SVELTE • TEMPLATE SYNTAX

{#snippet ...}

ON THIS PAGE

```
{#snippet name()}...{/snippet}
```

```
{#snippet name(param1, param2, paramN)}...{/snippet}
```

Snippets, and render tags, are a way to create reusable chunks of markup inside your components. Instead of writing duplicative code like this...

```
{#each images as image}
  {#if image.href}
    <a href={image.href}>
      <figure>
        <img src={image.src} alt={image.caption} width={image.width} height={image.height
        <figcaption>{image.caption}</figcaption>
      </figure>
    </a>
  {:else}
    <figure>
      <img src={image.src} alt={image.caption} width={image.width} height={image.height} /
      <figcaption>{image.caption}</figcaption>
    </figure>
  {/if}
{/each}
```

...you can write this:

```
{#snippet figure(image)}
  <figure>
```

```
{#each images as image}
  {#if image.href}
    <a href={image.href}>
      {@render figure(image)}
    </a>
  {:else}
    {@render figure(image)}
  {/if}
{/each}
```

Like function declarations, snippets can have an arbitrary number of parameters, which can have default values, and you can destructure each parameter. You cannot use rest parameters, however.

Snippet scope

Snippets can be declared anywhere inside your component. They can reference values declared outside themselves, for example in the `<script>` tag or in `{#each ...}` blocks (demo)...

```
<script>
  let { message = `it's great to see you!` } = $props();
</script>

{#snippet hello(name)}
  <p>hello {name}! {message}!</p>
{/snippet}

{@render hello('alice')}
{@render hello('bob')}
```

...and they are 'visible' to everything in the same lexical scope (i.e. siblings, and children of those siblings):

```
    <!-- this is fine -->
    {@render y()}
  {/snippet}

  <!-- this will error, as `y` is not in scope -->
  {@render y()}
</div>

<!-- this will also error, as `x` is not in scope -->
{@render x()}
```

Snippets can reference themselves and each other (demo):

```
{#snippet blastoff()}
  <span>🚀</span>
{/snippet}

{#snippet countdown(n)}
  {#if n > 0}
    <span>{n}...</span>
    {@render countdown(n - 1)}
  {:else}
    {@render blastoff()}
  {/if}
{/snippet}

{@render countdown(10)}
```

Passing snippets to components

Within the template, snippets are values just like any other. As such, they can be passed to components as props (demo):

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

  const fruits = [
```

```
</script>

{#snippet header()}
  <th>fruit</th>
  <th>qty</th>
  <th>price</th>
  <th>total</th>
{/snippet}

{#snippet row(d)}
  <td>{d.name}</td>
  <td>{d.qty}</td>
  <td>{d.price}</td>
  <td>{d.qty * d.price}</td>
{/snippet}

<Table data={fruits} {header} {row} />
```

Think about it like passing content instead of data to a component. The concept is similar to slots in web components.

As an authoring convenience, snippets declared directly *inside* a component implicitly become props *on* the component (demo):

```
<!-- this is semantically the same as the above -->
<Table data={fruits}>
  {#snippet header()}
    <th>fruit</th>
    <th>qty</th>
    <th>price</th>
    <th>total</th>
  {/snippet}

  {#snippet row(d)}
    <td>{d.name}</td>
    <td>{d.qty}</td>
    <td>{d.price}</td>
    <td>{d.qty * d.price}</td>
  {/snippet}
</Table>
```

part of the `children` snippet (demo):

App.svelte

```
<Button>click me<Button>
```

Button.svelte

```
<script>
  let { children } = $props();
</script>

<!-- result will be <button>click me</button> -->
<button>{@render children()}</button>
```

> Note that you cannot have a prop called `children` if you also have content inside the component — for this reason, you should avoid having props with that name

You can declare snippet props as being optional. You can either use optional chaining to not render anything if the snippet isn't set...

```
<script>
  let { children } = $props();
</script>

{@render children?.()}
```

...or use an `#if` block to render fallback content:

```
<script>
  let { children } = $props();
</script>

{#if children}
  {@render children()}
{:else}
  fallback content
```

Snippets implement the `Snippet` interface imported from `'svelte'`:

```
<script lang="ts">
  import type { Snippet } from 'svelte';

  interface Props {
    data: any[];
    children: Snippet;
    row: Snippet<[any]>;
  }

  let { data, children, row }: Props = $props();
</script>
```

With this change, red squigglies will appear if you try and use the component without providing a `data` prop and a `row` snippet. Notice that the type argument provided to `Snippet` is a tuple, since snippets can have multiple parameters.

We can tighten things up further by declaring a generic, so that `data` and `row` refer to the same type:

```
<script lang="ts" generics="T">
  import type { Snippet } from 'svelte';

  let {
    data,
    children,
    row
  }: {
    data: T[];
    children: Snippet;
    row: Snippet<[T]>;
  } = $props();
</script>
```

Programmatic snippets

for advanced use cases.

Snippets and slots

In Svelte 4, content can be passed to components using slots. Snippets are more powerful and flexible, and as such slots are deprecated in Svelte 5.

Edit this page on GitHub

PREVIOUS
{#await ...}

NEXT
{@render ...}