Svelte 5

Reactivity with Runes

inovex

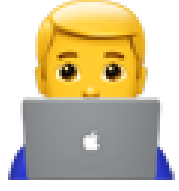
Svelte 4 Reactivity Revisited

Svelte identifies certain JS constructs and enhances them with meaning:

- `let`: creation of a local reactive state variable
- `=`: mutation of local state
- `$:`: derived local state
- `export let`: component properties

Problems and downsides (state on component level)

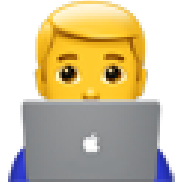
- The reactivity paradigm (let, =, $:) only works on the component root level
- State mutation restricted to equals (=)



Demo Legacy State

Problems and downsides (derived values)

- Derived values easily get out of sync
- Derived values do not properly react to (all) dependencies



Demo Legacy Derived

Svelte 5 Runes

...to the rescue.

"A letter or mark used as a mystical or magic symbol."

Runes

Instead of using (misusing) Javascript syntax elements to identify a certain use case, runes are introduced that are recognized by the svelte compiler.

Runes are functions with reserved names.

```
1 let count = $state(0)
```

Bonus-Feature: Typescript-Support!

Important Runes

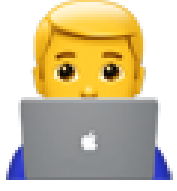
```
$state, $derived, $effect,
$props, $inspect
```


BUT IT
LOOKS LIKE REACT

$state

```
1 <script>
2   let count = $state<number>(0)
3 </script>
4
5 <h2>{count}</h2>
```

- `count` is a state variable, under the hood backed by a "Signal" the underlying programmatic primitive
- Arrays are implemented with a `Proxy` so any kind of mutation (push, pop, ...) can be tracked and directly communicated
- Typescript integration just works

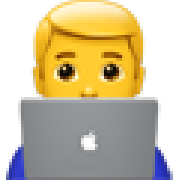


Demo $state

$derived

```
1 <script>
2   let count = $state(0)
3
4   let doubled = $derived(count * 2)
5 </script>
6
7 <h2>{count} / {doubled}</h2>
```

- `doubled` is also a normal state variable, also backed by a signal.
- It updates automatically whenever one of the dependent state variables are mutated
- Only a expression, not a function call



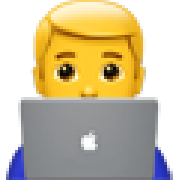
Demo $derived

$effect

```
<script>
  let count = $state(0)

  $effect(() => {
    console.log(`count has been updated to ${count}`)
  })

  $effect(() => {
    console.log('mount')

    return () => console.log('unmount')
  })
</script>

<h2>{count}</h2>
```

$effect

- function expression
- runs after every DOM updates
- the framework identifies dependencies on runtime
- the defacto replacement for all lifecycle hooks:
  - onMount
  - onDestroy
  - afterUpdate
  - (beforeUpdate)



Demo $effect

⚠️ $effect ⚠️

Pay attention when converting "old" lifecycle hooks to $effect!

```
<script>
  let count = $state(0)
  let doubled = $state(0)

  $effect(() => {
    doubled = count * 2
  })
</script>
```

```
<script>
  let count = $state(0)
  let doubled = $derived(count * 2)
</script>
```

⚠️ $effect ⚠️

Rich Harris @Rich_Harris · Sep 15

one of the reasons Svelte 5 has a rune called '$**effect**' (rather than something like 'watch' or 'autorun' or whatever) is to discourage you from actually using it

> 
> **Ben Lesh** @BenLesh · Sep 15
>
> No matter the framework, if you see something with “effect” in the name, try not to use it. Seriously. Do your best.

18 27 495 60K

$props

Use "normal" destructuring to access component properties. No more use of the irritating `export` syntax.

```
<script>
  let { optionalProp = 42, requiredProp } = $props();
</script>
```

```
<script>
  // no more $$restprops
  let { a, b, c, ...everythingElse } = $props();
</script>
```

```
<script>
  interface MyProps {
    a: string
  }

  let { a }: MyProps = $props();
</script>
```

$inspect

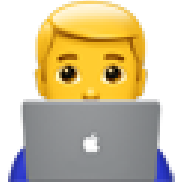
```
<script>
  let count = $state(0)

  $inspect(count)
</script>

<button onclick={() => count++}>
  increment
</button>
```

Only executed in development, logs the initial value and on every update.

```
"init"
0
"update"
1
"update"
2
```

Reuse of known paradigms

The reactivity paradigm can now be applied in a reusable way.



Demo Reactivity Pattern

Things that became obsolete and their replacement

Obsolete	Replacement
`export let foo`	`let {foo} = $props`

lifecycle methods	`onMount => $effect`
	`beforeUpdate => $effect.pre`
	`afterUpdate => $effect`
	`onDestroy => return value from $effect`

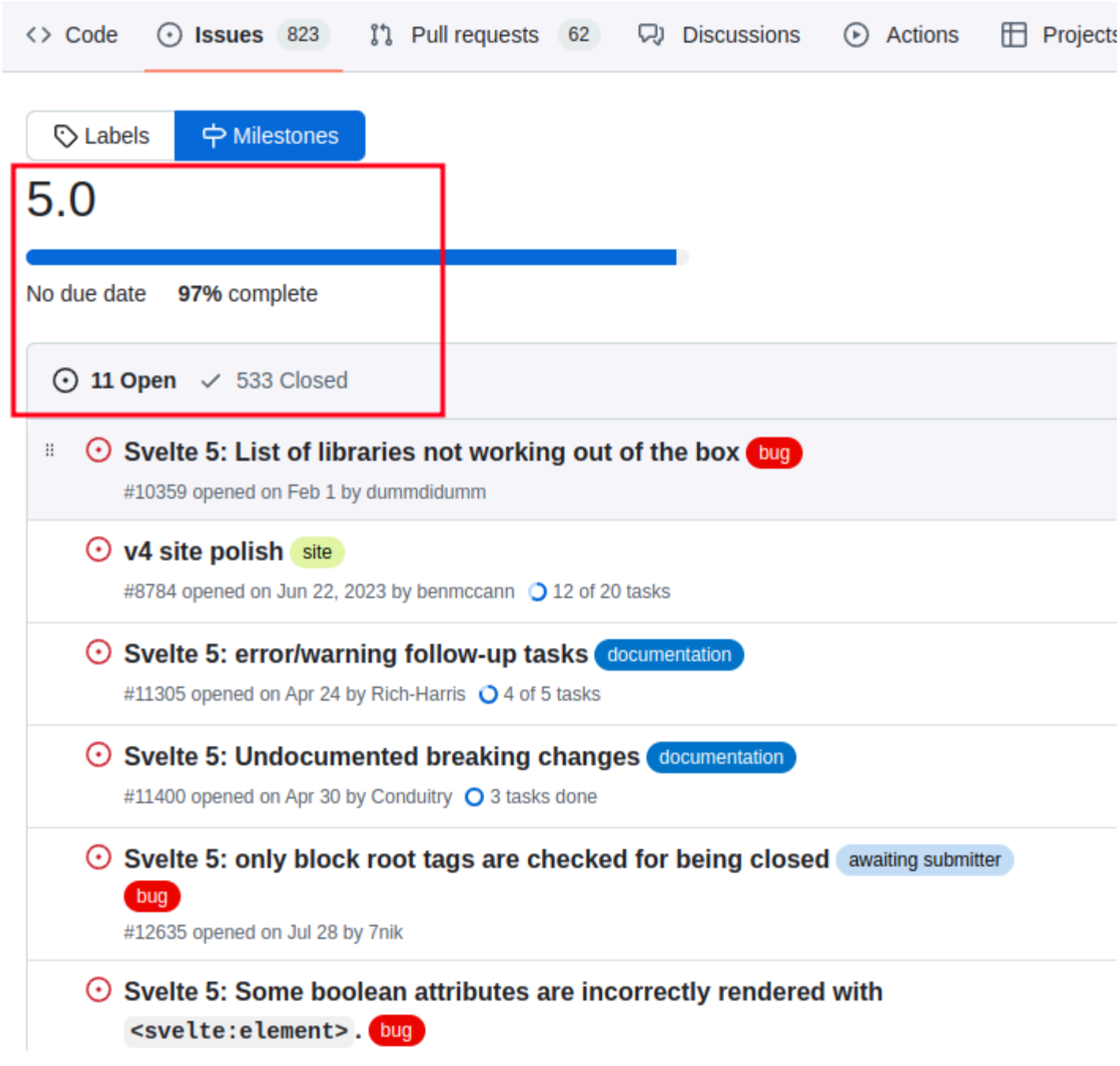
`$: doubled = count * 2`	`let doubled = $derived(count * 2)`
Stores and store apis	`$state`
`$$restProps`	`let {...restProps} = $props`

More Svelte 5 Changes

- Snippets will replace slots
- Event handlers via `onclick` instead of `on:click`
- Support for nested CSS syntax
- Deprecation of `afterUpdate`, `beforeUpdate`, `createEventDispatcher`

Release

- No Release date yet (originally was April 2024)
- Already be usable via the next RC
- Not recommended for production


Code
Issues 823
Pull requests 62
Discussions
Actions
Projects
Labels
Milestones
5.0
No due date 97% complete
11 Open
533 Closed
Svelte 5: List of libraries not working out of the box bug
#10359 opened on Feb 1 by dummdidumm
v4 site polish site
#8784 opened on Jun 22, 2023 by benmccann 12 of 20 tasks
Svelte 5: error/warning follow-up tasks documentation
#11305 opened on Apr 24 by Rich-Harris 4 of 5 tasks
Svelte 5: Undocumented breaking changes documentation
#11400 opened on Apr 30 by Conduitry 3 tasks done
Svelte 5: only block root tags are checked for being closed awaiting submitter
bug
#12635 opened on Jul 28 by 7nik
Svelte 5: Some boolean attributes are incorrectly rendered with
<svelte:element>. bug

Resources

- Svelte 5 Introduction: https://svelte.dev/blog/runes
- Svelte 5 Playground: https://svelte-5-preview.vercel.app/
- Official Documentation: https://svelte.dev/docs