PREACT

FAST 3KB ALTERNATIVE TO REACT WITH THE SAME MODERN API

PREACT : AN INTRODUCTION

THINNEST POSSIBLE VIRTUAL DOM

NO TRANSPILATION REQUIRED

SMALL SIZE

AUTOMATIC BATCH UPDATES

PORTABLE & EMBEDDABLE

REACT COMPATIBILITY

DIFFERENCE BETWEEN REACT AND PREACT

PREACT	REACT
Preact does not implement a synthetic event system – *onInput* should be used instead of *onChange*	Heavy usage of Synthetic event since the browsers have their own APIs implementations
Integration of "preact/debug“ package with *Preact Developers tool*	React Developer Tool is an isolated packages
Native support for ES Modules	Heavy toolkit for React code in browser
Arguments in Component.render(props, state)	State and Props managed internally by React. No explicit argument supplied.
Raw HTML attribute/property names eg. *class* instead of *className*	Does not differentiate HTML attributes in JSX Code

LET’S GET OUR HANDS DIRTY

1 Preact CDN

- Preact is packaged to be used directly in the browser
- Doesn't require any build or tools
- Include script type as module
- Refer preact CDN in JavaScript file - https://unpkg.com/preact?module

2 Preact CLI

- Preact CLI is an off-the-shelf solution for building Preact applications that is optimized for modern web development
- Built on standard tooling projects like Webpack, Babel and PostCSS
- Does not require any configuration

WORKING WITH PREACT

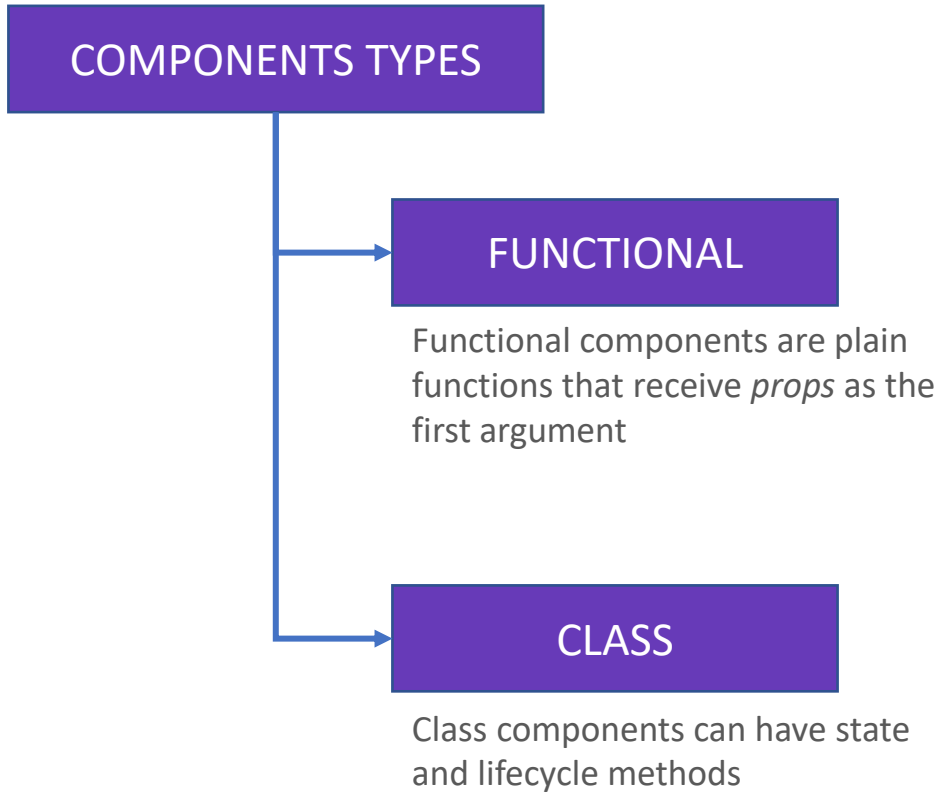
COMPONENTS, HOOKS, FORMS, REFERENCES AND MORE

COMPONENTS

Components represent the basic building block in Preact

Components are fundamental in making it easy to build complex UIs from little building blocks

Responsible for attaching state to the rendered output



HOOKS

The hooks API makes it possible to neatly extract the logic for state and side effects, and also simplifies unit testing that logic independently from the components that rely on it

useState	This hook accepts an argument, this will be the initial state. When invoked this hook returns an array of two variables. The first being the current state and the second being the setter for our state
useReducer	Compared to useState it's easier to use when you have complex state logic where the next state depends on the previous one
useMemo	With the hook we can memoize the results of that computation and only recalculate it when one of the dependencies changes
useCallback	The hook can be used to ensure that the returned function will remain referentially equal for as long as no dependencies have changed

MORE HOOKS

useRef	To get a reference to a DOM node inside a functional components there is the
useContext	To access context in a functional component we can use the hook, without any higher-order or wrapper components
useEffect	is the main way to trigger various side-effects. You can even return a cleanup function from your effect if one is needed
useErrorBoundary	Whenever a child component throws an error you can use this hook to catch it and display a custom error UI to the user

WORKING WITH FORMS

Forms in Preact work much the same as they do in HTML. You render a control, and attach an event listener to it

Uncontrolled Components - every form control will manage the user input themselves

Controlled Components - The component doesn't manage the value itself there, but something else higher up in the component tree

WORKING WITH REFS

Refs allow you to direct reference to the DOM-Element or Component that was rendered by Preact.

createRef	The createRef function will return a plain object with just one property: current
	Whenever the render method is called, Preact will assign the DOM node or component to current
Callback Refs	Another way to get the reference to an element can be done by passing a function callback

CONTEXT API

Context allows you to pass a value to a child deep down the tree without having to pass it through every component in-between via props

Context can be thought of a way to do pub-sub-style updates in Preact

createContext(initialValue) function returns a Provider component that is used to set the context value and a Consumer which retrieves the value from the context

TIPS & TRICKS

Code splitting and lazy loading
Working with Portal
React Compatibility layer - preact/compat
Use of useCallback and useMemo Hooks
Accessing real DOM, if required