Context, Refs, memo, lazy, Suspense

```
// createContext
const WeatherContext = React.createC
const App = ({ children }) => {
  const [weather, setWeather] = Reac
  const [error, setError] = React.us
  React.useEffect(() => {
    api.getWeather(...)
      .then(setWeather)
      .catch(setError)
  }, [])
  const contextValue = { weather, er
  return (
    <WeatherContext.Provider value={
      {children}
    </WeatherContext.Provider>
  )
}
const SomeChild = () => {
  const { weather } = React.useConte
  console.log(weather)
  return null
}

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

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

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

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

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

Valid Return Types

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

Strict mode (detecting deprecations, side effects)

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

Fragment

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

Hooks

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

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

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

```
    React.useEffect(() => {...}, [])
// useContext (Global state)
    const Context = React.createContext({ loa
    React.useContext(Context)
// useReducer (Use over useState for more com
    const initialState = { loaded: false }
    const reducer = (state = initialState, ac
    const [state, dispatch] = React.useReduce
      reducer,
      initialState
    )
// useCallback (Memoize functions)
    const handleClick = React.useCallback((e)
// useMemo (Memoize values)
    import { compute } from '../utils'
    const memoize = React.useMemo(() => compu
// useRef
    const timeoutRef = React.useRef()
    timeoutRef.current = setTimeout(() => {..
// useImperativeHandle (Customizes an assigne
    const MyComponent = (props, ref) => {
      const inputRef = useRef(null)
      React.useImperativeHandle(ref, () => in
      return <input type="text" name="someNam
    }
// useLayoutEffect (Fires after all DOM mutat
    React.useLayoutEffect(() => {...}, [])
// useDebugValue
    React.useDebugValue(10)
```

Default Props

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

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

Component States

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

// Function component state (useState/u
const MyComponent = (props) => {
  // With useState
  const [loaded, setLoaded] = React.use
  // With useReducer
  const [state, dispatch] = React.useRe
  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.Componer
  render() { return <Card {...th
}
```

Static Methods

```
// Returning object = New props re
// Returning null = New props do n
class MyComponent extends React.Co
  static getDerivedStateFromProps(
  state = {...}
}

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

// Listening to context from a cla
import SomeContext from '../SomeCo
```

Pointer Events

```
onPointerUp           onPointe
onPointerMove         onPointe
onGotPointerCapture   onLostPo
onPointerEnter        onPointe
onPointerOver         onPointe

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

```
const MyComp = ({ component: Com
const App = () => <MyComp compor
```

```
class MyCompmonent extends React.C
  static contextType = SomeContext
  componentDidMount() { console.lo
}

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