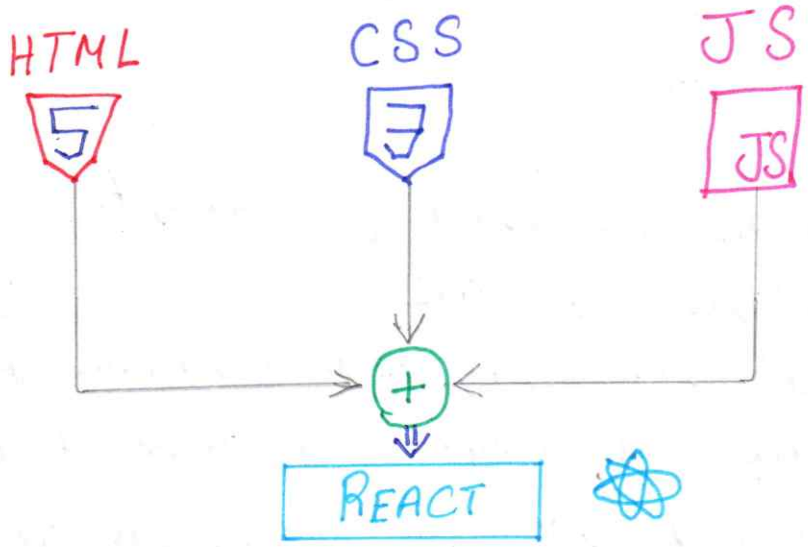
React

- React is a client-side JavaScript library.
- It is used for building modern, reactive userinterface for the web.
- Declarative, Component focused Approach.
- It is all about "Components".
-



- React allows you to create re-usable and reactive Components consisting of HTML & JavaScript. (and CSS).
  - * React uses Something which is called declarative approach for building these Components.
    - It means that with react, we will not tell react that a certain HTML element should be created and inserted in a specific place on the user interface as we would be doing it with Vanilla JavaScript.

- Insted with React, we will always define the desired end State, the target State or possibely also different target States depending on different Conditions and its's then React's job to figure out which elements on the actual web page might needed to be added or removed or updated.

- And we don't write these Concrete DOM updating Instructions on our own as we would be doing it with just ~~Javascript~~ Javascript.

- Insted with React & React Components, we just define these end States and under which conditions, which State Should be used and then React will do all the rest under the hood.

JSX :-

- It is basically HTML Code inside of Javascript.
- It Stands for JavaScript XML
- JSX Code is not understand by the browsers because, generally we assigns a HTML tags to a Variable. So to Convert it to browsers understandable Javascript code, we use a tool like 'Babel'

```
import React from 'React';

return React.CreateElement('div', {},
  React.CreateElement('H2', {}, "let's get Started"),
  React.CreateElement(Expenses, {items: expenses})
);
```

Old-Way.

VS

```
return(
   <div>
   <H2> let's get Sta</H2>
<Expenses items={expenses}/>
</div>
);
```

New-Way.

Props :-

- We use 'props' in react to pass data from one component to another (from one parent component to child component(s)).
- Props is just a shorter way of saying properties, they are useful when we want the flow of data in our app to be dynamic.
- props are passed to the component in the same way as arguments passed in a function.
- Props are immutable, so we cannot modify the props from inside the component.

Composition :-

- It's a development pattern based on react's orignal component model where we build components from from other components using explicit defined props. or the implicit children prop.
- In terms of refactoring, react composition is a pattern that can be used to break a complex component down to smaller component, and then composing those smaller components to structure and complete your application.

- Custom component "Card":- We cannot simply use our custom components as wrappers around other kind of ~~components~~ content.

  * In order our wrapper 'card' to works we have to do some process as follows.

* In Card.js, when we accept props, we will use one special prop which is built into React, which every component receives, even if we never set it explicitly.

```
function Card (props){
    return <div className="card">{props.children} </div>
```

* Its a prop which value we wanna output b/w the { } tag of above div, Insted of the card component function. It is props.children.

* 'Children' is a reserved name and the value of this special children prop will always be the content between the opening and closing tag of our custom component.

```
eg        <Card  className='name'>

              - - - -  }
              - - - -  } value of children.prop.
              - - - -  }

          </Card>
```

* Make sure that a className can be set on our Card component So we add whatever is set as a classname on card to this className string, we're setting as a className on that 'div'

```
e.g.  function Card (props){
          const classes = 'card' + props.className;
          return <div className={classes}>{props.children}</div>
      }
```

* here, 'card' as a default class, which is always applied — « whitespace and props.className, so anything we receive as a className from outside is added to that string, and then we can then dynamically point at this "const classes". So by doing that we are making sure that

any value set on the className prop is added to this long string of class Names, which is then finally set on the div inside of the card.

React State & Events.

Handling Events :-

- In react we add event listners in JSX code within the element like 'onClick'.
- React Events are named using CamelCase, rather then lower case. and with JSX you pass a function as the event handler, rather than a String.

e.g-
```
Const ClickHandler = () => {
        Console.log(" Clicked !! ");        // function.
}

<button onClick ={ClickHandler}> Change </button>
```

States :-

- Once a Component function is called Initially, that will not called a Second time, because a clicked occured or a Variable change does not trigger that component function to run again.

- In order us to tell React that it Should run again. for that we need to import Something and that is 'useState' function, this is the function provided by the react library.

```
import React, {useState} from 'react';
```

- This function allows us to define values as state, where changes to those values should reflect in the component function being called again.

- We use 'useState' function inside of our component function and it is so called 'React Hook.' It wants a default state value, because by useState we basically creates a special kind of ~~value~~ variable, where changes will lead to component function to be called again.

```
useState(props.title);
```

- Now that special variable is created, now we need to use that variable and therefore useState also returns access to that variable. and plus it also returns a function which we can then call to assign a new value to that variable.

- Syntax :- The first element is the initial state and second one is a function that is used for updating that state.

```
Const [state, setState] = useState(initialState);
```

* UseState actually returns an Array where the first value is variable itself and second one is updating function. and above we are using Array destructuring to store the values in seperate variables which are 'Constant.' It always returns an array with two elements.

- We are not going to assing a value to that variable by equal sign '=', insted, we we assign a new value by calling 'setState' function. We have to do it like that way because, calling this function not just assign a new value to some variable, but that insted it is a special variable to begin with. It's managed by React somewhere in memory and when we call this state updating function this special variable not just receives a new value but the component function. In which we called this setState function will be executed again. And that is what we need, we wants to call this component function again when our state changes.

- When we call setState() function, we are actually telling react that we wanna assign a new value to this state and that then also tells React that the component in which this state was registered with useState should be re-evaluated.

- If we have data, which might change, and when changes to that data should be reflected on the user interface, then we need state. Because regular variables will not do the trick, with state, However we can set and change values and when they do change, React will re-evaluate the component in which the state was registered.

Two-Way Binding :- It means that for inputs we don't just listen to changes, but can also pass a the new value back into the Input. So that we can reset or change the input programmatically.

* for doing this, we need to set the 'value' attribut to input element. this will set the Internal value property, which every input element has, and we can set it to a new value. and we can bind it to that State variable.

* Because of this we are not just listning to changes in the input to update our state but we also feed the state back into the input, So that when we change the state, we also change input.

Value = {StateVar.}

Calling → setStateVar(' '); → empty String.

Child-To-Parent Component Communication :-

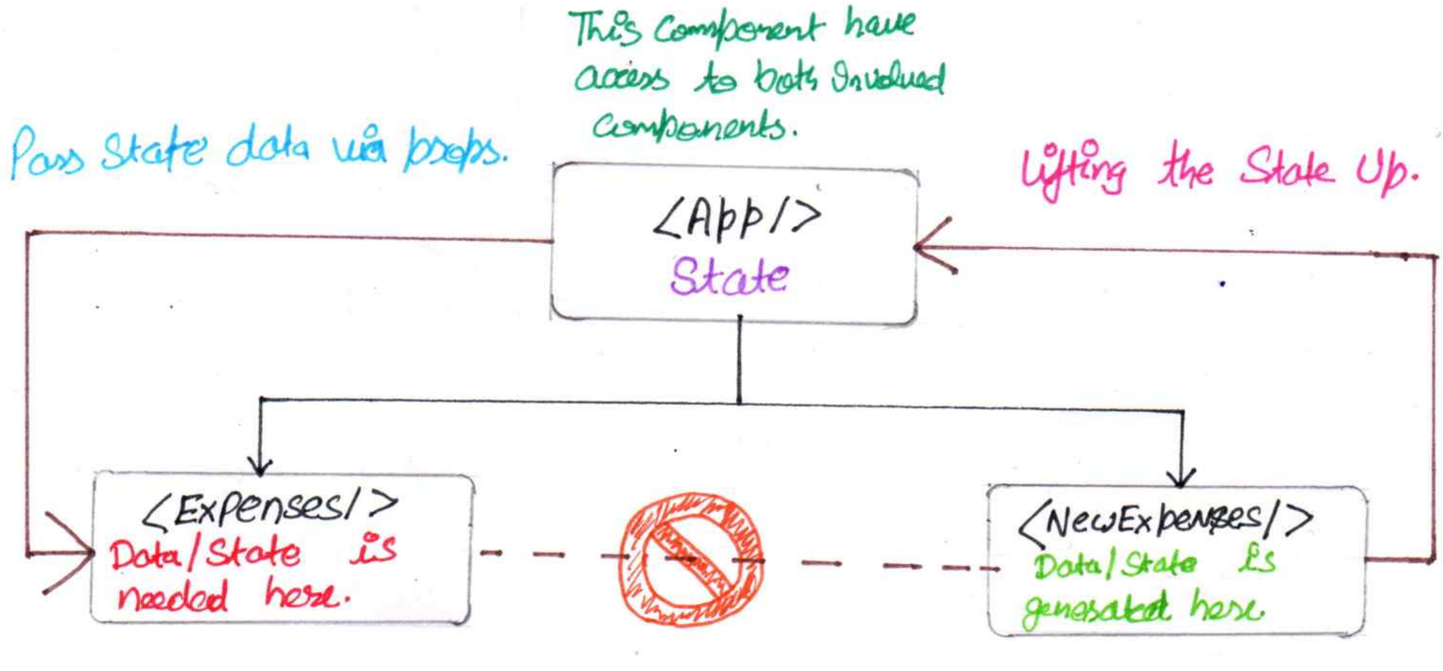
- We can create our own event props, and we can expect functions as values and that would allow us to pass a function from a parent component to a child component and then call that function inside of the child component and when we then call a function, we can pass data to that function as a parameter.

and that's how we can Communicate up from Child to parent.

Lifting State Up :-

- It is about moving data from a Child Component to Some parent Component to either use it ~~these~~ there or to then pass it down to Some other Child Component.

- The goal is to lift it up just as high as necessary in our component tree untill we have a Component which has both access to the Component that generates data as well as the Components that needs data.


This Component have access to both Involved components.
Pass State data via props.
Lifting the State Up.
<App/>
State
<Expenses/>
Data/State is needed here.
<NewExpenses/>
Data/State is generated here.