RIFERIMENTO RAPIDO SVELTE

Componenti, reattività, store, transizioni, SvelteKit

Componenti

Componente base

```
<script>
  let name = "world";
</script>
<h1>Hello {name}!</h1>
<style>
  h1 { color: purple; }
</style>
```

Struttura del componente

`<script>`	Logica del componente (JS/TS)
`Markup`	Template HTML con `{espressioni}`
`<style>`	CSS con scope (auto-applicato al componente)
`<script context="module">`	Eseguito una volta per modulo, non per istanza

Reattività

Assegnazioni reattive

```
<script>
  let count = 0;
  function increment() { count += 1; } // triggers re-render
</script>
<button on:click={increment}>{count}</button>
```

Dichiarazioni reattive

```
<script>
  let width = 10;
  let height = 5;
  $: area = width * height;       // recomputes when deps change
  $: console.log("area is", area); // reactive statement
</script>
```

$: contrassegna le dichiarazioni e le istruzioni reattive

Regole di reattività

`L'assegnazione attiva l'aggiornamento`	`count += 1` scatena un aggiornamento; anche `obj.x = 1`
`Mutazione di array`	Usare `arr = [...arr, item]` (riassegnare per attivare)
`Dichiarazione $:`	Si ricalcola automaticamente quando le variabili dipendenti cambiano
`Istruzione $:`	Esegue effetti collaterali in modo reattivo

Props

Dichiarazione e passaggio di props

```
<!-- Child.svelte -->
<script>
  export let name;
  export let greeting = "Hello";  // default value
</script>
<p>{greeting}, {name}!</p>

<!-- Parent.svelte -->
<Child name="Alice" />
```

Props con spread

```
<script>
  const props = { name: "Alice", greeting: "Hi" };
</script>
<Child {...props} />
```

Eventi

Eventi DOM

```
<button on:click={handleClick}>Click</button>
<input on:input={(e) => value = e.target.value} />
<form on:submit|preventDefault={handleSubmit}>
```

Modificatori di eventi

`preventDefault`	Chiama `e.preventDefault()`
`stopPropagation`	Interrompe il bubbling dell'evento
`once`	Il gestore si attiva solo una volta
`self`	Solo se `event.target` è l'elemento
`capture`	Usa la fase di cattura

Eventi dei componenti

```
<!-- Child.svelte -->
<script>
  import { createEventDispatcher } from "svelte";
  const dispatch = createEventDispatcher();
</script>
<button on:click={() => dispatch("greet", { text: "hi" })}>

<!-- Parent.svelte -->
<Child on:greet={(e) => alert(e.detail.text)} />
```

Binding

Binding bidirezionale

```
<input bind:value={name} />
<input type="checkbox" bind:checked={agreed} />
<select bind:value={selected}>
  <option value="a">A</option>
</select>
```

Binding di elementi e componenti

```
<div bind:this={element}></div>
<canvas bind:clientWidth={w} bind:clientHeight={h}></canvas>
<Child bind:value={childValue} />
```

Tipi di binding

`bind:value`	Valore di input/select/textarea
`bind:checked`	Stato della casella di controllo
`bind:group`	Gruppo radio/checkbox
`bind:this`	Riferimento all'elemento DOM
`bind:clientWidth/Height`	Dimensioni dell'elemento (sola lettura)

Store

Store scrivibile

```
// store.js
import { writable } from "svelte/store";
export const count = writable(0);

// Component — auto-subscribe with $
<script>
  import { count } from "./store.js";
</script>
<button on:click={() => $count += 1}>{$count}</button>
```

Metodi dello store

```
count.set(10);               // set value
count.update(n => n + 1);    // update from current
const unsub = count.subscribe(v => console.log(v));
```

Tipi di store

`writable(val)`	Store leggibile e scrivibile
`readable(val, fn)`	Sola lettura, impostato dalla funzione start
`derived(stores, fn)`	Calcolato da altri store
`$store`	Sintassi di sottoscrizione automatica nei componenti

Transizioni

Transizioni integrate

```
<script>
  import { fade, slide, fly } from "svelte/transition";
  let visible = true;
</script>
{#if visible}
  <div transition:fade>Fades in/out</div>
  <div in:fly={{ y: 200 }} out:fade>Fly in, fade out</div>
{/if}
```

Opzioni di transizione

`fade`	Opacità da 0 a 1
`fly`	Anima offset x/y + opacità
`slide`	Scivola dentro/fuori (altezza)
`scale`	Scala e dissolvenza
`draw`	Animazione del tratto SVG
`duration`	Durata della transizione in ms
`delay`	Ritardo prima dell'avvio

Slot

Slot predefiniti e con nome

```
<!-- Card.svelte -->
<div class="card">
  <slot name="header">Default header</slot>
  <slot>Default content</slot>
</div>

<!-- Usage -->
<Card>
  <h2 slot="header">Title</h2>
  <p>Body content goes here</p>
</Card>
```

Props degli slot

```
<!-- List.svelte -->
{#each items as item}
  <slot {item} index={item.id} />
{/each}

<!-- Usage -->
<List {items} let:item let:index>
  <p>{index}: {item.name}</p>
</List>
```

Contesto

Impostazione e lettura del contesto

```
<!-- Parent.svelte -->
<script>
  import { setContext } from "svelte";
  setContext("theme", { color: "dark" });
</script>

<!-- Descendant.svelte -->
<script>
  import { getContext } from "svelte";
  const theme = getContext("theme"); // { color: "dark" }
</script>
```

Contesto vs Store

`Contesto`	Limitato all'albero dei componenti, non reattivo per default
`Store`	Globale, reattivo, importabile ovunque
`Contesto + Store`	Passa uno store via contesto per reattività con scope

SvelteKit Nozioni di base

Routing basato su file

```
src/routes/
  +page.svelte          <!-- / -->
  about/+page.svelte    <!-- /about -->
  blog/[slug]/+page.svelte  <!-- /blog/:slug -->
```

Funzioni load

```
// +page.js (runs on client & server)
export async function load({ params, fetch }) {
  const res = await fetch(`/api/posts/${params.slug}`);
  return { post: await res.json() };
}
```

File chiave

`+page.svelte`	Componente pagina
`+page.js / +page.ts`	Funzione load client/universale
`+page.server.js`	Load solo server / azioni form
`+layout.svelte`	Layout condiviso
`+error.svelte`	Pagina di errore
`+server.js`	Endpoint API (GET, POST, ...)