALcase

PROPS = attributes passed to the TAG by the OUTER/PARENT/Containing component
= these attributes are MADE AVAILABLE to the component by REACT Framework
= **c1(props)** // parameter to functional component
= class component gets a property **this.props**

Event handling -
onClick = {name of the handler } //never call the handler
REGISTER the handler that will be called by REACT on the event

WE can ask the REACT FRAMEWORK to pass event object to the handler
Event object provides INFO(attributes) about the TAG on which event occurred
event.target.value

HOOK = Library function of REACT framework

let arr = useState(initial value of variable)

arr[0] = state variable = variable whose value will be PRESERVED after rerender

arr[1] = function to modify the state variable = when we change the value of the state variable using this function - the component is RERENDERED if the value changes

WHY to RERENDER ? To show the latest values of the variables!!

let [num, setnum] = useState(0)

InputComp

get the num1 num2 textfields and operation drop down list from the user

Calculator props num1 num2 and operation

- Perform the operation and show result
  - 12 + 10 = 22
  - 13 * 10 = 130

Conditional Rendering = RENDER only if condition is true

And

op1	Op2	Result
0	0	0
1	0	0
0	1	0
1	1	1

Op1 && op2

if op1 is false then no need to evaluate op2 = OPTIMIZATION

List Rendering =

Use the map function

```
let anotherarr = arr.map( (element)=>{
        Return a new element that corresponds to the element passed
} )
```

Map calls the lambda function once for each element in the arrray

Map will return another list that contains the new elements

DOM = browser

React is not directly changing the DOM

=== it uses Virtual DOM

=== it will do the changes in Virtual DOM

it will diff the earlier DOM with new DOM

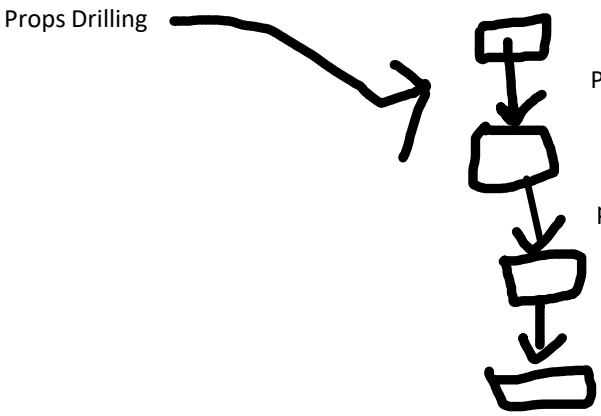
whatever differences are there ONLY those are rerendered /changed in the DOM

For Lists - if it gets uniques ID for each element then it can only update the changed elements ONLY avoiding unnecessary DOM updates!! BETTER

Higher Order Components = Wrapping one component into another

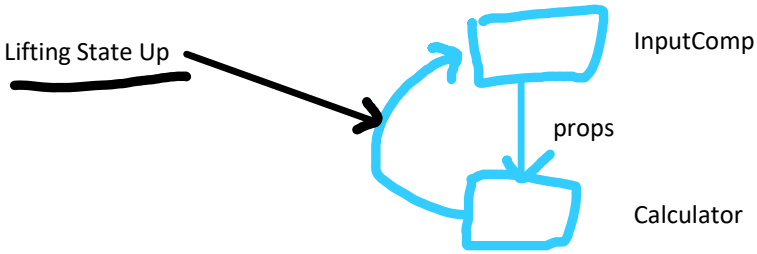
--- add a feature to my component without changing the component

---



Communication between the OUTER/Parent component to INNER/Child component through PROPS
If the level of NESTED components increases then PROPS DRILLING becomes invonvenient to manage

---



Ex ----

EmpManager Component
Array of Employees
[
{eid:12, empName:'ww' ,dept:'sw'},
{eid:12, empName:'ww' ,dept:'sw'},
{eid:12, empName:'ww' ,dept:'sw'},
]

Show the list of employess in a table

Add Employee Component
Eid : textfield
Ename: textfield
Ename : select / textfield
Add-button = user clicks on this button - a new employee object should be added to the array

---

Update a record in the array in the EmpManager

UpdateComponent
- show the values of the record in textfields in the update component
- keep the id field as readonly
- when user changes the name or dept
clicks update button --- the record in the table should be updated

Let use a form = we can submit a form = **WE NEVER LEAVE THE PAGE**

MPA

```
<html>
 <body>
   <form  action="anotherpage"  >
            <input  name="uname"  />
            <input name="password" />
            <input type="submit"   value="save"  />
   </form>
  </body>
 </html>
```

SPA

```
<div>
<form  onSubmit={handler}  >
            <input  name="uname"  />
            <input name="password" />
            <input type="submit"   value="save"  />
   </form>

</div>
```

CONTROLLED FIELDS = value = {statevariable } MUST give onChange(usestatesetter)

Life Cycle callbacks ------ functions which can be implemented by us
and called by react , **WHEN ? Whenever the suitable activity happens**

Lifecycle of react components is managed
React framework is managing the components -

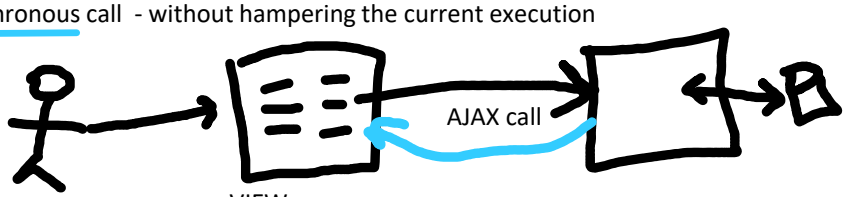
MOUNTING the component on the DOM
- It will create objects of the class components
- It will call the function of the functional componets

UPDATING
- --- when the props change RERENDER
- --- when state changes RERENDER
- ---- event handling RERENDER

UNMOUNT from the DOM

Ajax call using React

Asynchronous call - without hampering the current execution



VIEW
Is not refreshed
Not received in response

User interaction is SEAMLESS

---

axios library for asynchronous server calls   (AJAX call)

- download the library in our project
  cd project library > npm install axios

ASYNCHRONOUS = DELAYED EXECUTION , not executed in the sequence

the function will run sometime later depending on the CB queue!!

```
//SEQUENTIAL EXECUTION
M1()
Let x = M2()
M3(x)
```

IF we are interested in the VALUE changed by that function
HOW can access value
WHERE can we access that changed VALUE !!!!

JS provides ways to access values changed by ASYNC callbacks---

1. **Promises**
2. Asych await
3. observables

Promise = inbuilt JS object
It lives in 3 states
- pending promise
- fulfilled promise (RESOLVED)
- unfulfilled promise (REJECTED)

Promise has a useful API = then()
p.then(cb1, cb2) // we r registering two callbacks
if the promise resolves call cb1
if the promise rejected call cb2