

Référence rapide Svelte

Composants, réactivité, stores, transitions, SvelteKit

Composants

Composant de base

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

Structure d'un composant

<script>	Logique du composant (JS/TS)
Markup	Template HTML avec {expressions}
<style>	CSS limité (auto-délimité au composant)
<script context="module">	Exécuté une fois par module, pas par instance

Réactivité

Affectations réactives

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

Déclarations réactives

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

\$: marque les déclarations et instructions réactives

Règles de réactivité

Affectation déclenche	count += 1 déclenche la mise à jour; obj.x = 1 aussi
Mutation de tableau	Utiliser arr = [...arr, item] (réaffecter pour déclencher)
\$: déclaration	Se recalcule automatiquement quand les variables référencées changent
\$: instruction	Exécute des effets de bord de façon réactive

Props

Déclarer et passer des props

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

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

Diffusion de props

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

Événements

Événements DOM

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

Modificateurs d'événements

preventDefault	Appelle e.preventDefault()
stopPropagation	Arrête la propagation de l'événement
once	Le gestionnaire ne s'exécute qu'une seule fois
self	Seulement si event.target est l'élément
capture	Utiliser la phase de capture

Événements de composants

```
<!-- 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)} />
```

Liaisons

Liaison bidirectionnelle

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

Liaisons d'élément et de composant

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

Types de liaisons

bind:value	Valeur d'input/select/textarea
bind:checked	État de la case à cocher
bind:group	Groupe de boutons radio/cases à cocher
bind:this	Référence à l'élément DOM
bind:clientWidth/Height	Dimensions de l'élément (lecture seule)

Stores

Store inscriptible

```
// 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>
```

Méthodes de store

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

Types de stores

writable(val)	Store en lecture-écriture
readable(val, fn)	Lecture seule, défini par la fonction de démarrage
derived(stores, fn)	Calculé à partir d'autres stores
\$store	Syntaxe d'abonnement automatique dans les composants

Transitions

Transitions intégrées

```
<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}
```

Options de transition

fade	Opacité de 0 à 1
fly	Animer le décalage x/y + opacité
slide	Glisser en entrée/sortie (hauteur)
scale	Agrandir et fondre
draw	Animation de tracé SVG
duration	Durée de la transition en ms
delay	Délai avant le démarrage

Slots

Slots par défaut et nommés

```
<!-- 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 de 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>
```

Contexte

Définir et obtenir le contexte

```
<!-- 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>
```

Référence rapide Svelte

Contexte vs Stores

Contexte	Limité à l'arbre de composants, pas réactif par défaut
Stores	Global, réactif, importable partout
Contexte + Store	Passer un store via le contexte pour une réactivité délimitée

Bases de SvelteKit

Routage basé sur les fichiers

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

Fonctions de chargement

```
// +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() };  
}
```

Fichiers clés

+page.svelte	Composant de page
+page.js / +page.ts	Fonction de chargement client/universelle
+page.server.js	Chargement côté serveur uniquement / actions de formulaire
+layout.svelte	Mise en page partagée
+error.svelte	Page d'erreur
+server.js	Point d'API (GET, POST, ...)