**Solid Primitives**

The basic building blocks of SolidJS:

1. createSignal:

* the most fundamental primitive
* purpose is to build a reactive state variable
* an initial value is optional here
* it returns a tuple having 2 values (getter() to receive value, setter() to set received value)
* (count, setCount)
* Can be used with any data-type

CODE PERFORMED ON SOLIDJS PLAYGROUND

CODE:

|  |
| --- |
| import {render} from "solid-js/web";  import {createSignal} from "solid-js";  const [count,setCount]=createSignal(0);//starting point is set for count  function increment(){    setCount(count()+1);//on pressing button to increment, init no. will be incremented by 1  }  function decrement(){    setCount(count()-1);//on pressing button to decrement, init no. will be decremented by 1  }  function Counter()  {    return(<div>    <button onClick={decrement}>-</button>    <span>{count()//value of number displayed and changed    }    </span>     <button onClick={increment}>+</button>    </div>);  }   render(()=><Counter/>,document.getElementById("app")!)//to call the function for rendering on o/p |

OUTPUT:

|  |
| --- |
|  |

1. createEffect:

* purpose is to build a signal tied with an asynchronous function

CODE:

|  |
| --- |
| import {render} from "solid-js/web";  import {createSignal, createEffect} from "solid-js";  const [count,setCount]=createSignal(0);  createEffect(()=>{console.log("Count has been updated: ",count());});  function Counter()  {    return(<div>    <button onClick={()=>{setCount(count()-1)}}>-</button>    <span>{count()//value of number displayed and changed    }    </span>     <button onClick={()=>{setCount(count()+1)}}>+</button>    </div>);  }   render(()=><Counter/>,document.getElementById("app")!)//to call the function for rendering on o/p |

OUTPUT:

|  |
| --- |
|  |

1. createMemo:

to create a memoized value calculated by a function defined by developer

CODE:

|  |
| --- |
| import {render} from "solid-js/web";  import {createSignal, createMemo} from "solid-js";  const [count,setCount]=createSignal(0);//define count and update it using setCount  const dc=createMemo(()=>count()\*2);//creates a memoized value=val\*2  function Counter()  {    return(<div>    <p>Count:{count()}</p>    <p>DoubleCount:{dc()}</p>    <button onClick={()=>{setCount(count()-1)}}>-</button>     <button onClick={()=>{setCount(count()+1)}}>+</button>    </div>);  }   render(()=><Counter/>,document.getElementById("app")!)//to call the function for rendering on o/p |

OUTPUT:

|  |
| --- |
|  |

WHAT IS MEMOIZATION?

IT’S A CACHING TECHNIQUE.

In programming, memoization is an optimization technique that makes applications more efficient and hence faster. It does this by storing computation results in cache, and retrieving that same information from the cache the next time it's needed instead of computing it again.

1. createRoot:

to create a root element and mount the SolidJS application inside it.

CODE:

|  |
| --- |
| import { render } from "solid-js/web";  import {createSignal, createRoot} from "solid-js";  function App(){  const [count,setCount]=createSignal(0);//define count and update it using setCount    return(<div id="root">      <p>Count:{count()}</p>    <button onClick={()=>{setCount(count()-1)}}>-</button>     <button onClick={()=>{setCount(count()+1)}}>+</button>    </div>);  }  createRoot(()=>{    const rootElement=document.getElementById("root");    if (rootElement)    {  //Mount App component inside Root component  render(()=><App/>,rootElement)    }  }) |

OUTPUT:

|  |
| --- |
| N/A |

1. createRenderEffect:

to create a render effect that runs after the component renders.

CODE:

|  |
| --- |
| import { render } from "solid-js/web";  import {createSignal, createRenderEffect} from "solid-js";  function App(){  const [count,setCount]=createSignal(0);//define count and update it using setCount  createRenderEffect(()=>{console.log("RenderEffect triggered: ",count())});    return(<div>      <button onClick={()=>{setCount(count()-1)}}>-</button>     <p>Count:{count()}</p>     <button onClick={()=>{setCount(count()+1)}}>+</button>    </div>);  }   render(()=><App/>,document.getElementById("app")!)//to call the function for rendering on o/p |

OUTPUT:

|  |
| --- |
|  |

1. createDeferred

Creates a readonly that only notifies downstream changes when the browser is idle. timeoutMs is the maximum time to wait before forcing the update.

|  |
| --- |
| import { createDeferred } from "solid-js";  function createDeferred<T>(  source: () => T,  options?: {  timeoutMs?: number;  equals?: false | ((prev: T, next: T) => boolean);  }  ): () => T; |

1. createComputed:

Creates a reactive computation that runs immediately before render, mainly used to write to other reactive primitives

CODE:

|  |
| --- |
| import { render } from "solid-js/web";  import { createSignal,createComputed } from "solid-js";    function App() {    const [count, setCount] = createSignal(0);    const dc = createComputed(() => count() \* 2);    return (      <div>        <p>Count: {count()}</p>        <p>Doubled Count: {dc()}</p>        <button onClick={() => setCount(count() + 1)}>Increment</button>        <button onClick={() => setCount(count() - 1)}>Decrement</button>      </div>    );  }  render(() => <App />, document.getElementById('app'),); |

OUTPUT:

|  |
| --- |
|  |

USE OF createContext and createStore to make a color-picker

|  |
| --- |
| Theme.tsx  import { createContext, useContext } from "solid-js";  //createContext= used to create a context object  //useContext= hook to access the context value  import {createStore} from 'solid-js/store' //used to manage State in solidjs  export type ThemeContextState={      readonly color:string;      readonly title:string;  };  //this is an object with 2 properties, both being string type called color and title  export type ThemeContextValue=[      state:ThemeContextState,//type of state is ThemeContextState      actions:{          changeColor:(color:string)=>void;//objects that take string type arg and return void          changeTitle:(title:string)=>void;      }  ];//array storing state and actions related to the theme context==== array of objects  const defaultState={      color:"#00ff00",      title:"fallback title",  }//defines default state for ThemeContext  const ThemeContext=createContext<ThemeContextValue>([      defaultState,      {  changeColor:()=> undefined,  changeTitle:()=> undefined,      },  ]);//createContext is used to create theme context object. receives generic type ThemeContextValue and init it with  //default state and actions  export const ThemeProvider:ParentComponent<{color?:string; title?:string}>=(props)=>{const [state,setState]=createStore({      color:props.color??defaultState.color,      title:props.title??defaultState.title,  });//parent component provides theme context to children === optional parameters like color and title received  //createStore to store init state  const changeColor=(color:string)=>setState("color",color);  const changeTitle=(title:string)=>setState("title",title);  return(      <ThemeContext.Provider value={[state,{changeColor,changeTitle}]}>{props.children}</ThemeContext.Provider>  );  }  export const useTheme=()=>useContext(ThemeContext)//export custom hook to be used by main  main.tsx  import { render } from "solid-js/web";  import { ThemeProvider, useTheme } from "./theme";  function App() {    const [theme, { changeColor , changeTitle }] = useTheme();    return (      <>        <h1          style={{            color: theme.color,          }}        >          {theme.title}        </h1>        <h2 style={{            color: theme.color,          }}>Change color</h2>        <input          type="color"          name="color"          value={theme.color}          onInput={(e) => changeColor(e.currentTarget.value)}        />        <br/>         <h2 style={{            color: theme.color,          }}>Change text</h2>        <input          type="title"          name="title"          value={theme.title}          onInput={(e) => changeTitle(e.currentTarget.value)}        />      </>    );  }  render(    () => (      <ThemeProvider color="#ff00ff" title="Check">        <App />      </ThemeProvider>    ),    document.getElementById("app")!  ); |

Output

|  |
| --- |
|  |