Introduction to programming in JS

IIKG1002/IDG1011 – Front-end web development

Johanna Johansen
johanna.johansen@ntnu.no

- Syllabus
  - Duckett, J., Ruppert, G., & Moore, J. (2014). JavaScript & jQuery : interactive front-end web development. Wiley.
    - NOTE: we will use it only for the JavaScript part
    - https://archive.org/details/javascriptjqueryjonduckett
  - David Flanagan. (2020). JavaScript: The Definitive Guide, 7th Edition. O'Reilly Media, Inc.
    - Note: used to complement the book by Duckett and Moore
  - Mozilla Developer: https://developer.mozilla.org/en-US/docs/Learn

    https://developer.mozilla.org/en-US/docs/Web/JavaScript

- JavaScript ≠ Java
- Key concepts in computer programming and how these apply to the JavaScript language
- How JavaScript is used to create more engaging, interactive, and usable websites
- We learn vanilla JavaScript, no related library
- Basics of the vocabulary and syntax of the JavaScript language

- How Document Object Model (DOM) lets you access and change the content of a web page with JavaScript
- Events that are used to trigger certain parts of the JavaScript code
- Application Programming Interfaces (APIs) such as for geolocation and storage
- Error-handling and debugging of your JS code
- We we learn how to design and write scripts from scratch

Front-end development

- What the users see and interact with
  - people interact with web browser
  - as opposed to back-end which is related to web server and data base
- As front-end developers we are concerned with
  - different browser support
  - different types of devices are used by users
  - different user needs, special needs → web accessibility, universal design

- There are three basic languages that are used to create web pages: HTML, CSS, and JavaScript
- HTML
  - is the content layer
  - the HTML code is saved in .html files
  - this is the content of the page
  - with HTML markup we give the content structure and meaning (semantics)
  - we use tags to annotate content, e.g, <h1></h1>, <img>

- CSS
  - the presentation layer
  - .css files
  - we create rules which are associated with HTML elements
  - the rules state how the HTML content is presented
    - e.g., the font type, size, color, background colors, box dimensions, etc.

- JavaScript
  - the behavior layer
  - .js files
  - it adds interactivity to the page
  - how the page behaves when the user interacts with elements on the page: clicking, scrolling, filling in forms

- We call them layers because they build on top of each other
- we use separate files for each language
- the HTML pages link to the .js and .css files
- keeping these as separate layers allows for **progressive enhancement**

Exercise

- Read this article

  https://alistapart.com/article/understandingprogressiveenhancement/

- Note down what **progressive enhancement** is, as well as **graceful degradation**.
- What is the difference between the two?

Progressive enhancement

- provides a baseline of essential content and functionality to as many users as possible
- creates a design that achieves a simpler-but-still-usable experience for users of older browsers and devices with limited capabilities,
  - the capabilities of each device and connection speeds varies
  - some people browse with JS turned off
  - even if the user cannot load the JS, it still has access to the content

Progressive enhancement

- starting with the HTML layer
  - you make sure that all users have access to the most important part of your page, the content
  - it loads quickly on slow connections
  - you can have access to it from all devices

Progressive enhancement

- Separated CSS
  - the same style sheet can be used for all the pages of a web site
    - no duplicated code, easier to maintain, faster to load
  - use different style sheets with the same content, i.e., different visuals for different accessibility requirements

Progressive enhancement

- The usability and experience of interacting with the web page is in the end enhanced with JS
  - separated .js files
    - reuse the code on several pages
    - easier to maintain
    - faster loading times
- A design that progresses the user experience up to a more-compelling, fully-featured experience for users of newer browsers and devices with richer capabilities

Key programming concepts

What is a script?

- **A script:** a series of instructions that a computer can follow to achieve a goal
- E.g., following the instructions in a recipe, following manuals, handbooks, guides
- Simple scripts deal with one scenario, more complex deal with several scenarios and tasks that need to be performed
- Following only some of the steps based on a certain event/case
  - A browser uses only some parts of the scripts based on how the user interacts with the page
  - E.g., the guides for installing an application on our computer; different ones depending on the OS on our computer

Creating a script

1) Specify the goal and list the tasks to be performed BY THE SCRIPT
   - the big picture of what we want to achieve
2) Designing the script
   - A computer needs detailed instructions and any information it needs to perform the task
   - Break each task in smaller steps that the computer will perform one at a time
   - The computer needs more details than us humans

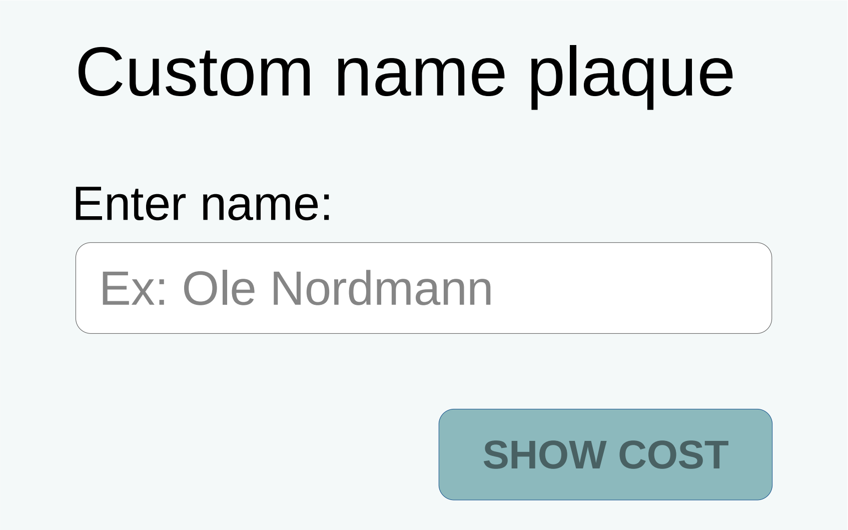
Creating a script

2) Designing the script

- might be tempting to start coding right away; design the script before writing it
- can use flowcharts

Example – designing a script

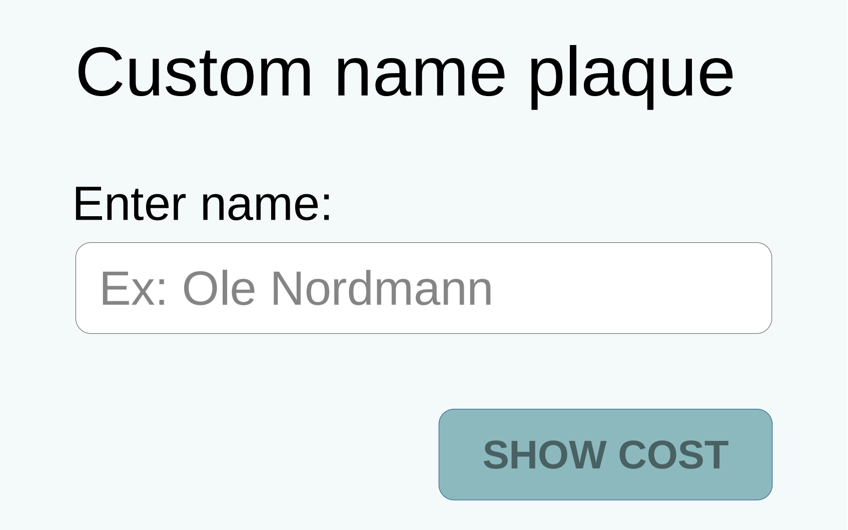
- The goal: calculating the cost of a name plaque, where the customers are charged by letter
- Detailing the goals:
  - customers can enter the name for the plaque
  - each letter costs 15 Kr
  - when the a user enters a name, show how much it will cost



Custom name plaque

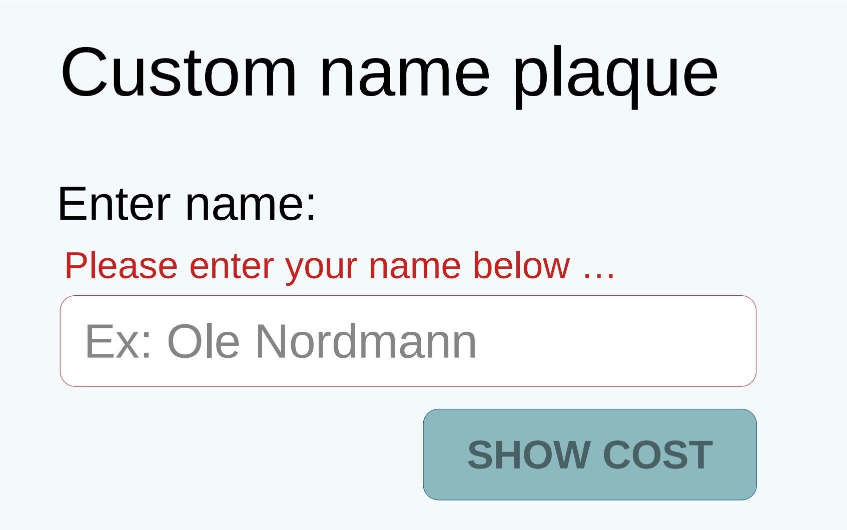
Enter name:

Ex: Ole Nordmann



Custom name plaque

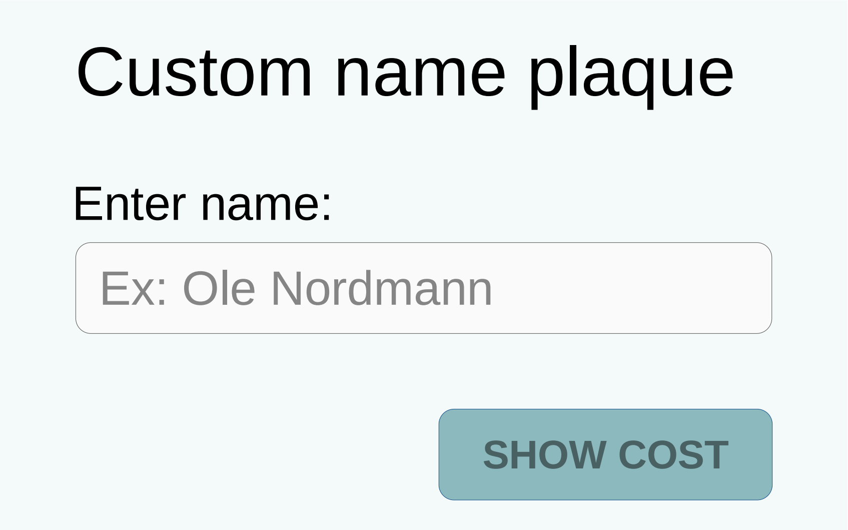
Enter name:



Custom name plaque

Enter name:

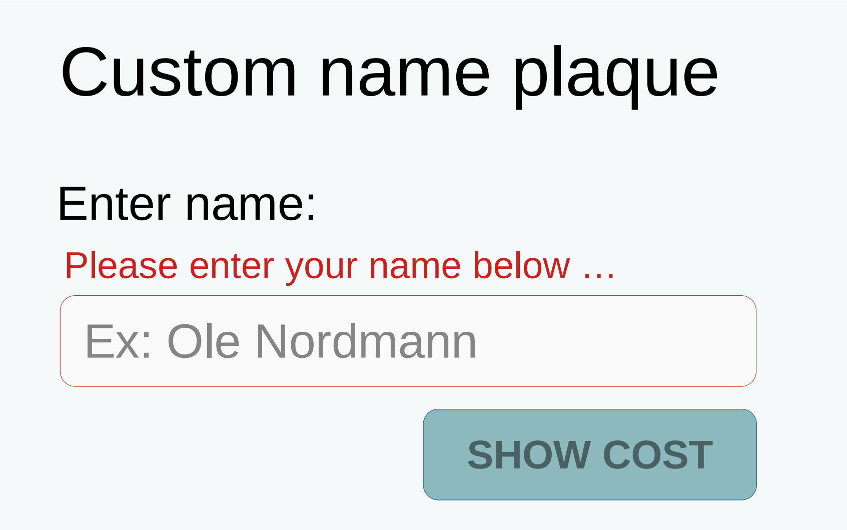
Please enter your name below …



Custom name plaque

Enter name:

Ex: Ole Nordmann

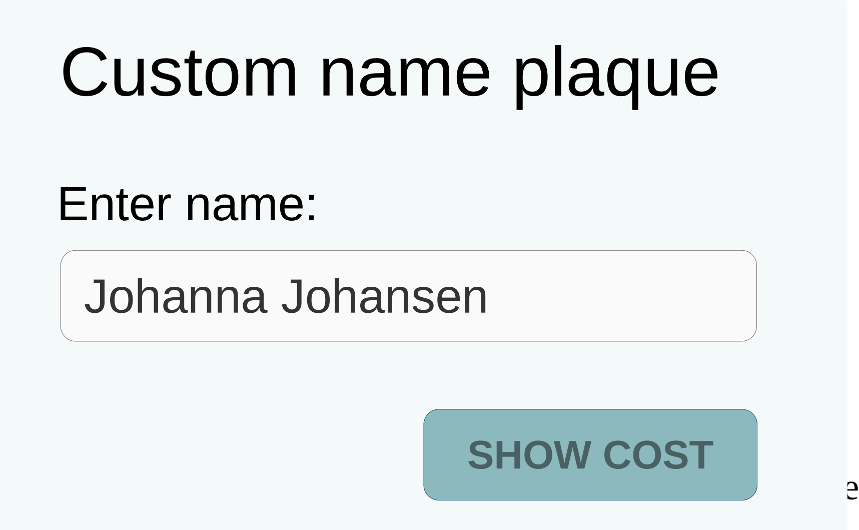


Custom name plaque

Enter name:

Please enter your name below …

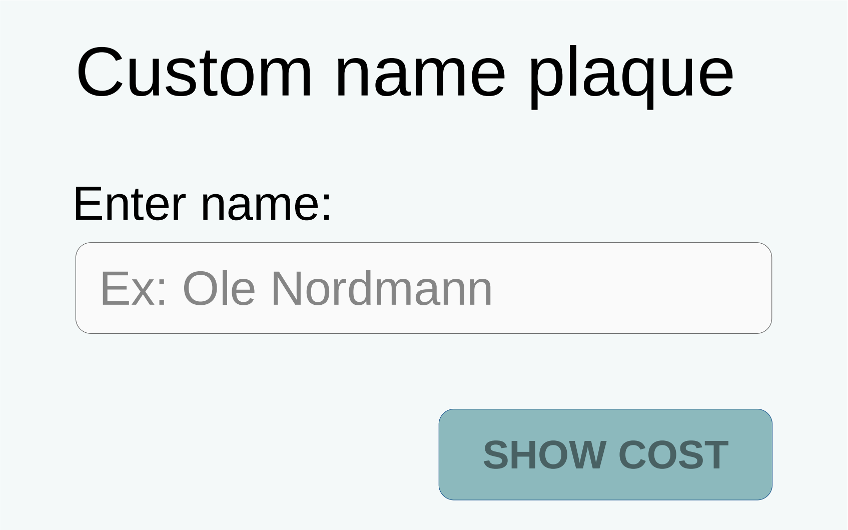
Ex: Ole Nordmann



Custom name plaque

Enter name:

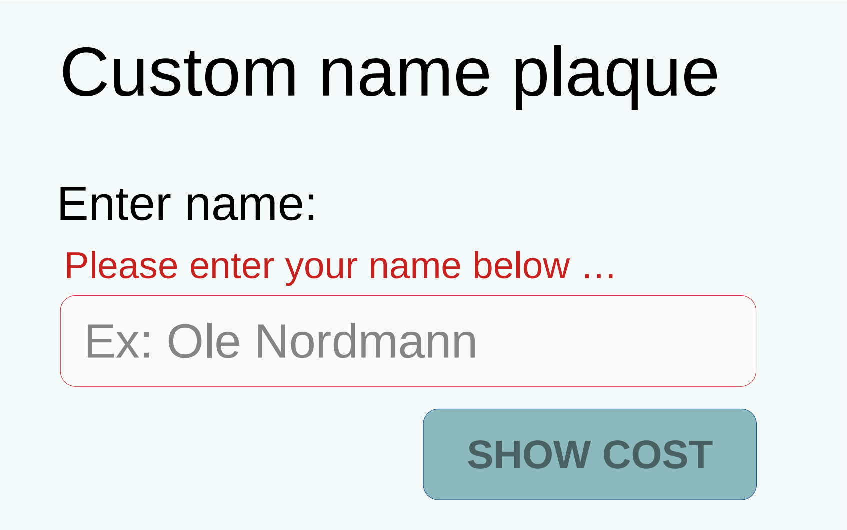
Johanna Johansen



Custom name plaque

Enter name:

Ex: Ole Nordmann

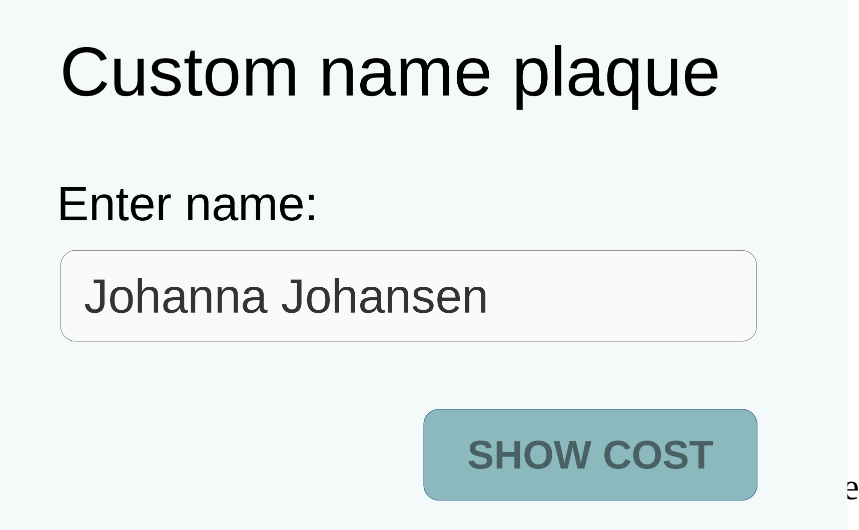


Custom name plaque

Enter name:

Please enter your name below …

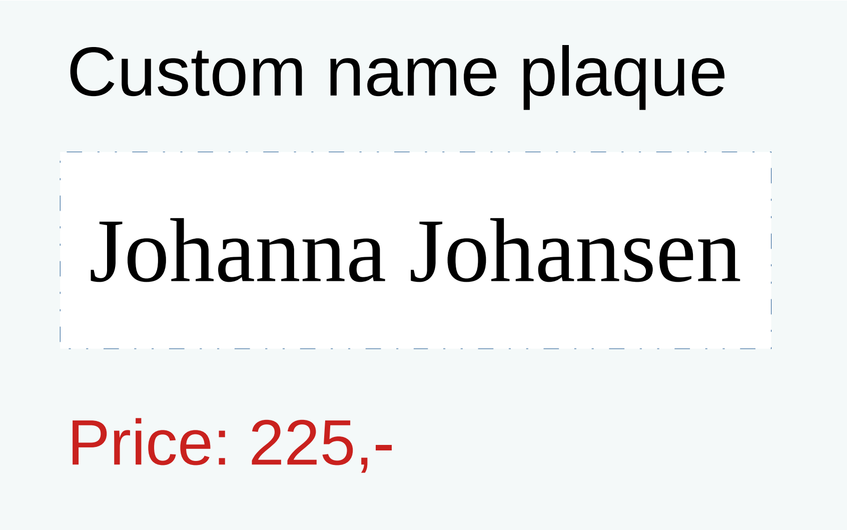
Ex: Ole Nordmann



Custom name plaque

Enter name:

Johanna Johansen



Custom name plaque

Johanna Johansen

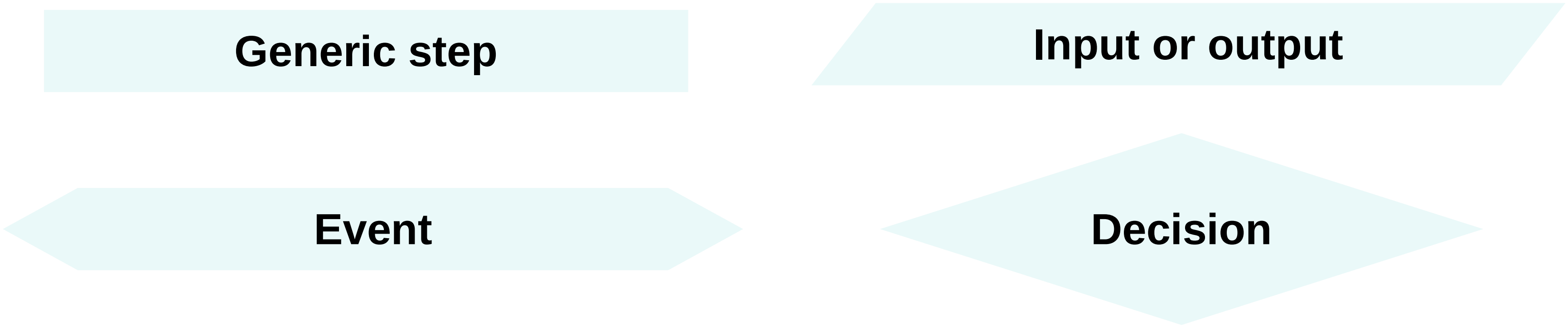
Price: 225,-

Example – designing a script

- Create a list of tasks that have to be performed in order to achieve these goals:
    1) The script is triggered when the “Show Cost” button is clicked
    2) It collects the name entered into the form field
    3) It checks that the user has entered a value
    4) If the user has not entered anything, a message will appear telling them to enter a name
    5) If a name has been entered, calculate the cost of the sign by multiplying the number of letters by the cost per letter
    6) Display to the user how much the plaque costs

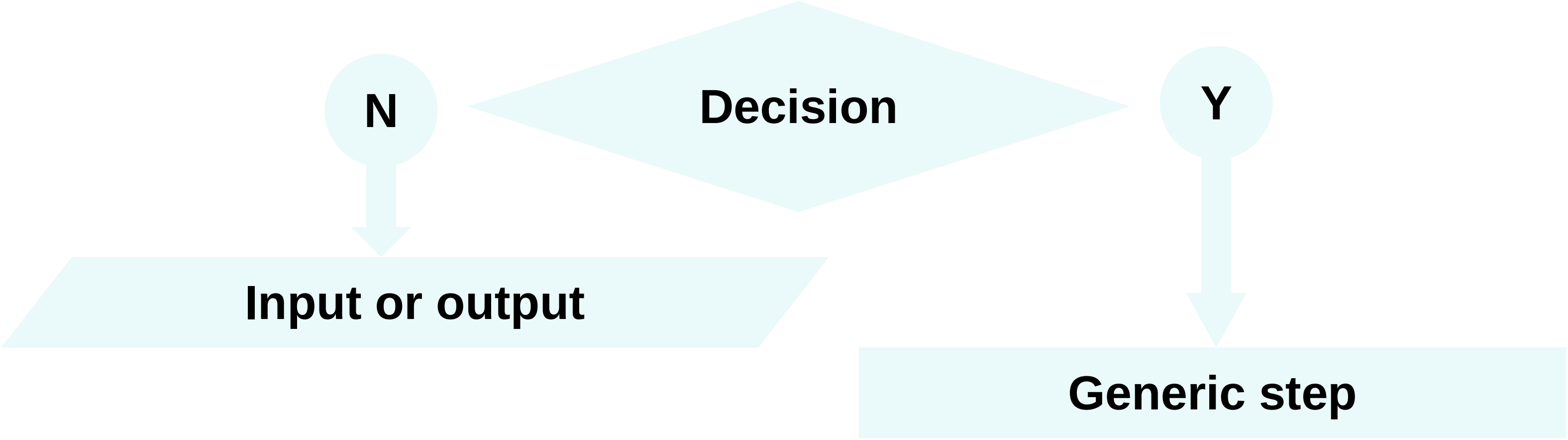
Example – designing a script

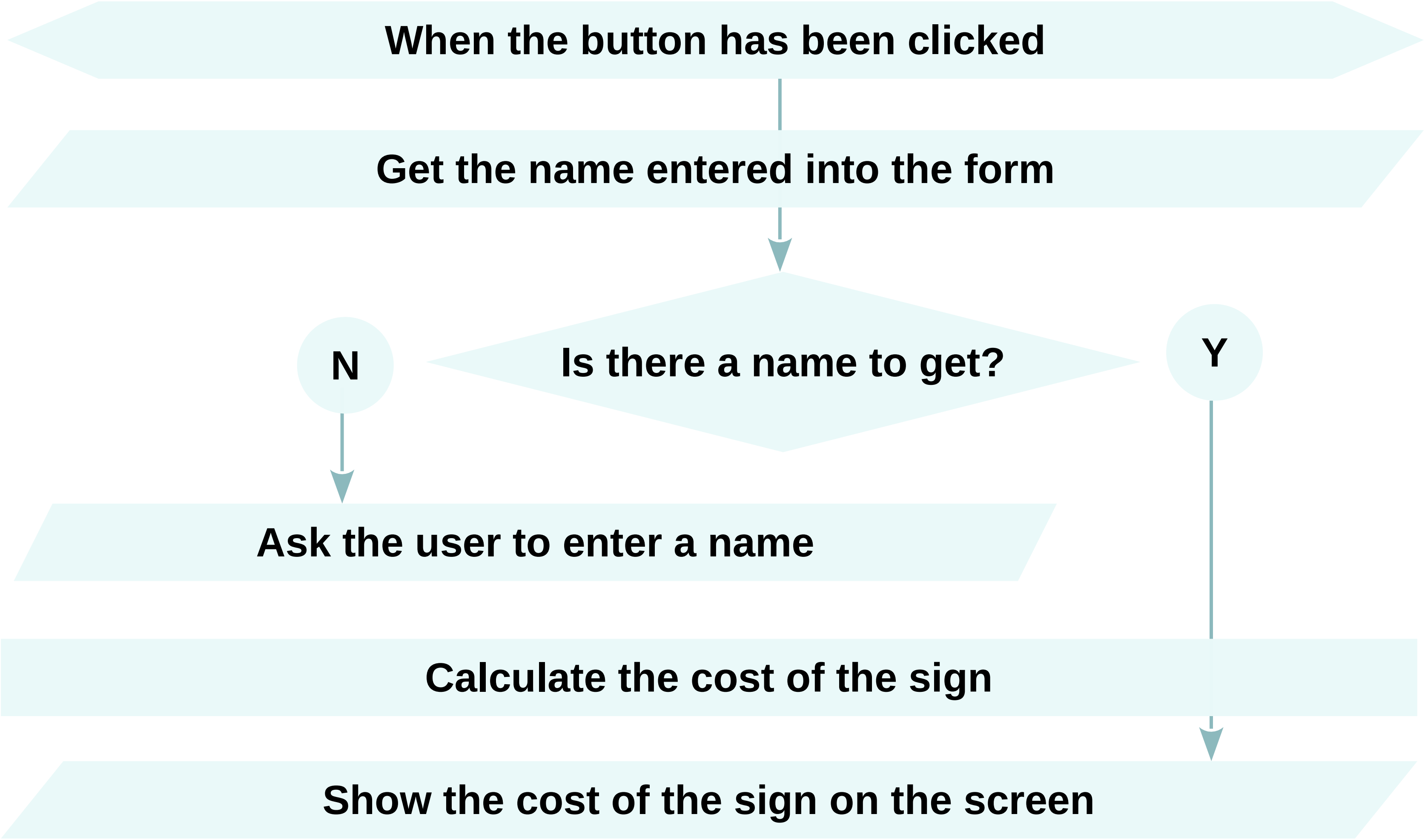
- **Flowcharts**
  - shows how the tasks fit together and the paths between each step
  - **Arrows** → the script moves from one task to the next
  - The different shapes represent different types of tasks



Example – designing a script

- Flowchart
  - Decisions → cause the code to follow different paths





When the button has been clicked

Get the name entered into the form

Is there a name to get?

N

Y

Ask the user to enter a name

Calculate the cost of the sign

Show the cost of the sign on the screen

Exercise

- Design, similarly to the example shown, your own script on a topic of your choice
  - specify the main goal
  - break it down into smaller tasks
  - create a flowchart for the script

Creating a script

3) Coding the script

- each of these steps are translated in a programming language that the computer understands; JavaScript in our case
- any language has a
  - **vocabulary**: the words the computer understands
  - **syntax**: how to put those words together
- a computer uses a **programmatic** approach to problem-solving

Creating a script

- Programmatic approach
  - they follow a series of instructions, one after another
  - they need to be told any detail of what they are expected to do
  - they are very logical and obedient; they follow the instructions without question
- Besides learning the language itself, one has to learn how to write instructions that a computer can follow

Creating models – objects

- The things from the real world need to be modeled so that the computer understands them
- **Objects** are created to represent physical things
  - e.g., if creating an application for booking an hotel or an application for calculating the speed of a car; both the hotel and the car are represented as objects
- If we need to model two objects of the same type, we create **instances** of the respective object
  - e.g., several instances of the hotel object or the car object

Creating models – properties

- Each object has it own
  - properties
  - events
  - methods
- **Properties** are the characteristics of the object
  - e.g., a car will have a make (brand), color, engine size, a speed, fuel type, etc.
  - e.g., a hotel will have a name, a number of rooms, will have / or not a pool, ratings, etc.

Creating models – properties

- Each property has a **name** and a **value**
- The name/value pairs tells you something about each individual **instance** of the object;
- The instances have the same name, but their values are different
  - E.g., for a hotel → name: Thon Hotel, rating: 3 stars, rooms: 65
- Name/value pairs are used a lot in programming; in the case of HTML and CSS:
  - HTML elements→ an attribute is like a property; an attribute has a name and a value

    ```
    <p class=“fruit”>peach</p>
    ```
  - CSS rules → property name and value

    ```
    .fruit {color: pink;}
    ```

Creating models – events

- Events model the interaction of people with these objects
- Through these interactions the values of the properties can be changed
  - e.g., people booking rooms in a hotel makes the value of a for example *booked* property to change; i.e., the number of booked rooms increases

Creating models – events

- Programmers decide which events they want to respond to
  - e.g., pressing the “Book room” button at the interface of the booking application
  - When a specific event happens, the event can be used to trigger a specific section of the code
  - There are many such events that can happen; in the script the programmers specify
    - which event they want to respond to
    - which part of the script should be run for these specific events

Creating models – methods

- **Methods** model what the people do with the objects
- Methods perform tasks using the properties of the object; i.e., retrieving or updating the values of the properties
- Represents the task that is to be done and the instructions used to achieve the respective task
  - e.g., `makeBooking()` → increases the value of *booked* property

Creating models

- The events, methods, and properties of an object are related to each other
  - Events can trigger methods
  - Methods can retrieve, add, or update an object's properties

Object type: `hotel`		
Event fired	Method called	Property changed
`book`	`makeBooking()`	`booked: 22`
reservation is made	increases the value of the number of the rooms booked	the value of the booked property is no longer 21 but 22

Modeling in web development

- Web browsers create models of
  - the page they are showing
    - `document` object
    - property for document, e.g., `location` → the URL of the current page
  - the browser window that the page is shown in
    - `window` object
    - property for window, e.g., `title` → what is in between the opening and closing <title> HTML tags

The `document` object

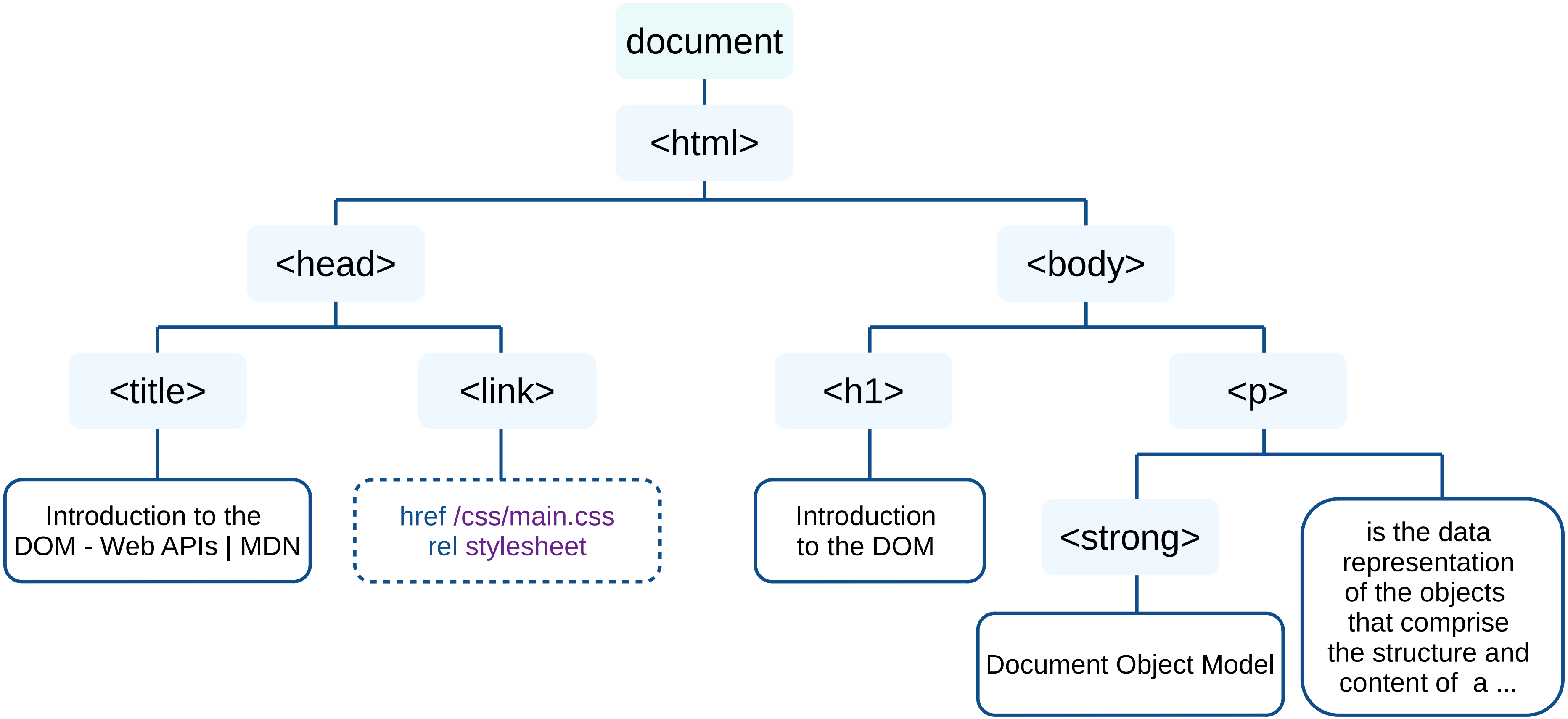
- is one object out of several that all major browsers have support for
- allows us to
  - access and change what content users see on the page
  - respond to the user interaction with the page
- has properties, methods, and events like any other object
  - already existing ones , i.e., implemented by the people who created the browser

The `document` object

  - properties → described the characteristics of the current web page; e.g., title of the page
  - methods → performs tasks such as getting information from a specific HTML element or adding new content to the page
  - events → e.g., user clicking on a button on the page
- one object is created for each HTML element on the page
- all these objects are described in the **Document Object Model (DOM)**

Document Object Model (DOM)

- How a browser works with our pages
  - when it receives the HTML page it creates a model of it
    - each page of a web site is seen as separate document
    - the model it creates looks like a **family tree**
    - the *document* object is at the top, representing the whole page
    - every object located within a document is a **node**
    - an node can be of type *element*, *text*, or *attribute* node



document

<html>

<head>

<body>

<title>

<link>

<h1>

<p>

Introduction to the DOM - Web APIs | MDN

href /css/main.css
rel stylesheet

Introduction to the DOM

<strong>

Document Object Model

is the data representation of the objects that comprise the structure and content of a ...

Document Object Model (DOM)

- How a browser works with our pages
  - ...
  - the model is stored in the memory
  - it shows the page on the screen with the help of a **rendering engine**
    - the rendering engine processes the CSS rules and applies them to each corresponding element

- How a browser works with your page
  - ...
  - the browser uses an **interpreter** for the JavaScript files to translate them into instructions the computer can follow
    - JavaScript is an **interpreted programming language**
    - each line of code is translated one-by-one as the script is run
- The JS code that you are going to write, uses the model that the browser creates for the web page

Starting with writing scripts

Linking/including a script

- the HTML `<script>` element is used to load the JavaScript file into the page
  - the the value of the attribute `src` is the path to the script file
  - tells the browser to find and load the script file
- This is doing the same job as the <link> element for CSS
  - It applies the JavaScript to the page, so it can have an effect on the HTML

***NOTE:** additional information on linking is provided later in the course; i.e., use of `defer` and `async`*

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp; Co.</title>
    <link rel="stylesheet"
href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp; Co.</h1>
    <script src="js/add-content.js"></script>
    <p>For all orders and inquiries please call
<em>555-3344</em></p>
  </body>
</html>
```

Linking/including a script

- **Good practice/ OLD practice *:** the link to the js file is usually included just before the closing </body> tag, near the bottom of the HTML file
- The position of the <script> element can affect how quickly a web page seams to load
  - In early days the <script> was placed in the <head> tag of the html page
    - this can make pages seem slower to load
  - when a browser starts to download a js file
    - stops all other downloads
    - pauses laying out the page

  until the script has finished loading and been processed

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp; Co.</title>
    <link rel="stylesheet"
href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp; Co.</h1>
    <p>For all orders and inquiries please call
<em>555-3344</em></p>
  <script src="js/add-content.js"></script>
  </body>
</html>
```

* *Later in the course: use of `defer` and `async`*

Linking/including a script

- The browser reads code in the order it appears in the file
  - Placed at the end of the document it will not affect the rendering of the rest of the page
  - If the JavaScript loads first and it is supposed to affect the HTML that hasn't loaded yet, we will be getting errors.

Linking/including a script

- The JS code is added directly into the html between the opening and closing `<script>` tags
  - the `src` attribute is no longer needed
  - the `document.write()` is one way to write content into the document
- This method mixes HTML with JavaScript and is not recommended
- When the browser comes across a <script> element, it stops to load the script

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp; Co.</h1>
    <script>document.write('<h3>Welcome!</h3>');
    </script>
    <p>For all orders and inquiries please call
<em>555-3344</em></p>
  </body>
</html>
```

Exercise

On pages 46-49 of the syllabus book
Duckett, J., Ruppert, G., & Moore, J. (2014). *JavaScript & jQuery : interactive front-end web development*. Wiley.

https://archive.org/details/javascriptjqueryjonduckett

- Follow the steps (1-8) to experiment with including/linking a js file.
- You do not need to understand the JavaScript code yet
- The code shown in the book can be downloaded from https://javascriptbook.com/code/ (The ABC of programming)
- Use a text editor to edit your code for now

```
document.write('Welcome!');
```

- example of **calling** the method of an object
  - `document`
    - is an object representing the entire page
    - is already implemented by the browser
    - the document object has several methods and properties – **members** of the object
    - the members can be accessed by using a dot between the object and the member

```
document.write('Welcome!');
```

- example of **calling** the method of an object
  - …
  - `write()`
    - is the method of the document object and allows new content to be written into the page
    - a method requires some information to work with, which this is given in the (); what to write into the page

Basic JavaScript instructions

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

```
var today = new Date();
var hourNow = today.getHours();
var 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

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:
  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 x 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

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

```
var quantity = 3;
```

  - this is what you will be doing most of the time; is quicker

Naming variables

- 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

- 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

- 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

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 usename, 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
  - 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');
```

Next time we continue with
other basic JavaScript instructions