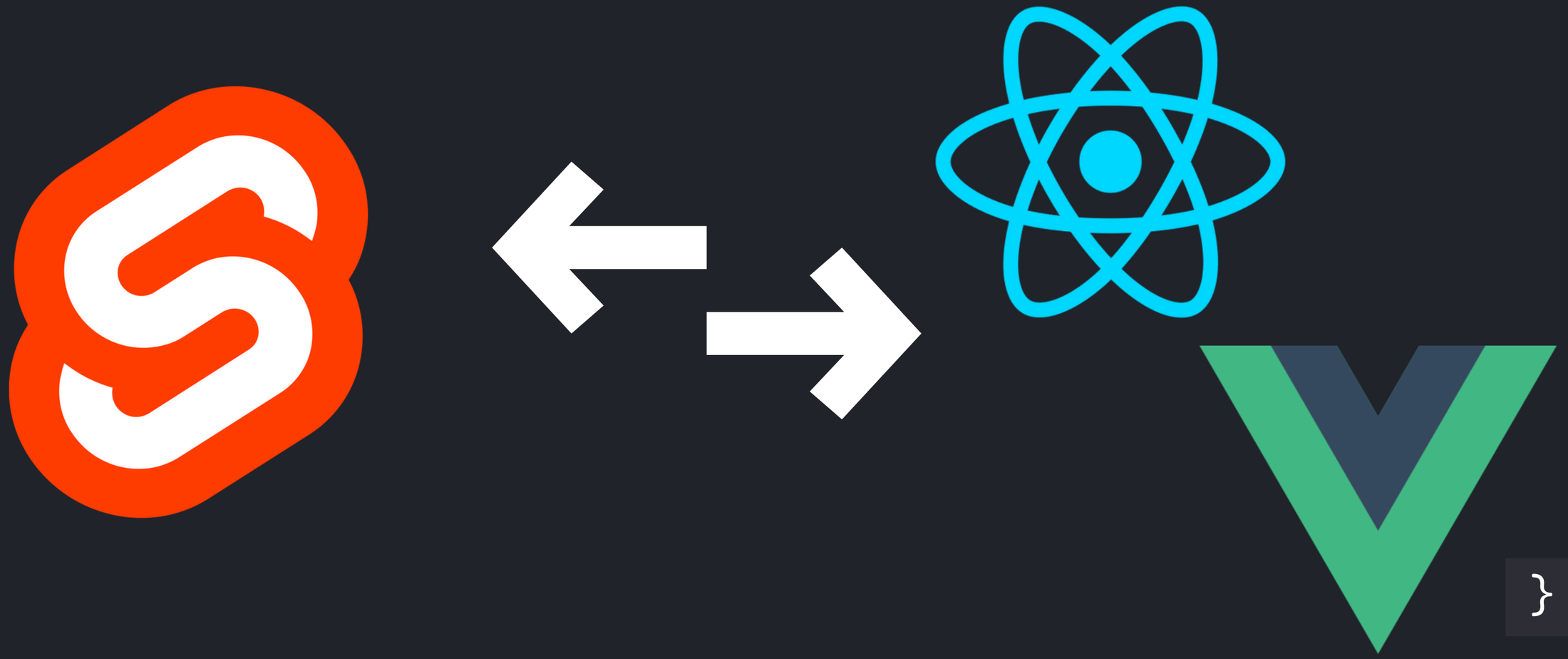
Tutorial Svelte{

```
<Por="Daniel Bonilla"/>
<Por="Ricardo Veloza"/>
<Por="Julian Sanchez”/>
<Por="Daniel Galindo"/>
```

}

¿Qué es Svelte? {

SVELTE ES UN FRAMEWORK FRONTEND. PERO, ¿QUÉ LO DIFERENCIA DE OTROS FRAMEWORKS/LIBRERIAS COMO REACT O VUE?



}

Tiempo de Ejecución vs Tiempo de Compilación{

FRAMEWORKS COMO REACT Y VUE REALIZAN GRAN PARTE DEL TRABAJO EN EL NAVEGADOR DEL USUARIO MIENTRAS SE EJECUTA LA APLICACIÓN, SVELTE TRASLADA ESA CARGA DE TRABAJO A UN PROCESO DE COMPILACIÓN QUE SE EJECUTA AL CONSTRUIR LA APLICACIÓN.

APLICACIONES MENOS PESADAS Y MÁS RÁPIDAS.

}

Simplicidad y Familiaridad {

SVELTE UTILIZA HTML, JAVASCRIPT Y CSS CON POCAS EXTENSIONES. LA CURVA DE APRENDIZAJE ES CORTA, YA QUE INTRODUCE POCOS CONCEPTOS NUEVOS Y PERMITE TRABAJAR CON LENGUAJES FAMILIARES.

Svelte 4

Name.svelte

```svelte
<script>
  let name = "John";
  name = "Jane";
</script>

<h1>Hello {name}</h1>
```

React

Name.jsx

```jsx
import { useEffect, useState } from "react";

export default function Name() {
  const [name, setName] = useState("John");

  useEffect(() => {
    setName("Jane");
  }, []);

  return <h1>Hello {name}</h1>;
}
```

}

HTTPS://COMPONENT-PARTY.DEV/

Adiós al virtual DOM{

A DIFERENCIA DE OTROS FRAMEWORKS, SVELTE NO UTILIZA EL VIRTUAL DOM, QUE PUEDE SER COSTOSO EN APLICACIONES COMPLEJAS. COMO COMPILADOR, SVELTE OPTIMIZA Y ACTUALIZA EL DOM DIRECTAMENTE DURANTE LA COMPILACIÓN, LOGRANDO UNA MAYOR EFICIENCIA.

}

¿Qué tiene Svelte con la reactividad y el paradigma de programación reactiva? {

LA REACTIVIDAD ES CLAVE EN SVELTE. LAS INTERFACES SE ACTUALIZAN AUTOMÁTICAMENTE EN RESPUESTA A CAMBIOS EN EL ESTADO DE LA APLICACIÓN. DECLARAS QUÉ VARIABLES SON REACTIVAS Y SVELTE ACTUALIZA LA INTERFAZ DE MANERA DECLARATIVA, SIMPLIFICANDO EL DESARROLLO.

}

Hola Mundo en Svelte {

SVELTE UTILIZA EL PRINCIPIO DE SINGLE FILE COMPONENT. ESTO ES, QUE EN UN MISMO ARCHIVO, TIENES UN COMPONENTE CON SU MARCADO (HTML), SU FUNCIONALIDAD (JAVASCRIPT) Y SU ESTILO (CSS).

LINK EJEMPLO

}

Añadiendo estilos CSS {

```svelte
<script>
  let name = 'world';
  // Comentario
</script>

<h1>Hello {name}!</h1>
<!— Comentario—>

<style>
  h1 {
    color: #09f;
  }
  /* Comentario */
</style>
```

LINK EJEMPLO

}

El Estado{

EL "ESTADO" SE REFIERE A CUALQUIER DATO QUE PUEDE CAMBIAR EN UNA APLICACIÓN, COMO EL CONTENIDO DE UN FORMULARIO O UN CONTADOR DE CLICS. SVELTE, COMO COMPILADOR, DETERMINA AUTOMÁTICAMENTE SI UNA VARIABLE ES PARTE DEL ESTADO DEL COMPONENTE, SIMPLIFICANDO SU GESTIÓN.

```
<script>
  let name = 'world';

  setTimeout(function () {
    name = 'Svelte'
  }, 3000)
</script>

<h1>Hello {name}!</h1>

<style>
  h1 {
    color: #09f;
  }
</style>
```

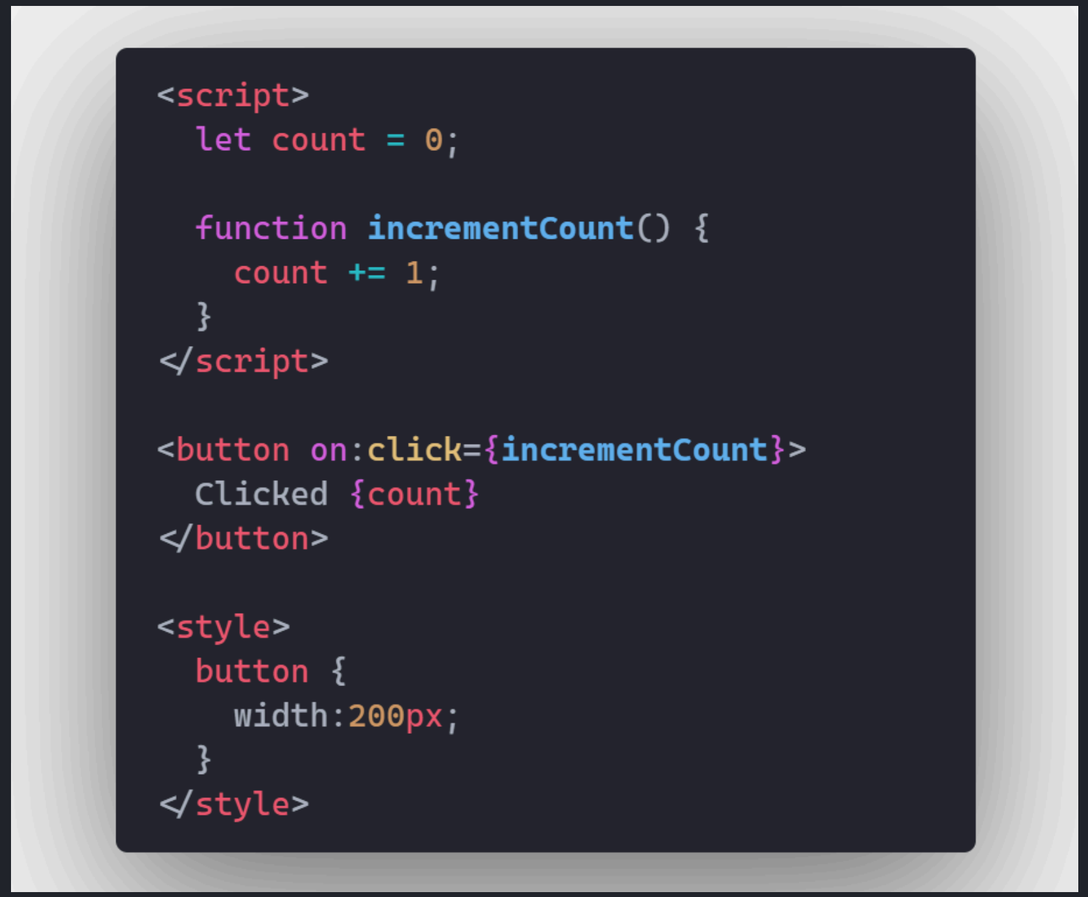
LINK EJEMPLO

}

Reactividad en Svelte{

EN SVELTE, PARA DECLARAR UN ESTADO SÓLO TENEMOS QUE DECLARAR UNA VARIABLE USANDO LET Y ASIGNARLE EL VALOR INICIAL QUE TIENE. DESPUES, PODREMOS CAMBIAR EL VALOR DE ESTA VARIABLE Y SVELTE SE ENCARGARÁ DE ACTUALIZAR LA UI PARA REFLEJAR LOS CAMBIOS.

LINK EJEMPLO



```
<script>
  let count = 0;

  function incrementCount() {
    count += 1;
  }
</script>

<button on:click={incrementCount}>
  Clicked {count}
</button>

<style>
  button {
    width:200px;
  }
</style>
```

}

¿Qué pasa si queremos que una variable dependa de otra que es reactiva? {

EN SVELTE, EL OPERADOR $ SE UTILIZA PARA INDICAR QUE UNA EXPRESIÓN ES REACTIVA. POR EJEMPLO, AL USAR $MESSAGE, SVELTE DETECTARÁ QUE LA EXPRESIÓN DEPENDE DE LA VARIABLE COUNT Y LA ACTUALIZARÁ AUTOMÁTICAMENTE CADA VEZ QUE COUNT CAMBIE.

LINK EJEMPLO

```
<script>
  let count = 0;
  //let message = count % 2 === 0 ? 'Es par' : 'Es impar'
  $:  message = count % 2 === 0 ? 'Es par' : 'Es impar'

  function incrementCount() {
    count += 1;
  }
</script>

<!— //<span> { count % 2 === 0 ? 'Es par' : 'Es impar' }
</span> —>
<span> { message }</span>
<button on:click={incrementCount}>
  Clicked {count}
</button>


<style>
  button {
    width:200px;
  }
</style>
```

}

Componentes {

HOY EN DÍA, TODOS LOS FRAMEWORKS DEL FRONTEND ESTÁN BASADOS EN LA COMPONETIZACIÓN DE LA INTERFAZ.ESTO SIGNIFICA QUE LA UI SE DIVIDE EN PEQUEÑOS COMPONENTES PARAMETRIZABLES QUE SE PUEDEN REUTILIZAR Y COMPONER PARA CREAR LA INTERFAZ FINAL.

LINK EJEMPLO

```
<!— Button.svelte—>
<script>
  import Button from './Button.svelte'
  let name = 'world';
</script>

<h1>Hello {name}!</h1>
<Button />
```

}

Props {

LAS PROPS PERMITEN QUE LOS COMPONENTES SEAN REUTILIZABLES AL CAMBIAR SU COMPORTAMIENTO DESDE FUERA. PARA PASAR UNA PROP, SIMPLEMENTE LA AÑADIMOS AL COMPONENTE, COMO TEXT EN UN BOTÓN. SVELTE USA EXPORT LET PARA RECUPERAR LAS PROPS, Y TAMBIÉN PERMITE DEFINIR VALORES POR DEFECTO.

LINK EJEMPLO

```
<!— app.svelte —>
<script>
  import Button from './Button.svelte'
  let name = 'world';
</script>

<h1>Hello {name}!</h1>

<Button/>
<Button text="Enviar" />
<Button text="Cancelar" />
```

}

Props {

```svelte
<!-- button.svelte -->
<script>
  export let text = 'Click me'
</script>

<button> { text }</button>

<style>
  button {
    color: skyblue;
  }
</style>
```

LINK EJEMPLO

}

Spread Props {

CUANDO UN COMPONENTE RECIBE MUCHAS PROPS, SVELTE PERMITE PASAR TODAS LAS PROPS COMO UN OBJETO USANDO LA SINTAXIS DE SPREAD PROPS. ESTO FACILITA EL MANEJO DE MÚLTIPLES PROPIEDADES SIN TENER QUE PASARLAS INDIVIDUALMENTE.

LINK EJEMPLO

```
<!-- app.svelte -->
<script>
  import Button from
'./Button.svelte'
  let name = 'world';

  const props = {
    type: 'submit',
    text: 'Enviar',
    disabled: true
  };

</script>

<h1>Hello {name}!</h1>

<Button/>
<Button {...props} />
```

}

Spread Props {

```svelte
<!-- button.svelte -->
<script>
  export let type = 'button';
  export let text = 'Click me';
  export let disabled = false;
</script>

<button type={type} disabled={
disabled}>
  {text}
</button>

<style>
  button {
    color: skyblue;
  }
</style>
```

LINK EJEMPLO

}

Lógica {

SVELTE FACILITA LA IMPLEMENTACIÓN DE LÓGICA A TRAVÉS DEL USO DE JAVASCRIPT ESTÁNDAR.

ESTA FUNCIONALIDAD SE PUEDE IMPLEMENTAR A TRAVÉS DEL USO DE LA ETIQUETA SCRIPT O CON LA IMPLEMENTACIÓN DE MÓDULOS DE JAVASCRIPT

LINK EJEMPLO

```
<script>
  let a = 5;
  let b = 3;
  let result = a + b;

  function calculateSum(x, y) {
    return x + y;
  }

  let sum = calculateSum(10, 20);
</script>

<p>La suma de {a} y {b} es {result}</p>
<p>La suma calculada es {sum}</p>
```

}

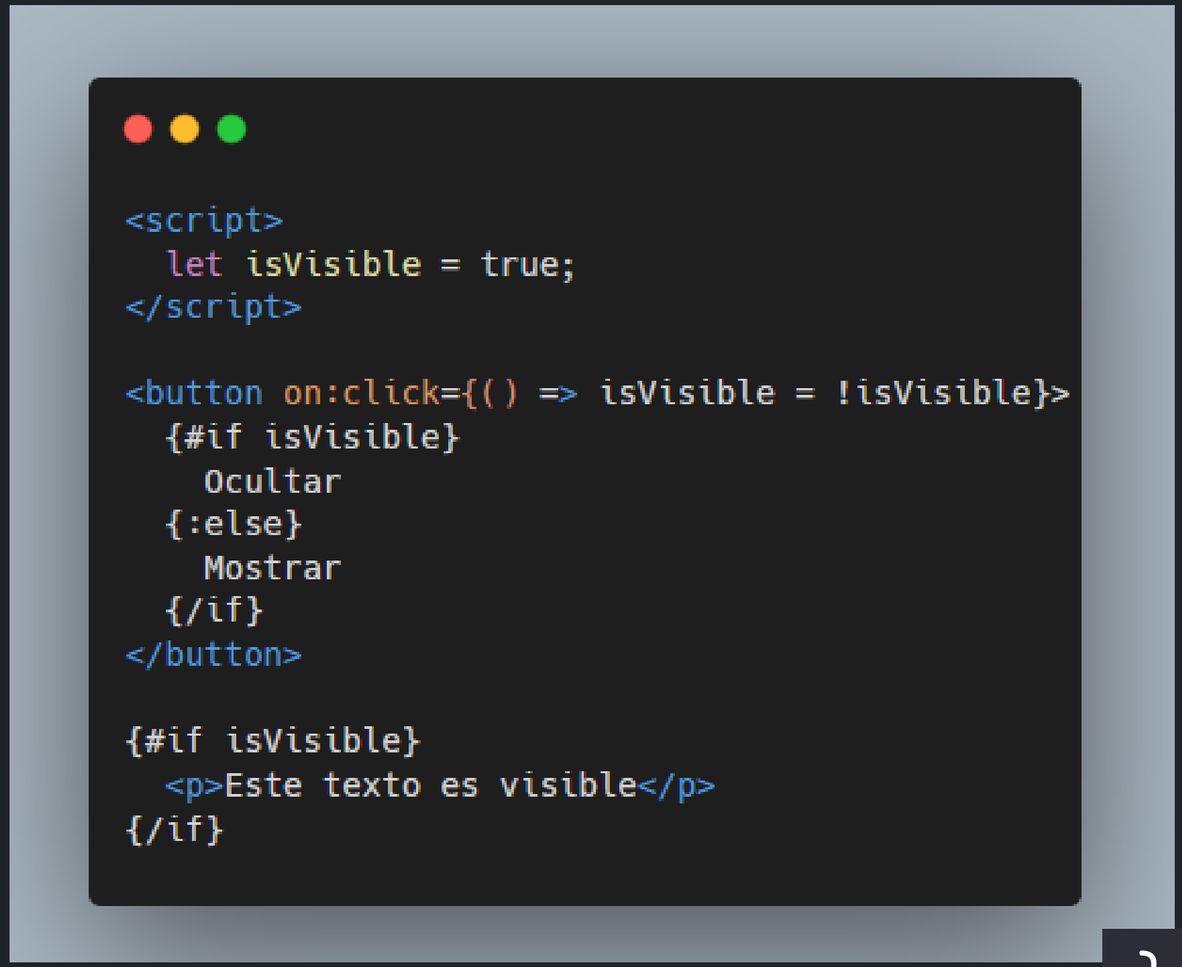
Lógica Reactiva {

EN SVELTE EXISTE UNA ALTERNATIVA A LA SINTÁXIS CLÁSICA DE JAVASCRIPT PARA IMPLEMENTAR LÓGICA RELACIONADA A LOS BUCLES Y LOS CONDICIONALES DE MANERA REACTIVA.

ESTA SE DA A TRAVÉS DEL USO DE LA SINTÁXIS {#IF} O {#EACH} PARA CADA CASO. CUANDO SU FUENTE DATOS CAMBIA, SVELTE ACTUALIZA LOS ELEMENTOS REINDERIZADOS DE FORMA AUTOMÁTICA.

LINK EJEMPLO

LINK EJEMPLO



```svelte
<script>
  let isVisible = true;
</script>

<button on:click={() => isVisible = !isVisible}>
  {#if isVisible}
    Ocultar
  {:else}
    Mostrar
  {/if}
</button>

{#if isVisible}
  <p>Este texto es visible</p>
{/if}
```

}

Ejemplo uso de lógica

PARA EJEMPLIFICAR DE MANERA UN POCO MÁS COMPLEJA EL USO DE LA LÓGICA EN SVELTE CREAREMOS UN CAMBIO DE DIVISAS.

LINK EJEMPLO

```
<script>
  export let rates = {};
  export let baseCurrency = 'USD';
</script>

<ul>
  {#each Object.entries(rates) as [currency, rate]}
    <li>{baseCurrency} = {rate.toFixed(2)} {currency}</li>
  {/each}
</ul>
```

}

Eventos {

SVELTE FACILITA EL MANEJO DE EVENTOS PROPORCIONANDO UNA SINTAXIS CLARA Y DIRECTA. UTILIZANDO LA DIRECTIVA ON:, PUEDES ADJUNTAR FÁCILMENTE CONTROLADORES DE EVENTOS A LOS ELEMENTOS DEL DOM.

EN SVELTE SE PUEDE PASAR PARÁMETROS A LOS MANEJADORES DE EVENTOS MEDIANTE FUNCIONES FLECHA

LINK EJEMPLO

```
<script>
  let count = 0;

  function handleClick() {
    count += 1;
  }
</script>

<button on:click={handleClick}>
  Contador: {count}
</button>
```

}

Reenvío de eventos{

SVELTE PERMITE QUE UN COMPONENTE HIJO NOTIFIQUE A SU COMPONENTE PADRE O A OTROS COMPONENTES SOBRE EVENTOS ESPECÍFICOS QUE HAN OCURRIDO. ESTO SE LOGRA UTILIZANDO LA FUNCIÓN CREATEEVENTDISPATCHER PARA CREAR Y DISPARAR EVENTOS PERSONALIZADOS.

LINK EJEMPLO

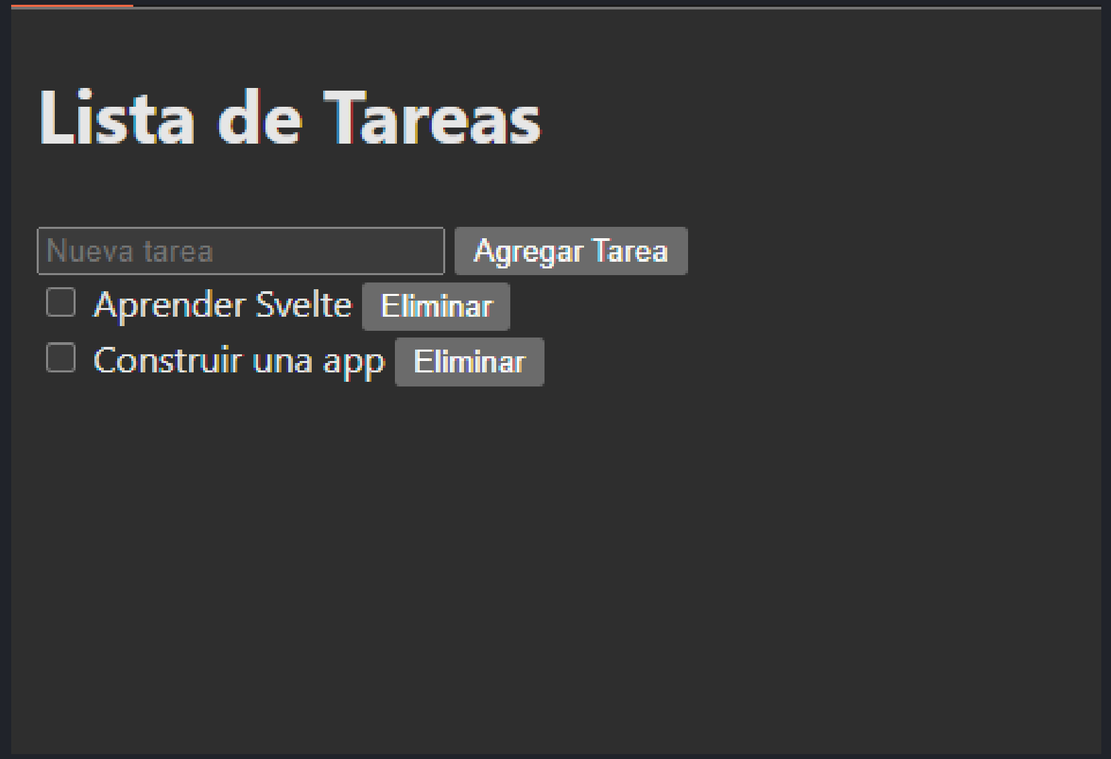
```svelte
<!-- Child.svelte -->
<script>
  import { createEventDispatcher } from 'svelte';

  const dispatch = createEventDispatcher();

  function sendMessage() {

    dispatch('message', { text:
                          'Hola desde el
                          componente hijo!' });
  }
</script>

<button on:click={sendMessage}>
  Enviar mensaje al padre
</button>
```

}

Ejemplo eventos {

PARA EJEMPLIFICAR EL USO DE EVENTOS EN UN ÁMBITO UN POCO MÁS COMPLEJO UTILIZAREMOS LA CÁLSICA FUNCIONALIDAD DE LA CREACIÓND DE UNA LSITA DE TAREAS

LINK EJEMPLO



}

Formularios {

ASÍ COMO SE DEBE RECIBIR INFORMACIÓN DESDE EL SERVIDOR AL NAVEGADOR, TAMBIÉN ES NECESARIO QUE ESTA PUEDA SER ENVIADA EN LA DIRECCIÓN OPUESTA.

POR ELLO, ENTRA EN JUEGO LA ETIQUETA <FORM>

LINK EJEMPLO

LINK TUTORIAL

```svelte
<!-- App.svelte -->
<script>
  let name = '';
  let email = '';
  let password = '';
  let acceptTerms = false;
  let formValid = false;

  $: formValid = name.length > 0 && email.length > 0 && password.length > 0 && acceptTerms;

  function handleSubmit() {
    if (formValid) {
      console.log({
        name,
        email,
        password,
        acceptTerms
      });
    } else {
      console.log('Form is not valid');
    }
  }
</script>
```

}

Lifecycle(onMount) {

TODOS LOS COMPONENTES TIENEN UN CICLO DE VIDA QUE INICIA CUANDO ESTE SE CREA Y TERMINA CUANDO SE DESTRUYE. EN SVELTE ES POSIBLE CORRER CÓDIGO EN MOMENTOS CLAVE DE ESTE CICLO.

EL PRIMERO DE ELLOS ES *ONMOUNT()* *QUE SE EJECUTA CUANDO EL COMPONENTE SE RENDERIZA POR PRIMERA VEZ EN EL DOM.*

LINK EJEMPLO

```
<script>
        import { onMount } from 'svelte';

    let photos = [];

    onMount(async () => {
        const res = await fetch(`/tutorial/api/album`);
        photos = await res.json();
    });
</script>
```

}

Lifecycle(onDestroy) {

ONDESTROY() SE EJECUTA CUANDO EL COMPONENTE SE DESTRUYE.

EN OCASIONES, ES CONVENIENTE LIMPIAR EL COMPONENTE CUANDO NO SEA RELEVANTE PARA EVITAR "MEMORY LEAKS".

LINK EJEMPLO

```
import { onDestroy } from 'svelte';

export function onInterval(callback, milliseconds) {
    const interval = setInterval(callback, milliseconds);

    onDestroy(() => {
        clearInterval(interval);
    });
}
```

}

Lifecycle(beforeUpdate, afterUpdate) {

- *BEFOREUPDATE()* SE EJECUTA INMEDIATAMENTE ANTES DE QUE SE REALICE UN CAMBIO EN EL DOM.
- POR OTRO LADO, SU CONTRAPARTE *AFTERUPDATE()* SE UTILIZA PARA EJECUTAR CÓDIGO UNA VEZ QUE EL DOM ESTÁ SINCRONIZADO CON LOS DATOS

LINK EJEMPLO

```
<script>
    import Eliza from 'elizabot';
    import { beforeUpdate, afterUpdate } from 'svelte';

    let div;
    let autoscroll;

    beforeUpdate(() => {
        autoscroll = div && div.offsetHeight + div.scrollTop > div.scrollHeight -
20; });

    afterUpdate(() => {
        if (autoscroll) div.scrollTo(0, div.scrollHeight);
    });
</script>
```

}

¿Qué es un store en Svelte?{

Son una forma de administrar datos compartidos entre diferente componentes en una aplicación.

Permite a los componentes acceder y actualizar el mismo conjunto de datos sin tener que pasarlos explícitamente como props.

}

Beneficios de usar stores{

01

Facilita la administración del estado

02

Mejora la reutilización de código

03

Optimización del rendimiento

}

Tipos de stores {

Hay tres tipos principales de stores en Svelte

- Writable store
- Readable store
- Derived store

Link Ejemplos prácticos

}

```
Gracias {

}
```