

tt()

Schedule

1. Events
2. Modules (import and export)
3. Async data fetching
4. All together!



Schedule

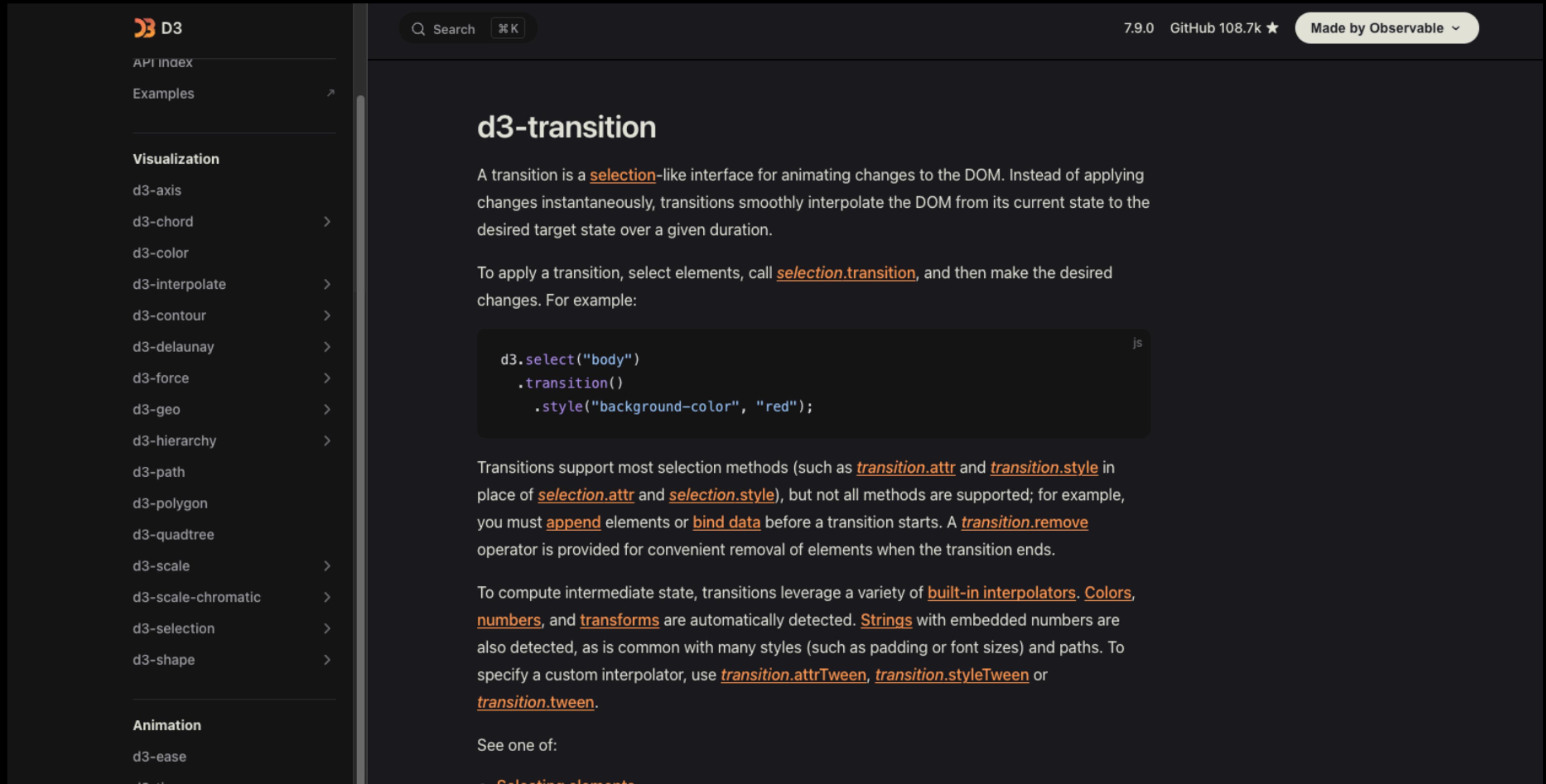
1. Events
2. Modules (import and export)
3. Async data fetching
4. All together!



Events

1. Mouse events (clicks, mouseover etc.)
2. Keyboard events (keypress etc.)
3. Touch events (longpress etc.)
4. Drag & Zoom events (drag etc.)

Events (transitions)



The screenshot shows the d3.js API documentation for transitions. The left sidebar has a dark theme with a navigation menu. The main content area has a light background and displays the following information:

d3-transition

A transition is a [selection](#)-like interface for animating changes to the DOM. Instead of applying changes instantaneously, transitions smoothly interpolate the DOM from its current state to the desired target state over a given duration.

To apply a transition, select elements, call [selection.transition](#), and then make the desired changes. For example:

```
js
d3.select("body")
  .transition()
  .style("background-color", "red");
```

Transitions support most selection methods (such as [transition.attr](#) and [transition.style](#) in place of [selection.attr](#) and [selection.style](#)), but not all methods are supported; for example, you must [append](#) elements or [bind data](#) before a transition starts. A [transition.remove](#) operator is provided for convenient removal of elements when the transition ends.

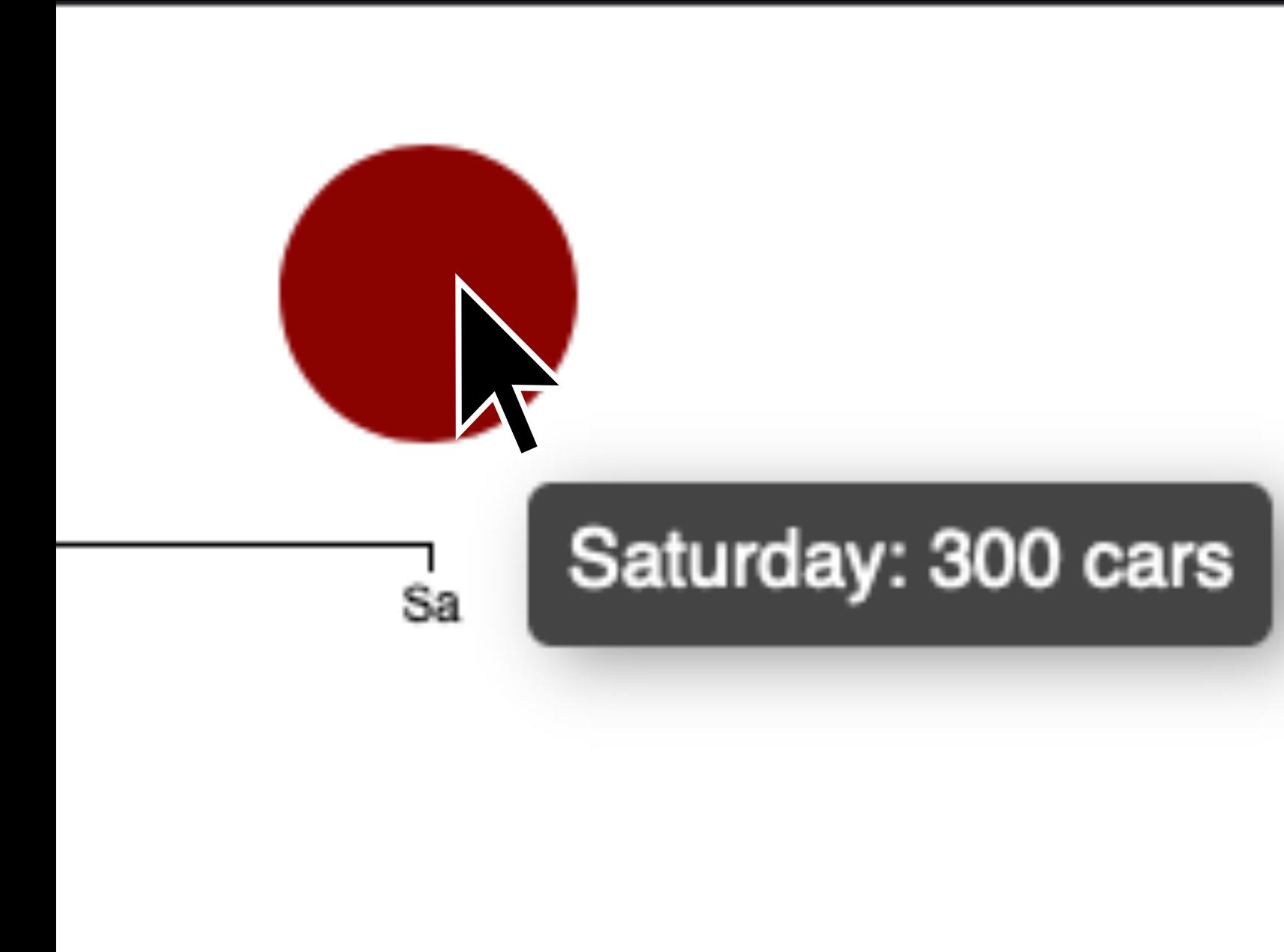
To compute intermediate state, transitions leverage a variety of [built-in interpolators](#). [Colors](#), [numbers](#), and [transforms](#) are automatically detected. [Strings](#) with embedded numbers are also detected, as is common with many styles (such as padding or font sizes) and paths. To specify a custom interpolator, use [transition.attrTween](#), [transition.styleTween](#) or [transition.tween](#).

See one of:

- [Selecting elements](#)

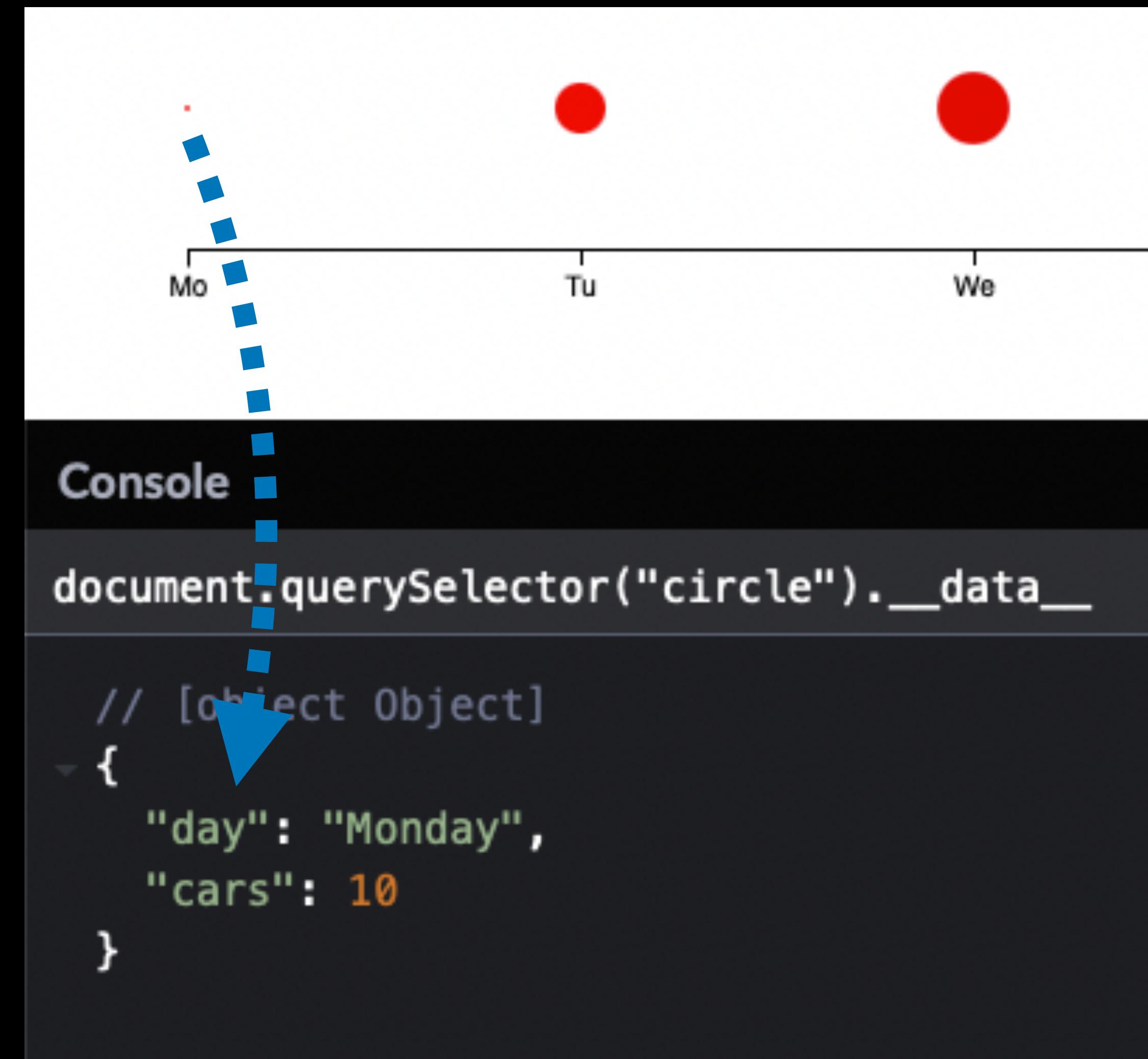
Events

Events help you to add interactivity to your graphs. For example you can add a tooltip or a side panel showing additional details.



Event data

D3 has a magical feature: it adds a `__data__` object to all DOM elements you created so you have access to the original data in your event.



```
Mo Tu We
Console
document.querySelector("circle").__data__
// [object Object]
{
  "day": "Monday",
  "cars": 10
}
```

Event binding



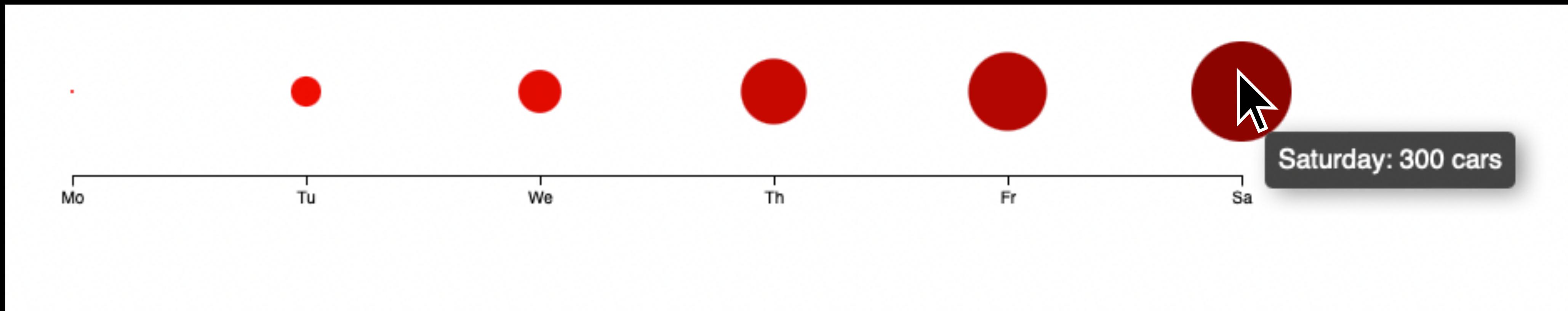
```
d3.select("#scale1")
  .selectAll("circle")
  .data(dataSet)
  .join("circle")
  .on("mouseover", (e, d) =>
    d3.select("#tooltip")
      .style("opacity", 1)
      .text(` ${d.day}: ${d.cars} cars`)
  )
  .on("mousemove", (e) =>
    d3
      .select("#tooltip")
      .style("left", e.pageX + 15 + "px")
      .style("top", e.pageY + 15 + "px")
  )
```

You add events by calling `d3.on()`. D3 will call your event function with two parameters:

1. Event data
2. Object data used during `d3.join()`

Tooltip demo

<https://codepen.io/dandevri/pen/azdrEQb>



Huiswerk

- Maak je visualisatie

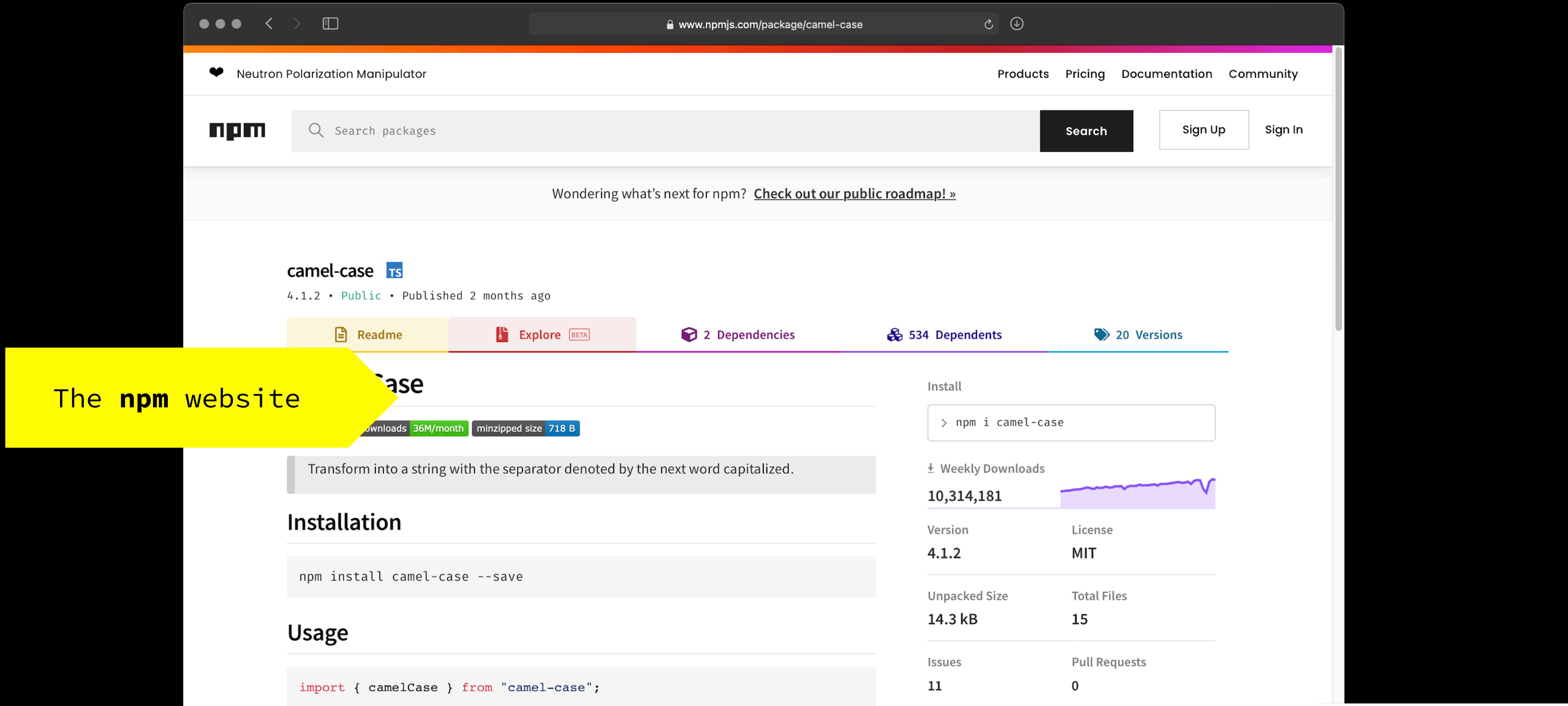
Schedule

1. Events
2. Modules (import and export)
3. Async data fetching
4. All together!



Package manager

npm is a package manager for the JavaScript programming language. It is the default package manager for [...] Node.js. It consists of a command line client, also called npm, and an online database of [...] packages, called the npm registry.



The npm website

www.npmjs.com/package/camel-case

Neutron Polarization Manipulator

Products Pricing Documentation Community

npm Search packages Sign Up Sign In

Wondering what's next for npm? [Check out our public roadmap! »](#)

camel-case TS

4.1.2 • Public • Published 2 months ago

Readme Explore BETA 2 Dependencies 534 Dependents 20 Versions

Transform into a string with the separator denoted by the next word capitalized.

Installation

```
npm install camel-case --save
```

Usage

```
import { camelCase } from "camel-case";  
  
camelCase("string"); //→ "string"
```

Install

```
npm i camel-case
```

Weekly Downloads

10,314,181

Version License

4.1.2 MIT

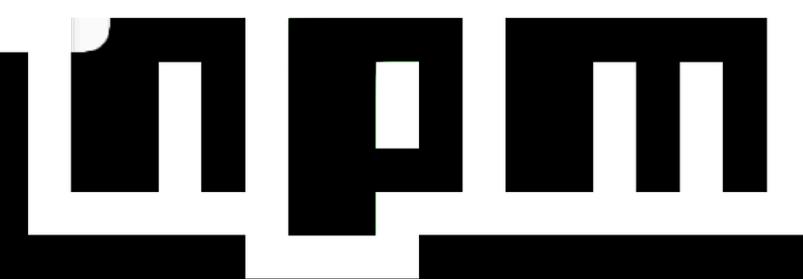
Unpacked Size Total Files

14.3 kB 15

Issues Pull Requests

11 0

Homepage



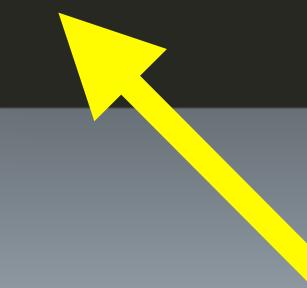
Import modules

```
import * as d3 from "d3";  
import { onMount } from 'svelte';
```

Modules



```
async function request(url) {  
  let res = await fetch(url);  
  return await res.json();  
}  
  
export default request
```



export default

Modules

```
● ● ●  
import CONFIG from './config.js';  
import request from './request.js';  
import makeHtml from './make.js';  
  
const data = await request(CONFIG.url);
```

Import default

Modules

- `require` : Function-based syntax:

javascript

 Copy code

```
const module = require('module-name');
```

- `import` : Declarative syntax (similar to other languages):

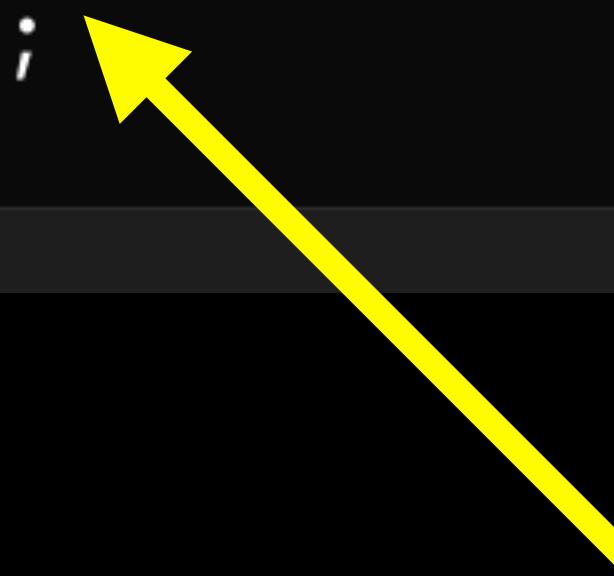
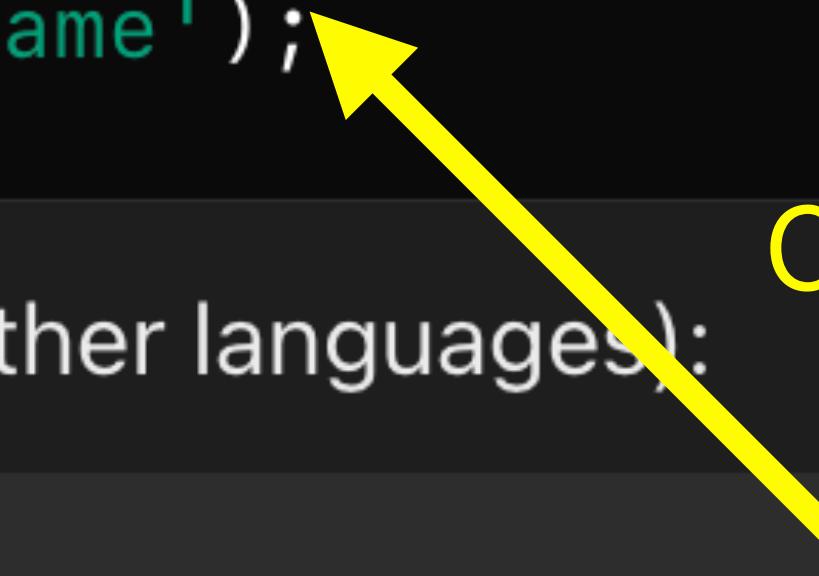
javascript

 Copy code

```
import module from 'module-name';
```

CommonJS

ES Modules



Modules

- You can export a function or variable from any file
- There are two types of exports, named and default

Named exports



// Individually

```
export const name = "Robert";
export const age = 29;
```

// All at once as an object

```
const name = "Robert";
const age = 29;
```

```
export { name, age }
```

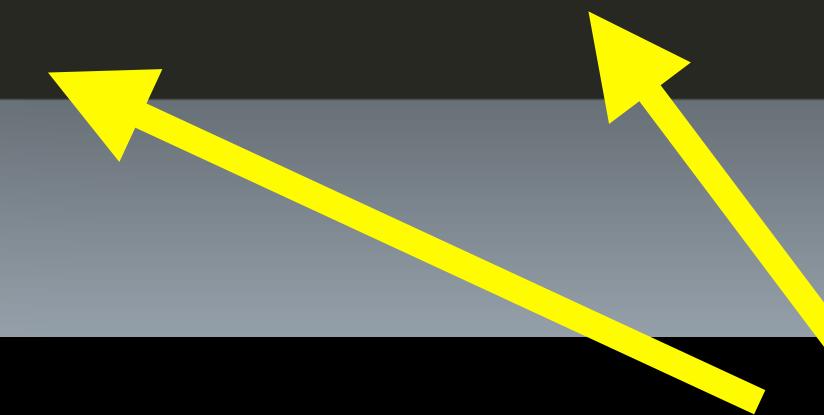


Exporting as an object, due to the {}

Named imports



```
import { name, age } from "./robert.js";
```



Importing as an object, due to the {}

Named imports

```
● ● ●  
import * as robert from "./robert.js";  
console.log(robert.age) // 30
```

Import everything using “as” to name it

Import modules

```
import * as d3 from "d3";
```

```
import { scales, selection } from 'd3';
```

Components

component === module

```
import Bar from './components/Bar.svelte';
```

Modules = functional programming

Referential transparency: The function always gives the same return value for the same arguments. This means that the function cannot depend on any mutable state

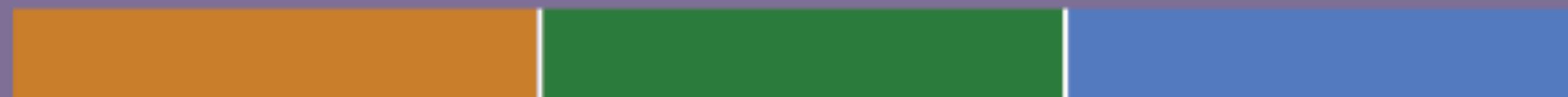
Side-effect free: The function cannot cause any side effects. Side effects may include I/O (e.g., writing to the console or a log file), modifying a mutable object, reassigning a variable, etc.

Schedule

1. Events
2. Modules (import and export)
- 3. Async data fetching**
4. All together!



SYNCHRONOUS LOAD



ASYNCHRONOUS LOAD



Fetching in JavaScript

- ❖ **Callbacks (XMLHttpRequest)**
- ❖ **Promises (Fetch)**
- ❖ **Async / Await (Fetch)**

Sync vs async

why?

Getting data from a resource (API) takes time. It needs to fetch the resource, parse it etc. But also, what if the data isn't available (no internet connection e.g.) how should errors be handled?

Separation of concerns

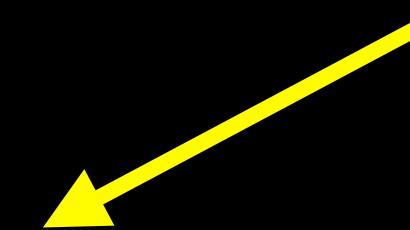
1. Data **fetchen** (fetch API)
2. Data **transformeren** (functional, map filter)
3. **Renderen van de Charts** (d3, scales)

Utility function

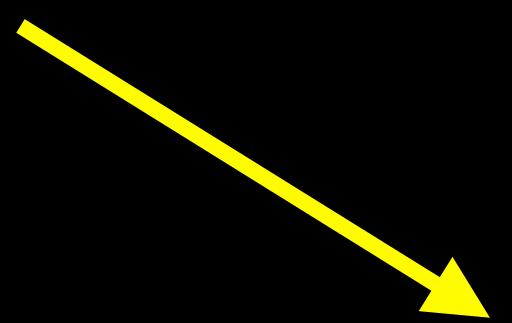
dataFetch.js



dataClean.js



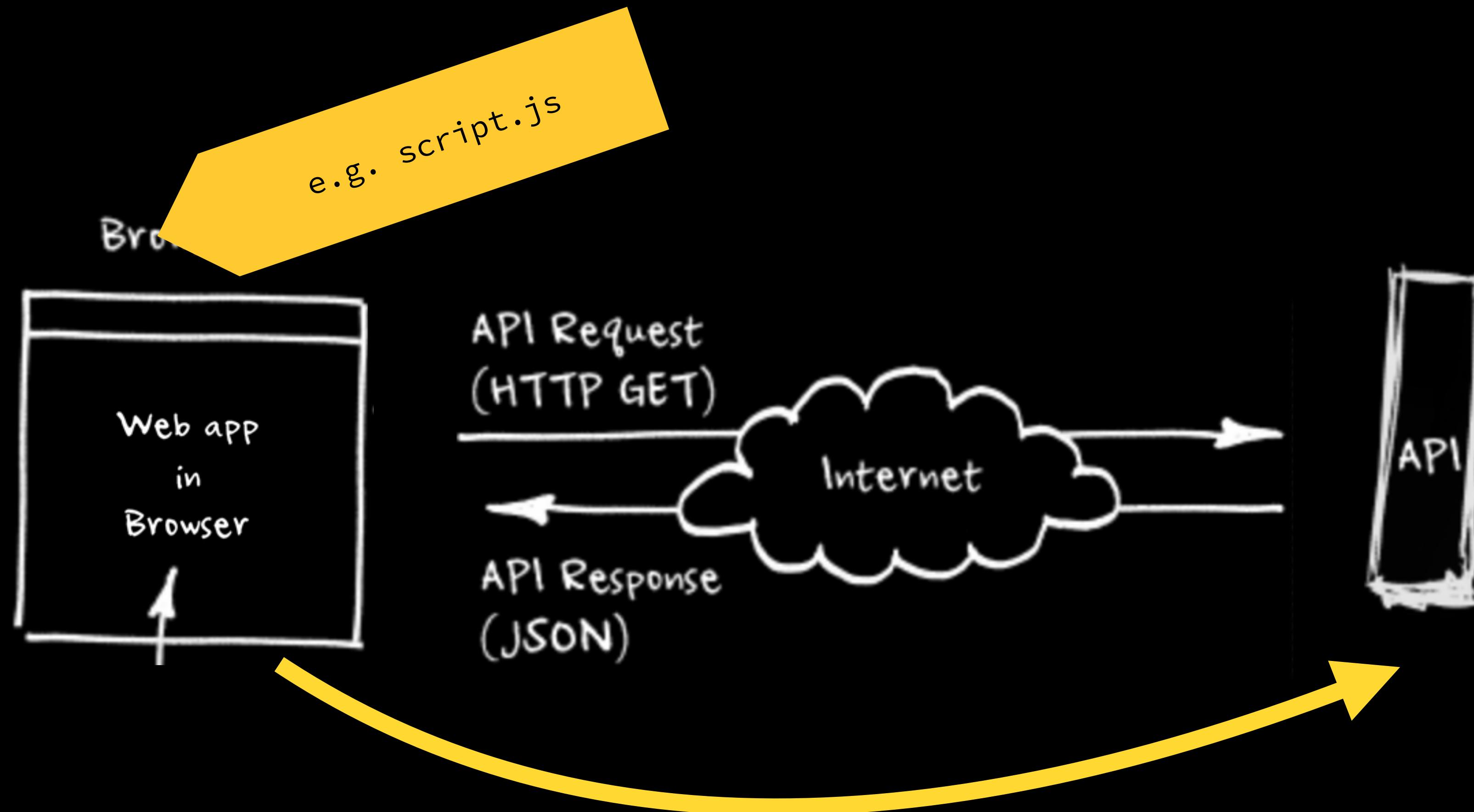
barChart.svelte



pieChart.svelte

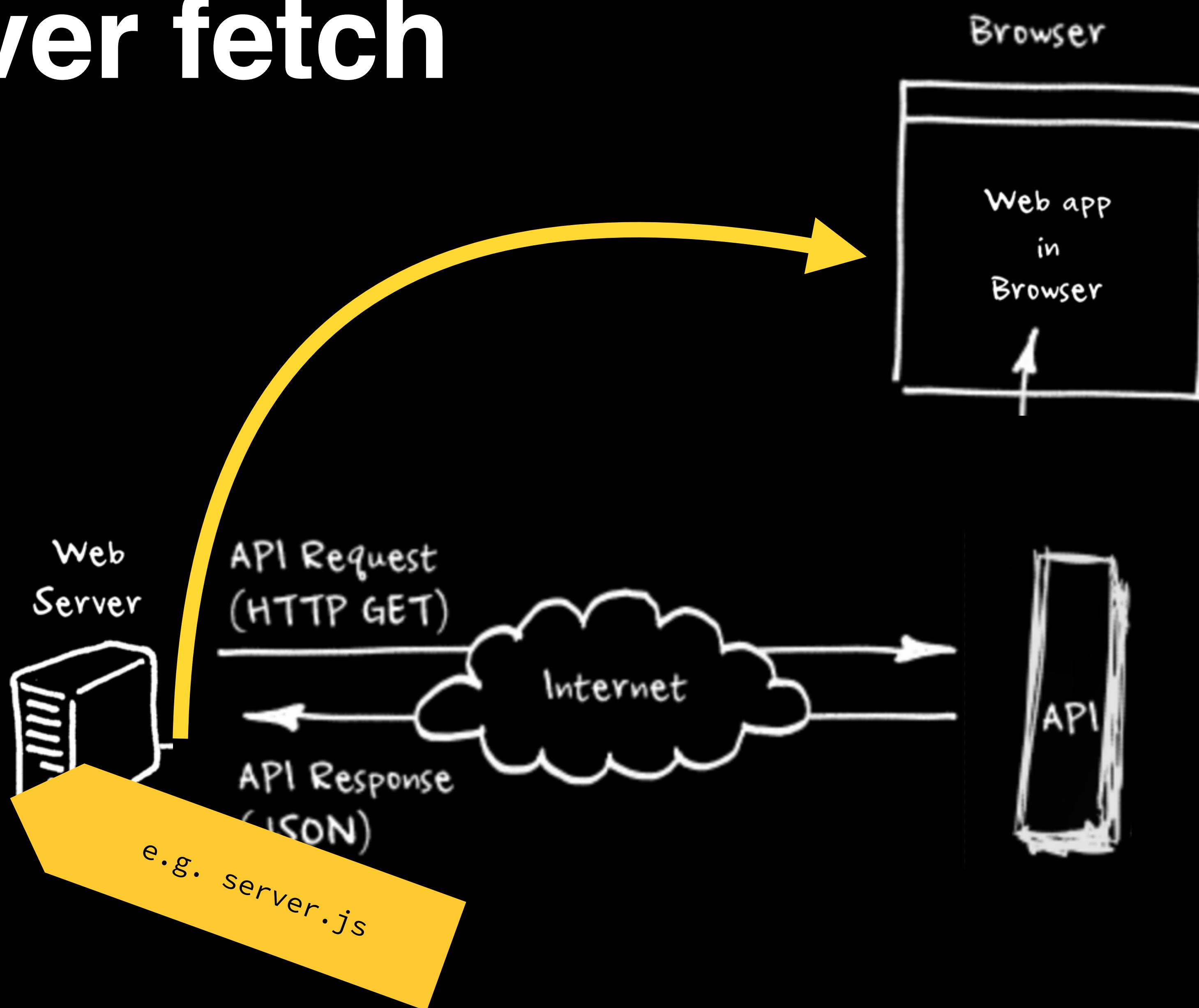
Client fetch

API



Server fetch

API



.env

```
# .env file
USER_ID="239482"
USER_KEY="foobar"
NODE_ENV="development"
```

server.js

```
require('dotenv').config();

process.env.USER_ID; // "239482"
process.env.USER_KEY; // "foobar"
process.env.NODE_ENV; // "development"
```

Data fetch module in Svelte

Methode	Reactief	Bereik	Gebruik
1. Stores	✓	Globaal	App-brede state
2. Import/export	✗	Globaal	Constanten, helpers
3. Load + data	✓ (SSR)	Pagina/layout	Serverdata, API's

https://svelte.dev/docs/kit/load

The screenshot shows the Svelte documentation page for 'Loading data'. The page is dark-themed with white text. At the top, there's a navigation bar with links for 'SVELTE', 'Docs', 'Tutorial', 'Packages', 'Playground', and 'Blog'. On the right side of the navigation bar are icons for search, GitHub, and other social media. The main content area has a sidebar on the left with sections for 'Getting started', 'Core concepts', 'Build and deploy', and 'Advanced'. The 'Core concepts' section includes a link to 'Loading data', which is highlighted in red. The main content area has a heading 'Loading data' and a paragraph explaining that before a `+page.svelte` component is rendered, data is loaded using `load` functions. Below this, there are two code snippets: one for `+page.ts` showing a `load` function that returns a `post` object with `title` and `content` properties, and another for `+page.svelte` showing the component structure with `script` and `h1` tags. At the bottom, there's a note about legacy mode and a link to the previous version of the documentation.

Getting started

- Introduction
- Creating a project
- Project types
- Project structure
- Web standards

Core concepts

- Routing
- Loading data**
- Form actions
- Page options
- State management
- Remote functions

Build and deploy

- Building your app
- Adapters
- Zero-config deployments
- Node servers
- Static site generation
- Single-page apps
- Cloudflare
- Cloudflare Workers
- Netlify
- Vercel
- Writing adapters

Advanced

- Advanced routing
- Hooks
- Errors
- Link options
- Service workers
- Server-only modules
- Snapshots
- Shallow routing
- Observability
- Packaging

On this page

- Loading data
- Page data**
- Layout data
- page.data
- Universal vs server
- Using URL data
- Making fetch requests
- Cookies
- Headers
- Using parent data
- Errors
- Redirects
- Streaming with promises
- Parallel loading
- Rerunning load functions
- Implications for authentication
- Using getRequestEvent
- Further reading

Page data

A `+page.svelte` file can have a sibling `+page.ts` that exports a `load` function, the return value of which is available to the page via the `data` prop:

```
src/routes/blog/[slug]/+page.ts
import type { PageLoad } from './$types';

export const load: PageLoad = ({ params }) => {
  return {
    post: {
      title: `Title for ${params.slug} goes here`,
      content: `Content for ${params.slug} goes here`
    }
  };
}
```

```
src/routes/blog/[slug]/+page.svelte
<script lang="ts">
  import type { PageProps } from './$types';

  let { data }: PageProps = $props();
</script>

<h1>{data.post.title}</h1>
<div>{@html data.post.content}</div>
```

Legacy mode Before version 2.16.0, the props of a page and layout had to be typed individually:

```
✓ week-3
  ✓ dataLoading
    > .svelte-kit
    > node_modules
      node_modules
        > static
        ⚡ .gitignore
        📄 .npmrc
        {} jsconfig.json
        {} package-lock.json
        {} package.json
        ⓘ README.md
        JS svelte.config.js
        ⚡ vite.config.js
```

Schedule

1. Events
2. Modules (import and export)
3. Async data fetching
- 4. All together!**



Huiswerk

- * Kies een strategie voor data fetchen
- * Zorg ervoor dat je fetchen async doet
- * Doe functioneel (pure functie met return)

**Uncaught SyntaxError
Unexpected end of input**