# React 19 Cheatsheet (WIP)

## Trends in React 19

- Encourages typescript
- Improves common struggles
- Enables performance enhancements
- Adds server APIs for realizing React's architecture vision
- Adds Client APIs essential for server-client interactions
- Adds Client APIs simplifying common use cases like mutating data by submitting forms, useful for any React app

## Migrating to React 19

Upgrade guide:
https://react.dev/blog/2024/04/25/react-19-upgrade-guide

### 1. (Install React 18.3)

React 18.3 is identical to 18.2, but includes warning about deprecated APIs and necessary changes.

```
npm install --save-exact react@18.3.0 react-dom@^18.3.0
```

### 2. Install React 19

Install React 19 with your preffered package manager.

```
npm install --save-exact react@^19.0.0 react-dom@^19.0.0
```

### 3. Run all codemods

Run the React 19 migration recipe, and the typscript migration codemod.

```
npx codemod@latest react/19/migration-recipe
npx types-react-codemod@latest preset-19 ./path-to-app
```

### 4. Remove deprecated functionality

Follow the React 19 upgrade guide to identify and remove any deprecated APIs and necessary changes.

```
// Deprecated
import PropTypes from 'prop-types';

function Heading({text}) {
return <h1>{text}</h1>;
}
Heading.propTypes = {
text: PropTypes.string,
};
Heading.defaultProps = {
text: 'Hello, world!',
};
```

### 5. Check that the app works!

Build your app, run your tests, and run the app. Check for errors in the console.
If all is well, congratulations! You're now in 19 :)

```
npm run build
npm run test
npm run dev
```

## React 19 features

Docs:
https://react.dev/reference/react

Release post:
https://react.dev/blog/2024/12/05/react-19

### Improvements

#### Async script support

Render async scripts anywhere in your component tree, with automatic deduplication.

```
<script async src="https://example.com/script.js"
```

#### Document metadata support

Automatically hoists <title>, <meta>, and <link> tags to the <head>.

```
<title>My blog</title>
<meta name="author" content="Aurora" />
```

#### Stylesheets with precedence

Support for inserting stylesheets with precedence in concurrent rendering environments.

```
<link rel="stylesheet" href="foo.css" precedence="default"
<link rel="stylesheet href="bar.css" precedence="high"
```

#### Resource preloading APIs

Preload resources like fonts, scripts, and styles to optimize performance.

```
preload('https://example.com/font.woff', { as: 'font' })
preconnect('https://example.com')
```

#### ref as a prop

Pass refs directly as props in function components, removing the need of forwardRef.

```
<MyInput ref={inputRef}
```

#### Ref callback cleanup

Ref callbacks can now return a cleanup function.

```
<input ref={(ref) => () => console.log('cleanup')} />
```

#### Streamlined context API

Use <Context> directly instead of <Context.Provider>

```
<LanguageContext value="nb-NO">{children}</LanguageContext>
```

#### useDeferredValue initial value

The useDeferredValue() hook now supports an initial value.

```
const deferredValue = useDeferredValue(value, 'initialValue')
```

#### Custom elements support

React now fully supports custom elements and handles properties/attributes consistently.

```
<custom-element prop1="value" />
```

#### Better error reporting

Automatically de-duplicates errors, and introduces onCaughtError and onUncaughtError handlers for root components.

```
onCaughtError: (error, errorInfo) => {
console.error(
'Caught error', error, errorInfo.componentStack
)
},
onUncaughtError: (error, errorInfo) => {
console.error(
'Uncaught error', error, errorInfo.componentStack
)
}
```

#### Hydration improvements

Improved error logging for hydration errors, providing a detailed diff when mismatches occur. In addition, unexpected tags in <head> and <body> are skipped during hydration to avoid mismatch errors.

```
Uncaught Error: Hydration failed
<App>
<span>
+ Client
- Server
```

### Server APIs

#### React Server Components

Server-rendered components that execute at build time or per request, and can import client-side code.

```
import { LanguagePicker } from './language-picker.tsx';

export async function Settings() {
const locales = await db.getLocales()
return (
<div>
<h1>Settings</h1>
<LanguagePicker locales={locales} />
</div>
)
}
```

#### "use client"

Marks code that can be referenced by the server component and can use client-side React features.

```
'use client'

import { saveLanguage } from './settings-actions.tsx';

export function LanguagePicker({locales}) {
const [locale, setLocale] = useState(currentLocale)

return (
<Select value={currentLocale} options={locales}
onSelect={async (newLocale) => {
setLocale(newLocale)
await saveLanguage(locale)
}}
/>
)
}
```

#### "use server"

Marks server-side functions callable from client-side code.

```
'use server'

export async function saveLanguage(locale) {
await db.updateLanguage(locale)
}
```

#### cache()

Cache the result of a data fetch or computation to reuse it across components, avoiding data coupling.

```
const cachedFn = cache(fn);
```

### Client APIs

*NEW: New understanding of actions. Ricky?*

#### Actions

Functions called inside the improved startTransition returned from useTransition() hook create Actions, and should be named accordingly.

Actions can be async and are useful for handling pending states, errors, optimistic updates, and sequential requests automatically. Subsequent actions after an async action must be wrapped in an additional startTransition.

```
const [isPending, startTransition] = useTransition();
const updateNameAction = () => {
startTransition(async () => {
const error = await updateName();
startTransition(() => {
setError(error)
})
})
})
```

#### <form/>

Extensions to the native HTML form element, including the improved action={} property which uses Actions by default.

```
<form action={deleteAction}>
<button type="submit">Delete</button>
</form>
```

#### useFormStatus()

Access information about of a parent form without prop drilling.

```
const { pending, data, method, action } = useFormStatus();
```

#### useActionState()

Update state based on the result of a form action and enable behaviors like automatic ordering and progressive enhancement.

```
const [state, formAction, isPending] = useActionState(fn, initialState, permalink?);
```

#### useOptimistic()

Show optimistic state while an async action is undeway. The dispatch function must be called inside the transition containing the action.

```
const [optimisticState, addOptimistic] = useOptimistic(state, updateFn);
```

TODO: why "add" and not "set" naming convention?

#### use()

Read resources like promises or context during render, useful for conditionally consuming a context or suspending "use client" components with a fallback. Does not support reading promises created at render.

```
const message = use(messagePromise);
const theme = use(ThemeContext);
```

This section is roughly based on the React 19 Cheat Sheet by Kent. C Dodds, but contains adjusted examples and explanations, and adds missing features.