```
<form onSubmit={handleSubmit}>
    :
</form>

handleSubmit(event) {
    validate(value)  → validating input values
    event.preventDefault()
}
```

To have control over the form values when form is submitted, use onSubmit prop in <form> tag.

onSubmit callback receives event parameter

Day-3

* Controlled vs. Uncontrolled Components

→ Uncontrolled Components

↳ Inputs are like standard HTML input.

↳ How to get value? (of the input typed)

↓

Using a React ref. → pull the value from the field when needed.

e.g.

```
const Form = () => {
    const inputRef = useRef(null)
```

```
const handleSubmit = () => {

    const inputValue = inputRef.current.value
    // do something with the value
}

return (
    <form onSubmit={handleSubmit}>
        <input ref={inputRef}
               type="text"
        />
    </form>
)
}
```

→ Mostly, try to use Controlled Components.

→ `<input type="file" />`

↓

always an uncontrolled component. because its value is read-only and can't be set programmatically.

e.g.
```
<form onSu={hanSub}>
    <input type="file" ref={fileInput} />
</form>

const fileInput = useRef(null)

const hanSub = (e) => {
    e.preventDefault()
    const files = fileInput.current.files
}
```

Miracle
Pg.No :
Date: / /20

Controlled C.	Uncontrolled C.
↓	↓
Component's state handles form data	DOM handles form data

→ Events:

onBlur(): Triggered when the user moves the cursor away from the element.

onFocus(): Triggered when element receives focus (cursor on the element) Opposite of onBlur event.

✡ React Context

Props	State
↓	↓
Passed to the component	Managed within the component.

→ Similarities bet^n props and state:

1) Plain JS objects
2) Deterministic (this means the component always generates the same output for the same combination of props and state)
3) Trigger render updates (both changes on trigger or render)

→ Props:

- Properties (full form)
- Component's configuration
- They are received from parents in the tree
- Immutable
- A component cannot change its props but it is responsible for putting together the props of its child components.

→ State:

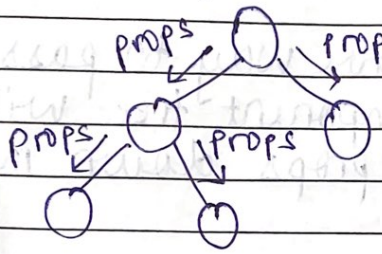
- This object is a way to allow react to determine when it should re-render.
- Has a default value
- Private

Components

Stateless	Stateful
• Props only, no state	• Both props and state
• Easy to follow and test	• Charge of client-server communication, data processing and responding to user events

✡ What is Context, and why is it used?

→ In a typical React app, data is passed from parents to children via props in a top-down fashion.



→ However, there are certain type of data that is needed by many components within an app.

→ In this case, using props is not always effective.

→ So, the alternative way of passing data is called Context

e.g. Theme switcher (dark-light mode)

→ Props drilling problem

Passing data through all component tree levels (even if components do not need it) i.e. passing through props.

Miracle
Pg.No :
Date: / /20

Parent component has to drill down props all the way to the children that need to consume them.

→ How to solve this problem?

Context API

→ Provides alternative way to pass data through the component tree without having to pass props down manually at every level.

→ Useful for global state

e.g. Website:

Blog App

Hello _____

What is Computer?

⋮ } Blog

Written by _____.

Here, both the blanks should have the name of the current logged in user.

Miracle
Pg.No :
Date: / /20

→ Code :

Components : <Header /> → title & hello
<LoggedInUser /> → user authentication
<Page />
↳ Blog name title
Blog
Written by ____

Hello ____ → <LoggedInUser/> ma lakhelu che.
i.e.

```
function LoggedInUser() {
    <h3> Hello .... </h3>
}
```

So, basically Header func. ma Blog App title and <LoggedInUser /> che.

→ Now, 2 components need to know about authenticated user name.
① LoggedInUser
② Page

→ Because an authenticated user falls into the nature of global data, shared across several components, that needs to be this is a clear example where context is perfect tool for the job.

① Create new file "UserContext.js"

```
import { createContext } from "react"
```

function that gives you a new context object back.

```
const UserContext = createContext(undefined)
```

func. value is a default argument here we set it to undefined

Provider comp.

```
export const UserProvider = ({children}) => {
    return <UserProvider>
    return <UserContext.Provider value={{user}}></UserContext.Provider>
```

has a value prop.

```
    const [user] = useState({
        name: "John",
        email: "john@gmail.com",
        dob: "01/01/2002",
    })
}
```

just for eg. we have added hard coded data to understand.

```
export const useUser = () => useContext(UserContext)
```

To provide a way for components to subscribe to the context,

Create a custom hook that wraps the useContext() hook

↓

A way to consume a context value.

→ Now, wrap <App/> comp. in App.js with <UserProvider />

```
<UserProvider>
    <App />
</UserProvider>
```

→ Now, to use the user details in <LoggedInUser> and <Page> comp,

```
func. Logg... () {
    const {user} = useUser()
    return (
        <p> Hello {user.name} </p>
```

Same for <Page /> component.