FRONTEND

A frontend fejlesztés helyzete

Miről lesz szó?

- Frontend fejlesztés eszközkészlete
- CSS keretrendszerek, preprocesszor
- Package manager
- JS keretrendszerek, State management, Build toolok, Teszt keretrendszerek
- UI library-k
- Code quality tools
- CI/CD
- Microfrontend
- Trendek, problémák

15 ÉVNYI SZENVEDÉSSEL ÉS NÉMI ELFOGULTSÁGGAL

A Frontend helyzete napjainkban

„Minimális” tudás eszköztára

RÉGEN

- HTML, CSS, JS, jQuery (és te voltál az Isten)

MOST

- HTML, CSS, JS
- CSS keretrendszer
- CSS preprocesszor
- Package manager
- JS keretrendszer
- State management
- Build tool
- Teszt keretrendszer
- UI library
- Code quality tools
- És még bőven van mit tanulni: https://roadmap.sh/frontend

CSS


CSS
Global
HD

CSS keretrendszerek – Pro/Contra

PRO

- Nem kell minden egyes elemet saját kezűleg formázni
- Nem kell a responsive-itással foglalkozni
- Változók segítségével testre szabható sok alapértelmezés. (színek, eltartások, árnyékok, stb)
- Sok esetben dinamikus elemeket is kapunk. (nem pure css kertrendszer, JS is van benne)
- Moduláris

CONTRA

- Sablonos oldalak
- Meg kell tanulni a class-eket (azaz eleinte sokat kell a doksit túrni)
- Időnként kézzel kell felülírni a stílusokat
- Néha szívás a frissítés
- Elfelejted a CSS-t

CSS keretrendszerek

- Bootstrap
- Foundation
- Bulma
- Materialize
- MUI
- Tailwind (utility class 💩)
- UIKit
- Primer
- https://github.com/troxler/awesome-css-frameworks

CSS pre-processzorok

A CSS-t egészítik ki a programozásból ismert vezérlési szerkezetekkel, függvényekkel, változókkal (már van natívan is) modulrendszerrel (már van natívan is),

- LESS
- Stylus
- **SASS/SCSS**
- PostCSS

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
```

```
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```

JavaScript

JavaScript keretrendszerek és társaik

- Prototype.js
- Dojo
- Mootools
- YUI
- jQuery
- Knockout
- ExtJS
- AngularJS
- Backbone
- Ember
- Meteor
- Aurelia
- Preact
- Alpine
- Solid
- Stencil
- Lit
- Polymer
- **Vue**
- **React**
- **Angular**
- **Svelte**
- Solid
- Quick
- Nuxt
- Next
- Remix
- Blitz
- RedwoodJS
- Blazor (webASM)

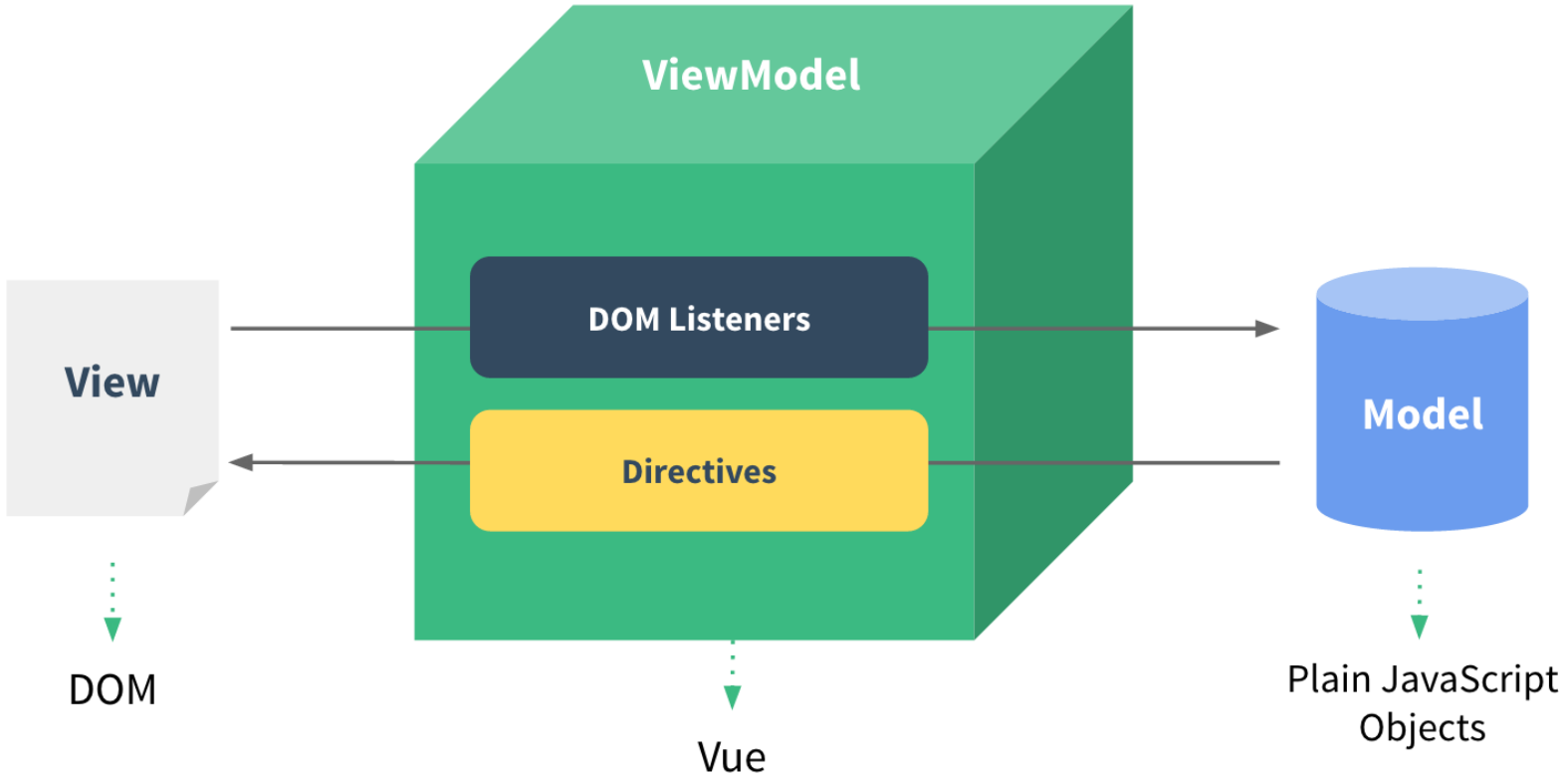
Trendek

Dinamikusan változnak!

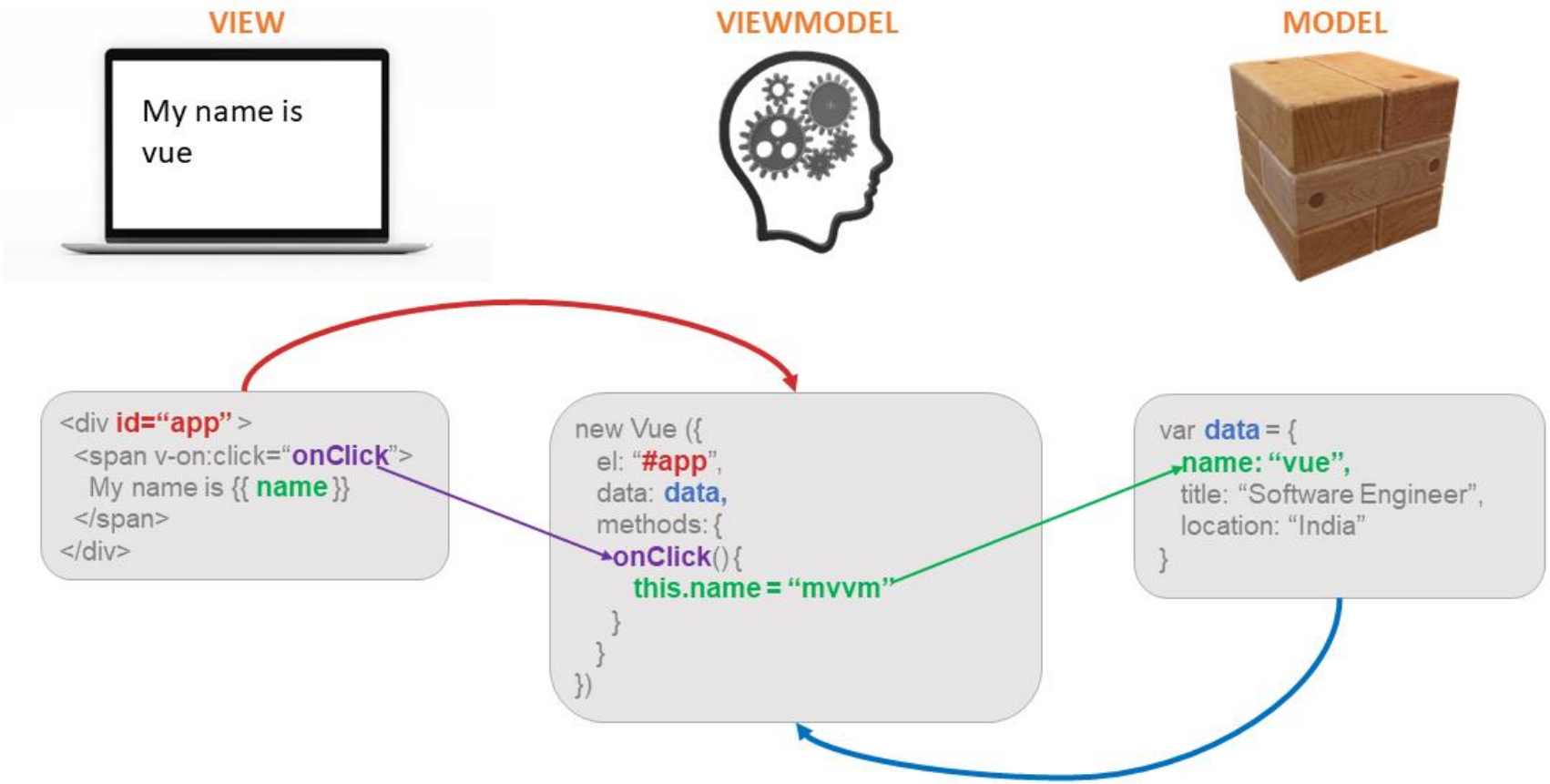
Mert az eszközök dinamikusan változnak!

Értem én, hogy gőzmozdony, de mi „hajcsa”?

Model-View-ViewModel



Model-View-ViewModel (egy ideális világban)



Virtual DOM (React, Vue)

1. A DOM leképzése js object szinten
2. Ha valami módosul, akkor a Virtual DOM-ot módosítja, ez gyorsabb, mint a DOM módosítása
3. Megnézi, mit kell a DOM-ban módosítani, azaz a Virtual DOM előző állapotához képest mi változott
4. Módosítja a DOM megfelelő részét

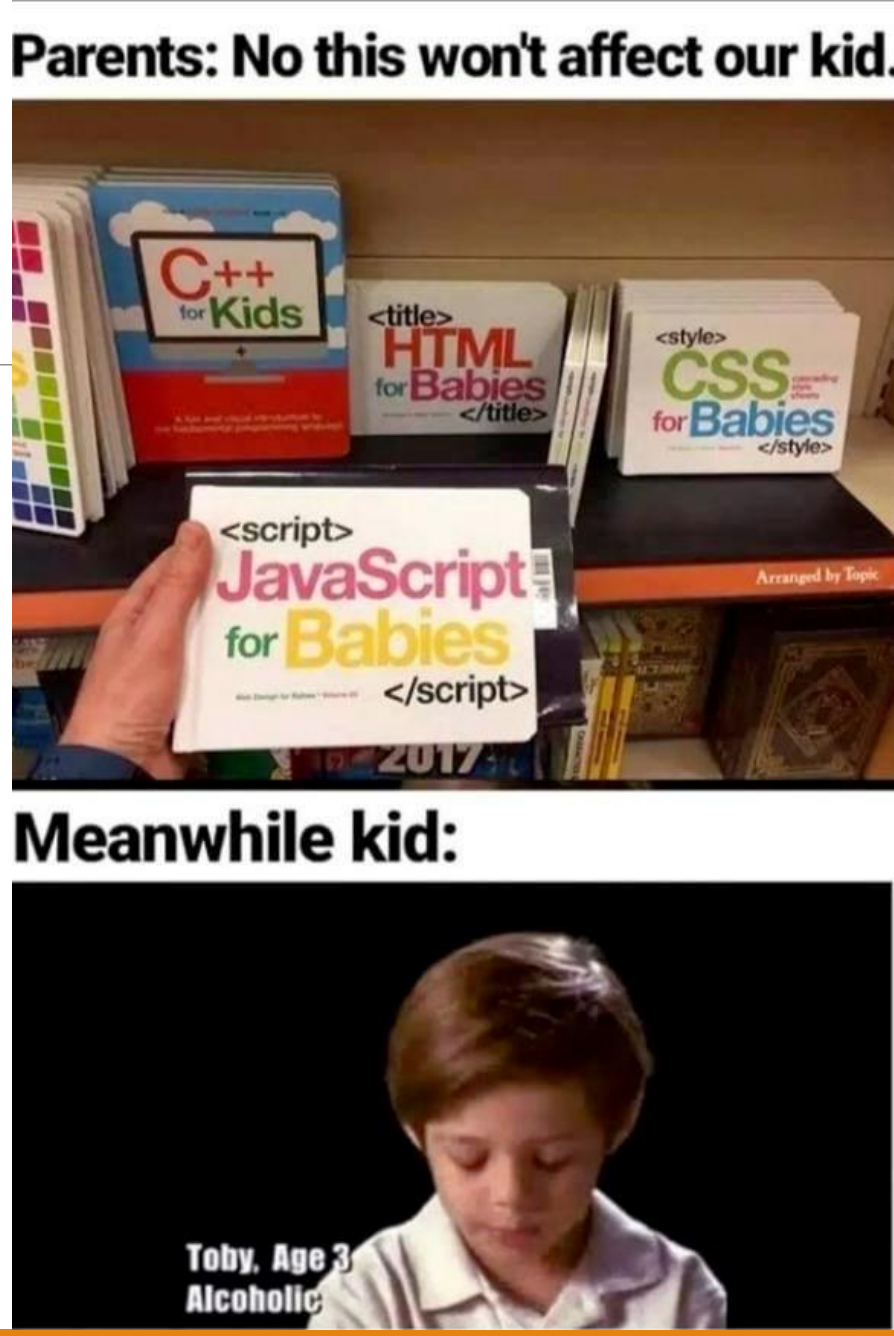
Virtual DOM = „pure overhead” - **Rich Harris,** creator of Svelte

Mikor melyiket használjam?

Egyiket se!

Maradj backenden!

Komolyan!



Mikor melyiket használjam? Komolyra fordítva a szót

- Kicsi vagy közepes projekteknél:
  - Vue
  - React
- Nagy projekteknél:
  - Next
  - Nuxt
  - Angular

Web komponensek

- Egyedi HTML elemek készítése
- Saját működést lehet hozzájuk írni
- Saját stílust tudunk hozzárendelni, ami csak a komponensre érvényes
- Shadow DOM-ot használ, ami a valódi DOM-ba lesz felcsatolva
- **A Shadow DOM nem egyenlő a Virtual DOM-mal**

Web komponensek – szabványosítási kísérlet

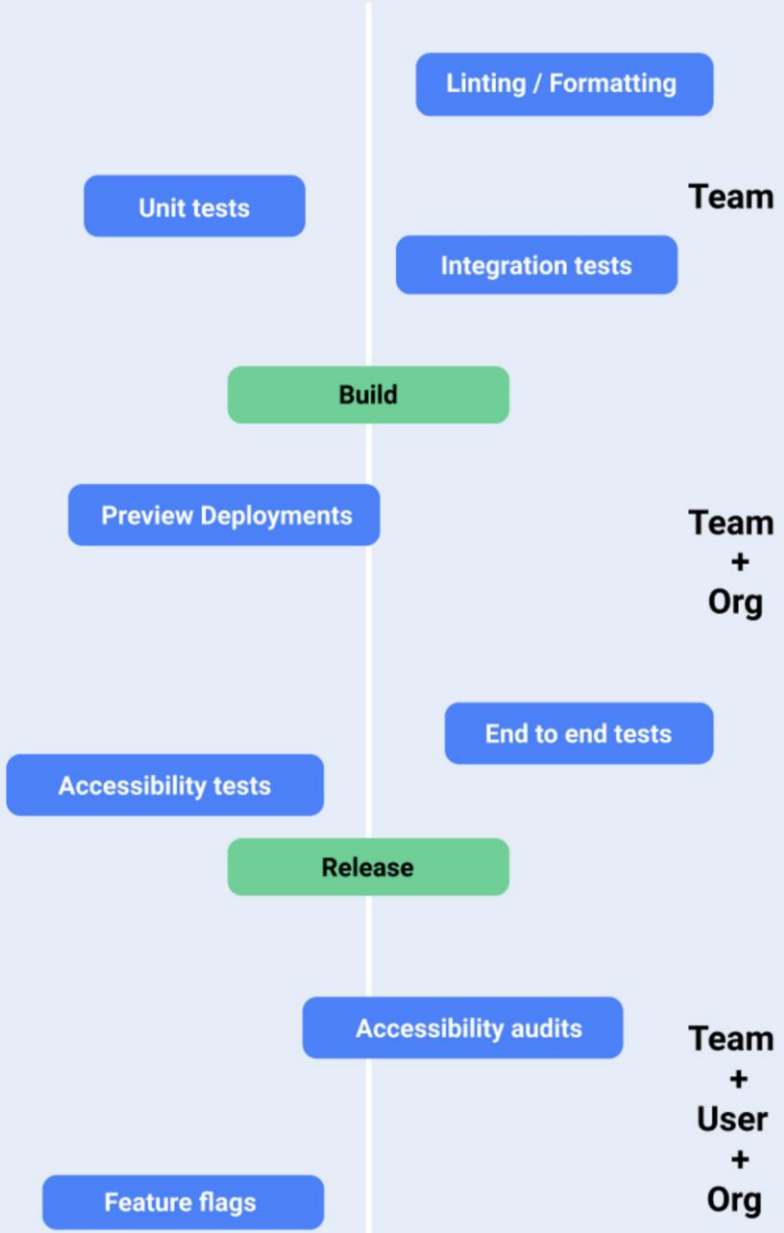
```
class MyComponent extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `<h1>Hello world</h1>`;
  }
}

customElements.define('my-component', MyComponent);
```

lion

Build toolok, task runnerek

- Grunt
- Gulp
- Webpack
- Rollup
- Parcel
- Snowpack
- **esbuild (Go-ba írták)**
- **Vite (esbuild-et használ)**


Linting / Formatting
Unit tests
Team
Integration tests
Build
Preview Deployments
Team
+
Org
End to end tests
Accessibility tests
Release
Accessibility audits
Team
+
User
+
Org
Feature flags

CI/CD

Linterek

- JsLint (2002)
- JsHint (JSHint fork, sokkal testre szabhatóbb)
- JSCS (2014-2016, az ESLinthez csatlakoztak a fejlesztők)
- **Eslint**
- Google Closure Compiler (2015-től elavult)
- TSLint (az ESLint része már)

Testing

RÉGEN

- Unit, integration:
  - Test runner: Karma
  - Testing framework: Mocha
  - Assertion library: Chai
  - Code coverage: Istambul
- e2e:
  - Protractor

MOST

- Unit, integration:
  - Jest
- e2e:
  - Cypress

Akadálymentesítés

- Szabvány: https://www.w3.org/WAI/fundamentals/accessibility-intro/
- 3 szint: A, AA, AAA: https://www.w3.org/WAI/WCAG2AA-Conformance
- Checklist: https://www.a11yproject.com/checklist/

Csomagkezelők, függőségek kezelése

- **npm**
- **yarn**
- **pnpm**
- bower
- jspm

Verziószámozás

Semantic versioning

- https://docs.npmjs.com/about-semantic-versioning
- https://docs.npmjs.com/cli/v8/configuring-npm/package-json#dependencies

Code status	Stage	Rule	Example version
First release	New product	Start with 1.0.0	1.0.0
Backward compatible bug fixes	Patch release	Increment the third digit	1.0.1
Backward compatible new features	Minor release	Increment the middle digit and reset last digit to zero	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit and reset middle and last digits to zero	2.0.0

Minden egyéb

Szerveroldali kommunikáció

- XMLHttpRequest/fetch API
- Websocket (Kétirányú)
- SSE (egyirányú)
- Push API (az oldal bezárása után is él)

Programozási paradigmák

- Procedurális
- Funkcionális
- Objektumorientált
- Reaktív

CSR, SSR, Hydration, SSG, Pre-Rendering meg a kutya…

- **CSR**: Client Side Rendering
- **SSR**: Server Side Rendering
- **SSG**: markdawnból vagy egyéb forrásokból töltik be a sablonba a tartalmat. Elkészítik a html-t (build time) és a böngésző felé már ezt küldik el. Elküldi a JS-t is. Ha ez letöltődött és inicializálódott, le lesz cserélve az eddigi statikus content. Ezt nevezik **Hydration-nek**.

https://medium.com/@prashantramnyc/server-side-rendering-ssr-vs-client-side-rendering-csr-vs-pre-rendering-using-static-site-89f2d05182ef

Microfrontend, vannak lehetőségek

Működés szempontjából megvalósítható:

- Web components-ek segítségével
- Webpack: Module Federation
- Angular: Lazy-loading feature modules
- Átlinkelés
- Framek

Problémák: kódszervezés, egy csomó plusz konfig, mi van ha nem Angulart, vagy Webpacket használok, state megosztása?

Éleseben nem használnám.

Webassembly

Jött…

…Látott…

… aztán ennyi volt

MS Blazor

UI libraryk

Csak pár fontosabb:

- Angular: MUI
- Vue: Vuetify
- React: MUI/Ant

Error monitoring

- Sentry (error tracking)
- Logrocket (user interaction, session replay, etc.)
- Exceptionless (error discovering)

Problémák – mert vannak

- Változó eszközkészletek
- Inkonzisztens kód
- Egyre nagyobb alkalmazások, egyre komplikáltabb kód, egyre több eszköz
- Keretrendszerek frissítése (AngularJS -> Angular, React before, and after v16.8)
- Függőségek kezelése: sok függőség = sok lehetséges problémaforrás
- A megfelelő CSS/JS tudás nélkül kezdenek el keretrendszereket használni. Nincs meg a megfelelő tanulási görbe. Nincs code review, nem tanulnak a hibákból.

Callback, Promise, async await (+generátorok)

```
// CALLBACK  HELL
function addOneTo(number, callback) {
    let result = number + 1;
    if (callback) {
        setTimeout(function () {
            callback(result);
        }, 1000)
    }
}

addOneTo(5, function (res1) {
    addOneTo(res1, function (res2) {
        addOneTo(res2, function (res3) {
            addOneTo(res3, function (res4) {
                addOneTo(res4, function (res5) {
                    console.log(res5);
                });
            });
        });
    });
});
```

```
// PROMISE

function addOneTo(number) {
  let result = number + 1
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve(result), 1000)
  })
}

addOneTo(5)
  .then((res1) => addOneTo(res1))
  .then((res2) => addOneTo(res2))
  .then((res3) => addOneTo(res3))
  .then((res4) => addOneTo(res4))
  .then((res5) => {
    console.log(res5)
  })
```

```
// ASYNC AWAIT

function addOnTo(number) {
  let result = number + 1
  return new Promise((resolve, reject) => {
    resolve(result)
  })
}

async function main() {
  const res1 = await addOnTo(5)
  const res2 = await addOnTo(res1)
  const res3 = await addOnTo(res2)
  const res4 = await addOnTo(res3)
  const res5 = await addOnTo(res4)
  console.log(res5)
}

main()
```

Hasznos linkek

MDN

ECMAScript

JavaScriptToday

2ality

James Sinclair

JavaScript Info

CSSTricks

Ahmad Shadeed

Modern CSS