

Kodkvalitet

- Validering och hjälp
 - <https://validator.w3.org/>
 - <http://www.css-validator.org/>
- JSLint är ett analysverktyg som används för att kontrollera om koden klarar kodreglerna för JS.
 - <https://www.jshint.com/>
 - <https://jshint.com/>

Felsökning

- Att hitta fel i en sida kan ofta vara komplicerat, speciellt i större projekt eller ramverk.
- Många filer inkluderas, både interna och externa (t ex tredjepartsbibliotek).

Felsökning

- Vad är en bugg?
 - Syntax-fel
 - Logiska fel
 - Exekveringsfel
 - Feature

Felsökning

- Syntax-fel
 - När man använt språket på fel sätt, när JS-tolken inte förstår våra instruktioner.
- Skrivfel
- Utelämnade tecken, osynliga tecken
- Felstavade identifierare (variabler, funktioner, konstanter klasser)
- Odefinierade variabler

Replace a semicolon (;) with a greek question mark (;) in your friend's JavaScript and watch them pull their hair out over the syntax error.



Felsökning

- Logiska fel
 - Vi använder språket korrekt men får inte det resultat vi har tänkt oss, t ex svar som blir fel bara för en viss sorts indata (specialfall).
 - Ett logikfel är ett fel i ett program som gör att det fungerar felaktigt, men inte att det avslutas onormalt (kraschar). Ett logikfel ger oönskad utdata eller annat beteende.

Felsökning

- Exekveringsfel (krasch)
- Dividera med noll
- Array utanför index
- Oändliga loopar
- Minnesläckor

```
#include <stdio.h>

int main(void)
{
    int a = 0;
    while (a < 10) {
        printf("%d\n", a);
        if (a == 5)
            printf("a equals 5!\n");
        a++;
    }
    return 0;
}
```

```
01 class Example {
02     public static void main(String[] args) {
03         char[] matrix = new char[] {'H', 'e', 'l', 'l', 'o'};
04         System.out.println(matrix);
05
06         /* Print each letter of the char array in a separate line. */
07         for(int i = 0; i <= matrix.length; ++i) {
08             System.out.println(matrix[i]);
09         }
10     }
11 }
```

Felsökning

- Feature
 - Ibland är en bugg inte en bugg utan ett missförstånd i vad en funktion verkligen ska göra.
 - Användaren förväntar sig att något ska hända som systemet inte är tänkt att hantera.
 - Viktigt att tänka på vid gränssnittsdesign.

Exempel

- Koda i fel fil/fel miljö
- Spindeln
- Städarskan

Felsökning

- Hur hittar man en bugg?
 - Isolering & uteslutning
 - Spårutskrifter
 - Debug-verktyg

Isolering & uteslutning

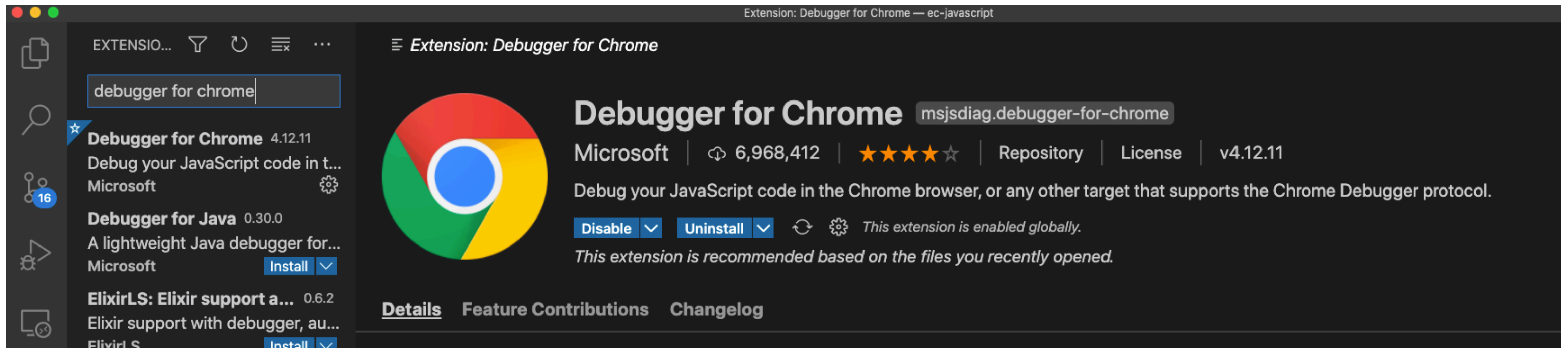
- Att isolera och utesluta buggar innebär att man testar så mycket man kan i både gränssnitt och kod för att ta bort faktorer som kan påverka problemet.
- Det kan t ex vara att testa vilka data som krävs för att återskapa buggen.
- Är det bara om funktionen får en viss typ av data? (Sträng, int, negativt tal osv.)
- Uppstår buggen bara under vissa omständigheter?
- Kan man kommentera bort kodrader och se om problemet fortfarande kvarstår?

Spårutskrifter

- Man kan skriva ut variabler, uppbyggda strängar osv för att undersöka vad de innehåller.
- `console.log(variable);`

Debug i VSC

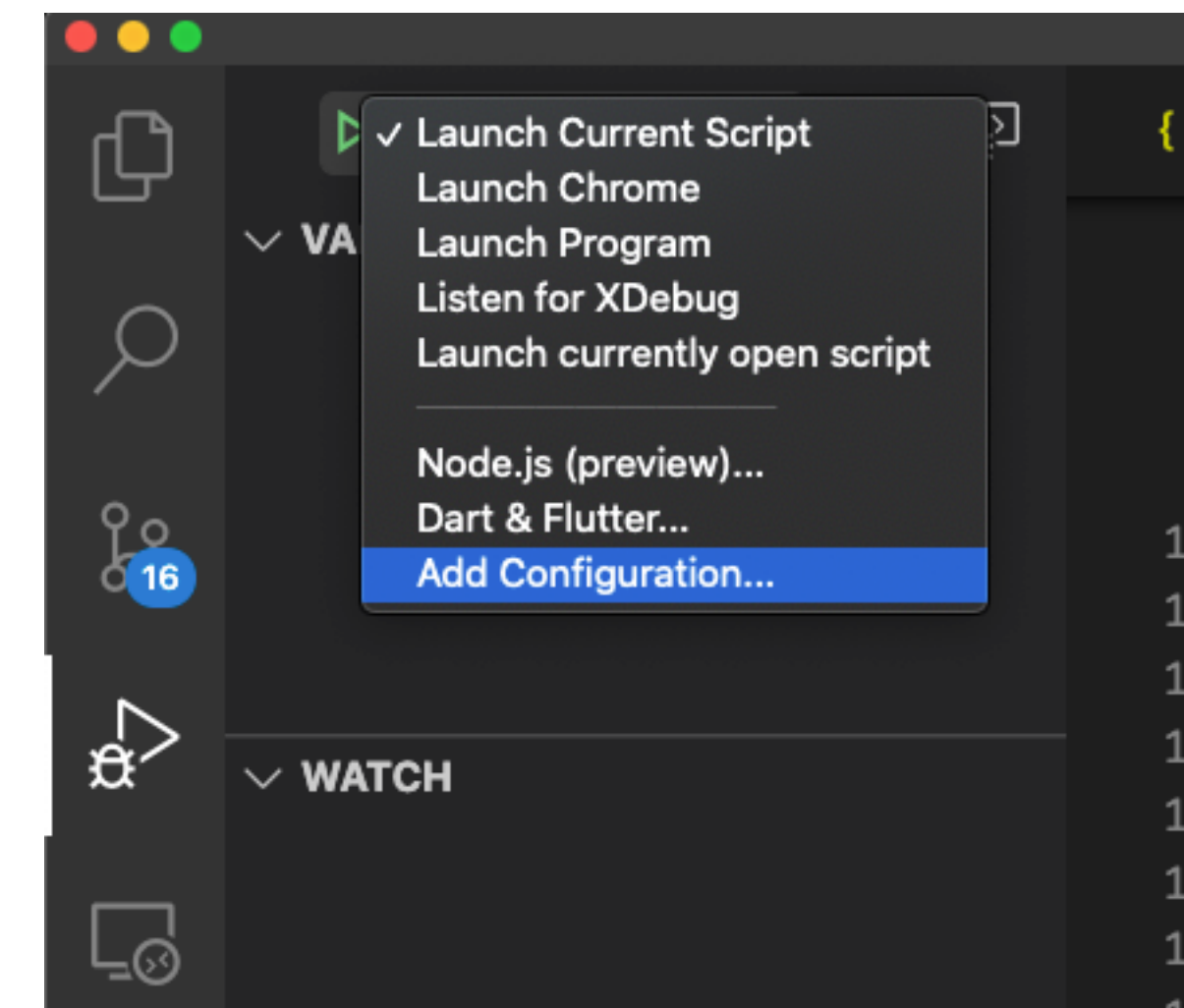
- Vi har tidigare använt debuggern i VSC när vi har jobbat med Node.
- För browser-debug får vi fixa lite extra.
- Installera extensionet "Debugger for Chrome"



Konfiguration

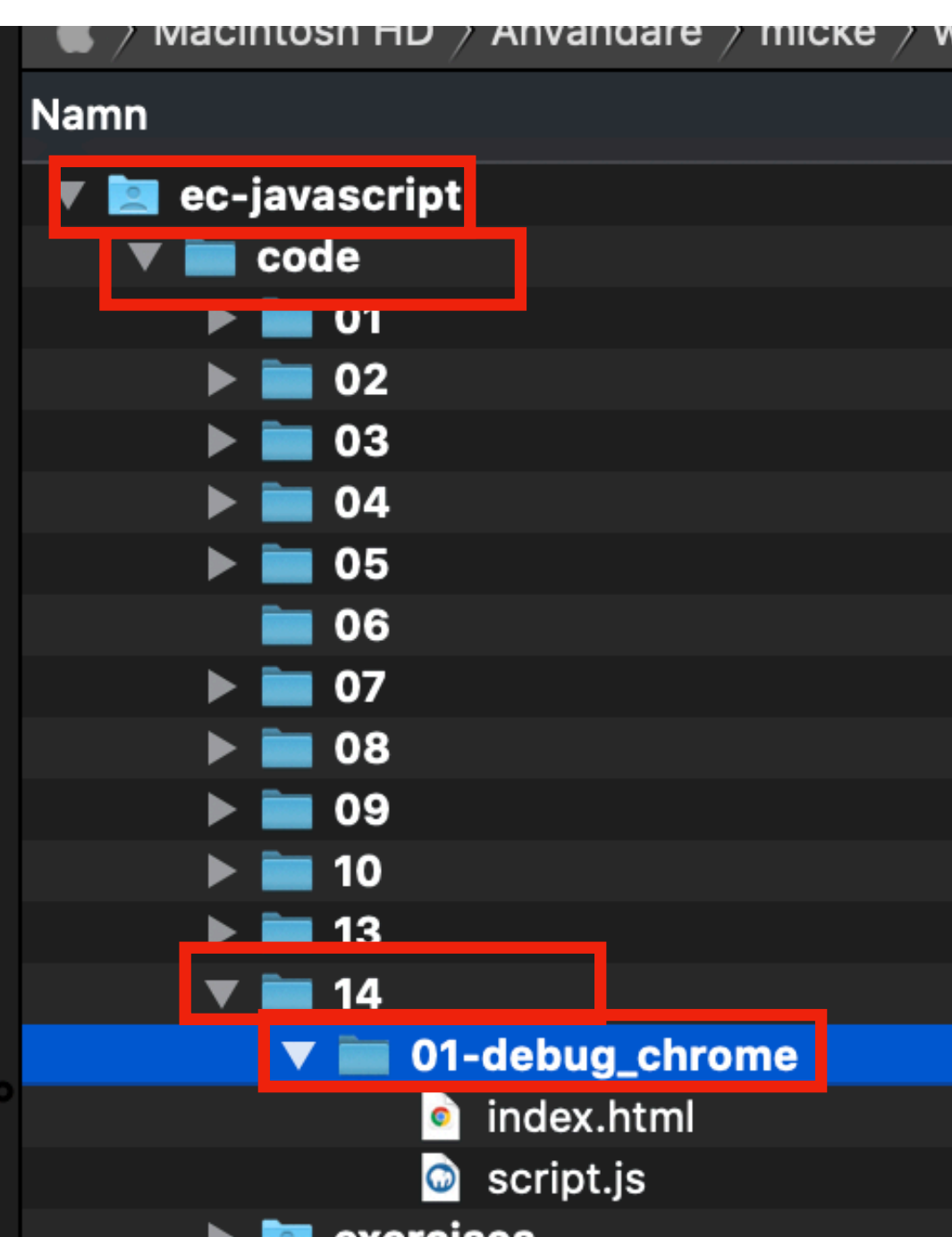
- Man kan konfigurera på lite olika sätt:
<https://github.com/microsoft/vscode-chrome-debug>
- Vi ska konfigurera vår debugger att starta chrome med vårt skript.

```
{  
  "name": "Launch Current Script",  
  "request": "launch",  
  "type": "chrome",  
  "url": "http://127.0.0.1:5500/${relativeFile}",  
  "webRoot": "${workspaceFolder}"  
},
```



Webbserver

- En webbrowser utgår normalt från en rot-katalog.
- Adresser på webbrowsern kommer att "mappa" mot adresser i vårt filsystem.
- I det här fallet har jag öppnat mappen `ec-javascript` som min rot, där ligger alla filer jag har för detta projekt.



- Sökvägen till min "debug-fil" utifrån roten blir:
`/Users/micke/www/projects/ec_education/ec-javascript/
code/14/01-debug_chrome/index.html`
- URL:en genom Liveserver blir:
`http://127.0.0.1:5500/code/14/01-debug_chrome/index.html`

fil-rot

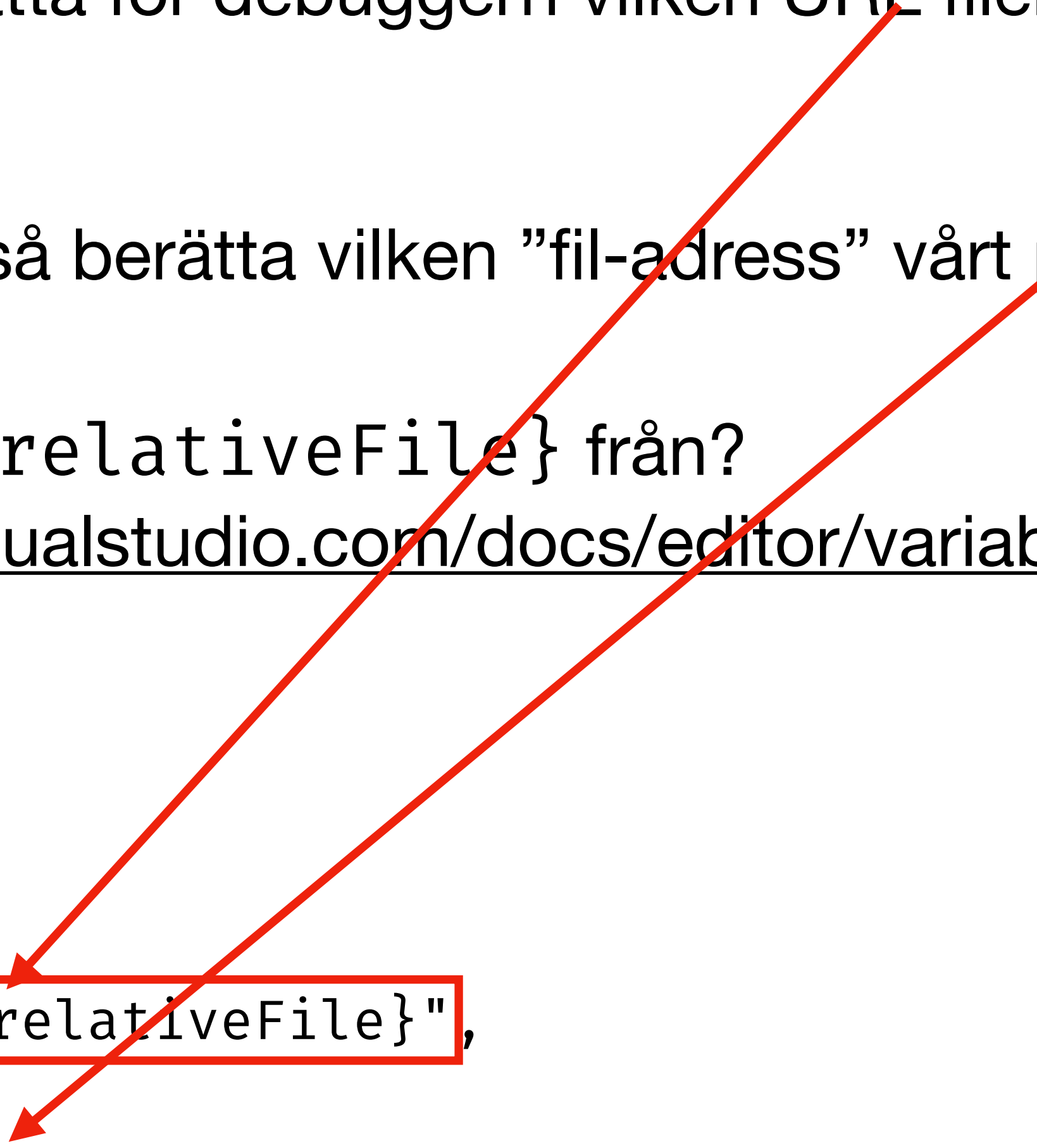
Sökväg

www-rot

Konfiguration

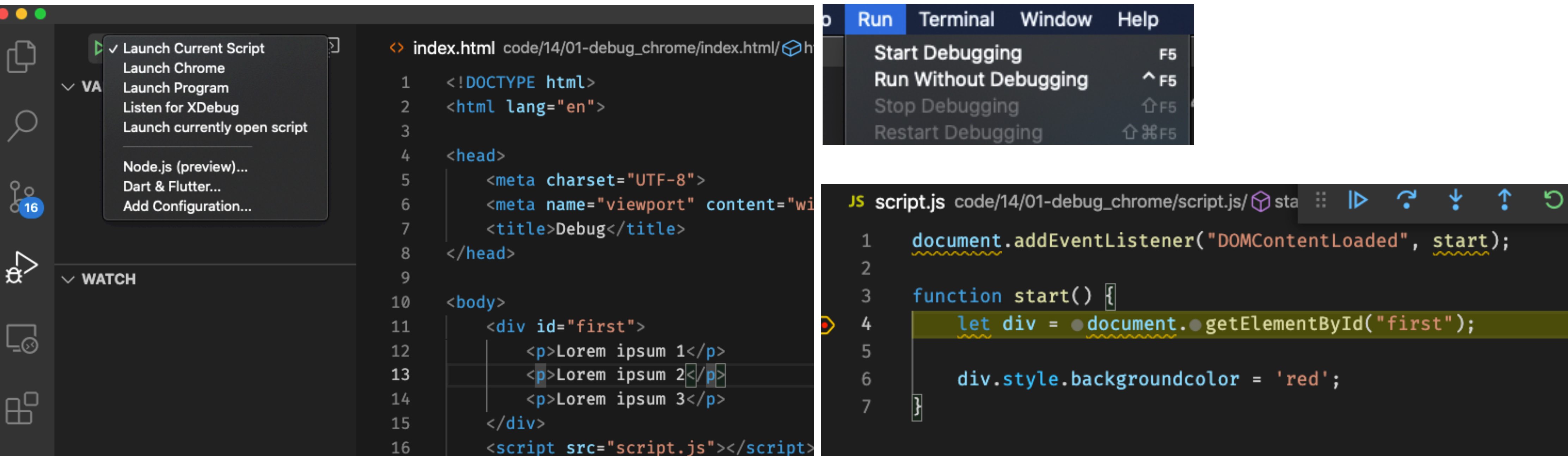
- Vi behöver berätta för debuggern vilken URL filen vi vill debugga har.
- Vi behöver också berätta vilken "fil-adress" vårt projekt har.
- Var kommer `${relativeFile}` från?
<https://code.visualstudio.com/docs/editor/variables-reference>

```
{  
  "name": "Launch Current Script",  
  "request": "launch",  
  "type": "chrome",  
  "url": "http://127.0.0.1:5500/${relativeFile}",  
  "webRoot": "${workspaceFolder}"  
},
```



Debugga

- Nu bör vi vara klara att köra! Set en breakpoint, välj "Launch Current Script" och "Start Debugging".



What is an error in JavaScript?

- An error in JavaScript is an object, which is later thrown to halt the program.
- To create a new error in JavaScript we call the appropriate constructor function. For example, to create a new, generic error we can do:
- `const err = new Error("Something bad happened!");`
- Once created, the error object presents three properties:
 - `message`: a string with the error message.
 - `name`: the error's type.
 - `stack`: a stack trace of functions execution.

```
const wrongType = TypeError("Wrong type given, expected number");  
  
wrongType.message; // "Wrong type given, expected number"  
wrongType.name; // "TypeError"
```

Types of errors

- Error
- EvalError
- InternalError
- RangeError
- ReferenceError
- SyntaxError
- TypeError
- URIError

TypeError

```
const name = "Jules";  
name = "Caty";
```

```
// TypeError: Assignment to constant  
variable.
```

When we select non-existent HTML elements in the page:

```
Uncaught TypeError: button is null
```

SyntaxError

```
va x = '33';  
// SyntaxError: Unexpected identifier
```

DOMException

```
document.body.appendChild(document.cloneNode  
(true));
```

```
// Uncaught DOMException: Node.appendChild:  
May not add a Document as a child
```

- <https://developer.mozilla.org/en-US/docs/Web/API/DOMException>

What is an exception?

- An error object becomes an exception only when it's thrown.
- To throw an exception in JavaScript we use `throw`, followed by the error object:

```
const wrongType = TypeError("Wrong type given, expected number");  
throw wrongType;
```

- The short form is more common, in most code bases you'll find:

```
throw TypeError("Wrong type given, expected number");
```

- or

```
throw new TypeError("Wrong type given, expected number");
```

Where to throw an Exception

- It's unlikely to throw exceptions outside of a function or a conditional block.

```
function toUppercase(string) {  
  if (typeof string === "string") {  
    throw TypeError("Wrong type given, expected a string");  
  }  
  
  return string.toUpperCase();  
}
```

What to throw

- Technically, you could throw anything in JavaScript, not only error objects:

```
throw Symbol();  
throw 33;  
throw "Error!";  
throw null;
```

- However, it's better to avoid these things: always throw proper error objects, not primitives.
- By doing so you keep error handling consistent through the codebase. Other team members can always expect to access `error.message` or `error.stack` on the error object.

What happens when we throw an exception?

- Exceptions are like an elevator going up: once you throw one, it bubbles up in the program stack, unless it is caught somewhere.

```
function toUppercase(string) {  
  if (typeof string === "string") {  
    throw TypeError("Wrong type given, expected a string");  
  }  
  
  return string.toUpperCase();  
}
```

```
toUppercase(4);
```

```
// _____
```

```
Uncaught TypeError: Wrong type given, expected a string  
    toUppercase http://localhost:5000/index.js:3  
    <anonymous> http://localhost:5000/index.js:9
```

Stack trace

- The stack trace goes from bottom to top. So here:

```
toUpperCase http://localhost:5000/index.js:3  
<anonymous> http://localhost:5000/index.js:9
```

- We can say:
 - something in the program at line 9 called `toUpperCase`
 - `toUpperCase` blew up at line 3
- In addition to seeing this stack trace in the browser's console, you can access it on the `stack` property of the `error` object.

Handling Exceptions

- If the exception is uncaught, that is, nothing is done by the programmer to catch it, the program will crash.
- When, and where you catch an exception in your code depends on the specific use case.
- For example you may want to propagate an exception up in the stack to crash the program altogether. This could happen for fatal errors, when it's safer to stop the program rather than working with invalid data.

Error handling for regular functions

```
function toUppercase(string) {  
  if (typeof string === "string") {  
    throw TypeError("Wrong type given, expected a string");  
  }  
  return string.toUpperCase();  
}
```

```
toUppercase(4);
```

- Here the engine calls and executes toUppercase. All happens synchronously. To catch an exception originating by such synchronous function we can use try/catch/finally:

```
try {  
  toUppercase(4);  
} catch (error) {  
  console.error(error.message);  
  // or log remotely  
} finally {  
  // clean up  
}
```

try/catch/finally

```
try {  
  toUppercase(4);  
} catch (error) {  
  console.error(error.message);  
  // or log remotely  
} finally {  
  // clean up  
}
```

- Usually, `try` deals with the happy path, or with the function call that could potentially throw.
- `catch` instead, captures the actual exception. It receives the error object, which we can inspect (and send remotely to some logger in production).
- The `finally` statement on the other hand runs regardless of the function's outcome: whether it failed or succeeded, any code inside `finally` will run.

Läs mer

- <https://www.valentinog.com/blog/error/>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control flow and error handling](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling)

Mer om Quality Assurance

- Använd en kodstandard
- Parprogrammera (åtminstone ibland)
- Peer review
- Tester (ffa enhetstester)

Kodstandard

Coding standards are a set of guidelines, best practices, programming styles and conventions that developers adhere to when writing source code for a project.

<https://codeahoy.com/2016/05/22/effective-coding-standards/>

- <https://standardjs.com/>
- <https://google.github.io/styleguide/jsguide.html>

Parprogrammera

“Pair programmers: Keep each other on task. Brainstorm refinements to the system. Clarify ideas. Take initiative when their partner is stuck, thus lowering frustration. Hold each other accountable to the team’s practices.

Pairing”

Kent Beck, eXtreme Programming

- https://medium.com/@weblab_tech/pair-programming-guide-a76ca43ff389
- <https://medium.com/kayvan-kaseb/the-importance-of-pair-programming-for-developers-4a8b3325f62f>

Peer review

Code Review, or Peer Code Review, is the act of consciously and systematically convening with one's fellow programmers to check each other's code for mistakes

<https://smartbear.com/learn/code-review/what-is-code-review/>

- <https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/>
- <https://medium.com/palantir/code-review-best-practices-19e02780015f>

I Yatzy-grupperna, byt projekt med varandra

- Varje grupp tittar på en annan grupps repo.
- Ägna 30 minuter åt att titta igenom repot. Förstå hur koden fungerar. Skriv upp förslag på ändringar. Annan struktur, andra variabelnamn, andra kommentarer, annan kodstandard, andra lösningar.
- Ägna 15 minuter åt att ge den andra gruppen feedback.
- Var konstruktiva! Och ta inte feedbacken som att ni har gjort något dåligt, fundera öppet och ärligt om det finns något i feedbacken ni kan lära er av och bli bättre av.

Automatiska tester - kod som testar kod

Unit testing is the process of testing the implemented code at a module level. After every new module development, the entire suite of test cases is run to ensure that no existing modules are affected by the developed module.

<https://geekflare.com/javascript-unit-testing/>

- <https://mochajs.org/>

Hitta fel

- Uppgiften består av att få en given kod att fungera.
- Tänk på att det inte bara finns syntax-fel.
- <https://yh.pingpong.se/courseld/11861/content.do?id=5225412>

CSS Preprocessors

T ex SCSS & Less

- Är CSS ett programmeringsspråk?
- Preprocessorer används för att utöka möjligheterna med CSS, t ex:
 - Variabler
 - Nesting
 - Mixins
 - Loopar
 - Arv
- Olika för olika preprocessorer
- Kompileras till CSS

SCSS / Sass

- Syntactically awesome style sheets
- Style sheet language
- SassScript - a simple scripting language used in Sass files

SCSS / Sass

- There are two syntaxes available for Sass. The first, known as SCSS (Sassy CSS) and used throughout this reference, is an extension of the syntax of CSS. This means that every valid CSS stylesheet is a valid SCSS file with the same meaning. This syntax is enhanced with the Sass features described below. Files using this syntax have the `.scss` extension.
- The second and older syntax, known as the indented syntax (or sometimes just “Sass”), provides a more concise way of writing CSS. It uses indentation rather than brackets to indicate nesting of selectors, and newlines rather than semicolons to separate properties. Files using this syntax have the `.sass` extension.

SCSS

- Installera

- Alt 1:

```
npm install -g sass
```

- Alt 2 (Mac):

```
brew install sass/sass/sass
```

- Massor av alternativ:

- <https://sass-lang.com/install>

SCSS

- You can also watch individual files or directories with the `--watch` flag. The watch flag tells Sass to watch your source files for changes, and re-compile CSS each time you save your Sass. If you wanted to watch (instead of manually build) your `input.scss` file, you'd just add the watch flag to your command, like so:

```
sass --watch input.scss output.css
```

- You can watch and output to directories by using folder paths as your input and output, and separating them with a colon. In this example:

```
sass --watch app/sass:public/stylesheets
```

- Sass would watch all files in the `app/sass` folder for changes, and compile CSS to the `public/stylesheets` folder.

SCSS

- Variabler

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

SCSS

- Nesting

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  
  li { display: inline-block; }  
  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
  }  
}
```

SCSS

- Partial Sass files contain little snippets of CSS that you can include in other Sass files.
- Great way to modularize your CSS and help keep things easier to maintain.
- A partial is a Sass file named with a leading underscore i e `_partial.scss`.
- The underscore lets Sass know that the file is only a partial file and that it should not be generated into a CSS file.
- Sass partials are used with the `@use` rule.

SCSS

- You can split it up however you want with the @use rule.
- This rule loads another Sass file as a module, which means you can refer to its variables, mixins, and functions in your Sass file with a namespace based on the filename.
- When you use a file you don't need to include the file extension.

```
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

SCSS

- Mixins

```
@mixin transform($property) {  
  -webkit-transform: $property;  
  -ms-transform: $property;  
  transform: $property;  
}  
  
.box { @include transform(rotate(30deg)); }
```

SCSS

- @extend lets you share a set of CSS properties from one selector to another.
- It helps keep your Sass very DRY.
- A placeholder class is a special type of class that only prints when it is extended.

```
/* This CSS will print because %message-shared is extended. */
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// This CSS won't print because %equal-heights is never extended.
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}

.error {
  @extend %message-shared;
  border-color: red;
}

.warning {
  @extend %message-shared;
  border-color: yellow;
}
```

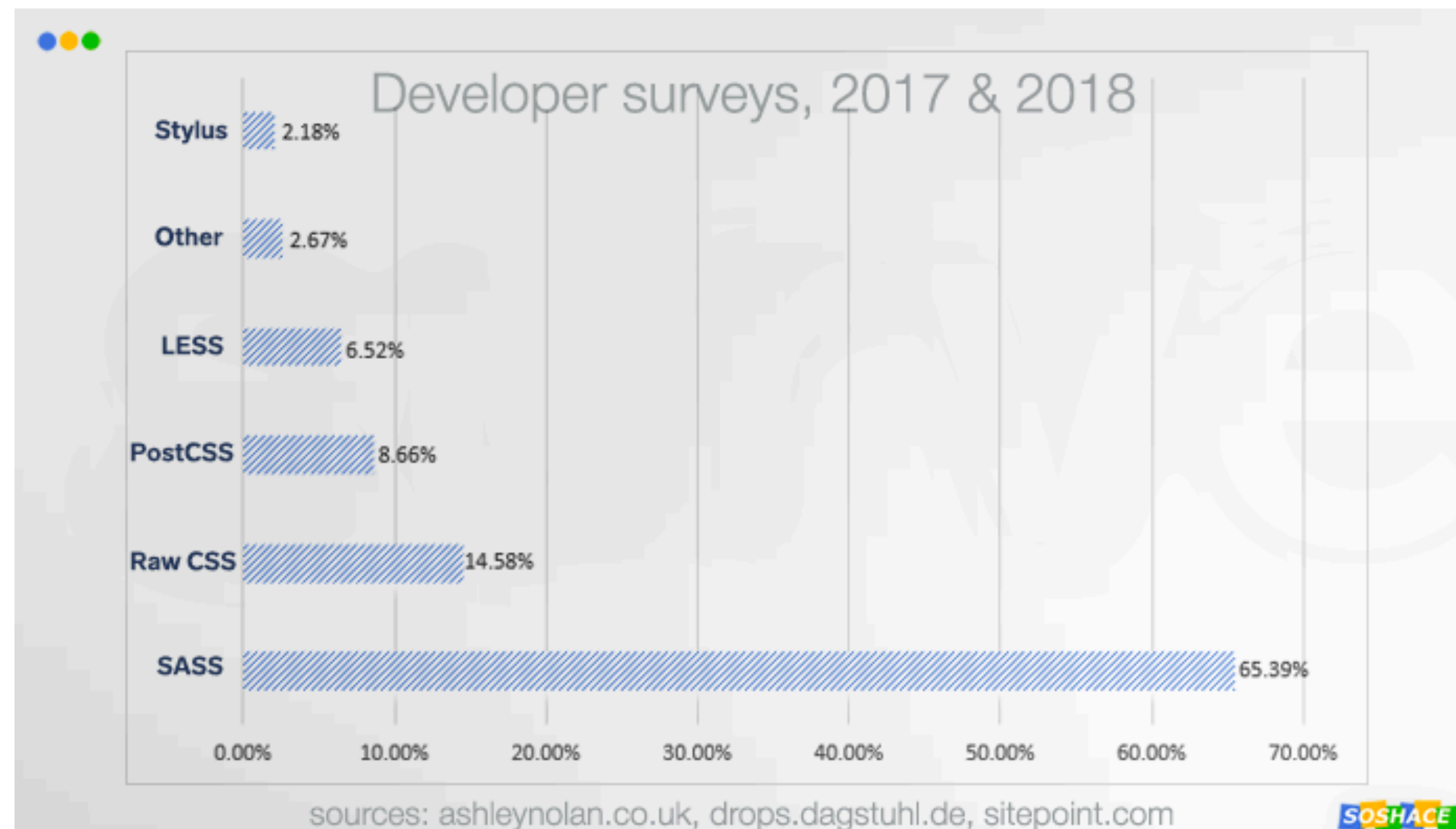

SCSS

- Sass has a handful of standard math operators like +, -, *, /, and %.

```
.container {  
  width: 100%;  
}  
  
article[role="main"] {  
  float: left;  
  width: 600px / 960px * 100%;  
}  
  
aside[role="complementary"] {  
  float: right;  
  width: 300px / 960px * 100%;  
}
```

Less eller SCSS?

- <https://blog.soshace.com/en/css/sass-vs-less-which-css-preprocessor-to-choose-in-2019/>
- <https://css-tricks.com/sass-vs-less/>



Andra preprocessorer

- Stylus
<http://stylus-lang.com/>
- PostCSS
<https://postcss.org/>

CSS

- Behöver man alls ha en preprocessor?
- Beror lite på vad man har behov av. CSS utvecklas och får hela tiden nya funktioner.
- CSS Grid, Flexbox...
- CSS custom properties
- <https://cathydutton.co.uk/posts/why-i-stopped-using-sass/>

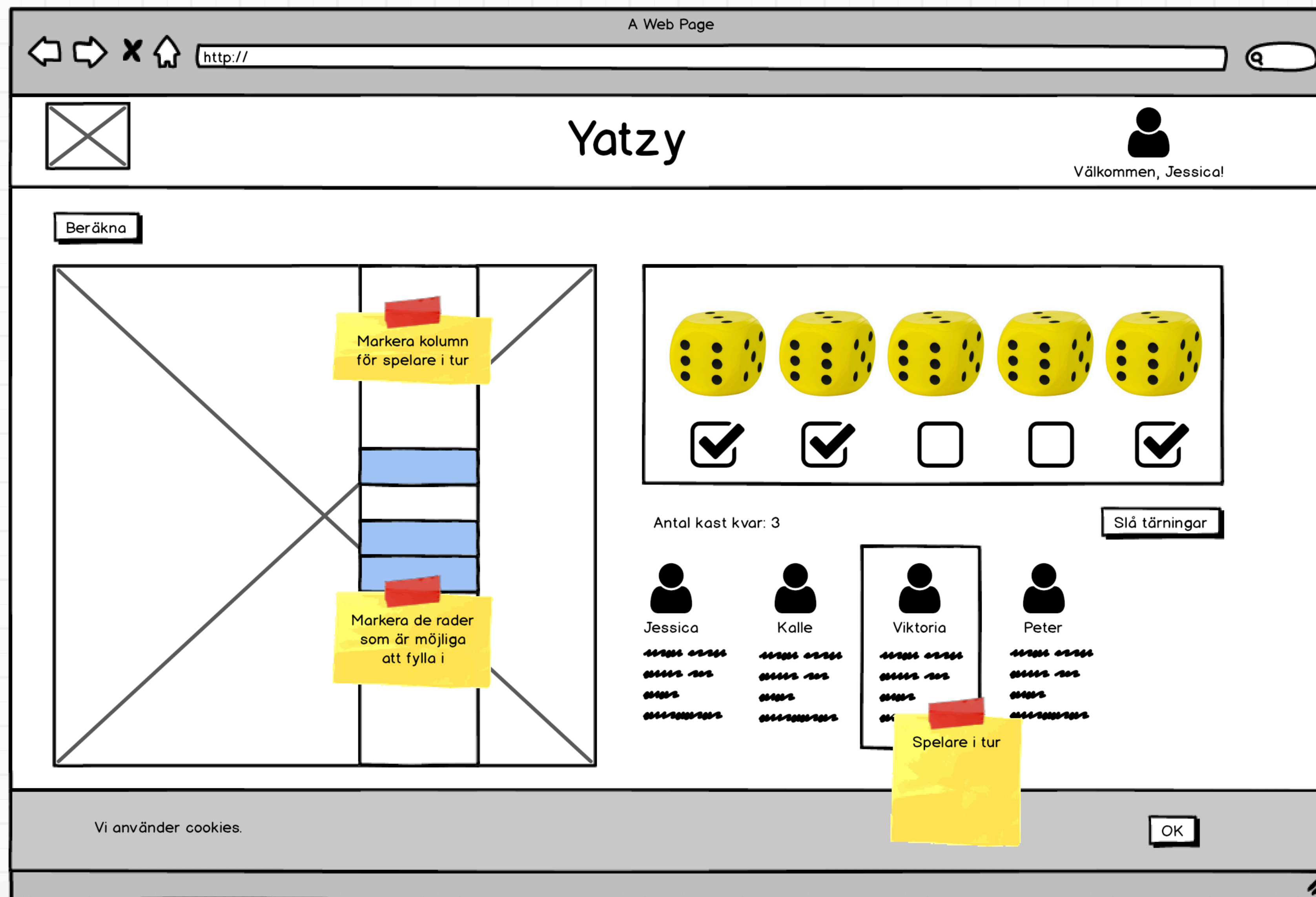
SCSS

- *Uppgift:* Installera stöd för SCSS i Visual Studio Code.
- https://code.visualstudio.com/docs/languages/css#_transpiling-sass-and-less-into-css
- Bygg en enkel sida och prova på några av funktionerna i SCSS, eller följ någon av dessa tutorials.
 - <https://1stwebdesigner.com/learn-sass-tutorial/>
 - <https://www.tutorialspoint.com/sass/>

SCSS

- Bygg en enkel sida och prova på några av funktionerna i SCSS, eller följ någon av dessa tutorials.
- <https://1stwebdesigner.com/learn-sass-tutorial/>
- <https://www.tutorialspoint.com/sass/>

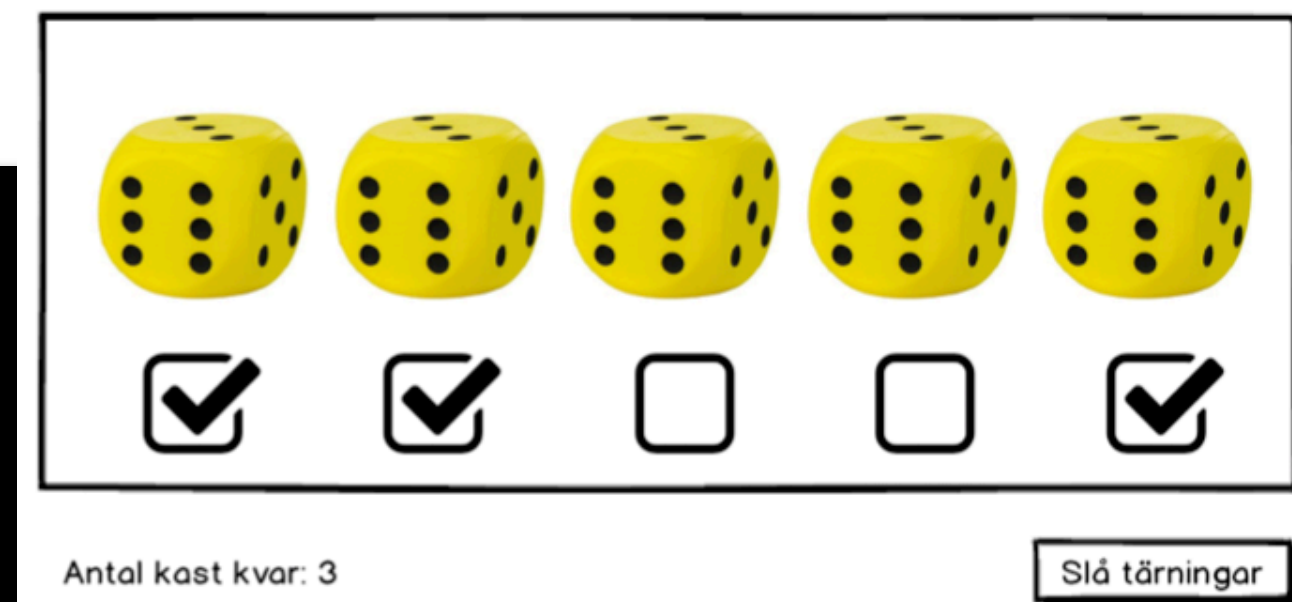
Yatzy - layout



Yatzy

- Gör ett tärningsformulär med fem text-rutor, fem kryssrutor och en knapp.
 - När man klickar på knappen ska alla rutorna få ett nytt tal (simulera tärningskast).
 - *Extra uppgift:* De tärningar som är "kryssade" ska inte få ett nytt tal.
 - *Extra uppgift:* Håll reda på hur många kast spelaren har kvar. (Tre från början.)
 - *Extra uppgift:* Fundera på hur man skulle kunna göra för att få till bilden nedan, att visa en tärning med rätt antal ögon istället för ett tal i en textruta.
 - *Extra uppgift:* Gör en funktion som tar en array med fem tal som parameter och returnerar sant om talen innehåller en kåk. (3 av samma + 2 av samma)

```
var slump = Math.random();  
console.log(slump);
```



Övningar

- Address Book
 - <https://yh.pingpong.se/courseId/11861/content.do?id=5221094>
- Todo List
 - <https://yh.pingpong.se/courseId/11861/content.do?id=5209549>