

Svelte Core Concepts



Essential features of the
Svelte Framework

Svelte is a tool for building fast web Front Ends

It is similar to JavaScript frameworks such as React and Vue, which share a goal of making it easy to build slick interactive user interfaces.

But there's a crucial difference:

- Svelte converts your app into ideal JavaScript at build time, rather than interpreting your application code at run time.
- This means you don't pay the performance cost of the framework's abstractions, and you don't incur a penalty when your app first loads.

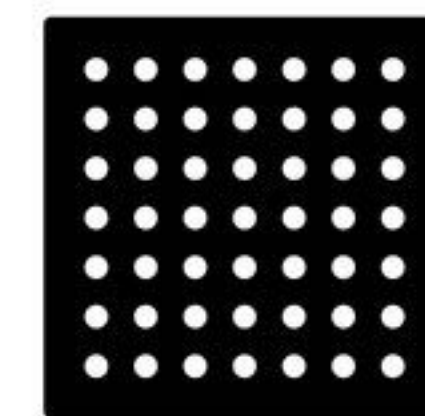
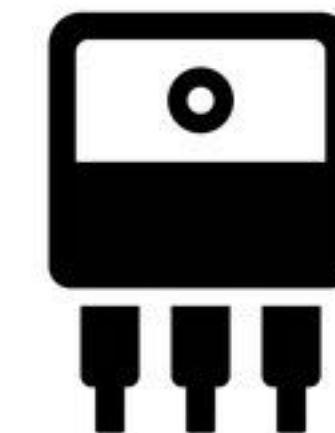
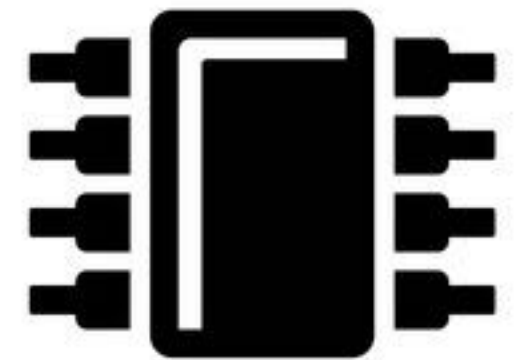
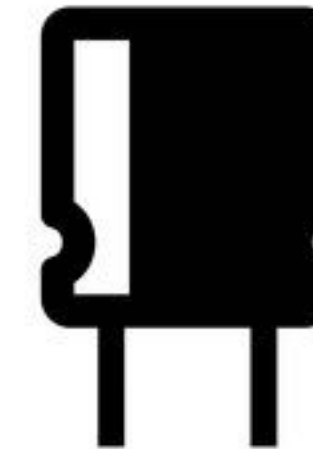
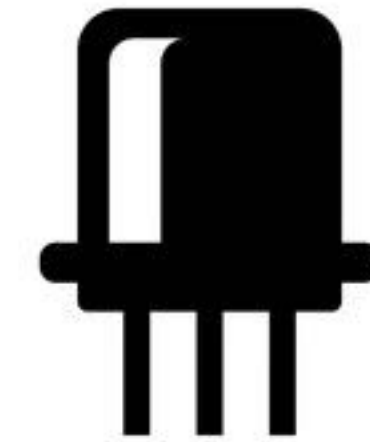
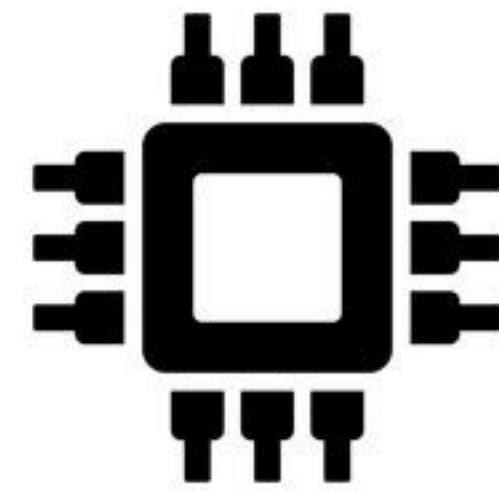


What is Svelte?

You can build your entire app with Svelte, or you can add it incrementally to an existing codebase. You can also ship components as standalone packages that work anywhere, without the overhead of a dependency on a conventional framework.

Svelte Components

- Modern Web development is very much focused on components,
- What is a component?
 - A component is an atomic part of the application that is self-contained and optionally references other components to compose its output.
 - It's a compartmentalized part of the application. A form can be a component. An input element can be a component. The whole application is a component.
- Svelte components contain all that's needed to render a piece of the UI.



State

Props

Logic

Events

Bindings

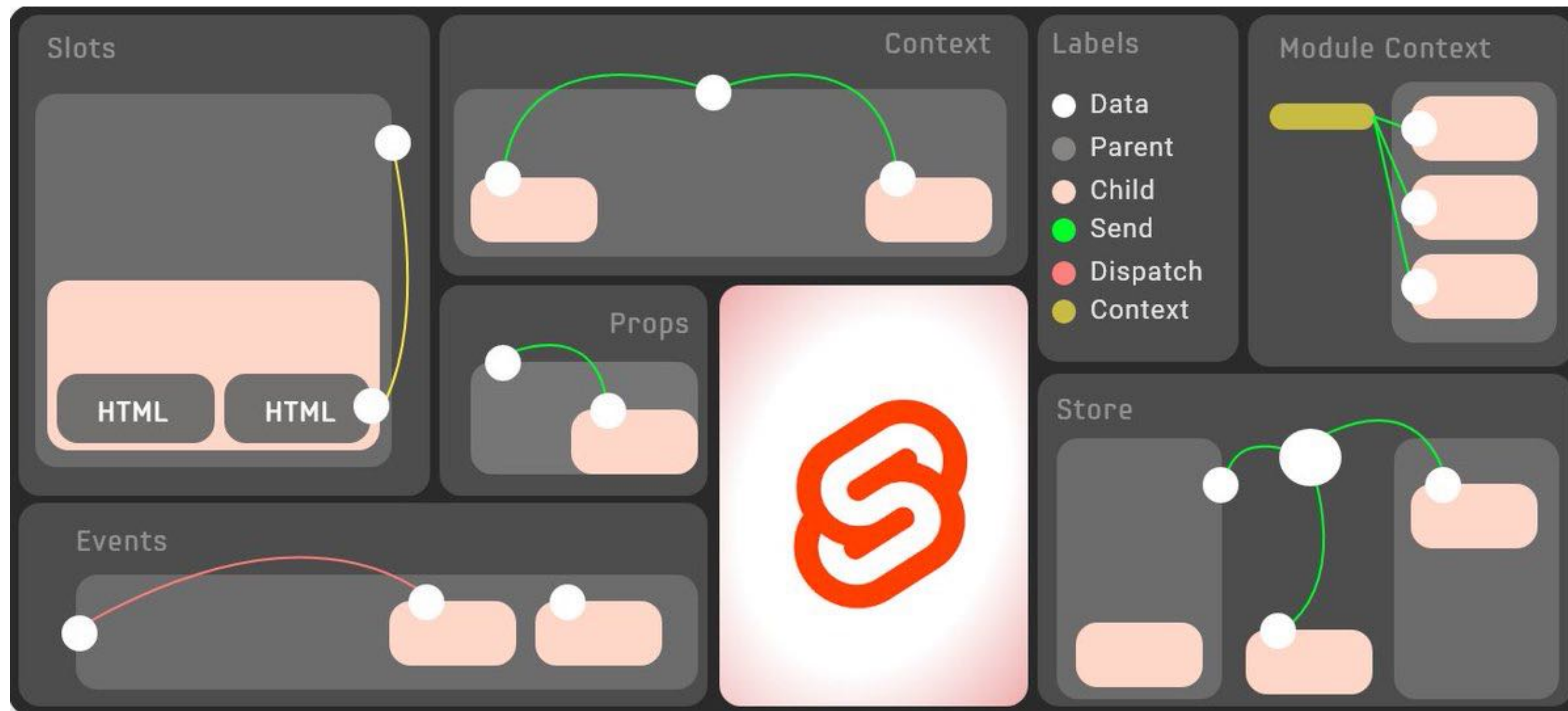
Stores

Context

Lifecycle

Reactivity

Key Svelte Concepts



- **Components:** Primary building block for application. Must be named *SomeComponent.svelte*
- **State:** Any data that's needed to make the component render what it's meant to render
- **Props:** To pass data from a component down to its children
- **Logic:** To specify way of expressing logic, like conditionals and loops.
- **Events:** Define a listener for a DOM event directly
- **Bindings:** To create a two-way binding between data and the UI.

Key Svelte Concepts

- **Context:** A feature to allow a component communicate with multiple descendants
- **Stores:** A feature to allow unrelated components to talk to each other
- **Lifecycle:** To specify way a component can be notified of important events in its lifecycle
- **Reactivity:** Be notified dynamically when changes to occur a variable

Svelte Components (1)

- Every Svelte component is declared in a .svelte file containing (markup), the behavior (JavaScript), and the presentation (CSS).
- This is considered a sane way to define a piece of the UI because you don't need to search for the items that affect the same element across various files.

→ **Components:** Primary building block for application. Must be named *SomeComponent.svelte*

```
<script>
export let name;
</script>

<style>
h1 {
  color: purple;
}
</style>

<h1>The dog name is {name}!</h1>
```

Dog.svelte

Svelte Components (2)

- Any JavaScript must be put in the script tag.
- The CSS you have in the style tag is scoped to the component and does not "leak" outside. If another component has an h1 tag, this style will not affect that.
- The html is presented without enclosure
- All three are optional

```
<script>
export let name;
</script>

<style>
h1 {
  color: purple;
}
</style>

<h1>The dog name is {name}!</h1>
```

Dog.svelte

Components (3)

- A component can be used by other components.

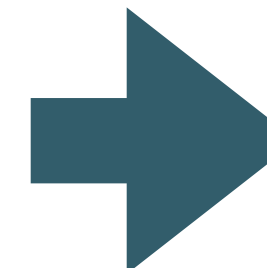
```
<script>  
export let name;  
</script>
```

```
<style>  
h1 {  
  color: purple;  
}  
</style>
```

```
<h1>The dog name is {name}!</h1>
```

Dog.svelte

- The component must be 'imported'...
... and then it can be used as an element



```
<script>  
import Dog from './Dog.svelte'  
</script>  
  
<Dog />
```

App.svelte

State (2)

→ **State:** Any data that's needed to make the component render what it's meant to render

- Don't need to do anything special to update the state of a component.
- All you need is an assignment. A simple JavaScript assignment, using the = operator for example.
- Say you have a count variable. You can increment that using, simply, `count = count + 1` , or `count++` :

```
<script>
let count = 0

const incrementCount = () => {
  count++
}
</script>

{count} <button on:click={incrementCount}>+1</button>
```

Props

You can import a Svelte component into any other component

Use the newly imported component in the markup, like an HTML tag

The parent component can pass data to the child component using props. Props behave similarly to attributes in plain HTML, and they are a one-way form of communication.

In the SignupForm component, you need to export the disabled prop

→ **Props:** To pass data from a component down to its children

```
<script>
import SignupForm from './SignupForm.svelte';
</script>
```

```
<SignupForm />
```

```
<SignupForm disabled={true}/>
```

```
<script>
  export let disabled
</script>
```

Logic

- In Svelte components you can create a loop using the `{#each}{/each}` syntax
- Conditional logic is via the `{#if}{/if}`

→ **Logic:** To specify way of expressing logic, like conditionals and loops.

```
<script>
let goodDogs = ['Roger', 'Syd']
</script>

{#each goodDogs as goodDog}
  <li>{goodDog}</li>
{/each}
```

```
<script>
let isRed = true
</script>
```

```
{#if isRed}
  <p>Red</p>
{:else}
  <p>Not red</p>
{/if}
```

Events

➔ **Events:** Define a listener for a DOM event directly

To listen to the click event, pass a function to the on:click attribute

Svelte passes the event handler as the argument of the function

```
<script>
const doSomething = () => {
  alert('clicked')
}
</script>

<button on:click={doSomething}>Click me</button>
```

Bindings

A variable from the component state (name),
can be bound to a form field:

isChecked (boolean) is bound to a check box

➔ **Bindings:** To create a two-way binding between data and the UI.

```
<script>
let name = ''
</script>

<input bind:value={name}>
```

```
<script>
let isChecked
</script>

<input type=checkbox bind:checked={isChecked}>
```


Context

→ **Context:** A feature to allow a component communicate with multiple descendants

The context API is provided by 2 functions which are provided by the svelte package: `getContext` and `setContext`

You set an object in the context, associating it to a key

In another component you can use `getContext` to retrieve the object assigned to a key:

```
<script>
import { setContext } from 'svelte'

const someObject = {}

setContext('someKey', someObject)
</script>
```

```
<script>
import { getContext } from 'svelte'

const someObject = getContext('someKey')
</script>
```

Stores

Import *writable*, create, initialise and export a **store** - from a .js file (not .svelte)

→ **Stores:** A feature to allow unrelated components to talk to each other

```
import { writable } from 'svelte/store'
export const username = writable('Guest')
```

In any component, import the store

```
<script>
import { username } from './store.js'
</script>
```

Update store (write)

```
username.set('new username')
```

Be notified if any other component has updated the store (read)

```
username.subscribe(newValue => {
  console.log(newValue)
})
```

Lifecycle

- *onMount* fired after the component is rendered
- *onDestroy* fired after the component is destroyed
- *beforeUpdate* fired before the DOM is updated
- *afterUpdate* fired after the DOM is updated

→ **Lifecycle:** To specify way a component can be notified of important events in its lifecycle

```
<script>
  import { onMount, onDestroy, beforeUpdate, afterUpdate } from 'svelte'
</script>
```

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

  onMount(async () => {
    //do something on mount
  })
</script>
```

Reactivity

- Listen for changes on count using the special syntax \$:
- This defines a new block that Svelte will re-run when any variable referenced into it changes.

→ **Reactivity:** Be notified dynamically when changes to occur a variable

```
<script>
let count = 0

const incrementCount = () => {
  count = count + 1
}

$: console.log(`${count}`)
</script>

{count} <button on:click={incrementCount}>+1</button>
```


← Introduction / Basics →

Introduction

Welcome to the Svelte tutorial. This will teach you everything you need to know to build fast, small web applications easily.

You can also consult the [API docs](#) and the [examples](#), or – if you're impatient to start hacking on your machine locally – the [60-second quickstart](#).

What is Svelte?


Svelte is a tool for building fast web applications.

It is similar to JavaScript frameworks such as React and Vue, which share a goal of making it easy to build slick interactive user interfaces.


But there's a crucial difference: Svelte converts your app into ideal JavaScript at *build time*, rather than interpreting your application code at *run time*. This means you don't pay the performance cost of the framework's abstractions, and you don't incur a penalty when your app first loads.

<https://svelte.dev/tutorial/basics>

Svelte Handbook



THE
SVELTE
HANDBOOK



Flavio Lopes

The book is available at

<https://flaviocopes.com/page/download-svelte-handbook/>

Svelte Core Concepts



Essential features of the
Svelte Framework