

INTRODUKTION TIL JAVASCRIPT

SDBG Forår 2020

AGENDA

- Hvad er Javascript?
- Hvordan er det anderledes end andre programmeringssprog.
- Hvordan Javascript afvikles.
- Javascript sproget.
- Document object model (DOM)
- Events
- Debugging
- Øvelser

INTENDED LEARNING OUTCOMES

Efter kurset kan du:

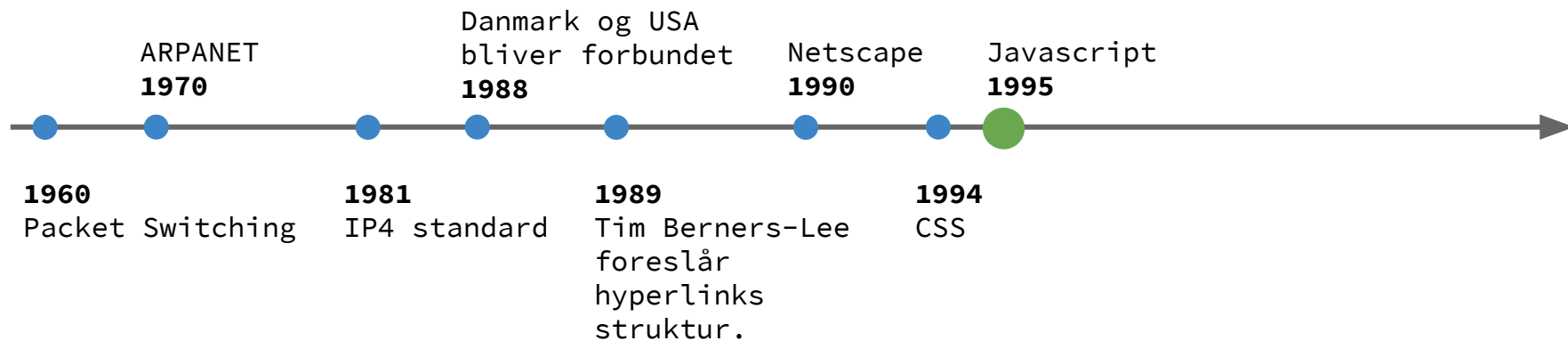
- levere et overblik over, hvad user experience er, og hvordan det kan måles
- redegøre for designmetoder til udarbejdelse af brugergrænseflader
- designe en grænseflade systematisk ud fra en designmetode
- udarbejde prototyper af brugergrænsefladen og vurdere hensigtsmæssigheden af forskellige slags prototyper i givne situationer
- teste og analysere user experience med en prototype
- forklare hvordan en webbrowser er opbygget og fungerer
- **udarbejde en semantisk korrekt skalerbar brugergrænseflade ved brug af HTML og CSS**
- implementere logik der muliggør brugerrejsen gennem de forskellige dele af løsningen ved brug af JavaScript og API-kald

INTENDED LEARNING OUTCOMES

Efter kurset kan du:

- levere et overblik over, hvad user experience er, og hvordan det kan måles
- redegøre for designmetoder til udarbejdelse af brugergrænseflader
- designe en grænseflade systematisk ud fra en designmetode
- udarbejde prototyper af brugergrænsefladen og vurdere hensigtsmæssigheden af forskellige slags prototyper i givne situationer
- teste og analysere user experience med en prototype
- forklare hvordan en webbrowser er opbygget og fungerer
- **udarbejde en semantisk korrekt skalerbar brugergrænseflade ved brug af HTML og CSS**
- **implementere logik der muliggør brugerrejsen gennem de forskellige dele af løsningen ved brug af JavaScript** og API-kald

JAVASCRIPT



JAVASCRIPT

Javascript hed oprindeligt LiveScript, men allerede 3 måneder efter launch skiftede de navn.

Samme år (1995) kom Java til verdenen, så en del spekulation mener navnesammenfaldet kom fra marketingafdelingen.

I dag er Javascript som sprog blevet standardiseret under navnet ECMAScript.

JAVA OG C#

For at sætte Javascript i perspektiv vil vi sammenligne med Java sproget da de 2 sprog udkom på samme tid og har navnesammenfald.

Der er store ligheder mellem Java og C# så de fleste sammenligninger vil kunne gøres mellem C# og Javascript.

JAVASCRIPT

Javascript er et script sprog.

```
console.log('Hello World');
```

Der er ikke behov for noget entrypoint, da det blot er den første linje der kan eksekveres som bliver det.

Java og C# har brug for et standardiseret entrypoint af hensyn til deres respektive VMs.

```
class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```


JAVASCRIPT VS JAVA

Java	Javascript
Statiske typer	Dynamiske typer
Klassebaseret	Prototypebaseret
Kun metoder (alle funktioner tilhører klasse/objekt)	Funktioner og metoder
Kompileres til bytecode i særskilt trin	Fortolkes og afvikles direkte i browser
Udviklet af Sun	Udviklet af Netscape

STATISKE VS DYNAMISKE TYPER

I Java og C# ved man altid hvilken type en variabel er:

```
String navn = "Villads";  
Car honda = new Car();  
int total = 453;
```

I javascript er der ingen typer på variabler:

```
var navn = "Villads";  
var honda = new Car();  
var total = 453;
```

KLASSER VS PROTOTYPER

Alt i Java er baseret på klasser:

```
class MyClass {  
    // Variabler og metoder ...  
}
```

Javascript er baseret på prototyper også selvom class keyword bruges.

```
const MyObject = {  
    metode1: (x, y) => { return x + y; }  
}
```

```
const myInstance = new MyObject();
```

```
MyObject.prototype.metode1 = (x, y) => {  
    return x * y;  
};
```

```
myInstance.metode1(5, 5); // Giver 25
```

METODER VS FUNKTIONER

Java tillader kun metoder i klasser.

```
class MyClass {  
    public int method1() {  
        return 1;  
    }  
}
```

Javascript tillader både funktioner og metoder.

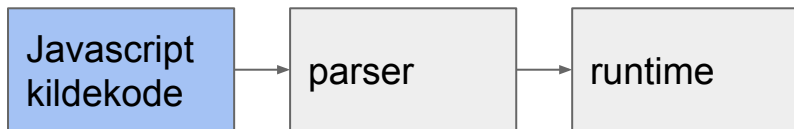
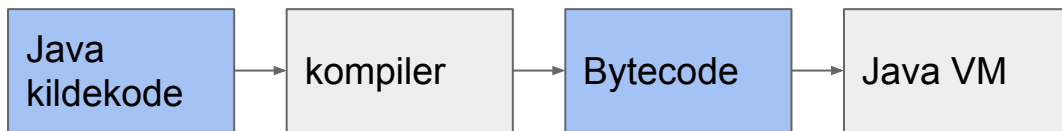
```
function func1() {  
    return 1;  
}
```

eller

```
class MyClass {  
    method1() {  
        return 1;  
    }  
}
```

Java kompileres til bytecode der efterfølgende eksekveres i en VM

Javascript fortolkes af browseren der så eksekverer kode baseret på de instruktioner i en "mini VM".



* **Vanilla JavaScript** er når der ikke bygges videre på frameworks som Angular, Vue, React eller brugere kompilere som Babel eller TypeScript.

JAVASCRIPT RUNTIMES

JavaScript kører i en engine/VM. Alle browsere har én.

Eksempler:

- **SpiderMonkey** (Firefox)
- **V8** (Chromium – Chrome, Brave)
- **JavaScriptCore** (Safari)
- **Chakra** (IE/Edge) – Edge er senere blevet skrevet om til at bruge Chromium.

AFVIKLING AF JAVASCRIPT

Via browserens udviklingsværktøjer:



```
>> console.log('Hello world!');  
← undefined  
Hello world!
```

Via Node:



EDITORS

Det er en fordel med en god editor der kan hjælpe en med udviklingen



Visual Studio Code



Sublime



Atom



WebStorm

DATA TYPER

Følgende datatyper eksisterer i JavaScript:

- Boolean
- Number
- String
- Object
- Null
- Undefined
- Symbol
- Function
- Regex
- Date
- Array

TRUTHY OG FALSY

Når der laves sammenligninger i JS, vil nogle typer bliver konverteret automatisk for at lave sammenligningen:

`'' == 0` og `'' !== 0` er begge true.

`1 == true` og `1 !== true` er begge true.

`null == undefined` og `null !== undefined` er begge true.

En værdi er truthy medmindre den kan konverteres til boolean false. Eksempler på falsy værdier:

`false`, `0`, `null`, `undefined`, `''`, `""`, `NaN`

VÆRDIER

Javascript har ingen types som i C#. Variabler erklæres derfor ikke med en type, men med en egenskab.

```
const UNCHANGABLE_VALUE = 1234;  
let changable_local_value = 'locals';  
var function_scoped_value = false;
```

Brug *let* og *const*.

Brug helst kun *var* til globaler.

*** Note omkring const og objekter.** Et objekt i en const variabel kan godt ændres/mutes men et variabelen kan ikke ændres til at pege på et andet objekt.

NAVNGIVNING

Best practise:

1) Hold jer til engelske navngivning. Variabler kan ellers godt bestå af alle unicode tegn, inkl æøå.

2) Variabler kan ikke have mellemrum, så vælg en konsistent måde at navngive på:

```
var camelCase = '';
var snake_case = '';
var PascalCase = '';
const SNAKE_CASE_CAPS = '';
```

FUNKTIONER

Funktioner kan erklæres på 3 måder:

```
function funcName() {}
```

```
var|const|let funcName = function() {}
```

Nyeste måde (ES6), såkaldte arrow functions:

```
var|const|let funcName = () => {}
```

OBJEKTER

I javascript kan objekter defineres når de erklæres:

```
const person = {  
  name: 'John',  
  age: 30,  
  'non-smoker': true  
};
```

Det kendes også som **J**ava**S**cript **O**bject **N**otation: JSON. Et meget brugt format i til at overfører data i API kald.

OBJEKTER

Objekt variabler kan tilgås med dot-notation eller bracket-notation:

`person.name`

`person['name']`

`person.non-smoker` **FEJL!**

`person['non-smoker']`

ARRAY

Array kan erklæres som literals:

```
const arrayWithElements = ['strings', 1234, {name:'John'}];
```

Brug push for at tilføje entries:

```
arrayWithElements.push('element value');
```

Brug brackets for at tilgå entries:

```
let secondElement = arrayWithElements[1];
```


ARRAY

Arrays kan manipuleres med splice-metoden:

Fjerne elementer:

```
const removed = arrayWithElements.splice(4, 1);
```

Indsæt:

```
arrayWithElements.splice(2, 0, ['new 1', 'new 2']);
```

CONTROL FLOWS

Javascript har mange af de samme control flows som andre sprog har:

- if-else
- switch
- for
- while

CONDITIONS

if-else konstruktion:

```
if (condition) {  
  
} else if (condition) {  
  
} else {  
  
}
```

Switch konstruktion:

```
switch (expression) {  
  case 'value':  
    break;  
  case 'value1':  
  case 'value2':  
    break;  
  default:  
    break;  
}
```

LOOPS

For loop:

```
for (let i = 0; i < arr.length; i++) {  
  
}
```

for-of:

```
for (const element of array1) {  
  
}
```

While loop:

```
while (condition) {  
  
}
```

Do-while:

```
do {  
  
} while (condition);
```

SCOPES

Scopes findes også i Javascript og har indflydelse på hvor en variabel kan tilgås fra.

Et scope er defineret ved brugen af tuborg-klammer:

```
{  
  // scoped  
}
```

SCOPES

Eksempel på et scope:

```
let val = 'string';  
  
{  
  let otherVal = 'string';  
  console.log(val); // 'string'  
}  
  
console.log(otherVal); // undefined
```

SCOPES

var trigger *variable hoisting*.

```
showState();

function showState() {
  alert("Function declaration");
}

showState();

var showState = function() {
  alert("Function expression");
};

showState();
```

"Sorteret" kode efter parsing. Funktioner og Variabler bliver "løftet" til toppen.

```
function showState() { // moved to the top (function declaration)
  alert("Function declaration");
}

var showState; // moved to the top (variable declaration)

showState();

showState();

showState = function() { // left in place (variable assignment)
  alert("Function expression");
};

showState();
```

JAVASCRIPT I BROWSERE

De seneste år er browserne blevet mere og mere enige om hvordan Javascript skrives.

Men det er altid godt at bruge caniuse.com til at tjekke om en indbygget funktion eller syntax er understøttet i alle browsere. Det er typisk Internet Explorer 11 der ikke er med.

Canibase kan også bruges til at tjekke HTML og CSS syntax support.

ES6 classes are syntactical sugar to provide a much simpler and clearer syntax to create objects and deal with inheritance.

Usage

% of all users

Global

$$94.63\% + 0.13\% = 94.76\%$$
[illegible]

INDKLUDER JAVASCRIPT I HTML

For at Javascript kan blive afviklet, skal det naturligvis indkluderes på jeres HTML side.

1) inline `<script>` tag

```
<script> ... </script>
```

2) attributter på elementer

```
<a href="javascript:alert('Hello world');">Link tekst</a>
```

3) ekstern js-fil (mest praktisk og anbefalet)

```
<script src="source-code.js"></script>
```

DOM TREE

Document Object Model eller DOM er det interface Javascript kan bruge til at tilgå vores HTML side programmatisk.

Som vi så i Intro til HTML, er HTML meget lig XML og via elementer og deres underelementer opbygges et hirarkisk træ.

DOM TREE

```
<!DOCTYPE html>
<html lang="en"> [event] [scroll]
  <head> [...] </head>
  <body class="logged-in env-production min-width-lg intent-mouse">
    <div class="position-relative js-header-wrapper ">
      <a class="p-3 bg-blue text-white show-on-focus js-skip-to-content" href="#start-of-content">Skip to content</a>
      <span class="Progress progress-pjax-loader position-fixed width-full js-pjax-loader-bar"> [event] </span> [flex]
      <header class="Header" role="banner"> [flex]
        <div class="Header-item"> [flex]
          <a class="Header-link" href="https://github.com/" data-hotkey="g d" aria-label="Homepage " data-ga-click="Header, go to dashboard, icon:logo">
            <svg class="octicon octicon-mark-github v-align-middle" height="32" viewBox="0 0 16 16" version="1.1" width="32" aria-hidden="true">
              <path fill-rule="evenodd" d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59 4.07 5.58 10.85 3.5 13.58 0 4.22-3.58 8-8 8z"></path>
            </svg>
          </a>
        </div>
        <div class="Header-item Header-item--full"> [event] </div> [flex]
        <div class="Header-item"> [event] </div> [flex]
        <div class="Header-item position-relative"> [event] </div> [flex]
        <div class="Header-item position-relative mr-0"> [event] </div> [flex]
      </header>
    </div>
```

Hver node (element) har 0, 1 eller flere.childNodes.

DOM API

Udover vores DOM tree som give mulighed for at få fat i elementer, har vi en række metoder i DOM API'en til at manipulere DOM træet.

Find element med id-attribute med værdien "myID"

```
document.getElementById('myID')
```

Find elementer ud fra en CSS-selector.

```
document.querySelector('div.header > span')
```

DOM API

Opret element i hukommelsen:

```
document.createElement(<tag-navn>)
```

Feks:

```
const ulElement = document.createElement('ul');
```

DOM API

Opret et nyt stykke tekst i hukommelsen. Tekst har intet tag, men er stadig et element i DOM'en.

```
document.createTextNode(<tekst>)
```

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      </script>
  </head>
  <body>
    <header>
      <h1>This is a motherfucking website.</h1>
      <aside>And it's fucking perfect.</aside>
    </header>
    <h2>Seriously, what the fuck else do you want?</h2>
    <p>
      You probably build websites and think your shit is special. You think
      your 13 megabyte parallax-ative home page is going to get you some
      fucking Awwward banner you can glue to the top corner of your site. You
      think your 40-pound jQuery file and 83 polyfills give IE7 a boner
      because it finally has box-shadow. Wrong, motherfucker. Let me describe
      your perfect-ass website:
    </p>
    <ul>
      <li>Well guess what, motherfucker:</li>
    </ul>
    <p>It's fucking lightweight</p>
    <p>It's fucking lightweight</p>
    <p>It's fucking lightweight</p>
```

DOM API

Lægger et element ind under et andet:

```
<element>.appendChild(<element>)
```

Feks:

```
ulElement.appendChild(liElement);
```


DOM API

Opret/ændre attribut på element

```
<element>.setAttribute(<attribut-navn>, <værdi>)
```

Feks:

```
element.setAttribute('class', 'header');
```

FORM DATA

Form elementer er komplekse elementer med ekstra felter.

```
<input type="text" value="Tekstfelt værdi" />
```

Værdien kan tilgås via:

```
const textFieldValue = inputTextElement.value
```

Og ændres via samme felt:

```
inputTextElement.value = 'Ny værdi'
```

FORM DATA

Select elementer:

```
<select>
  <option>Valg 1</option>
  <option>Valg 2</option>
  <option selected>Valg 3</option>
</select>
```

Værdien kan tilgås via:

```
const selectIndex = selectElement.selectedIndex
const selectValue =
  selectElement.options[selectIndex].value
```

Og ændres via samme felt:

```
selectElement.selectedIndex = -1
```

TIMERS

Javascript har to indbyggede metoder til at planlægge kørsel af kode i fremtiden.

- `setTimeout()`
- `setInterval()`

TIMERS

SetTimeout kører den angivet kode efter antal ms.

```
setTimeout(<function>, <ms>)
```

Feks:

```
const timerId = setTimeout(refreshData, 2500);
```

Fjern timer vha timerId og clearTimeout():

```
clearTimeout(timerId);
```

TIMERS

`setInterval` kan bruges til at køre kode igen og igen med et fastlagt tid imellem hver kørsel startes.

```
setInterval(<function>, <ms>)
```

Feks:

```
const intervalId = setInterval(oneSecondPassed, 1000);
```

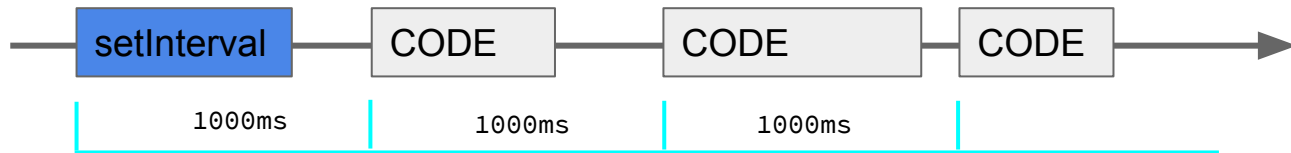
Fjern interval vha `intervalId` og `clearInterval()`:

```
clearInterval(intervalId);
```

TIMER TIMING

Tid fra en timeout startes til den køre måles fra `setTimeout` returnere og til den refererede funktion bliver kaldt.

Tid i mellem intervaller måles fra funktionen kaldes til den kaldes næste gang, lige meget hvor lang tid funktionen tager at afvikle.



EVENT LISTENERS

Javascript køres så snart det læses af browseren.

Det er ikke altid det man vil, ofte vil man gerne gøre noget når personer interagerer med websitet.

Det er her Event Listeners kommer ind i billedet.

EVENT LISTENERS

Event listenere sættes typisk på a, button og form-tags, men kan sættes på næsten alle typer af elementer.

```
const myButton = document.getElementById('myButton');  
myButton.addEventListener('click', <function>);
```

Eller

```
<button onclick="<function>">Min knap</button>
```

FORM EVENTS

form-elementer har flere events end almindelige elementer.

Kør kode når personen forsøger at sende formen (submit knap):

```
formElement.addEventListener('submit', <function>);
```

DOM READY

En vigtig note:

Browseren opbygger DOM'en for os asynkront og det er derfor ikke tilgængeligt i sin helhed før den sidste linje html er indlæst.

Derfor er der også et event for hvornår DOM er fuldt klar til brug.

DOM READY

En vigtig note:

Browseren opbygger DOM'en for os asynkront og det er derfor ikke tilgængeligt i sin helhed før den sidste linje html er indlæst.

Derfor er der også et event for hvornår DOM er fuldt klar til brug.

Nemlig 'domready' på Window objektet.

DOM READY

Kode der kaldes når HTML siden er parsed færdig af browseren og alle elementer er tilgængelig i DOM'en.

```
window.addEventListener('domready', () => {});
```

Med dette event kan vores kode nemt ligges i separate filer.

DEBUGGING

Vi skriver jo alle perfekt kode ... meeeenn...

Måde der kan debugges i Javascript:

- `console.log()`
- `alert()`
- `debugger;`
- break og log points.

DEBUGGING

`Console.log` skriver til browserens log i Devtools.

Den er god at bruge til at skrive forskellige beskeder ud og den blokerer ikke browseren, som f.eks. `alert()` metoden gør.

```
console.log('Besked der skal vises når linjen rammes.');
```

DEBUGGING

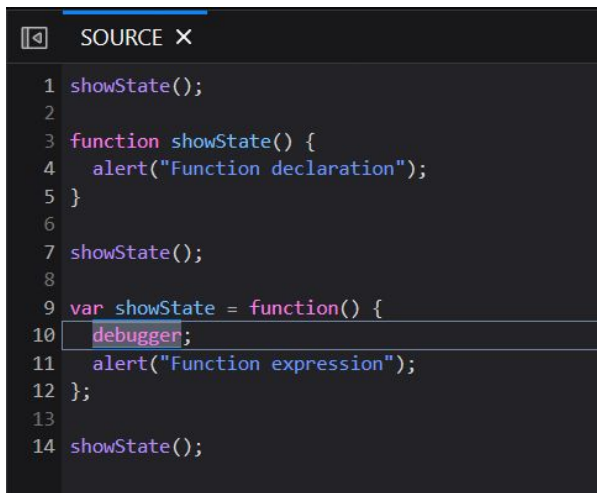
Old school

Brug af alert-funktionen til debugging er en meget afbrydende teknik, man slipper dog for at åbne devtools.

```
alert('Besked der skal vises i alert-boksen');
```


DEBUGGING

Brug *debugger* til at stoppe browseren og gå i debug mode.



A screenshot of a code editor window titled "SOURCE X". The editor displays JavaScript code with line numbers 1 through 14. The code defines a function `showState()` and then calls it. A `debugger;` statement is placed on line 10, which is highlighted with a light blue background. The code is as follows:

```
1 showState();
2
3 function showState() {
4   alert("Function declaration");
5 }
6
7 showState();
8
9 var showState = function() {
10  debugger;
11  alert("Function expression");
12 };
13
14 showState();
```

DEBUGGING

Brug devtools interaktivt til at sætte breakpoints.



```
1 showState();
2
3 function showState() {
4   console.log("declaration");
5 }
6
7 showState();
8
9 var showState = function() {
10  console.log("Function expression");
11  debugger;
12 };
13
14 showState();
```

ØVELSER

Cykelstur!

I stedet for fysiske øvelseslokaler, forsøger vi os på Discord. Erfaringer fra andre hold er gode.

Link: <https://discord.gg/XDyRx4F>

Tak til TAs for at være proaktive 🙌