Web programming

Variables Conditions and Functions

Events

Programmering i Javascript

“When learning a new programming language, it’s important to try ... examples ..., then modify them and try them again to test your understanding of the language. ”

David Flanagan, JavaScript: The Definitive Guide, 7th Edition, ch1.1

Arrangement

Basic functions of Javascript

- Data Types (ch3, JavaScript: The Definitive Guide, 7th Edition)
- Variables (ch3.10, JavaScript: The Definitive Guide, 7th Edition)
- Terms (ch5.3, JavaScript: The Definitive Guide, 7th Edition)
- Iteration (ch5.4, JavaScript: The Definitive Guide, 7th Edition)
- Functions (ch8, JavaScript: The Definitive Guide, 7th Edition)
- Features specific to HTML web pages
- Declaration of Javascript code
- Events (ch13, JavaScript: The Definitive Guide, 7th Edition)

Invocation of Javascript

Javascript can be used as a server-side language, but in this course it is client-side javascript as part of a web page we will use (Advanced Web Programming, a continuation course that is currently not given)

We can call javascript code in the browser in three different ways (ch15.1.1, JavaScript: The Definitive Guide, 7th Edition)

- In **<script>** in **<head>** at the top of the document
  - We can see the code at the same time as we see our document
  - Suitable for small pieces of code that we want to execute before the document is loaded
- Linked to external file in header
  - The program code is in an external .js file that we link with a **<script src='fil.js'>** tag
  - Suitable when the program code becomes larger, but we lose the overview we get when we can see both code and html at the same time
- **<script>** tags included in the body document
  - The program code is in the middle of the html code
  - Suitable for small snippets that we want to be executed immediately when the element is loaded
  - In our code examples we use the first variant in the task, you can use the variant you prefer

Code Example **Onload Javascript**

```
<!DOCTYPE html>
<html>
<head>
   <title>HTML Example Call Javascript</title>

   <script type="text/javascript" >
       function loaded()
       {
           alert("Hello Head");
       }
   </script>

   <script src="extern.js" > </script>
</head>
<body onload="loaded()">
   <script>alert('Hello Scripttag')</script>
</body>
</html>
```

Flow Control

Conditional statements ... are the decision points of your code, and they are also sometimes known as “branches.”

— David Flanagan, JavaScript: The Definitive Guide, 7th Edition, ch5.3

The looping statements are those [conditional statements] that bend that [execution] path back upon itself to repeat portions of your code.

— David Flanagan, JavaScript: The Definitive Guide, 7th Edition, ch5.4

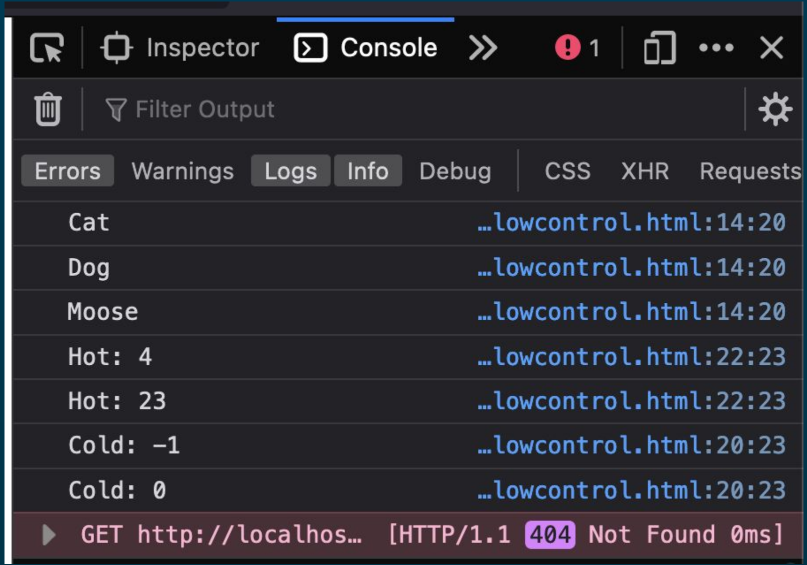
Iteration with For

Using an iteration, we want to make it clear which element we are working with. For example, below, we want to iterate over a list of animals. In each round of the iteration will deal with an animal, an animal (singular). Both variants do exactly the same thing.

```
var animals = [“cat”,”mouse”,”dog”];

for (var i=0;i<animals.length;i++) {
   var animal = animals[i];
   console.log(animal);
}

var animals = [“cat”,”mouse”,”dog”];
for (animal of animals) {
   console.log(animal);
}
```

Code Example **Flow control**

```
function loaded()
{
    var animals = ["Cat","Moose","Dog"];

    for (var i=0;i<animals.length;i++) {
       var animal = animals[i];
       console.log(animal);
    }

    var temperatures=[4,23,-1,0];
    for (temperature of temperatures) {
       if(temperature<=0){
          console.log("Cold: "+temperature);
       }else{
          console.log("Hot: "+temperature);
       }
    }
}
```



```
Cat                                    …lowcontrol.html:14:20
Dog                                    …lowcontrol.html:14:20
Moose                                  …lowcontrol.html:14:20
Hot: 4                                 …lowcontrol.html:22:23
Hot: 23                                …lowcontrol.html:22:23
Cold: -1                               …lowcontrol.html:20:23
Cold: 0                                …lowcontrol.html:20:23
▶ GET http://localhos…  [HTTP/1.1 404 Not Found 0ms]
```

Conditions with **if / else if / else**

We can let the execution take different paths depending on conditions.

```
if(temperatur < 0){
       console.log("Låg");
   } else if (temperature == 0) {
       console.log("Noll");
   } else{
       console.log("Varm");
   }
```

Data types

- Numeric Types (ch 3.2.1 Integer Literals, ch 3.2.2 Floating-Point Literals)
  - 1, 2, 3, 4 1.0 3.345
- Strings (ch 3.3.1 String Literals)
  - Enclosed with quotes " or ' in newer javascript you can use ` to make template strings
- Booleans / Undefined values (ch 3.4 and ch 3.5)
  - true, false
  - null, undefined

Deklaration av Variabler

Tre olika nyckelord var, const, let. I kursen så behöver man inte använda annan deklaration än med var.

I Javascript i likhet med php behöver man **inte deklarera** variabler i förväg men det resulterar ofta i fel

```
      stad="Grums";
```

Var - In older versions of Javascript var is the only way to declare variables - declared variables are reachable within the **same function**

```
      var height;
```

Let - More restrictive declaration, variable is only reachable with the **same block**

```
      let height;
```

Const - Is a variable that **can not be updated**

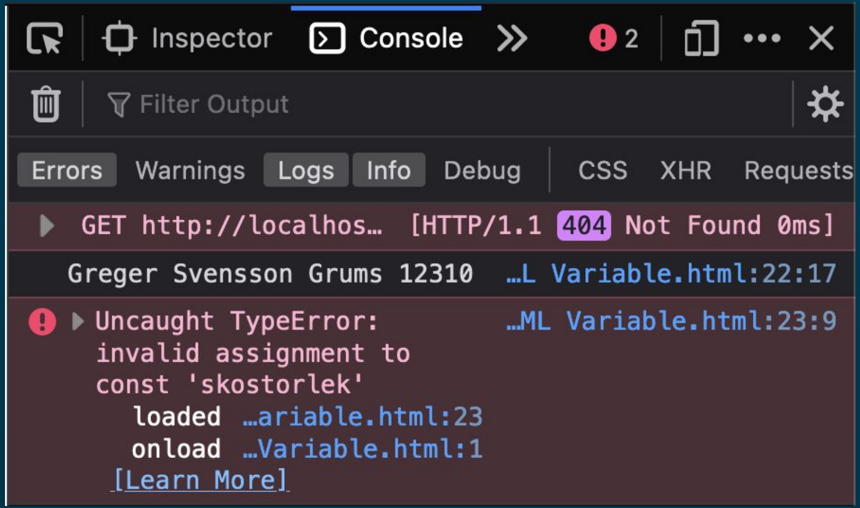
```
      const height=200;
```

We assign values to variables using =

```
      let lastName="Olsson";
```

Kodexempel **Variabler**

```
const skostorlek="123";

function loaded()
{
    var firstName="Greger";
    var lastName="Svensson";
    stad="Grums";

    if(firstName=="Greger"){
        let lastName="Olsson";
    }
    alert(firstName+" "+lastName+" "+stad);

    console.log(firstName,lastName,stad,skostorlek+10);
    skostorlek=skostorlek+5;
}
```



```
GET http://localhos… [HTTP/1.1 404 Not Found 0ms]
Greger Svensson Grums 12310                    …L Variable.html:22:17
Uncaught TypeError:                            …ML Variable.html:23:9
invalid assignment to
const 'skostorlek'
    loaded …ariable.html:23
    onload …Variable.html:1
  [Learn More]
```

Functions

A function is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times.

— David Flanagan, JavaScript: The Definitive Guide, 7th Edition, ch8

- In older versions of javascript there was a way to declare a function with the function keyword
- In new versions, => can be used to shorten the declaration.
- There are cases when arrow notation (=>) is not allowed to be used and there is a risk that we introduce more errors into the program code as it is faster to write and for some easier to read but because it is possible to write errors in more ways than with function, it can result in more errors in the code.

Declaration of Function

```
function loaded(a)
{
   alert(addhundred(200));
}

function addhundred(a)
{
   return a+100;
}

var addhundred = (function (a) {
 return a+100;
});

addhundred = (a) => {
 return a+100;
};

addhundred = (a) => a + 100;
```

Asynchronous Programmering

Most real-world computer programs, however, are significantly asynchronous. This means that they often have to stop computing while waiting for data to arrive or for some event to occur.

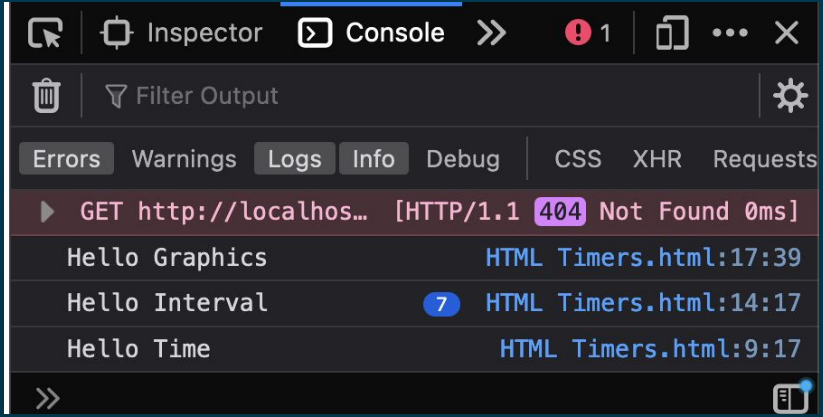
— David Flanagan, JavaScript: The Definitive Guide, 7th Edition, ch13

Timers

- A timer is a piece of code (a function) that executes after a time interval or with a certain interval between executions. (ch 13.1.1 Timers)
- A timeout is a timer that executes after a selectable time interval
- An interval timer is a timer that executes code with a certain time interval between executions.
- Request animationframe is a timer that is connected to drawing the graphics. If you use animationframe, animations get smooth movements.

Code Example for **Timers**

```
function timeoutFunc(){
    console.log("Hello Time");
    clearInterval(interval);
};

function intervalFunc(){
    console.log("Hello Interval")
};

function animationFunc(){ console.log("Hello Graphics") };

var timeout = setTimeout(timeoutFunc, 4000);

var interval = setInterval(intervalFunc, 500);

window.requestAnimationFrame(animationFunc);
```



```
▶ GET http://localhos… [HTTP/1.1 404 Not Found 0ms]
  Hello Graphics                  HTML Timers.html:17:39
  Hello Interval             7    HTML Timers.html:14:17
  Hello Time                       HTML Timers.html:9:17
```

Events

When an event occurs, we can ensure that code is executed. The term "event-driven" is sometimes used to describe applications that have a program flow where the occurrence of various events controls the order of program code execution.

An "event handler" is a javascript function that we defined that executes as soon as an event occurs on the document or element that the function is connected to.

Example of HTML events

- When a user clicks on an HTML element
- When a web page has finished loading
- Once an image has been loaded
- When the mouse cursor is over an HTML element
- When an input field has been changed
- Once an HTML form has been submitted
- When a user presses a key
- …

Code Example Static **Replace Content**

When the onclick event is fired, we add the contents of a text box with the id **skriv** to the span element with the id **output1**

```
    <script type="text/javascript">
      function clicked()
      {
        var divobj = document.getElementById("output1" );
        var str="<ul><li>"+document.getElementById("skriv").value+"</li></ul>";
        divobj.innerHTML=str;
      }
    </script>
  </head>
  <body>
    <span id="output1">Output Element</span>
    <input type="text" id="skriv"><button onclick="clicked()">Click here</button>
  </body>
```

Code Example **Dynamic Replace Content**

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Javascript Form Replace Content</title>
<script>
 function clicked()
 {
       var divobj = document.getElementById("output1" );
       var str="<ul><li>"+document.getElementById("skriv").value+"</li></ul>";
       divobj.innerHTML=str;
 }

 function init()
 {
       var button1=document.getElementById("button1");
       button1.addEventListner("click",clicked);
 }
</script>
</head>
<body>
 <span id="output1">Output Element</span>
 <input type="text" id="skriv"><button id="button1">Click here</button>
</body>
```

Webbprogrammering

DOM Manipulation and calling API

DOM Manipulation - Find the Element

The DOM is a representation of a document in the form of a number of objects that represent the document's elements

We can manipulate this tree of objects in different ways eg by replacing the content of the element or eg by changing the element's css properties

In order to change an element, we have to **find the element**

- We can **navigate** to the element from other elements
- We can retrieve an element based on its **ID**
- We can retrieve elements based on its **class**
- We can retrieve elements based on their **css selector**

Which method is most appropriate to use

- Navigation is almost never the "right" way to implement DOM manipulation - navigation is relatively fast but often very complicated
- **ID is fast** and works best when we want to access **an element specifically**
- **Class is relatively fast**, but it's not sure the element we're after **uses classes**
- Selector gives us the element based on the **css selector**. If the document is complex, selectors **can be very slow**.
    - querySelector is good if we want a specific element but don't know the ID of the element
    - querySelectorAll is good if we want to access multiple elements but is **slower than querySelector**

Code Example **Find Element**

```
var tstelements = document.getElementsByClassName("tstclass");
for(let element of tstelements){
    element.style.border = "2px solid green";
}

document.getElementById("tstE").style.border="4px dotted blue";

var tstelements = document.querySelectorAll(".dialog span");
for(let element of tstelements){
    element.style.background = "#9df";
}
```

DOM Manipulering - Modifiera (ch 15.3.4 Element Content)

När vi hittat till ett element så finns det flera olika saker vi kan göra med elementet

- Change content using innerHTML `element.innerHTML="<h1>Test</h1>";`
  We **change the html** inside an element with content constructed with javascript. The drawback is that we can inadvertently **inject javascript** code
- We can change the text content but this means that we can only modify text and not introduce new html
  `element.textContent="Test";`
- We can modify the document tree directly but the code will get complicated very fast. This approach can be more secure than using innerHTML.

```
let header = document.createElement("header");
header.append("World");

element.prepend(header);
```

- We modify the css of the element. This is very good for moving elements around or other basic modifications

```
element.style.position = "absolute";
element.style.left = `${x}px`;
```

- We can add or remove classes using the classList attribute. This allows us to make complex css modifications using snippets of javascript. `element.classList.add("mystyle");`

Replacing Content (ch 3.3.4 Template Literals)

```
function clicked()
{
 var divobj = document.getElementById("output1" );
 var str="<ul><li>"+document.getElementById("skriv").value+"</li></ul>";
 divobj.innerHTML=str;
}
```

Same code with template literals

```
function clicked()
{
 var divobj = document.getElementById("output1" );
 var str=`<ul>
             <li>${document.getElementById("skriv").value}</li>
         </ul>`;
 divobj.innerHTML=str;
}
```

Form tags compared to working without form tags

- If we use form tags, we are limited in how we make our interface
- The form tag must cover entire elements
- If we want a button for each row in a table, the form-tag must be in each row.
- We cannot start a shape in one div and end it in another
- If we want to use the same input in two different forms, this input must be duplicated and hidden (eg login and create customer can use the same login name input)

Code Example **Form tag**

We need to add `event.preventDefault();` which stops the default action of the form to execute. If we do not do this working with form tags in single page applications does not work.

```
function submitForm(event)
{
   event.preventDefault(); // This prevents the traditional submitting of the form
   var fname=document.getElementById("fname").value;
   var lname=document.getElementById("lname").value;

   alert("You sent |"+fname+"| |"+lname+"|.");
   // Do your AJAX or Fetch calls here ...
}
```

What does code that handles forms look like?

Depending on the type of form we are talking about, there are several different things our program code can do

- **Validate** what the user writes in the different parts of the form
- **Store** information in local storage (remember e.g. previously used search terms)
- **History** - what happens when we press the back button
- Collect the entered data and **pass the data** to one or more web services

Kodexempel **Validation of Characters**

```
<input id="input1" type="text" value="Must contain at character" />
<button onclick="clicked()" >Click here</button>
```

```
function clicked()
{
 var textinput = document.getElementById("input1");
 if(textinput.value.indexOf("@")==-1){
   alert("Email must contain @");
  }
}
```

Code Example **RegExp** (ch 11.3 Pattern Matching) med Keypress

```
  <input id="input1" type="text" onchange="validate()" onkeyup="validate()" value="Write
Only Numbers" />
```

```
function validate()
{
 var textinput = document.getElementById("input1");
 var val=textinput.value;
  // Match is null if there is no match
 if(val.match(/^\d+$/)!=null){
   textinput.style.backgroundColor = "#cfa";
 }else{
   textinput.style.backgroundColor = "#f57";
 }
}
```

Validation

- We look to see if each individual input follows the pattern of our data
- We can see what data type the data has
- We can see how long text values are
- We can search for characters like . & @ or similar
- We can use regular expressions to make complex conditions such as three numbers followed by a minus sign, three more numbers, followed by a space and three more numbers
- In the case of passwords we can for example use complex code (not using regular expressions) to for example count types of characters for example number of upper case characters, numbers or special characters

Local Storage (15.12 Storage)

- **Cookies** used to be popular, you could store up to one kilobyte of data in the browser and retrieve this data at a later time.
- **Local storage** (and session storage) allows us to store **larger amounts of data** and more varied data in the browser's memory.
- Local storage acts as a **global variable** that persists after we close the browser (but not between two different browsers on the same machine).

- For example, we can remember search terms so that we can get to the same input if we come back to the same computer after closing the browser. (distinct from history)

- Local storage has two functions to **store a value** with a specific name and a function to **read the value** with a specific name

Code Example **Local Storage**

```
var username = localStorage.getItem("username");
 if (username!=null && username!=""){
     alert('Welcome again '+username+'!');
 }else{
     username=prompt('Please enter your name:',"");
     localStorage.setItem("username",username);
 }
```

History Management (ch 15.10.4 History Management)

- The history management allows us to save information so that the back buttons in the browser work again. Even though we have built a single page application.
- History management is suitable for use e.g. on menus and search boxes etc.

- We have a function that **stores a new step** in the history
- We have a function that reacts to when the **history is updated**

If we have a generic showpage function that is used for all page navigation we can connect this to the history management

Code Example **History Management**

```
function historyChange(event)
{
   alert('History updated!');
   document.getElementById("outputValue").value+=JSON.stringify(event.state);
}

function updateHistory(token)
{
 history.pushState(token, "Titel: "+token, "");
}

function setupHistory()
{
 document.getElementById("outputValue").value=history.state;
 window.onpopstate = function(event) { historyChange(event); };
}
```

Collate Data to send to Browser

```
var input={
     ID: document.getElementById("customerIDC").value,
     firstname: document.getElementById("customerlnameC").value,
     lastname: document.getElementById("customeremailC").value,
     email: document.getElementById("customeremailC").value,
     address: document.getElementById("customeraddressC").value,
     auxdata: document.getElementById("customerauxdataC").value
}
console.log(input);
```

Promises and Fetch (ch 13.2.1 Using Promises)

- 2003 An employee at google discovered that there were features in Internet Explorer that could retrieve data **without reloading the web page**.
- Almost immediately, google started using this to provide search answers while the user typed in search boxes. (this was called **AJAX,** asynchronous Javascript and XML)
- The disadvantage of AJAX was, however, that the **calls are not placed in a queue**, which makes it complex to build applications if you have several ajax requests at the same time.
- Since 2016, browsers have had better and **better support for fetch**, which is a more capable implementation of AJAX that can handle more types of requests and more types of data
- Fetch uses **promises** .then which executes the code in the function when the data has arrived. The retrieval of the information is divided into several phases with their own promises
- Fetch has extensive support for **error handling**

Store Customer

We collate the form data and send it to the server using fetch.

```
fetch('../booking/makecustomer_XML.php', {
 method: 'POST', // or 'PUT'
 headers: {'Content-Type': 'application/json'},
 body: JSON.stringify(input)
})
.then(function(response) {                        // first then()
   if(response.ok) return(response.text());
   throw new Error(response.statusText);
}).then(function(text){
   ResultBookingCustomer(new window.DOMParser().parseFromString(text, "text/xml"));
}).catch(function(error) {                          // catch
   alert('Request failed\n'+error);
});
```

Fetch with Await

```
async function makecustomer() {
   try {
        var result = await fetch('../booking/makecustomer_XML.php', {
         method: 'POST', // or 'PUT'
         headers: {'Content-Type': 'application/json'},
         body: JSON.stringify(input)
        });
        const json = await ResultBookingCustomer(new window.DOMParser().parseFromString(response.text(),
"text/xml"));;
   } catch (error) {
       alert('Request failed\n'+error);
   } finally {
       alert('Request failed\n');
   }
}
```