Context, Refs, memo, lazy, Suspense

```
// createContext
const WeatherContext = React.createCor
const App = ({ children }) => {
  const [weather, setWeather] = React.
  const [error, setError] = React.useS
  React.useEffect(() => {
    api.getWeather(...)
      .then(setWeather)
      .catch(setError)
  }, [])
  const contextValue = { weather, erro
  return (
    <WeatherContext.Provider value={co
      {children}
    </WeatherContext.Provider>
  )
}
const SomeChild = () => {
  const { weather } = React.useContext
  console.log(weather)
  return null
}

// createRef (Obtain a reference to a
const App = () => {
  const ref = React.createRef()
  React.useEffect(() => { console.log(
  return <div ref={ref} />
}

// forwardRef (Pass the ref down to a
const Remote = React.forwardRef((props
  <div ref={ref} {...props} />
))
const App = () => {
  const ref = React.createRef()
  return <Remote ref={ref} />
}

// memo (Optimize your components to d
const App = () => {...}
const propsAreEqual = (props, nextProp
  return props.id === nextProps.id
} // Does not re-render if id is the s
export default React.memo(App, propsAr

// lazy -> Dynamic import. Reduces bur
// + Code splitting
const MyComponent = React.lazy(() => i
const App = () => <MyComponent />

// Suspend rendering while components
// + Code splitting
import LoadingSpinner from '../Loading
const App = () => (
  <React.Suspense fallback={<LoadingSp
    <MyComponent />
  </React.Suspense>
)
```

Valid Return Types

```
const App = () => 'a basic strin
const App = () => 1234567890
const App = () => true
const App = () => null
const App = () => <div />
const App = () => <MyComponent /
const App = () => [
  'a basic string',
  1234567890,
  true,
  null,
  <div />,
  <MyComponent />,
]
```

Error

```
// Error boundary
class MyErrorBoundary extends R
  state = { hasError: false }
  componentDidCatch(error, info
  render() {
    if (this.state.hasError) re
    return this.props.children
  }
}
const App = () => (
  <MyErrorBoundary>
    <Main />
  </MyErrorBoundary>
)
```

Strict mode (detecting deprecations, side effects)

```
const App = () => (
```

Fragment

```
// Does not support key
const App = () => (
```

Hooks

```
// useState (Use over useReducer for basic sta
    const [state, setState] = React.useState(in
```

```
  <React.StrictMode>
    <div>
      <MyComponent />
      <OtherComponent />
    </div>
  </React.StrictMode>
)
```

```
  <>
    <MyComponent />
  </>
)
// Supports key attribut
const App = () => (
  <React.Fragment key="a
    <MyComponent />
  </React.Fragment>
)
```

```
// useEffect (Runs after components have mounte
    React.useEffect(() => {...}, [])
// useContext (Global state)
    const Context = React.createContext({ loade
    React.useContext(Context)
// useReducer (Use over useState for more compl
    const initialState = { loaded: false }
    const reducer = (state = initialState, act
    const [state, dispatch] = React.useReducer
      reducer,
      initialState
    )
// useCallback (Memoize functions)
    const handleClick = React.useCallback((e) =
// useMemo (Memoize values)
    import { compute } from '../utils'
    const memoize = React.useMemo(() => comput
// useRef
    const timeoutRef = React.useRef()
    timeoutRef.current = setTimeout(() => {...
// useImperativeHandle (Customizes an assigned
    const MyComponent = (props, ref) => {
      const inputRef = useRef(null)
      React.useImperativeHandle(ref, () => inpu
      return <input type="text" name="someName"
    }
// useLayoutEffect (Fires after all DOM mutati
    React.useLayoutEffect(() => {...}, [])
// useDebugValue
    React.useDebugValue(10)
```

Default Props

```
// Function component
const MyComponent = (props) =>
MyComponent.defaultProps = { f

// Class component
class MyComponent extends Reac
  static defaultProps = { frui
  render() { return <div {...t
}
```

Component States

```
// Class component state
class MyComponent extends React.Component
  state = { loaded: false }
  componentDidMount = () => this.setState
  render() {
    if (!this.state.loaded) return null
    return <div {...this.props} />
  }
}

// Function component state (useState/use
const MyComponent = (props) => {
  // With useState
  const [loaded, setLoaded] = React.useSt
  // With useReducer
  const [state, dispatch] = React.useRedu
  if (!loaded) return null
  React.useEffect(() => void setLoaded(tr
  return <div {...props} />
}
```

Importing Components

```
// default export
const App = (props) => <div {
export default App
import App from './App'

// named export
export const App = (props) =>
import { App } from './App'
```

Rendering Components

```
// Ways to render Card
const Card = (props) => <div {...

const App = ({ items = [] }) => {
  const renderCard = (props) => <
  return items.map(renderCard)
  // or return items.map((props)
}
```

Static Methods

```
// Returning object = New props requ
// Returning null = New props do not
class MyComponent extends React.Comp
  static getDerivedStateFromProps(pr
  state = {...}
}

// Return value is passed as 3rd arg
class MyComponent extends React.Comp
```

Pointer Events

```
onPointerUp             onPointerl
onPointerMove           onPointer
onGotPointerCapture     onLostPoi
onPointerEnter          onPointer
onPointerOver           onPointer

const App = () => {
  const onPointerDown = (e) =>
  return <div onPointerDown={onl
```

```
const App = (props) => <Card {...

class App extends React.Component
  render() { return <Card {...thi
}

const MyComp = ({ component: Comp
const App = () => <MyComp compone
```

```
  static getSnapshotBeforeUpdate(pre
}

// Listening to context from a class
import SomeContext from '../SomeCont
class MyCompmonent extends React.Com
  static contextType = SomeContext
  componentDidMount() { console.log(
}

// Enables rendering fallback UI bef
class MyComponent extends React.Comp
  state getDerivedStateFromError() {
  state = { error: null }
  componentDidCatch(error, info) {..
}
```

```
}
```

Test utils (act)

```
import { act } from 'react-dom/test-utils'
import MyComponent from './MyComponent'
const container = document.createElement('div')

// Synchronous
it('renders and adds new item to array', () => {
  act(() => {
    ReactDOM.render(<MyComponent />, container)
  })
  const btn = container.querySelector('button')
  expect(btn.textContent).toBe('one item')
  act(() => {
    button.dispatchEvent(new MouseEvent('click', { bubbles: true }))
  })
  expect(btn.textContent).toBe('two items')
})

// Asynchronous
it('does stuff', async () => {
  await act(async () => {
    // code
  })
})
```