* React Hooks
  + Hooks let you use different React features from your components. You can either use the built-in Hooks or combine them to build your own. This page lists all built-in Hooks in React.
  + State Hooks
    - State lets a component “remember” information like user input. For example, a form component can use state to store the input value, while an image gallery component can use state to store the selected image index.
    - useState
      * declares a state variable that you can update directly.
      * example: ‘’
    - useReducer
      * declares a state variable with the update logic inside a reducer function.
      * example: ‘’
  + Context Hooks
    - Context lets a component receive information from distant parents without passing it as props. For example, your app’s top-level component can pass the current UI theme to all components below, no matter how deep.
      * useContext
      * reads and subscribes to a context.
      * example: ‘’
  + Ref Hooks
    - Refs let a component hold some information that isn’t used for rendering, like a DOM node or a timeout ID. Unlike with state, updating a ref does not re-render your component. Refs are an “escape hatch” from the React paradigm. They are useful when you need to work with non-React systems, such as the built-in browser APIs.
    - useRef
      * declares a ref. You can hold any value in it, but most often it’s used to hold a DOM node.',
      * example: ‘’
    - useImperativeHandle
      * lets you customize the ref exposed by your component. This is rarely used.',
      * example: ‘’
  + Effect Hooks
    - Effects let a component connect to and synchronize with external systems. This includes dealing with network, browser DOM, animations, widgets written using a different UI library, and other non-React code.
    - useEffect
      * connects a component to an external system.
    - useLayoutEffect
      * fires before the browser repaints the screen. You can measure layout here.
      * example: ‘’
    - useInsertionEffect
      * fires before React makes changes to the DOM. Libraries can insert dynamic CSS here.
      * example: ‘’
  + Performance Hooks
    - A common way to optimize re-rendering performance is to skip unnecessary work. For example, you can tell React to reuse a cached calculation or to skip a re-render if the data has not changed since the previous render.
    - useMemo
      * lets you cache the result of an expensive calculation.
      * example: ‘’
    - useCallback
      * lets you cache a function definition before passing it down to an optimized component.
      * example: ‘’
    - useTransition
      * lets you mark a state transition as non-blocking and allow other updates to interrupt it.
      * example: ‘’
    - useDeferredValue
      * lets you defer updating a non-critical part of the UI and let other parts update first.
      * example: ‘’
  + Resource Hooks
    - Resources can be accessed by a component without having them as part of their state. For example, a component can read a message from a Promise or read styling information from a context.
    - use
      * use lets you read the value of a resource like a Promise or context.
      * example: ‘’