

# Basic JavaScript instructions

IIKG1002/IDG1011 – Front-end web development

Johanna Johansen  
johanna.johansen@ntnu.no

# Previously on scrips

- A script is a series of instructions that the computer can follow in order to achieve a goal
- Only one subset of all instructions might be used depending on the user interaction
- Computers solve the task **programmatically**; the instructions are followed one after another
- Before writing the script
  - define a goal
  - break down the goal into a series of tasks
  - a flowchart can help with planing and designing the script

# Previously on scrips

- To write instructions in a language that the web browser understands, i.e., JavaScript
  - learn the **vocabulary** – the words
  - and rules for how the words can be put together – the **grammar** and **syntax** of the language
- Start with key building blocks of JavaScript
- How these can be used to write some very basic scripts

# JavaScript – background

- Also known as **ECMAScript**
  - is a JavaScript **standard** → meant to ensure that web pages are handled the same across different web browsers
  - specifies language syntax and semantics
  - *Ecma International* is in charge of standardizing ECMAScript:  
<https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
- developed by Brendan Eich of Netscape; co-founded the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation

# JavaScript – background

- First version was released in 1997
- Commonly used for client-side scripting on the World Wide Web
  - language embedded in web browsers
- Increasingly being used for writing server applications and services using Node.js.

# JavaScript – background

- A **scripting language** or script language (e.g., python)
- is a programming language for a runtime system
- scripting languages are usually **interpreted at runtime** rather than **compiled**
  - **interpreted**
    - the code is run from top to bottom and the result of running the code is immediately returned
    - we don't have to transform the code into a different form before the browser runs it
    - the code is received in its programmer-friendly text form and processed directly from that
  - **Remember:** the browser uses an **interpreter** for the JavaScript files to translate them into instructions the browser can follow

# JavaScript – background

- **runtime** or execution time
  - is the final phase of a computer program's life cycle
  - the code is being executed on the computer's central processing unit (CPU) as machine code
  - "runtime" is the running phase of a program
- **compiling**
  - compiled languages are transformed (compiled) into another form before they are run by the computer
  - translated from a high-level programming language to a lower level language (e.g. assembly language, object code, or machine code) to create an executable program

# Statements

- **Remember:** a script is a series of instructions that a computer can follow one-by-one
- Each individual instruction or step is known as a **statement**
- **Good practice:** start each statement on a new line and end it with a semicolon
  - makes your code easier to read and follow
- Statements can be grouped into **code blocks**, by surrounding them with curly braces

```
const today = new Date();
const hourNow = today.getHours();
let greeting;

if (hourNow > 18) {
    greeting = 'Good evening!';
} else if (hourNow > 12) {
    greeting = 'Good afternoon!';
} else if (hourNow > 0) {
    greeting = 'Good morning!';
} else {
    greeting = 'Welcome!';
}

document.write('<h3>' + greeting +
'</h3>');
```



# Code writing style guide

- Good practice and code formatting guides

[https://developer.mozilla.org/en-US/docs/MDN/Writing\\_guidelines/Writing\\_style\\_guide/Code\\_style\\_guide](https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Writing_style_guide/Code_style_guide)

# Comments

- We write comments to explain what our code does
  - make our code easier to read and understand
  - this can help others who read/work with our code
  - help ourselves to understand our code when coming back to it later after several months
- Comments are not processed by the JavaScript interpreter
- Two types of comments, depending on how long the comment is or the specificity
  - multi-line comments
  - single-line comments
- Comments guidelines – a combination between these two guides:  
[https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code\\_guidelines/JavaScript#javascript\\_comments](https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript#javascript_comments)  
<https://github.com/airbnb/javascript#comments>

```
/** This script displays a greeting to the user based upon the current time.
 * This is an example from your syllabus book by Jon Duckett.
 */

var today = new Date();           // Create a new date object
var hourNow = today.getHours();   // Find the current hour
var greeting;

// Display the appropriate greeting based on the current time
if (hourNow > 18) {
    greeting = 'Good evening!';
} else if (hourNow > 12) {
    greeting = 'Good afternoon!';
} else if (hourNow > 0) {
    ...
}
```

# Variables

- Data/information that the script needs in order to do its job is stored in **variables**
- E.g., calculating the area of a rectangle
  - in math:  $width \times height = area$
  - to do this in our script we have first to save the value of the width and height in variables

# Variables

- This is a simple, quick to do operation for us humans
- For the computer we have to give the computer detailed instructions with each step it needs to do
  - 1) Remember the value for *width*
  - 2) Remember the value for *height*
  - 3) Multiply *width* by *height* to get the *area*
  - 4) Return the result to the user
- Any data/information we want to work with in our script needs to be remembered, by storing it in a variable

# Variables

- Before you can use a variable, you need to announce that you want to use it
  - **declare** a variable by giving it a name  
`var quantity;`
    - `var` is a **keyword** that the JavaScript interpreter knows that it is used to create a variable
    - the name of the variable (`quantity`) is called an **identifier**
  - **Good practice:** use variable names that describe the kind of data that the variable holds

# Naming variables/identifiers

- Variable names can be more than one word → the convention is to use **lowerCammelCase**
- Use concise, human-readable, semantic names

## Do this:

```
let playerScore = 0;  
let speed = distance / time;
```

## Not this:

```
let thisIsaveryLONGVariableThatRecordsPlayerscore345654 = 0;  
let s = d/t;
```

# Naming variables/identifiers

- JavaScript is case sensitive
- e.g., `myVariable` is not the same as `myvariable`
- if you have problems in your code, check the case!
- can contain Latin characters (0-9, a-z, A-Z) , the underscore (`_`) character, and the dollar sign (`$`)
- the name should start with a letter, dollar sign (`$`), or an underscore (`_`); not with a number



# Naming variables/identifiers

- Do not use JavaScript reserved words as your variable names
  - words that make up the actual syntax of JavaScript, e.g., **var**, **function**, **let**, and **for**
  - they tell the interpreter to do something
  - a complete list:  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical\\_grammar#keywords](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#keywords)

# Variables

- Tell the variable what information you would like it to store for you
  - we **assign a value** to the variable  
`var quantity;`  
`quantity = 3;`
  - equal sign (=) is an assignment operator;
    - it assigns a value to the variable
    - it can also update the value given to the variable

# Variables

- The value for a variable that it is not assigned a value is considered **undefined**
  - i.e., you will receive an error message of *undefined* when trying to use that variable
- You can declare and initialize a variable at the same time
  - `let quantity = 3;`
  - this is what you will be doing most of the time; is quicker

# Variables

- The **var** and **let** keywords
  - **let** was created in modern versions of JavaScript
  - **let** fixes some issues that var has, related to **hoisting**
  - There is no reason to use **var**, unless you need to support Internet Explorer 10 or older with your code.
- Use **let** instead of **var**!

```
/** you can actually declare a variable  
 * with var after you initialize it and  
 * it will still work  
 */
```

```
myName = 'Chris'; // initializing the  
variable; assigning it a value
```

```
function logName() {  
    console.log(myName);  
}
```

```
logName();
```

```
var myName; // declaring the variable
```

# Variables / Constants

- The term *variable* implies that new values can be assigned to it
  - the value associated with the variable may vary as our program runs
- **Constants**
  - we can permanently assign a value to a name
  - used for values that are unchanging
  - you must initialize them when you declare them
  - you can't assign them a new value after you've initialized them

# Shorthand for creating variables

```
let username = 'Johanna Johansen';
```

```
let message = 'Thank you for ordering a name plaque';
```

```
-----
```

```
let username, message;
```

```
username = 'Johanna Johansen';
```

```
let message = 'Thank you for ordering a name plaque';
```

```
-----
```

```
let username = 'Johanna Johansen', message = 'Thank you for ordering a  
name plaque';
```

# Exercise

- Read the first part of the article
  - *What is JavaScript?; the A high-level definition part, and So what can it really do?* (up to “And much more!”)
  - [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\\_steps/What\\_is\\_JavaScript](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript)
  - copy the code here and play with it in <https://codepen.io/pen/>
  - identify in the given JavaScript code the part that we have already learned about
  - try to assign a new value to the `const`  
`name = prompt('Your name');`

# Data types

- There are a few different types of data we can store in variables: *numbers, strings, and boolean* data types
  - other data types: *arrays, objects, undefined, and null*
- **numeric** data type
  - handles numbers; integers (whole numbers), negative numbers, and decimals
  - used for doing calculations
    - e.g., determining the size of the screen, moving the position of an element on a page



# Example

- Example from previous week:  
calculating the cost of a name plaque  
based on the number of characters

```
// Create three variables to store the  
information needed.
```

```
let price;  
let quantity;  
let total;
```

```
// Assign values to the price and quantity  
variables.
```

```
price = 15;  
quantity = 15;
```

```
// Calculate the total by multiplying the  
price by quantity.
```

```
total = price * quantity;
```

```
// Get the element with an id of cost.
```

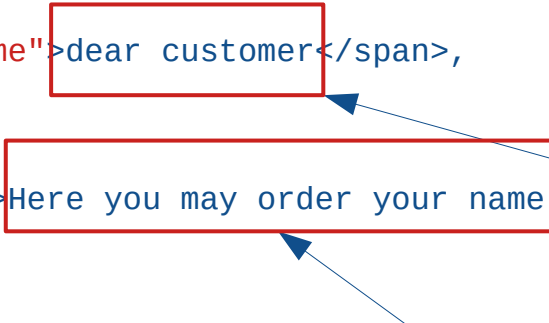
```
let el = document.getElementById('cost');  
el.textContent = `${total} kr`;
```

# Data types

- **string** data type
  - means any kind of text; words, sentences, and other characters
  - the string data type is enclosed within a pair of quotes; single or double quotes
    - choose one style and be consistent
  - Used to add new content into a page and they can contain HTML markup

# Example

```
<h1>Name plaque</h1>
<div id="content">
  <div id="title">
    Hello
    <span id="name">dear customer</span>,
  </div>
  <div id="note">Here you may order your name
plaque</div>
</div>
```



```
// Create variables to hold the name and note
text.
let username;
let message;

// Assign values to these variables.
username = 'Johanna Johansen';
message = 'Thank you for ordering a name plaque!';

// Get the element with an id of name.
let elName = document.getElementById('name');

// Replace the content of this element.
elName.textContent = username;

// Get the element with an id of note.
let elNote = document.getElementById('note');

// Replace the content of this element.
elNote.textContent = message;
```

# Example

```
let elNote =  
document.getElementById('note');
```

A variable used to hold a reference to an element in the HTML page (a node object)

```
// Create variables to hold the name and note text.  
let username;  
let message;  
  
// Assign values to these variables.  
username = 'Johanna Johansen';  
message = 'Thank you for ordering a name plaque';  
  
// Get the element with an id of name.  
let elName = document.getElementById('name');  
  
// Replace the content of this element.  
elName.textContent = username;  
  
// Get the element with an id of note.  
let elNote = document.getElementById('note');  
  
// Replace the content of this element.  
elNote.textContent = message;
```

# Example

- **Escaping** the quotation characters
  - using a **backwards slash** before any type of quote mark that appears within a string
  - tells the interpreter that the respective character is part of the string.

```
<h1>Name plaque</h1>
```

```
<div id="content">
```

```
<div id="title">
```

```
Hello
```

```
<span id="name">dear customer</span>!
```

```
</div>
```

```
<div id="note">Here you may order your name  
plaque</div>
```

```
</div>
```

```
...
```

```
// Assign values to these variables.
```

```
plaqueName = 'Johanna Johansen';
```

```
message = 'Thank you for ordering a name  
plaque.';
```

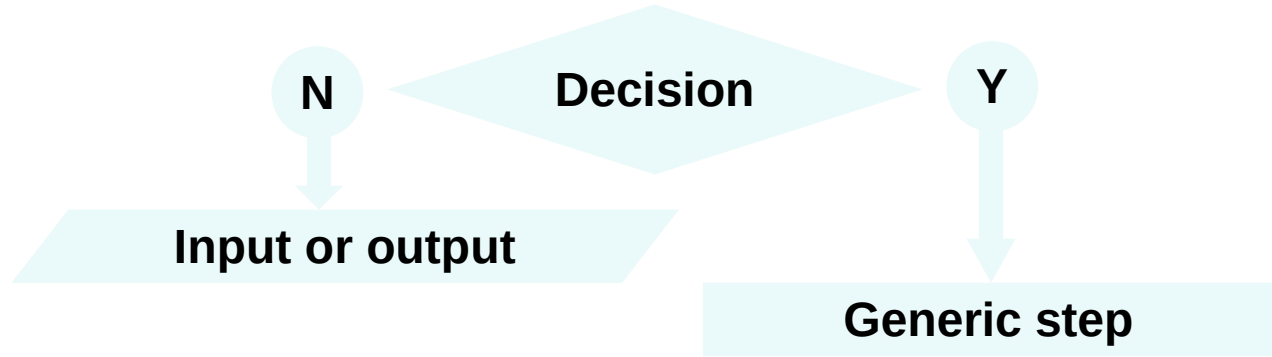
```
motto = 'Our motto is \"Your beautiful name  
deserves a beautiful name plaque.\"'
```

```
...
```

```
// Replace the content of this element.
```

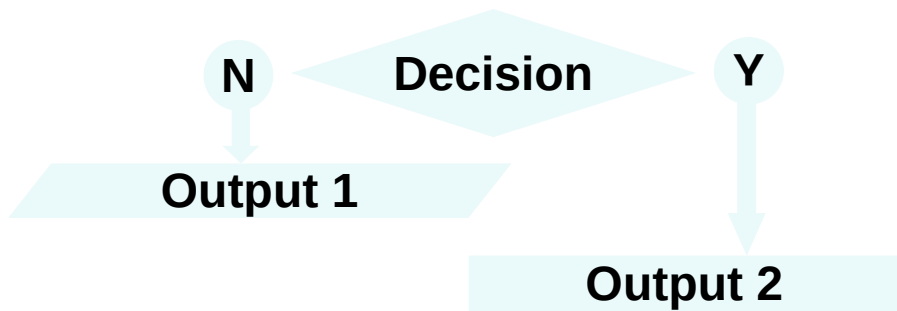
```
elNote.textContent = message + motto;
```

# Data types



- **boolean** data type
  - can have one of two values: *true* or *false*
- **Remember:**
  - that we talked about taking decisions in our flowchart
  - Boolean data is helpful with deciding which part of the script to run

# Example – storing a boolean in a variable



```
// create variables and assign their values
let inStock;
let message;
```

```
inStock = true;
```

```
if(inStock) {
    message = 'The product is in stock';
} else {
    message = 'Unfortunately, the product is
temporary not in stock';
}
```

```
// get the element that has an id of note
let elNote = document.getElementById('note');
```

```
// Replace the content of this element to
display if the product is in stock
elNote.innerHTML = message;
```

## Example – changing the value of a variable

```
// create variables and assign their values
let inStock = true;
let message;

/**
 * Some other processing might go here.
 * The script needs to change the value of the inStock
variable.
 */

inStock = false;

if(inStock) {
    message = 'The product is in stock';
} else {
    message = 'Unfortunately, the product is temporary not in
stock';
}
...
```



# Variables

- JS is **dynamically typed**
  - when declaring a variable, you do not need to specify what kind of data the variable will hold
  - the type is decided based on the value of a variable
  - a variable initially bound to a number may be reassigned to a string
  - the type is checked at the runtime

# Exercise

- Try out the examples offered by your syllabus book for data types
  - <https://javascriptbook.com/code/c02/>
  - Choose one of the following examples:
    - Using a Variable to Store a Number
    - Using a Variable to Store a String
    - Using Quotes Inside a String
    - Using a Variable to Store a Boolean
  - Copy both the html and js code in the <https://codepen.io/pen/> and play with it there
  - for the HTML file right click on the page and choose *View page source* to be able to copy the html code
    - NOTE: paste only the code between the <body> tags
    - For the *Using a Variable to Store a Boolean* example, open the Developer Tools (F12) and note what happen with the code example  
`<span id="stock"></span>` and `<span id="shipping"></span>`

# Arrays

- A special variable type that stores a list of values
  - one reference for multiple values
- Used when working with lists or a set of values related to each other
- Especially helpful for when you do not know how many items a list will contain
  - e.g., the number of items in an online shopping cart might differ from one customer to another

# Arrays

- Creating arrays:

- array literals

```
let shoppingList = ['cheese', 'pumpkin', 'spinach',  
  'bread'];
```

- the spread operator

- used to include the elements of one array within another array literal

```
let shoppingList = ['cheese', 'pumpkin', 'spinach', 'bread'];  
let shoppingListParty = [...shoppingList, 'chips', 'chocolate',  
  'juice']; // shoppingListParty == ['cheese', 'pumpkin', 'spinach',  
  'bread', 'chips', 'chocolate', 'juice']
```

# Arrays

- Creating arrays:
  - ...
  - the `Array()` constructor
    - `let shoppingList = new Array();` // equivalent to `let shoppingList;`
    - `let shoppingList = new Array('cheese', 'pumpkin', 'spinach', 'bread');` // equivalent to: `let shoppingList = ['cheese', 'pumpkin', 'spinach', 'bread'];`
  - array literals are almost always simpler to use

# Arrays

- Each item in an array are automatically given a number called **index**
- The index can be used to access specific items in the array

INDEX	VALUE
0	cheese
1	pumpkin
2	spinach
3	bread

```
let shoppingList;  
shoppingList= ['cheese',  
'pumpkin', 'spinach', 'bread'];
```

# Arrays

- Accessing items in an array

```
let diaryProducts;
```

```
diaryProducts = shoppingList[0]; // 'cheese' is the element with  
the index 0
```

- Getting the number of items in an array
  - using the *length property*

```
let nrListItems;
```

```
nrListItems = shoppingList.length; // 4
```

# Arrays

- Changing values in an array

```
let shoppingList = ['cheese', 'pumpkin', 'spinach', 'bread'];  
shoppingList[1] = 'salat'; // the elements in the list are now  
                             'cheese', 'salat', 'spinach', 'bread'
```



# Exercise

- Use the code on the page 71 (*Creating an array*) of your syllabus book and change it as following
  - add another array to your code named *colorKeywords* that should contain 5 more color keywords taken from the list here:  
[https://developer.mozilla.org/en-US/docs/Web/CSS/color\\_value#color\\_keywords](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#color_keywords)  
add the elements of the *colorKeywords* to the initial colors array by using the *spread operator*
  - change the color displayed to be one of the colors added with the spread operator instead of 'white'

# Expressions

- An **expression** *evaluates* (results in) to produce a value
- Simpler expressions
  - e.g., a variable evaluates to whatever value has been assigned to that variable  
`let color = 'blue';`
- Complex expressions are built from simpler expressions
- The most common way to build complex expressions out of simpler expressions is with the help of an **operator**

# Operators

- An **operator** combines the values of its **operands** in some way and evaluates to a new value
- Types of operators
  - assignment operator (=)
    - used to assign a value to a variable  
`price = 15;`

```
/* The name plaque example */  
let price;  
let quantity;  
let total;
```

```
price = 15;  
quantity = 15;
```

```
// total evaluates into one value 225  
(15 * 15 = 225)  
total = price * quantity;  
...
```

# Types of operators

- arithmetic operators

- used to perform basic math operations

```
total = 15 * 15 // the  
multiplication operator
```

- addition (+)
- subtraction (-)
- division (/)
- multiplication (\*)

- increment (++) → adds one to the current number

```
i = 10;  
i++; // 11
```

- decrement (--) → subtracts one from the current number

```
i = 10;  
i--; // 9
```

- remainder (%) – or modulo in languages such as Python → divides two values and returns the remainder

```
10 % 3 // 1
```

# Types of operators

- arithmetic operators
  - **order of the execution** of the operations → multiplications and divisions are performed before addition or subtraction  
`total = 2 + 4 * 10; // total is 42`
  - to change the order in which the operations are performed, place the calculations you want done first inside parentheses  
`total = (2 + 4) * 10; // total is 60`

# Types of operators

- string operator
  - the plus sign (+)
  - **concatenation** → joining together two or more strings to create one new string
  - use case: the name plaque → we would like to give the user the possibility to choose that the name s/he registered with on our website to be used for the plaque; the first name and last name of the user were stored separately and we would have to combine them in order to get all the letters in the name for our price calculation.

```
let firstName = 'Ivar';  
let lastName = 'Hendriksen';  
let fullName = firstName + ' ' + lastName; // the full name will be Ivar  
Hendriksen
```

# Types of operators

- string operator

- use case: we would like to display a personalized message to the user to thank him/her for the order

```
let firstName = 'Ivar';
```

```
let thanksMessage = 'Thank you ' + firstName + ' for  
your order and welcome back!'; // Thank you Ivar for  
your order and welcome back!
```

# Types of operators

- string operator

- combining numbers with strings results in strings
- e.g., code on slide 24 → calculating the price of the name plaque and displaying a message with the cost

```
let price;  
let quantity;  
let total;
```

```
price = 15;  
quantity = 15;
```

```
total = price * quantity;
```

```
let el = document.getElementById('cost');  
el.textContent = 'The price is ' + total  
+ ' kr'; // The price is 250 kr
```



# Creating strings

- by combining strings with expressions (eg., variable name) by using **template literals (template strings)**
  - use of backticks (`) instead of quotation marks
  - the variable names are included with `${varName}`

```
e1.textContent = 'The price is ' + total + ' kr'; // The price is 250 kr
```

```
e1.textContent = `The price is ${total} kr`; // The price is 250 kr
```
- **Note:** the recommended way – [https://developer.mozilla.org/en-US/docs/MDN/Writing\\_guidelines/Writing\\_style\\_guide/Code\\_style\\_guide/JavaScript#template\\_literals](https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Writing_style_guide/Code_style_guide/JavaScript#template_literals)

# Exercise

- Active learning: sizing a canvas box

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\\_steps/Math#active\\_learning\\_sizing\\_a\\_canvas\\_box](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#active_learning_sizing_a_canvas_box)

only the first three bullet points

- **TODO:** see how to use the *Console* in the *Developer Tools*, for implementing the exercise in this section:

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\\_steps/Math#its\\_all\\_numbers\\_to\\_me](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#its_all_numbers_to_me)

Next time we continue with  
functions, methods, and objects