Context, Refs, memo, lazy, Suspense

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

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

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

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

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

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

Valid Return Types

```
const App = () => 'a basic string'
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 Re
  state = { hasError: false }
  componentDidCatch(error, info)
  render() {
    if (this.state.hasError) ret
    return this.props.children
  }
}
const App = () => (
  <MyErrorBoundary>
    <Main />
  </MyErrorBoundary>
)
```

Strict mode (detecting deprecations, side effects)

```
const App = () => (
  <React.StrictMode>
```

Fragment

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

Hooks

```
// useState (Use over useReducer for basic state
    const [state, setState] = React.useState(ini
// useEffect (Runs after components have mounted
```

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

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

```
    React.useEffect(() => {...}, [])
// useContext (Global state)
    const Context = React.createContext({ loaded
    React.useContext(Context)
// useReducer (Use over useState for more comple
    const initialState = { loaded: false }
    const reducer = (state = initialState, actio
    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(() => compute(
// useRef
    const timeoutRef = React.useRef()
    timeoutRef.current = setTimeout(() => {...},
// useImperativeHandle (Customizes an assigned u
    const MyComponent = (props, ref) => {
      const inputRef = useRef(null)
      React.useImperativeHandle(ref, () => input
      return <input type="text" name="someName"
    }
// useLayoutEffect (Fires after all DOM mutation
    React.useLayoutEffect(() => {...}, [])
// useDebugValue
    React.useDebugValue(10)
```

Default Props

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

// Class component
class MyComponent extends React
  static defaultProps = { fruit
  render() { return <div {...th
}
```

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/useR
const MyComponent = (props) => {
  // With useState
  const [loaded, setLoaded] = React.useSta
  // With useReducer
  const [state, dispatch] = React.useReduc
  if (!loaded) return null
  React.useEffect(() => void setLoaded(tru
  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 {...pr

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

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

class App extends React.Component 
  render() { return <Card {...this
```

Static Methods

```
// Returning object = New props requi
// Returning null = New props do not 
class MyComponent extends React.Compo
  static getDerivedStateFromProps(pro
  state = {...}
}

// Return value is passed as 3rd argu
class MyComponent extends React.Compo
  static getSnapshotBeforeUpdate(prev
}

// Listening to context from a class 
```

Pointer Events

```
onPointerUp             onPointerDo
onPointerMove           onPointerCa
onGotPointerCapture     onLostPoint
onPointerEnter          onPointerLe
onPointerOver           onPointerOu

const App = () => {
  const onPointerDown = (e) => co
  return <div onPointerDown={onPo
}
```

```
}

const MyComp = ({ component: Compor
const App = () => <MyComp componen
```

```
import SomeContext from '../SomeConte
class MyCompmonent extends React.Comp
  static contextType = SomeContext
  componentDidMount() { console.log(t
}

// Enables rendering fallback UI befo
class MyComponent extends React.Compo
  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
  })
})
```