Context, Refs, memo, lazy, Suspense

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

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

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

// memo (Optimize your components t
const App = () => {...}
const propsAreEqual = (props, nextP
  return props.id === nextProps.id
} // Does not re-render if id is th
export default React.memo(App, prop

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

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

Valid Return Types

```
const App = () => 'a basic str
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 extend
  state = { hasError: false
  componentDidCatch(error, i
  render() {
    if (this.state.hasError)
    return this.props.childr
  }
}
const App = () => (
  <MyErrorBoundary>
    <Main />
  </MyErrorBoundary>
)
```

Strict mode (detecting deprecations, side effects)

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

Fragment

```
// Does not support ke
const App = () => (
  <>
    <MyComponent />
```

Hooks

```
// useState (Use over useReducer for basic s
    const [state, setState] = React.useState
// useEffect (Runs after components have mou
  React.useEffect(() => { ... }, [])
```

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

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

```
// useContext (Global state)
    const Context = React.createContext({ lo
    React.useContext(Context)
// useReducer (Use over useState for more co
    const initialState = { loaded: false }
    const reducer = (state = initialState, a
    const [state, dispatch] = React.useReduc
      reducer,
      initialState
    )
// useCallback (Memoize functions)
    const handleClick = React.useCallback((e
// useMemo (Memoize values)
    import { compute } from '../utils'
    const memoize = React.useMemo(() => comp
// useRef
    const timeoutRef = React.useRef()
    timeoutRef.current = setTimeout(() => {.
// useImperativeHandle (Customizes an assign
    const MyComponent = (props, ref) => {
      const inputRef = useRef(null)
      React.useImperativeHandle(ref, () => i
      return <input type="text" name="someNa
    }
// useLayoutEffect (Fires after all DOM muta
    React.useLayoutEffect(() => {...}, [])
// useDebugValue
    React.useDebugValue(10)
```

Default Props

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

// Class component
class MyComponent extends Re
  static defaultProps = { fr
  render() { return <div {..
}
```

Component States

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

// Function component state (useState/
const MyComponent = (props) => {
  // With useState
  const [loaded, setLoaded] = React.us
  // With useReducer
  const [state, dispatch] = React.useR
  if (!loaded) return null
  React.useEffect(() => void setLoaded
  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
}

const App = (props) => <Card {

class App extends React.Compone
  render() { return <Card {...
}
```

Static Methods

```
// Returning object = New props r
// Returning null = New props do
class MyComponent extends React.C
  static getDerivedStateFromProps
  state = {...}
}

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

// Listening to context from a cl
import SomeContext from '../SomeC
```

Pointer Events

```
onPointerUp            onPoint
onPointerMove          onPoint
onGotPointerCapture    onLostP
onPointerEnter         onPoint
onPointerOver          onPoint

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

```
const MyComp = ({ component: C
const App = () => <MyComp compo
```

```
class MyCompmonent extends React.
  static contextType = SomeContex
  componentDidMount() { console.l
}

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