

SIT774 Web Technologies and Development

Workshop Week 6 - JavaScript

Instructor : Ran Zhang

Unit chair : Michael Hobbs

Instructor: Ran Zhang & Ruyi Li



Creating interactive web pages using Javascript (6.1-6.4):

Investigate how you can create interactive web pages using Javascript.

Jumpstart Javascript (6.5-6.8): Explore the basic elements of Javascript and how it is used to program on the client-side.

Wrap up (6.9): Reflect on how you can use Javascript to build interactive web pages.

Let
If; if...else; switch;
For
While
Do...while
Function

6.1 Welcome back

Web Page

6.2 What is JavaScript?

Web Page

6.3 Using JavaScript in a web page

Web Page

6.4 Writing JavaScript code

Web Page

6.5 JavaScript elements

Web Page

6.6 JavaScript arrays

Web Page

6.7 JavaScript condition and function

Web Page

6.8 JavaScript switch statement

Web Page

6.9 Javascript-Object-Notation

Web Page

6.10 That's a wrap

Web Page

Ontrack task 6.1P

Ontrack task 6.2C + table

Ontrack task 6.3D + rating

HTML,CSS and JavaScript

Why Study JavaScript?

JavaScript is one of the 3 languages all web developers must learn:

1. **HTML** to **define the content** of web pages
2. **CSS** to **specify the layout** of web pages
3. **JavaScript** to **program the behavior** of web pages

6.2 JavaScript basic

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

What Can JavaScript Do?

JavaScript can change HTML content.

Click Me!

What Can JavaScript Do?

Hello JavaScript!

Click Me!

The <script> Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript in Body

My First JavaScript

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

- JavaScript is a **premier scripting or programming language** used today on the Web. It is used to enhance the functionality and appearance of web pages.
- JavaScript code is embedded in a HTML document by using the `<script>` tag.
- The text enclosed **between <script> and </script>** is the JavaScript code.
- JavaScript is a **client-side** scripting language.
- JavaScript is an **object-based** programming language.
- You can put it in the `<body>` or `<head>` Section.

JavaScript in <head> and <body> example

JavaScript in <head>

In this example, a JavaScript `function` is placed in the `<head>` section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

Demo JavaScript in Head

A Paragraph.

Try it

Demo JavaScript in Head

Paragraph changed.

Try it

JavaScript in <body>

In this example, a JavaScript `function` is placed in the `<body>` section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Demo JavaScript in Body</h2>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Demo JavaScript in Head

A Paragraph.

Try it

Demo JavaScript in Head

Paragraph changed.

Try it

6.3 JavaScript in a web page

External JavaScript

External JavaScript are written and saved in a separate document (with **.js** extension) and then linked to various Web documents.

The linked **.js** file will be loaded at runtime. The linking is implemented via the **src** attribute of a **<script>** tag.

Example

```
<script src="myScript.js"></script>
```

External file: myScript.js

```
function myFunction() {  
    document.getElementById("demo").innerHTML = "Paragraph changed.";  
}
```

External JavaScript **Advantages:**

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>  
<script src="myScript2.js"></script>
```

Using JavaScript in a web page Cont.

Internal JavaScript

Internal JavaScript are scripts enclosed between the tag `<script>` and `</script>` and embedded within the *head or body* section of the Web documents.

```
<script>  
    //script statement(s)here ...  
</script>
```

Inline JavaScript

Inline scripts are included within the values of attributes.

```
<html>  
  <head>  
    <title>inline javascript</title>  
  </head>  
  <body>  
    <input type="button" onclick="alert('hello')" value="click me!">  
  </body>  
</html>
```

6.4 JavaScript code and statement

- The **<type> attribute** of the <script> tag specifies the type of the script (e.g., text) as well as the scripting language (e.g., text) as well as the scripting language (e.g., JavaScript). **In HTML5, this attribute can be omitted.**
- It can be seen from the example that each JavaScript statement is ended with a **semicolon (;).**
- In this simple example, we will see some important concepts of JavaScript programming in action:

Specify script type declaration

```
<script type="text/javascript">  
let name;  
name = window.prompt ("Please enter your name");  
document.writeln ("<h1> Hello " + name + ",");  
document.writeln ("<br> Welcome to JavaScript World! </h1>");  
</script>
```

keyword JavaScript statement is ended with a semicolon

First statement:

- keyword: let;
- variable: name;
- declaration: let name


assignment operator object method

```
<script type="text/javascript">  
let name;  
name = window.prompt ("Please enter your name");  
document.writeln ("<h1> Hello " + name + ",");  
document.writeln ("<br> Welcome to JavaScript World! </h1>");  
</script>
```

Second statement:

- Object: window;
- method: prompt;
- assignment operator: =

JavaScript code and statement Cont.



The diagram shows a snippet of JavaScript code within a `<script>` block. Annotations with arrows point to specific parts: 'object' points to `document`, 'method' points to `writeln`, and 'string concatenation' points to the `+` operator in the string `"<h1> Hello " + name + ", "`.

```
<script type="text/javascript">
  let name;
  name = window.prompt("Please enter your name");
  document.writeln("<h1> Hello " + name + ", ");
  document.writeln("<br> Welcome to JavaScript World! </h1>");
</script>
```

Fourth statement

- Considering the third and fourth statements together, it can be interpreted by the browser as a normal HTML code and displayed as what you see in the example.
- On completing the third line of code, the text `<h1> Hello James,` has been written to the document. The fourth line of code writes the final html text as `
 Welcome to JavaScript World! </h1>`, which collectively be seen by the browser as:

`<h1> Hello James,
 Welcome to JavaScript World! </h1>`

Third statement:

- This statement calls the document object's `writeln` method to write a line of HTML5 markup in the HTML5 document.
- The `+` operator is for `string concatenation` that enables a string and a value of another data type (including another string) to be combined.
- A `string` is indicated by `double quotation marks(")`.
- In this example, if the value of the variable "name" is "James", the written line of the third statement will show:

`<h1> Hello James,`

It would then be rendered in the client's window if viewed in a web browser.

6.5 JavaScript elements

Let
If; if...else; switch;
For
While
Do...while
Function

6.5 JavaScript elements

Web Page



'let' vs 'var' variable declaration types

The keyword **var** also exists in the language but due to poor data typing and scope controls (i.e., a variable declared with **var** can be accessed from anywhere in the JavaScript modules, making it open for corruption). It is therefore encouraged as good programming practice to use **let**. For example:

- a variable can be defined using **var** but the type of data it can hold can be changed:

```
var z = "Jane Doe"; // Declared with a 'string' data type
z = 3;              // Changed to an 'integer' data type -- allowed with a 'var'
```

'const' declarations

The final core variable declaration type in JavaScript is **const**. Unlike the *let* and *var* types, with a variable declared as a **const**, its value can't change (making it read-only). For example:

- a variable can be defined using **const** but must be initialised with a value and later cannot be changed:

```
const rate = 4.55; // Declared with as a 'float' data type
rate = 3.13;      // **ERROR** Can't change a 'const' value
```

Syntax – Control Structures

JavaScript provides three control structures.

- The **if** statement either performs (selects) an action if a condition is **true**, or skips the action if the condition is **false**.
- The **if ... else** statement performs an action if a condition is **true** and performs a different action if the condition is **false**.
- The **switch** statement performs one of many different actions, depending on the value of an expression.

if statement example structure

```
if (value > 10)
{
    ...
}
else if (value > 5)
{
    ...
}
else
{
    ...
}
```

JavaScript elements

switch statement structure (for multiple conditions)

```
switch (value)
{
    case 1:
        ... ;
        break;
    case 2:
        ... ;
        break;
    ...
    default:           // Important to have a 'default' check
        ... ;
}
```

Syntax – while Statement

Allows the programmer to specify that an action is to be repeated while some condition remains true.

The following is an example:

```
let counter = 0;
while (counter < 10)
{
    counter ++;
    ...
}
```

Syntax – do ... while Statement

- tests the loop-continuation condition after the loop body executes
- the loop body always executes at least once.

The following is a simple example:

```
let counter = 0;
do
{
    counter ++;
    ...
}
while (counter < 10)
```

Syntax - Functions

When writing Javascript code it is always good programming practice to break the code down into sections of code that may be reused in later parts of the code. To support this in Javascript functions are used to contain a set of code which can be called (invoked) from various locations in the program.

The format of a function definition is:

```
function function-name ( parameter-list ) {
    ... // JavaScript statements
    ...
    return value;
}
```

The *function-name* is defined by the programmer (similar to a variable name) which can be called from another location in the code. When calling a function, it is possible to pass data which the function can work on, this is achieved through the parameter-list. The body of the function are the various Javascript statements to do the 'work' of the function. Finally, each function should return, this can either be with or without some result. The function returns to the point in the code where the function was called.

6.6 JavaScript arrays and statement example

- An array is a special variable, which can hold **more than one value at a time**.
- Each value is stored in an element of the array and identified by the index of the array. Array indexes start with 0, i.e. the first element index is [0], the second element index is [1], and so on.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JavaScript Array Example</h2>
    <p>Here is the list of fruits in an array:</p>

    <div id="display"></div>

    <script>
      let fruits, outputText;

      fruits = ["Apple", "Banana", "Mango", "Orange", "Pear"];

      outputText = "<ul>";

      for (let i = 0; i < fruits.length; i++)
      {
        outputText += "<li>" + fruits[i] + "</li>";
      }

      outputText += "</ul>";

      document.getElementById("display").innerHTML = outputText;
    </script>
  </body>
</html>
```

The first statement: `let fruits, outputText;`
declares four variables to be used later.

The second statement: `fruits = ["Apple", "Banana", "Mango", "Orange", "Pear"];`
It creates an array which is assigned to the **variable** `fruits`.
So, the variable **fruits** is now an array.

The third statement:
The variable **outputText** in the third statement is to store the HTML code that will be generated by the JavaScript code.

The fourth statement:
Since we are going to list the fruits in an **unordered** list, the tag `` is therefore assigned to the variable `outputText` as the initial value.

JavaScript arrays and statement example Cont.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JavaScript Array Example</h2>
    <p>Here is the list of fruits in an array:</p>

    <div id="display"></div>

    <script>
      let fruits, outputText;

      fruits = ["Apple", "Banana", "Mango", "Orange", "Pear"];
      outputText = "<ul>";

      for (let i = 0; i < fruits.length; i++)
      {
        outputText += "<li>" + fruits[i] + "</li>";
      }

      outputText += "</ul>";

      document.getElementById("display").innerHTML = outputText;
    </script>
  </body>
</html>
```

This page will display as follows when it is opened in a browser:

JavaScript Array Example

Here is the list of fruits in an array:

- Apple
- Banana
- Mango
- Orange
- Pear

The **for-loop** statement:

- Picks up the array values one by one, using the local variable **i** as the iterator in the loop, from 0 to the number of elements in the array **fruits.length**.
- For each picked up value, it generates a list item HTML code by using the string concatenation operator +, and append it to the existing HTML code in the variable **outputText**.
- Please note the array value is picked up by indicating its array index, ie **fruits[i]**.
- The statement **outputText += "";** appends the closing unordered list tag to the existing HTML code generated and stored by the **for-loop**.
- So, the HTML code stored in **outputText** is ready to be displayed by the browser.

The last statement:

`document.getElementById("display").innerHTML = outputText;`
Displays the HTML code that is stored in the variable **outputText**.



JavaScript elements

6.5 Javascript elements

Elements	W3schools	Content
Comments	JS Comments	single-line comment: multi-line comments
Variables	JS Variables	let var const nothing
Arrays	JS Arrays/JS Arrays methods	pop push shift unshift length
Assignment operators	JavaScript Assignment	= += -= *= /=
Operators	JavaScript Operators/Arithmetic Operators	Addition Subtraction Multiplication Division
	JavaScript Operators/Comparison Operators	Equal to Not equal to Less than Less than or equal to Greater than Greater than or equal to
	JavaScript Operators/ Logical Operators	logical and logical or logical not
Control Structures	JS If Else	if else else if switch
for Statement	JS loop for	for
while Statement	JS loop while	while do while
Functions	JS Fuctions	fuctions

[HTML](#)
[CSS](#)
[JAVASCRIPT](#)
[SQL](#)
[PYTHON](#)
[PHP](#)
[BOOTSTRAP](#)
[HOW TO](#)
[W3.CSS](#)
[JAVA](#)
[JQUERY](#)
[C](#)
[C++](#)
[C#](#)
[R](#)

JS Tutorial

[JS HOME](#)
[JS Introduction](#)
[JS Where To](#)
[JS Output](#)
[JS Statements](#)
[JS Syntax](#)
[JS Comments](#)
[JS Variables](#)
[JS Let](#)
[JS Const](#)
[JS Operators](#)
[JS Arithmetic](#)
[JS Assignment](#)
[JS Data Types](#)
[JS Functions](#)
[JS Objects](#)
[JS Events](#)
[JS Strings](#)
[JS String Methods](#)
[JS String Search](#)
[JS String Templates](#)
[JS Numbers](#)
[JS Number Methods](#)
[JS Arrays](#)
[JS Array Methods](#)
[JS Array Sort](#)
[JS Array Iteration](#)
[JS Array Const](#)
[JS Dates](#)
[JS Date Formats](#)
[JS Date Get Methods](#)
[JS Date Set Methods](#)
[JS Math](#)

[Instant Grammar Checker](#)
[Try Now](#)

JavaScript Tutorial

[← Home](#)
[Next →](#)

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

JavaScript is easy to learn.

This tutorial will teach you JavaScript from basic to advanced.

[Start learning JavaScript now »](#)

Examples in Each Chapter

With our "Try it Yourself" editor, you can edit the source code and view the result.

Example

My First JavaScript

[Click me to display Date and Time](#)

Please go to [W3schools.com- JS Tutorial](#).

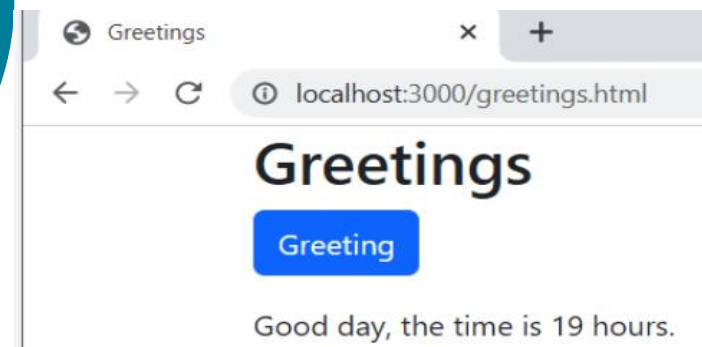
Use the left table and find out support concepts.

All the basic elements can be found.

You can use W3schools website to learn extra elements.



6.7 JavaScript condition and function example



- The example uses a predefined object in JavaScript, **Date**, to get the current hour data of your computer, and uses the condition control: **if-statement**, to decide what greeting message should be displayed.
- The **Date** object is created by the constructor **Date()** via the command **new Date()**.
- Then this created object uses the Date method **getHours()**, which is also predefined in JavaScript, to obtain the current time of your computer.
- Please note these operations are defined in the function, **myFunction()**, which means these operations will not be performed, unless the function is called somewhere in the web page.
- In this example, this function **myFunction()** is called by an inline JavaScript on the button element: `<button id="greetingButton">Greeting </button>`
- If we add an event handler by looking for the button element in the DOM to catch when the user clicks this button, we can set the action to call the function **myFunction()** to perform the operations defined in the function. As a result, a greeting message will display depending on your current computer time.
- If the user does not click the button, nothing will happen.

```
<body>
  <div class="container">
    <h1>Greetings</h1>

    <button id="greetingButton" class="btn btn-primary mb-3"> Greeting </button>

    <p id="greeting"></p>
  </div>

  <script>
    function myFunction() {
      console.log("Greetings Button Clicked!");
      let greeting;
      let time = new Date().getHours();

      if (time < 10) {
        greeting = "Good morning";
      } else if (time < 20) {
        greeting = "Good day";
      } else {
        greeting = "Good evening";
      }

      greeting += `, the time is ${time} hours.`;

      let myGreeting = document.getElementById("greeting");
      myGreeting.innerHTML = greeting;
    };

    let myButton = document.getElementById("greetingButton");
    myButton.onclick = () => {
      myFunction();
    };
  </script>
</body>
```


JavaScript Get Date Methods

Date Get Methods

Method	Description
getFullYear()	Get year as a four digit number (yyyy)
getMonth()	Get month as a number (0-11)
getDate()	Get day as a number (1-31)
getDay()	Get weekday as a number (0-6)
getHours()	Get hour (0-23)
getMinutes()	Get minute (0-59)
getSeconds()	Get second (0-59)
getMilliseconds()	Get millisecond (0-999)
getTime()	Get time (milliseconds since January 1, 1970)

Note 1

The get methods above return **Local time**.

Universal time (UTC) is documented at the bottom of this page.

Note 2

The get methods return information from existing date objects.

In a date object, the time is static. The "clock" is not "running".

The time in a date object is NOT the same as current time.

Please look at the [GET](#) methods
We will apply them next week!



6.8 JavaScript switch statement example

- **Switch statements** are used to perform different actions based on different conditions.
- The switch statement selects one of many blocks of code to execute depending on different conditions.
- The following example shows how the **switch** statement works.

The JavaScript Switch Statement

Use the **switch** statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {  
  case x:  
    // code block  
    break;  
  case y:  
    // code block  
    break;  
  default:  
    // code block  
}
```

```
<html>  
<body>  
  <p id="display"></p>  
  <script>  
    let day;  
    switch(new Date().getDay()) {  
      case 0:  
        day = "Sunday";  
        break;  
      case 1:  
        day = "Monday";  
        break;  
      case 2:  
        day = "Tuesday";  
        break;  
      case 3:  
        day = "Wednesday";  
        break;  
      case 4:  
        day = "Thursday";  
        break;  
      case 5:  
        day = "Friday";  
        break;  
      case 6:  
        day = "Saturday";  
    }  
    document.getElementById("display").innerHTML = "Today is " + day;  
  </script>  
</body>  
</html>
```

When this page is opened in a browser, you will see the day name like:

Today is Wednesday

This example uses the JavaScript object, **Date**, to get the day information of your computer (using **getDay()** method) and shows today's name accordingly.

After getting the day information via **new Date().getDay()**, the **switch** statement evaluates day information once via **switch(new Date().getDay())**, and compares it with the value of each case. If there is a match, the associated block of code is executed.

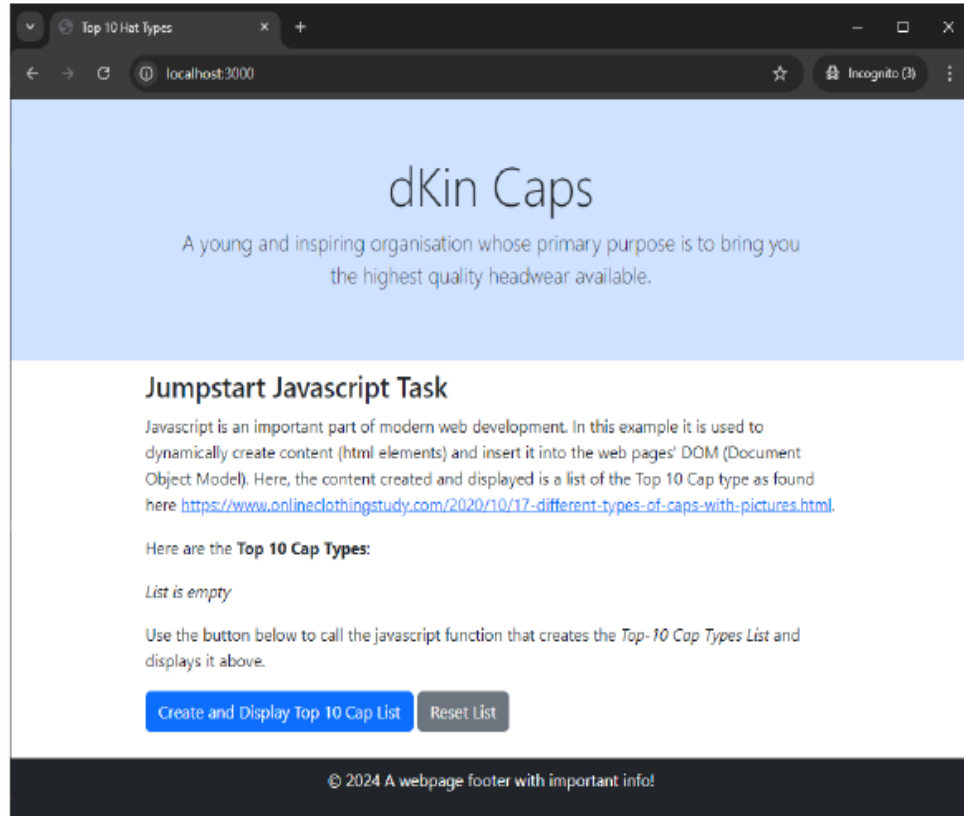
The **getDay()** method returns the weekday as a number between 0 and 6:

- Sunday=0
- Monday=1
- Tuesday=2
- ...

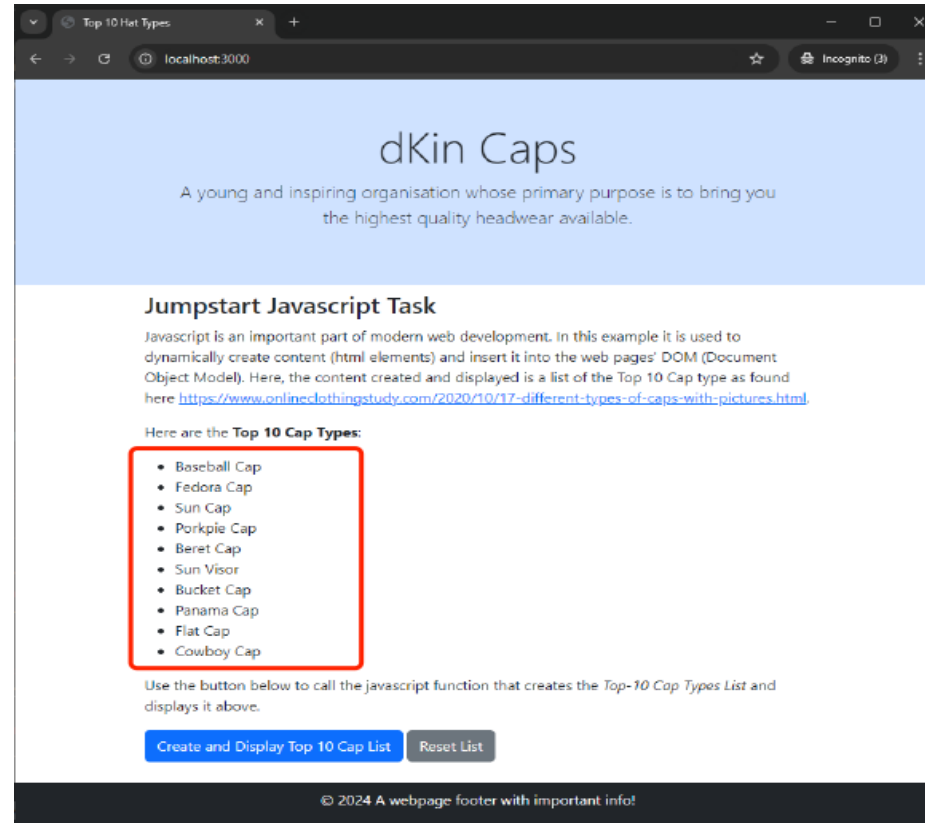
Ontrack task 6.1P: Jumpstart JavaScript

In this task, you are required to create **an unordered list** that is built dynamically and contains the Top 10 Cap Styles as found on the web. This list should be created when the user **clicks a button**.

A sample structure for the page (**before and after**) is shown below



Task6.1.1 Jumpstart Javascript - Cap List is empty



Task6.1.2 Jumpstart Javascript - Top 10 Cap List is populated



Requirements & Submission

Requirements:

```
function buildCapList() {
```

1. To complete this tasks, you are asked to create a **javascript function** called **buildCapList()** that you will add to a file called **CapList.js**.
2. We have provided you with the frontend HTML code (using the bootstrap framework) that you can use as a basis of this project. The html file can be found in the resources tab in this task's OnTrack page (located in bottom right hand corner of task-page)
3. You will also be required to add a javascript function called **removeCapList()** to delete/remove the list of caps, which is assigned to the Reset List button on the page.

```
function removeCapList() {
```

Hints

1. In the JavaScript function that builds the **unordered list** of popular cap types from an array of strings. You will need to **create an array of strings** (a string for each of the cap type names).
2. The **div** block within the webpage called **top_cap_list** is the section that you will be replacing with new HTML content, i.e., the newly created unordered list.
3. To replace the existing content, you must first **remove the current content** (initially a paragraph saying List is empty but can be previously added list of caps) by using the **removeChild()** method.
4. You can then add the cap list as a child of the **top_cap_list** div using the **appendChild()** method.
5. The fragment of javascript that defines the array holding the **10 cap types** is given below:

```
// List drawn from https://www.onlineclothingstudy.com/2020/10/17-different-types-of-caps-with-pictures.html
const capArray = [
  "Baseball Cap",
  "Fedora Cap",
  "Sun Cap",
  "Porkpie Cap",
  "Beret Cap",
  "Sun Visor",
  "Bucket Cap",
  "Panama Cap",
  "Flat Cap",
  "Cowboy Cap"
];
```

NOTE: In your browser's console (from the developers->inspect tool), check to see if there are any errors displayed from your page/javascript, especially in the case where you hit the *Create List* button a second time. This may require you to look at how you create/remove the existing content (or nodes) from the DOM.

You should submit:

- Screen-shot of the web page showing the output **before** the button has been clicked.
- Screen-shot of the web page showing the output of **after** the button clicked.
- The HTML file of the webpage task6-1.html
- The new JavaScript file CapList.js

Hints:

- Creating/Removing HTML Elements (Nodes/Child Nodes): https://www.w3schools.com/js/js_htmlDOM_nodes.asp
- JavaScript while Loop: https://www.w3schools.com/js/js_loop_while.asp
- JavaScript For Of: https://www.w3schools.com/js/js_loop_forof.asp

How to do

Task Details

6.1P Jumpstart JavaScript

Learn about JavaScript

Pass Task

Start Date — Aim to start this task by Mon 12 Aug.

Due Date — Aim to complete by Fri 23 Aug.

↓ Task Sheet

↓ Resources

1. Ontrack task 6.1 – Task details – Resources
2. Download “6.1P-resources.zip”
3. Open the task 6.1.html
4. Right check the **source code**

```
<!doctype html>
<html lang="en">

<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH" crossorigin="anonymous">

  <!-- You will need to create this new JS file and place it in
  the same directory as this HTML file (i.e., public_html), add it here -->

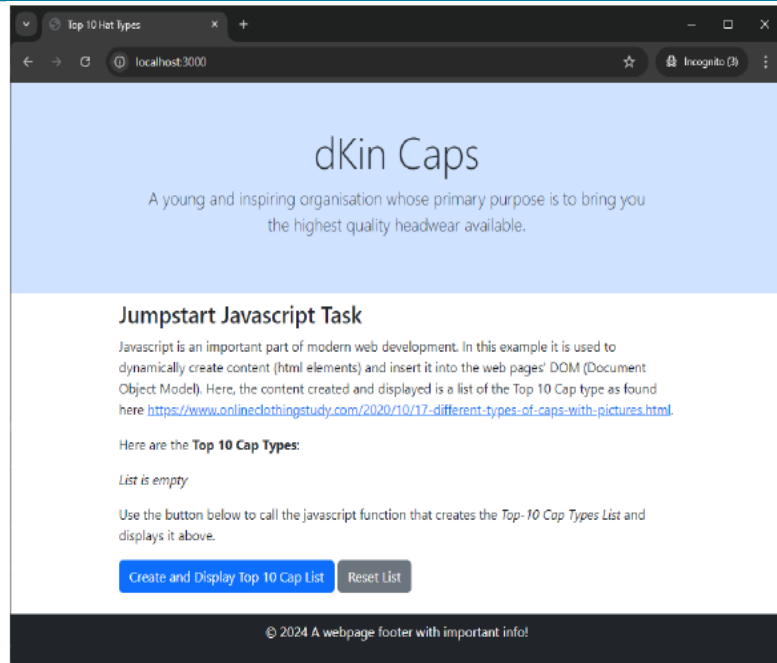
  <title>Top 10 Hat Types</title>
</head>

<body>
  <header>
    <div class="container-fluid bg-primary-subtle">
      <div class="col-sm-8 mx-auto text-center py-5">
        <h1 class="display-4">dKin Caps</h1>
        <p class="lead">A young and inspiring organisation whose
          primary purpose is to bring you the highest quality headwear available.</p>
      </div>
    </div>
  </header>

  <main>
    <div class="container">
      <h3 class="mt-2">Jumpstart Javascript Task</h3>
      <p>
        Javascript is an important part of modern web development. In this example
        it is used to dynamically create content (html elements) and insert it into the web pages' DOM (Document Object Model).
        Here, the content created and displayed is a list of the Top 10 Cap type as found here
        <a href="https://www.onlineclothingstudy.com/2020/10/17-different-types-of-caps-with-pictures.html"
          target="_blank">https://www.onlineclothingstudy.com/2020/10/17-different-types-of-caps-with-pictures.html</a>.
      </p>
      <p>
        Here are the <strong>Top 10 Cap Types</strong>:
      </p>
    </div>
  </main>
</body>
</html>
```

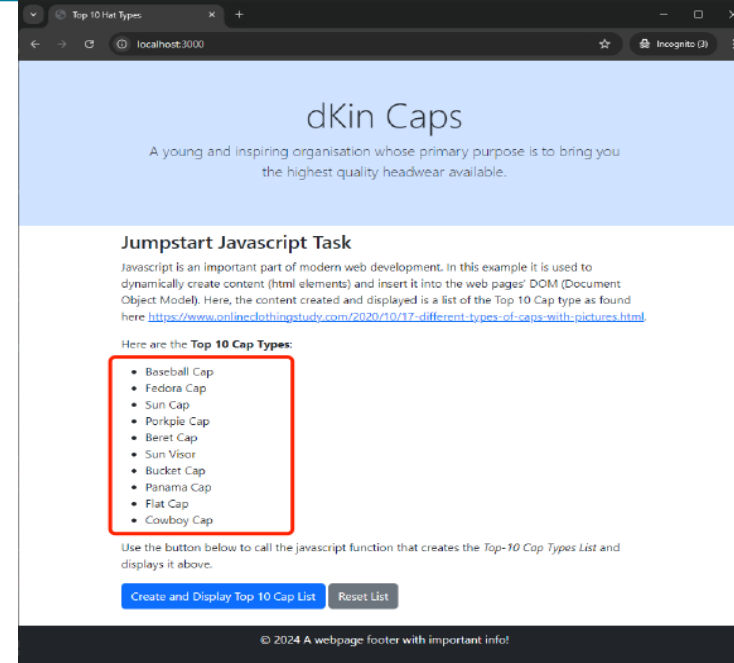
Example for task 6.1

Before

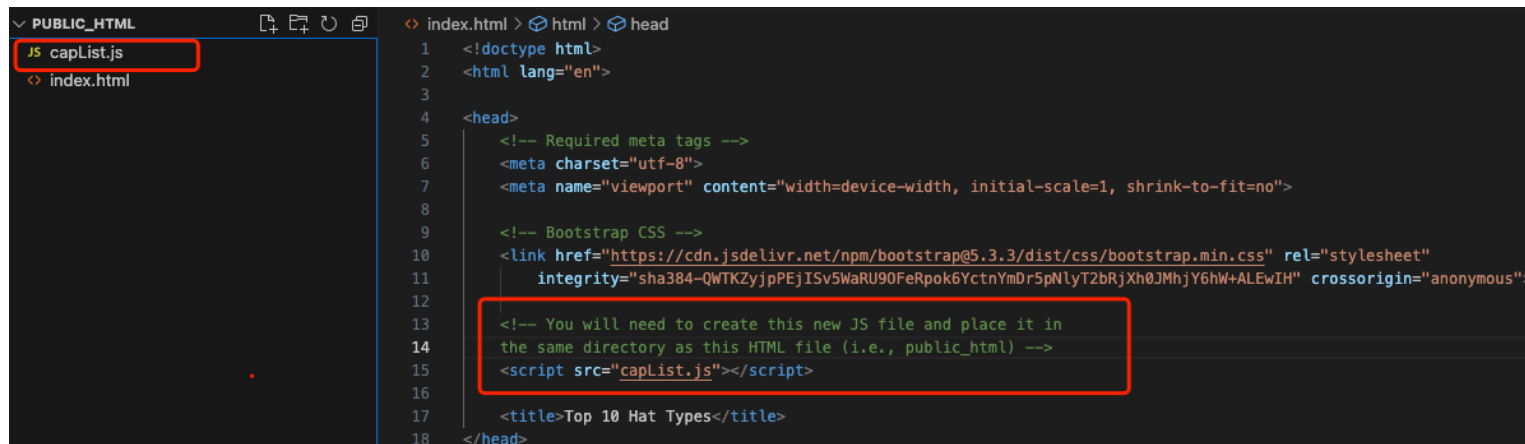


Task6.1.1 Jumpstart Javascript - Cap List is empty

After



Task6.1.2 Jumpstart Javascript - Top 10 Cap List is populated



Ontrack task 6.2C: JavaScript Loop

- In this task, you are asked to construct a Cap Wearing Planner in the form of a **table**, that is dynamically created based on (valid) **input from the user** entered through a **simple form**.
- You are provided with a **starting web page**, that contains:
 - **a form (two fields)**
 - **a list** that is already populated with one rating.

The sample starting page should look like this:

The screenshot shows a web browser window with the title 'Cap Wearing Planner' and the URL 'localhost:3000'. The page has a light blue header with the text 'dKin Caps' and a subtitle 'A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.' Below the header, the main content area is titled 'Javascript Loops: Cap Wearing Planner'. It contains a paragraph explaining the purpose of the planner and two bullet points listing constraints: 'the number of weeks to display should be between 1 and a maximum of 52' and 'the number of caps to display should be between 1 and a maximum of 10'. Below this, there is a section titled 'Input Fields' with two text input fields: 'How many weeks in the schedule?' and 'How many caps in the schedule?'. Below the input fields are two buttons: 'Create Cap Schedule' (blue) and 'Reset Cap Schedule' (grey). Below the buttons is a table with 8 columns: 'Week #', 'Day #1', 'Day #2', 'Day #3', 'Day #4', 'Day #5', 'Day #6', and 'Day #7'. The table is currently empty, and a message 'The plan is currently empty!' is displayed below it. At the bottom of the page, there is a footer with the text '© 2024 Ice Cream made chilled!'.

Cap Wearing Planner

localhost:3000

dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

Javascript Loops: Cap Wearing Planner

A web page can provide helpful services to users, as well as various ways to present complex information. Here, we want to provide a user with a planner for wearing their special caps. By asking for the number of weeks the plan should cover, and the number of caps to cycle through, a table is generated showing the sequence of caps to wear on which day of the week. **NOTE:** The user should only be able to input valid numbers for the input fields. For example, the following constraints should be applied:

- the number of weeks to display should be between 1 and a maximum of 52
- the number of caps to display should be between 1 and a maximum of 10

With an error being displayed if it is outside these bounds.

Input Fields

How many **weeks** in the schedule?

How many **caps** in the schedule?

Create Cap Schedule Reset Cap Schedule

Week #	Day #1	Day #2	Day #3	Day #4	Day #5	Day #6	Day #7
The plan is currently empty!							

© 2024 Ice Cream made chilled!

Task6.2.1: Initial page view

Requirements

The form accepts from the user two number values, the `Number of weeks` and the `Number of caps`, which relate to how long the planner should be, and the number of caps to cycle through, respectively. With this information a table with 8-columns is created that shows:

1. The week number
2. The type of cap to wear on that day of the week (cycling through the number of caps owned: maximum 10)
3. Highlighting the alternate cap sequences with those cells having a grey (`table-active`) and white backgrounds.

The following screenshots show the result of three example inputs. The first is a *3 week plan, with 5 caps*:

• the number of caps to display should be between 1 and a maximum of 10
With an error being displayed if it is outside these bounds.

Input Fields

How many **weeks** in the schedule? How many **caps** in the schedule?

[Create Cap Schedule](#) [Reset Cap Schedule](#)

Week #	Day #1	Day #2	Day #3	Day #4	Day #5	Day #6	Day #7
1	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap	Beret Cap	Baseball Cap	Fedora Cap
2	Sun Cap	Porkpie Cap	Beret Cap	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap
3	Beret Cap	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap	Beret Cap	Baseball Cap

© 2024 Ice Cream made chilled!

Task6.2.2: Input of 3 week plan, with 5 caps

Requirements

- The next is a 9 week plan, with 9 caps:

Input Fields

How many **weeks** in the schedule?

How many **caps** in the schedule?

Create Cap Schedule

Reset Cap Schedule

Week #	Day #1	Day #2	Day #3	Day #4	Day #5	Day #6	Day #7
1	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap	Beret Cap	Sun Visor	Bucket Cap
2	Panama Cap	Flat Cap	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap	Beret Cap
3	Sun Visor	Bucket Cap	Panama Cap	Flat Cap	Baseball Cap	Fedora Cap	Sun Cap
4	Porkpie Cap	Beret Cap	Sun Visor	Bucket Cap	Panama Cap	Flat Cap	Baseball Cap
5	Fedora Cap	Sun Cap	Porkpie Cap	Beret Cap	Sun Visor	Bucket Cap	Panama Cap
6	Flat Cap	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap	Beret Cap	Sun Visor
7	Bucket Cap	Panama Cap	Flat Cap	Baseball Cap	Fedora Cap	Sun Cap	Porkpie Cap
8	Beret Cap	Sun Visor	Bucket Cap	Panama Cap	Flat Cap	Baseball Cap	Fedora Cap
9	Sun Cap	Porkpie Cap	Beret Cap	Sun Visor	Bucket Cap	Panama Cap	Flat Cap

© 2024 Ice Cream made chilled!

- And the final screenshot shows a 1 week plan, with 2 caps:

be able to input valid numbers for the input fields. For example, the following constraints should be applied:

- the number of weeks to display should be between 1 and a maximum of 52
- the number of caps to display should be between 1 and a maximum of 10

With an error being displayed if it is outside these bounds.

Input Fields

How many **weeks** in the schedule?

How many **caps** in the schedule?

Create Cap Schedule

Reset Cap Schedule

Week #	Day #1	Day #2	Day #3	Day #4	Day #5	Day #6	Day #7
1	Baseball Cap	Fedora Cap	Baseball Cap	Fedora Cap	Baseball Cap	Fedora Cap	Baseball Cap

© 2024 Ice Cream made chilled!

Task6.2.5 Error message on invalid rating value

Task6.2.2: Input of 3 week plan, with 5 caps



Requirements

- In completing this task, the code used to dynamically **create (extend) the table** should be allocated within a javascript function.
- This function is called when the user clicks the Create Cap Schedule button.
- In adding to the list, the javascript code should take and process the input fields from the form (weekCountInput and capCountInput).
- The **week** count input **must be check** to be **within the 1..52 range** and if it is outside this range **an error message provided**.
- Similarly, the **cap** count input must be check to be within the **1..10 range** and if it is outside this range **an error message provided**.
- In the case of an error case for **either value**, **return the user to the input form** (return in the javascript function).

localhost:3000 says
Error: Number of WEEKS must be between 1 and 52 (value 66 entered)

OK

Javascript Loops: Cap Wearing Planner

A web page can provide helpful services to users, as well as various ways to present complex information. Here, we want to provide a user with a planner for wearing their special caps. By asking for the number of weeks the plan should cover, and the number of caps to cycle through, a table is generated showing the sequence of caps to wear on which day of the week. **NOTE:** The user should only be able to input valid numbers for the input fields. For example, the following constraints should be applied:

- the number of weeks to display should be between 1 and a maximum of 52
- the number of caps to display should be between 1 and a maximum of 10

With an error being displayed if it is outside these bounds.

Input Fields

How many **weeks** in the schedule?

How many **caps** in the schedule?

Create Cap Schedule Reset Cap Schedule

Week #	Day #1	Day #2	Day #3	Day #4	Day #5	Day #6	Day #7
The plan is empty...							

© 2024 Ice Cream made chilled!

KIN UNIVERSITY

Hints

In completing this task consider the following points:

- Include the array of cap names from Task 6.1P (list of 10 caps)
- Build your code in stages, i.e., get and check input from the user; find the fields within the HTML document that need to be updated (e.g., the ID `cap-planner-table-body`)
- The `input` fields of the form can be extracted using DOM commands such as
 - `let weekCountMax = parseInt(document.getElementById("weekCountInput").value);`
 - `let capCountMax = parseInt(document.getElementById("capCountInput").value);`
- The `weekCountMax` value inputted by the user should be checked to see it is **invalid** (i.e., `!weekCountMax` or `< 1` or `> 52`) and if it is an error message should be provided (using `window.alert()`) then returning from the function.
- The `capCountMax` value inputted by the user should be checked to see it is **invalid** (i.e., `!capCountMax` or `< 1` or `> 10`) and if it is an error message should be provided (using `window.alert()`) then returning from the function.
- It may be useful to have nested double loop to construct your table. Outer loop creates a row for each of the number of weeks, while the inner creates a table cell for each the 7 days.
- Keep track of the *cell count* and use the *modulus* operator to see if cycles of `capCountMax` have been reached before toggling between *greyed* and *white* backgrounds.
- Online references include:
 - (https://www.w3schools.com/jsref/dom_obj_table.asp)
 - (https://www.w3schools.com/js/js_arithmetic.asp) Modulus (remainder) operator

What will you submit?

You should submit:

- Screen-shot of the web page showing the output for 5 added valid ratings (total of 6 names/rating in list).
- Screen-shot of the page when an 'invalid' rating is entered (i.e., a rating of `-1` or `99`) and the prompt warning window is presented.
- The webpage's HTML file.
- The file with javascript code used to build the table.

```
// List drawn from https://www.onlineclothingstudy.com/2020/10/17-different-types-of-caps-with-pictures.html
const capArray = [
  "Baseball Cap",
  "Fedora Cap",
  "Sun Cap",
  "Porkpie Cap",
  "Beret Cap",
  "Sun Visor",
  "Bucket Cap",
  "Panama Cap",
  "Flat Cap",
  "Cowboy Cap"
];
```

NOTE: In your browser's console (from the developers->inspect tool), check to see if there are any errors displayed from your page/javascript, especially in the case where you hit the *Create List* button a second time. This may require you to look at how you create/remove the existing content (or nodes) from the DOM.

The Array of cap names are same as task 6.1P

1. Const capArray

2. **function resetCapPlan():** reset the table , clean all content, and show :”the plan is empty...”

➤ `document.getElementById("cap-planner-table-body")` - retrieves a specific element from the page

➤ `createElement` and `appendChild` - used to create new HTML elements (like `<tr>`, `<td>`, `<th>`) and append them to the page

3. **function buildCapPlan():** generates a table that displays the cap-wearing schedule for each week.

for Loops: In the `buildCapPlan` function, two nested for loops iterate through the number of weeks and days to dynamically generate the cap-wearing schedule for each week.

if Conditional Statements: These are used to validate user input (like range checks for `weekCountMax` and `capCountMax`) and to control whether a cell should be highlighted

```
function buildCapPlan() {  
  let capTable = document.getElementById("cap-planner-table-body");  
  
  let weekCountMax = parseInt(document.getElementById("weekCountInput").value);  
  if( !weekCountMax || weekCountMax < 1 || weekCountMax > 52 ) {  
    alert(`Error: Number of WEEKS must be between 1 and 52 (value ${weekCountMax} entered)`);  
    return;  
  }  
}
```

Example of `weekcountMax`,
please write the `capCountMax` by yourself



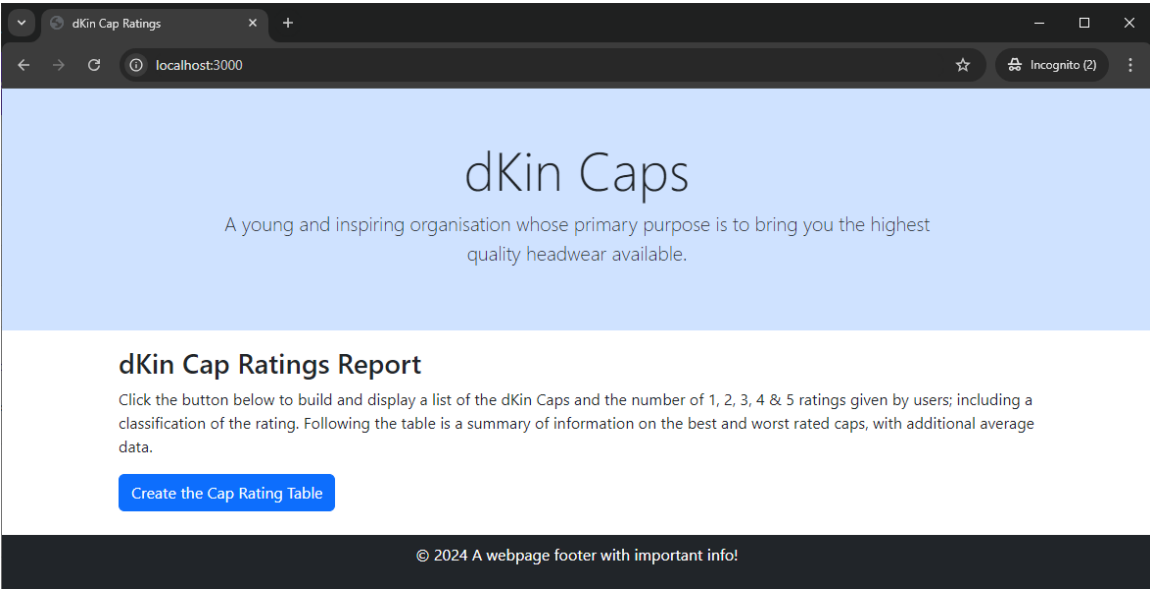
Ontrack task 6.3D: Conditions and Function Tasks

In this task you are asked to construct a table showing *Ratings Information* of customers of the ten **dKin Caps** types.

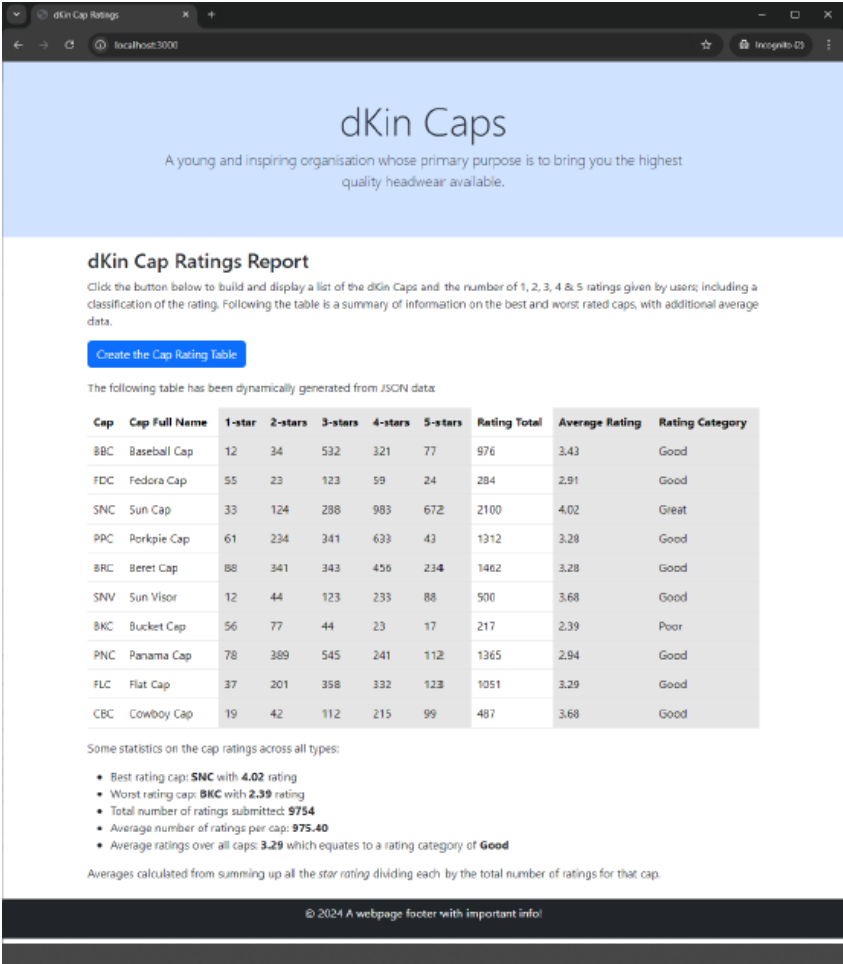
You will be provided with some raw data for the set cap ratings, which includes fields such as an abbreviation of the Cap type, the number of 1, 2, 3, 4, & 5 star ratings the cap has received.

In completing this task, the code used to dynamically build the table should be allocated within a javascript function. This function is called when the user clicks a button.

The initial webpage should appear as shown below:



When the user clicks the button to generate the table, it should appear as shown in the following two images below:



Task6.3.2: List of Cap Ratings



Ontrack task 6.3D: Requirements

You are provided with the raw data for the cap ratings in the form of a JSON object, that contains an array of `capratings`. This JSON object is shown below:

```
const capRatingListJSON = {  
  capratings: [  
    { cap:"BBC", stars: [12,34,532,321,77] },  
    { cap:"FDC", stars: [55,23,123,59,24] },  
    { cap:"SNC", stars: [33,124,288,983,672] },  
    { cap:"PPC", stars: [61,234,341,633,43] },  
    { cap:"BRC", stars: [88,341,343,456,234] },  
    { cap:"SNV", stars: [12,44,123,233,88] },  
    { cap:"BKC", stars: [56,77,44,23,17] },  
    { cap:"PNC", stars: [78,389,545,241,112] },  
    { cap:"FLC", stars: [37,201,358,332,123] },  
    { cap:"CBC", stars: [19,42,112,215,99] }  
  ]  
};
```

```
const capArray = [  
  "Baseball Cap", // BBC  
  "Fedora Cap", // FDC  
  "Sun Cap", // SNC  
  "Porkpie Cap", // PPC  
  "Beret Cap", // BRC  
  "Sun Visor", // SNV  
  "Bucket Cap", // BKC  
  "Panama Cap", // PNC  
  "Flat Cap", // FLC  
  "Cowboy Cap" // CBC  
];
```

For example, the **Cap Name** is provided in the data in an abbreviated form, which needs to be presented in the second column in full-form, i.e., BBC should be presented as Baseball cap. A javascript function that takes the *abbreviated capname* as input and returns the full name as a string could be used. An example of such a function outline is shown below:

```
function fullCapName( abbreviatedCapName ) {  
  // Should use the input 'abbreviatedCapName' and return a string  
  // with the full name (HINT: a switch() statement)  
}
```

In addition the **Ratings Total**, **Average Rating** and **Rating Category** fields are not provided in the data so needs to be constructed and displayed in the final columns.

For example, a javascript function should be developed that takes a respective cap's *stars* array as input and returns a *ratings total* could be used. An example of such a function outline is shown below:

```
function calcCapRatingTotal( capRatingsArray ) {  
  // Should use the input array and return the sum of the number of ratings submitted for that cap.  
}
```


Ontrack task 6.3D: Requirements Cont.

Furthermore, a similar function could be developed that returns the *average cap rating*:

```
function calcCapAverageRating( capRatingsArray ) {  
    // Should use the input array and return the average rating based on the caps' ratings.  
    // Use the formula:  
    //  
    // avgrating = (ratings[0] * 1.0 + ratings[1] * 2.0  
    //             + ratings[2] * 3.0 + ratings[3] * 4.0  
    //             + ratings[4] * 5.0) / calcCapRatingTotal(ratings)  
}
```

The final column in the table is **Rating Category** which is dependent on the *cap's rating* value. The following table details how the activity should be chosen:

Cap Rating Range	Category Name
0 <= capRating < 2.5	'Poor'
2.5 <= capRating < 4.0	'Good'
4.0 <= capRating	'Great'

In calculating the Cap Category field, a javascript function should be developed that takes as input the user rating and using a cascading if / then / else if condition checks, returns the text (string) for the appropriate category from the table above, as shown in the code fragment below:

```
function capCategory( capRating ) {  
    // Should use the input 'capRating' and return a string  
    // with the full name (HINT: a cascading if/then/else if set of statements)  
}
```

Minimum / Maximum / Average Summary

The final section of the dynamically created table are the fields to show the various interesting features from the data. This includes:

- The cap type with the *best* rating and it's value
- The cap type with the *worst* rating and it's value
- The total (sum) of all ratings provided across all cap types
- The average number of ratings provided per cap
- The ice cream type with the least number sold and from which state
- The calculated *average cap rating* and the *average rating category* that this would equate to. This information should be displayed below the table.

Ontrack task 6.3D: Hints and Submission

Hints:

- Use the JSON array of objects provided to store the Cap Rating details and information.
- Define Javascript functions that builds and populates the table (and summary information) when the button is clicked, i.e., implements the `buildCapRatingList()` function. `function buildCapRatingList() {`
- Define a set support Javascript functions to return the *Full Cap Name*, *Total Number of Ratings*, *Average Cap Rating* and the *Average Cap rating category*.
- Use a table to organize the displayed data.
- Use additional variables to keep a total of the *total cap ratings submitted* and the *index* of the best & worst rated caps; these can be updated in the loop as you build the html to display the cap's row.

What will you submit?

You should submit:

- Screen-shot(s) of the web page showing the final table.
- HTML base file
- The javascript code to construct the table.

Good luck with your Week 6 Ontrack tasks!

