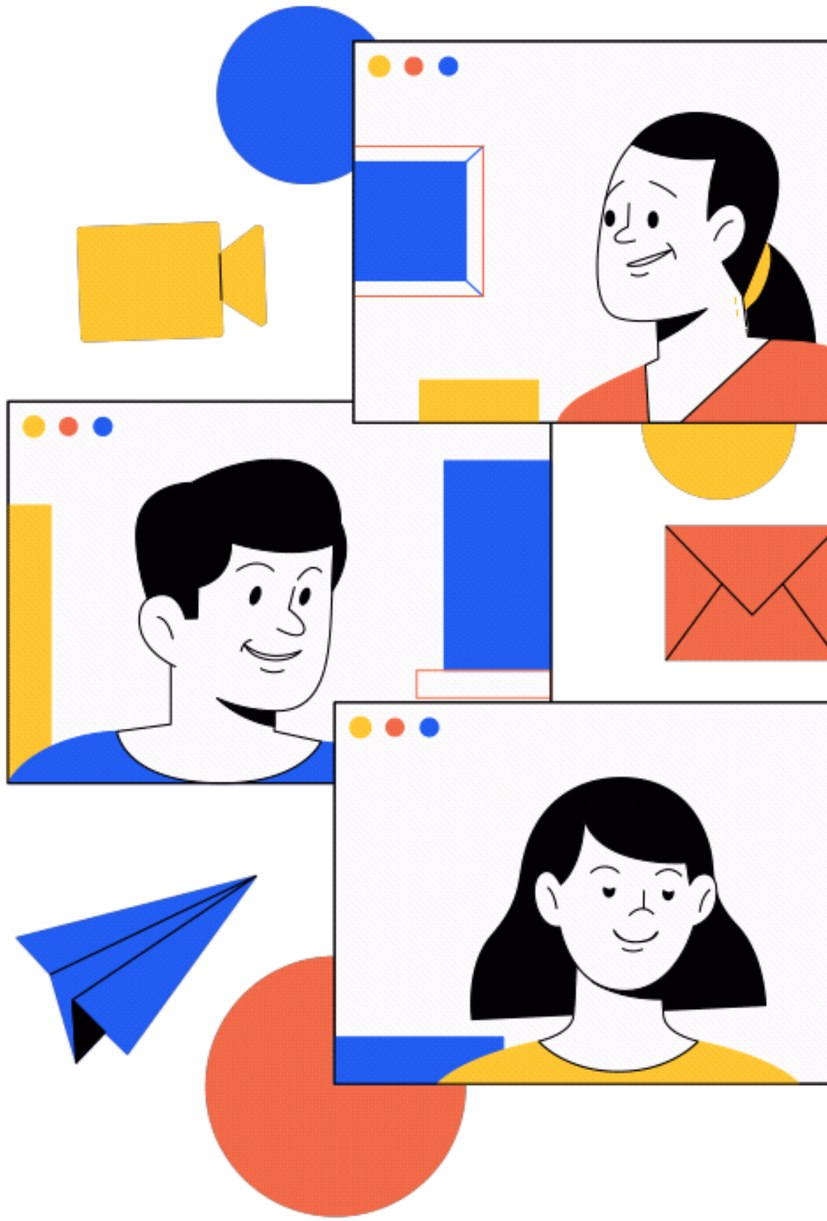
UDACITY

Week 4 - React Dev. Cross-Skilling ND

one last ride?



Ahmed Abdelbakey Ghonem

React Session Lead

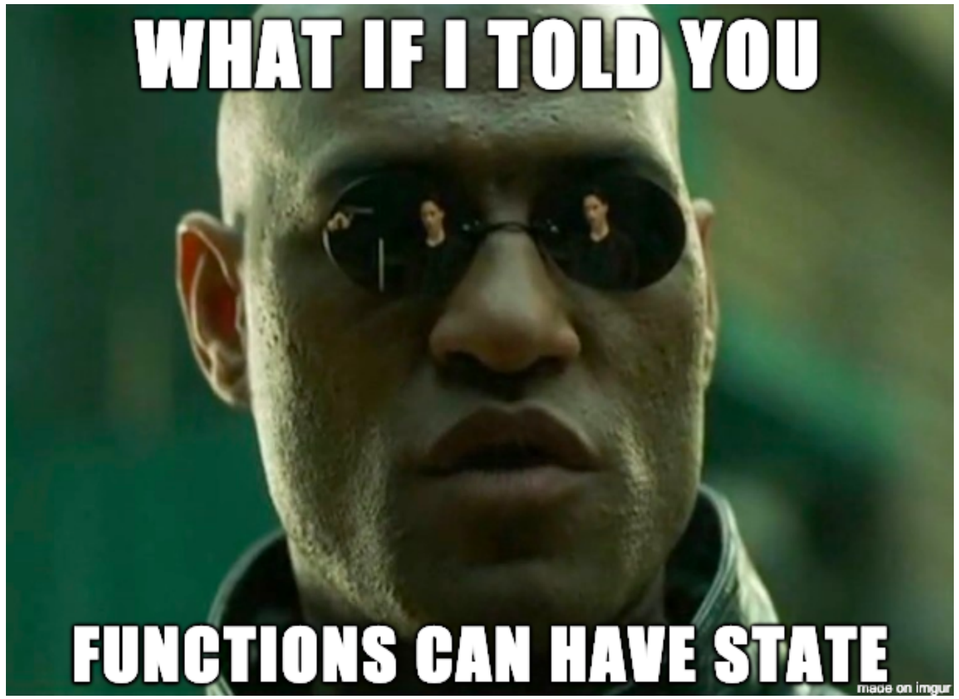
Agenda

What we'll cover in this session

- What are React Hooks?
- Rules of Hooks
- Most Famous Hooks
- Custom Hooks
- Live Demo
- What is Next?

React Hooks

- Hooks are the new feature introduced in the React 16.8 version.
- Hooks allows you to use **state** and other **React features** without writing a class.
- Hooks are the functions which **"hook into"** React state and **lifecycle features** from function components.
- Hooks does not work inside classes ✖



Rules of Hooks

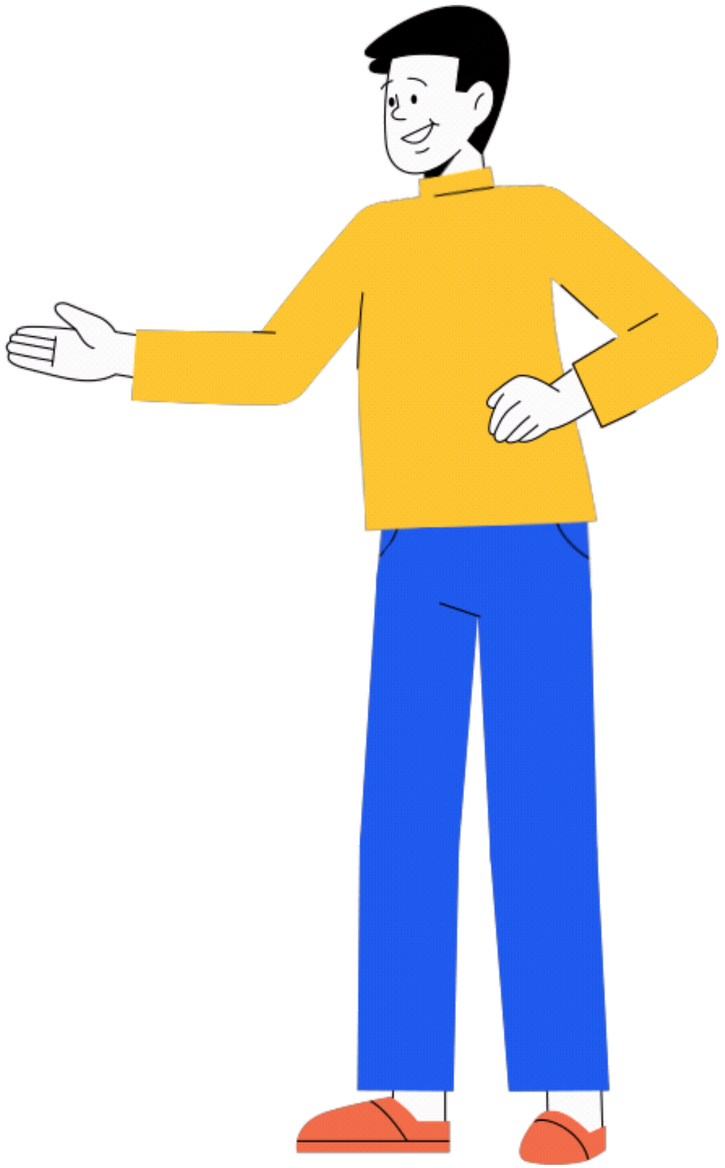
1. Only Call Hooks at the Top Level

✕ Don’t call Hooks inside loops, conditions, or nested functions. Instead, before any early returns.

2. Only Call Hooks from React Functions

Don’t call Hooks from regular JavaScript functions. Instead, you can:

- ✅ Call Hooks from React Function Components
- ✅ Call Hooks from a custom hook (later on the slides)



✕ Don’t call Hooks inside loops, conditions, or nested functions.

✕ **Bad practice**

```
const App = () => {

  // Nested functions
  const handler = () => () => {
    const [count, setCount] = React.useState(0);
  }

  return <h1>Do not call React hooks inside nested functions</h1>;
};
```

✕ **Bad practice**

```
const App = () => {

  for (let index = 0; index < 10; index++) {
    let [count, setCount] = React.useState(0);

  }

  return <h1>Do not call React hooks inside loops</h1>;
};
```

✕ **Bad practice**

```
const App = () => {

  if (true){
    let [count, setCount] = React.useState(0);
  }

  return <h1>Do not call React hooks inside conditions</h1>;
};
```

✅ Instead, Always use Hooks at the top level of a React function

✅ **Good practice**

```
const App = () => {
  const [count, setCount] = React.useState(0);
  React.useEffect(sideEffectCallback);
  const [person, setPerson] = React.useState({});

  // Loops, conditions, nested functions, etc...

  return <h1>Good example</h1>;
};
```

✕ Don’t call Hooks from regular JavaScript functions.

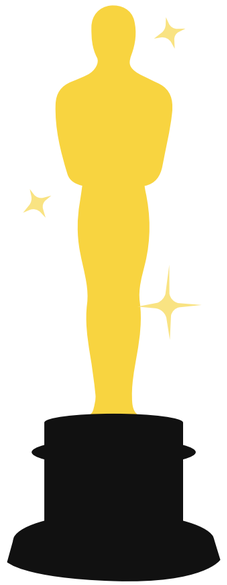
❌ **Bad practice**

```
function useCustomHook() {
  return [count, setCount] = React.useState(0);
}

function regularFunc() {
  const [count] = useCustomHook();
}
```

✅ Instead, Call hooks from React function component or Custom Hooks

✅ Good practice

```
function useCustomHook() {
  const [count, setCount] = React.useState(0);

  // on mount hook
  React.useEffect(() => {
    setInterval(() => {
      setCount(state => state + 1);
    }, 1000);
  }, []);

  return count;
}

const App = () => {
  const count = useCustomHook();

  // Loops, conditions, nested functions, etc...

  return <h1>Good example: {count}</h1>;
};
```



Most Famous/Used React Hooks

1. useState

- The React useState Hook allows us to track state in a function component.
- useState takes the ***initial state*** as its argument
- useState hook returns an array that contains two elements ***[currentState, setState]***

```
//Using a class component
class Message extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: '',
    };
  }
  render() {
    return <div>{this.state.message}</div>;
  }
}
//Using functional component and React Hooks
const Message = () => {
  const [message, setMessage] = useState('');

  return <div>{message}</div>;
};
```

2. useEffect

- The Effect Hook lets you perform side effects in function components.
- **Examples for Side effects**
  - Data Fetching
  - Manually changing DOM elements
  - Setting up subscription
- *useEffect* hook runs immediately after the component is mounted (similar to componentDidMount).
- *useEffect* will also run on changing any member of the dependency array. (similar to componentDidUpdate)

```
1  useEffect(()=>{
2      //do your side-effects here
3      return ()=>{
4          //clean up
5      }
6  },[dependencies])
```

useEffect Runtime

You can download this cheat sheet through this link

once

similar to componentDidMount

```
useEffect(() => {
  // put 'run once' code here
}, [])
```

pass empty array

on props change

similar to componentDidUpdate

```
function YourComponent({ someProp }) {
  useEffect(() => {
    // code to run when someProp changes
  }, [someProp]);
}
```

include all monitored props

after every render

similar to componentDidUpdate

```
useEffect(() => {
  // put 'every update' code here
})
```

no second argument

on state change

similar to componentDidUpdate

```
function YourComponent() {
  const [state, setState] = useState()
  useEffect(() => {
    // code to run when state changes
  }, [state])
}
```

include all state vars to watch

on unmount

similar to componentWillUmount

```
useEffect(() => {
  return () => {
    // put unmount code here
  }
})
```

return the cleanup function

daveceddia.com

useEffect Example

```react_example_1.js
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  },[count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}

```

3. useRef

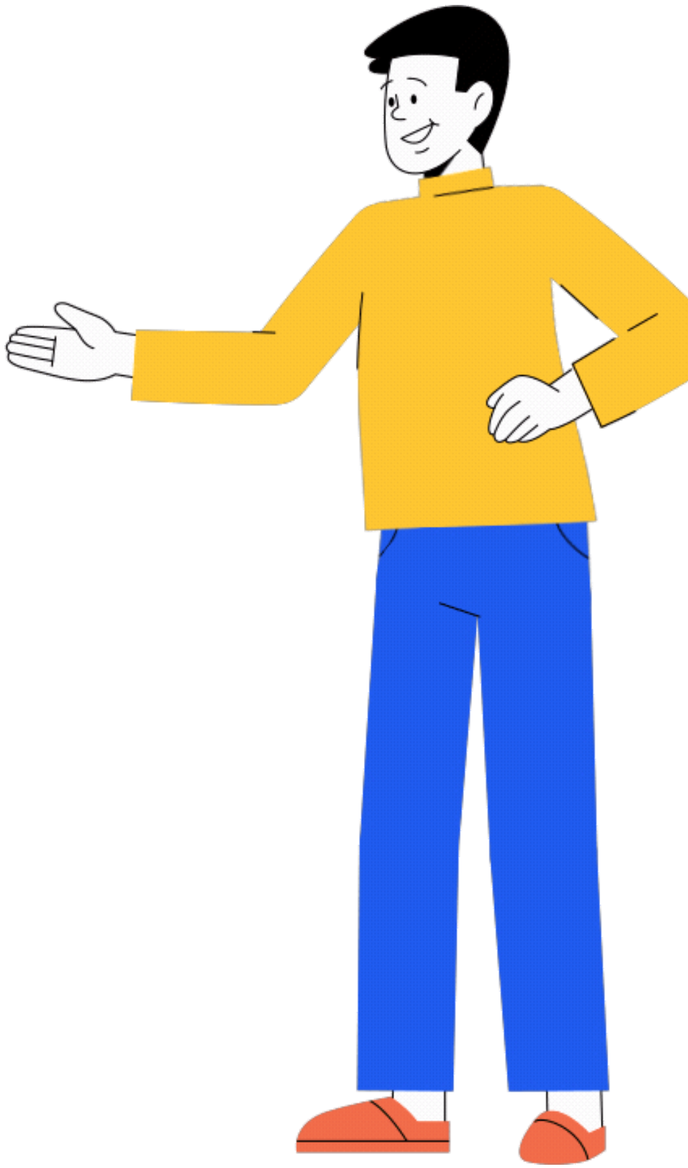
- **useRef** returns a mutable ref object whose ***.current*** property is initialized to the passed argument ***(initialValue)***.
- You can access the ref element by accessing ref.current property
- Mutating the ref's ***.current*** property does not cause a re-render
- ***useRef*** in a function component is similar to ***createRef*** in class component

useRef Example

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    // mutating the .current property doesn't cause a re-render
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type='text' />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );

```

Custom Hooks

Build your own hook right now



Custom Hooks

A custom Hook is a JavaScript function whose name starts with "use" and that may call other Hooks. and add some features like

1. Code Reusability

If you have the same logic being repeated a lot of components, Extract it to a custom hook and use it everywhere

2. Easier to Maintain ⚙

The code will be easier to maintain and fix in case of a bug or issue happening

3. Hooks are called in the same order

Using a custom hook will insure that the hooks (inside this custom hook) will run in the same order every render

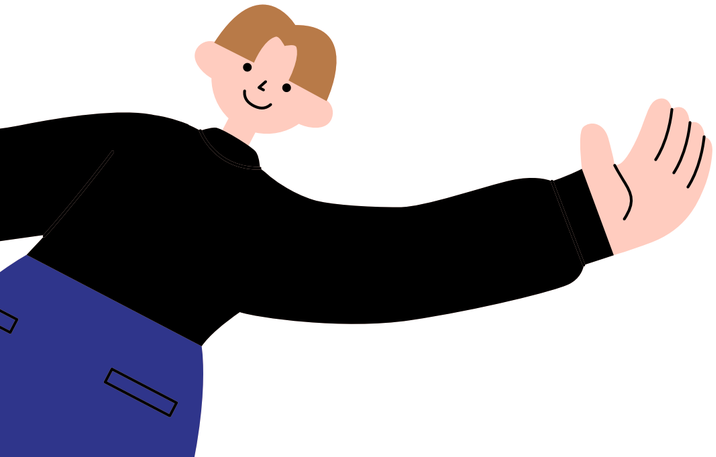
You can find so many pre-made custom hooks at [this link]

Its Demo Time

What is Next?

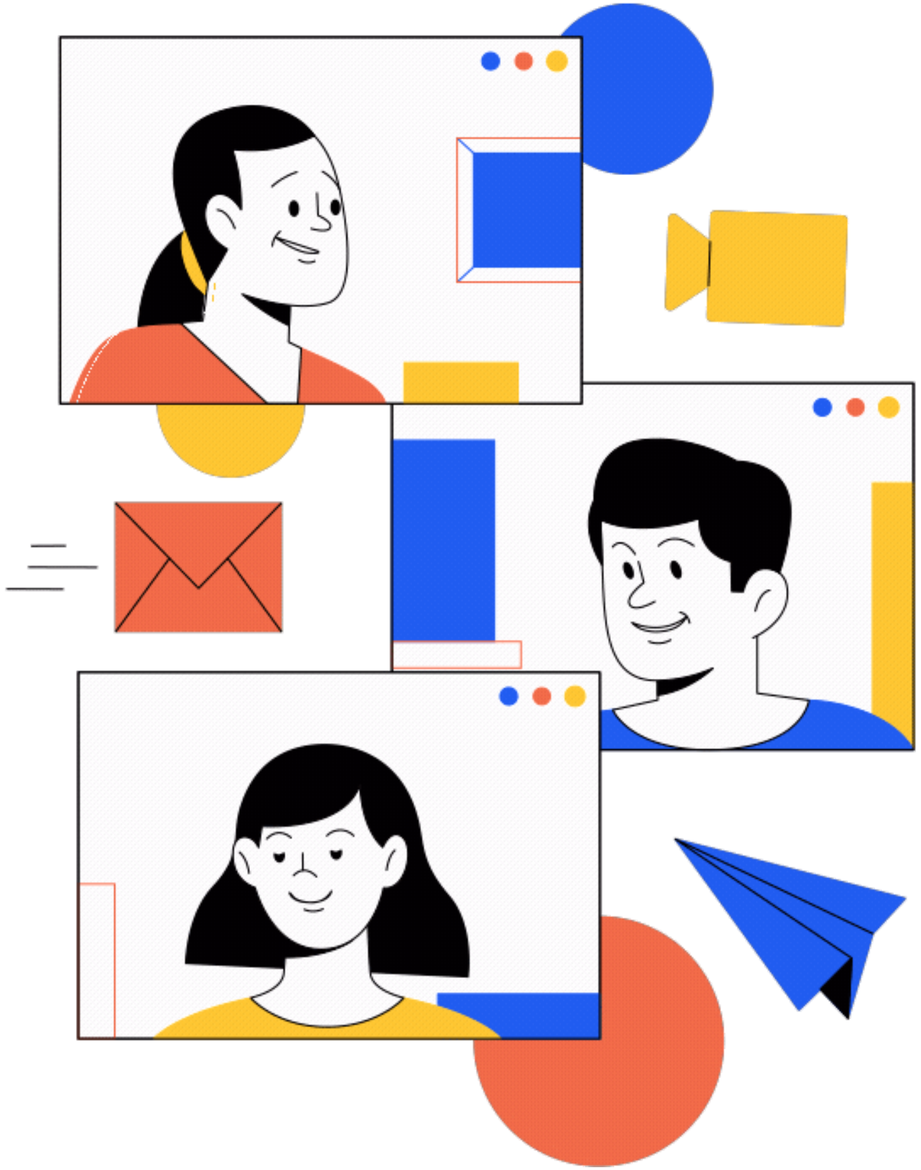
Where to go from here?

Any Questions?



Thank you!

For questions, requests and anything, please reach out to me on slack or email me at aghonem2011@gmail.com



Follow me on Github **@3ba2ii**

code and slides are found at this [github repo]