ReDI SCHOOL

Functions and DOM manipulation

Week 4

ReDI

Agenda

1. Homework Review
2. Continuation functions
3. DOM manipulation

Homework Review

I. You and your family are very busy today and don't have time to cook for dinner. You decided to order some food, but your partner and your kid have a different favourite restaurants.

   a. Write a function that takes 3 arguments, the name of the restaurant, food, and the amount.
   b. Outputs the result to your console with the text : You are ordering (amount) (food) from (name of the restaurant)
   c. call the function 3 times with different values for the argument

Functions

There are plenty of items in the real world that work as a function.

Imagine if you had to built a blender from scratch every time you want to make a smoothie

Functions

There are 3 main steps to use functions:

1. Declare the function
2. Pass arguments (optional step)
3. Call the function!

Functions

Declaring a function

To declare a function, you use the function keyword, followed by the function name, a list of parameters, and the function body as follows:

```
function sayHiToUser() {
  alert('Hello');
}
```

Functions

Declaring a function

The function name must be a valid JavaScript identifier. By convention, the function names are in **camelCase** and start with verbs like **getData**(), **fetchContents**(), and **isValid**().

A function can accept zero, one, or multiple parameters. In the case of multiple parameters, you need to use a comma to separate two parameters.

The following declares a function say() that accepts no parameter:

Functions

Declaring a function

The following declares a function **sayHiToUser**() that accepts no parameter:

```
function sayHiToUser() {
  alert('Hello');
}
```

Functions

Declaring a function

Inside the function body, you can write the code to implement an action. For example, the following say() function simply shows a message as an alert:

```
function sayHiToUser() {
  alert('Hello');
}
```

Functions

Passing arguments

The following declares a function named **square**() that accepts one parameter:

```
function square(a) {
    a * a;
}
```

Functions

Passing arguments

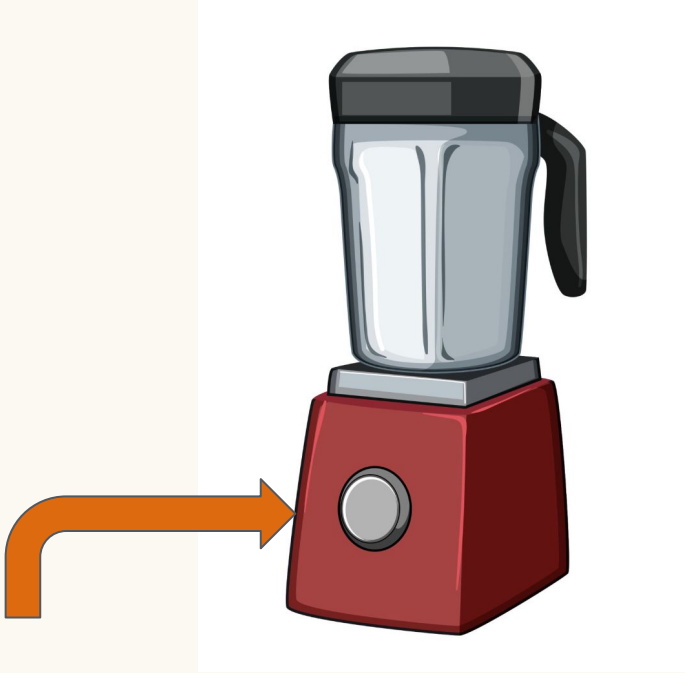
And the following declares a function named **add**() that accepts two parameters:

```
function add(a, b) {
  a + b;
}
```

Functions

Calling a function!

To use a function, you need to call it. It’s like pressing the button on the blender!



Functions

Calling a function!

To call a function, you use its name followed by arguments enclosing in parentheses like this:

```
sayHiToUser();
```

Functions

Calling a function!

If the function is expecting some parameters we need to pass them as arguments when we call it:

```
square(3);
```

```
add(2, 3);
```

Functions

Using a returned value

The purpose of the functions most of the time is to reuse the outcome of running some code. Every function in JavaScript implicitly returns undefined unless you specify a return value

```
function square(a) {
  return a * a;
}
```

Functions

Using a returned value

Now I can use the returned value. OBS: Think about the order in which the lines of code are executed

```
function square(a) {
  return a * a;
}

const doubledPrice = square(25);

console.log(doubledPrice);
```

Functions

Arrow functions

New Syntax

```
const convertToDogYears = (dogAge) => {
  const dogAgeInDogTime = dogAge * 7;
  console.log(`Your dog is ${dogAgeInDogTime} years old in dog years!`);
  return dogAgeInDogTime;
};
```

Old Syntax

```
function convertToDogYears2(dogAge) {
  const dogAgeInDogTime = dogAge * 7;
  console.log(`Your dog is ${dogAgeInDogTime} years old in dog years!`);
  return dogAgeInDogTime;
}
```

Functions

Main differences

- No *arguments object* in arrow functions
- Arrow functions do not create their own *this* binding
- Arrow functions *cannot* be accessed before initialization
- Arrow functions can have *implicit returns*

Functions

Arrow functions

Implicit return:

```
const convertToDogYearsShort = (dogAge) => dogAge * 7;
```

Explicit return:

```
const convertToDogYears = (dogAge) => {
  return dogAge * 7;
};
```

Regular function return:

```
function convertToDogYears2(dogAge) {
  return dogAge * 7;
}
```

Exercise

10 minutes

A. Transform the function used for your homework from a regular function to an arrow function

B. Try to write the shortest version possible

```
function orderFood(restaurantName, foodAmount, foodName) {
  return `You are ordering ${foodAmount} ${foodName} from  ${restaurantName}`;
}
```

Break

ReDI SCHOOL

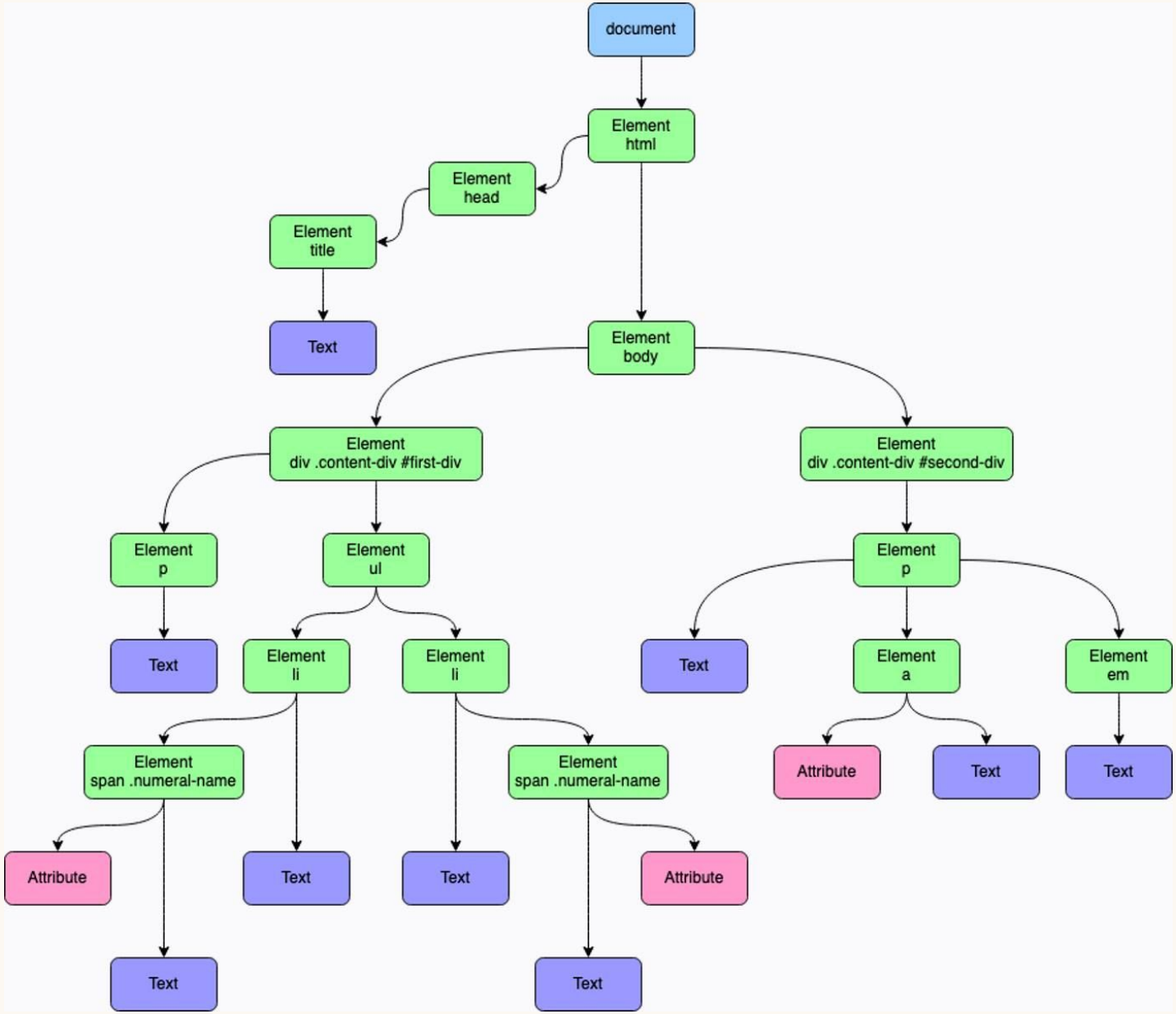
DOM manipulation

What is the DOM?

It stands for Document Object Model and in simple words is the representation of an HTML document in our browser.

The browser builds this representation in a ‘tree’ shape to easily manipulate the hierarchy of the HTML elements

DOM manipulation



DOM manipulation

Accessing the DOM

We can ‘manipulate’ or navigate this tree using JavaScript functions!

You can find elements, change their content, style them, or even add new elements via JavaScript. This makes possible for us to interact with web content, create dynamic websites, and build cool online experiences!

Imagine if an e-commerce had to update its HTML everytime a new product is added or sold out

DOM manipulation

ReDI SCHOOL

Using ID

```
// HTML:
// <div id="myElement">Hello, DOM!</div>

  const elementById = document.getElementById('myElement');
  console.log(elementById.textContent); // Outputs: Hello, DOM!
```

Using class name

```
// <div class="myClass">First Element</div>
// <div class="myClass">Second Element</div>
  const elementsByClass = document.getElementsByClassName('myClass');
  console.log(elementsByClass.length); // Outputs: 2
```

DOM manipulation

Using tag name

```
// HTML:
// <p>Paragraph 1</p>
// <p>Paragraph 2</p>

  const paragraphs = document.getElementsByTagName('p');
  console.log(paragraphs.length); // Outputs: 2
```

Using CSS Selectors

```
// HTML:
// <ul>
//   <li class="list-item" id="first-item">Item 1</li>
//   <li class="list-item" >Item 2</li>
// </ul>
const listItem = document.querySelector('.list-item');
const firstListItem = document.querySelector('#first-item');
console.log(listItem.textContent); // Outputs: Item 1
```

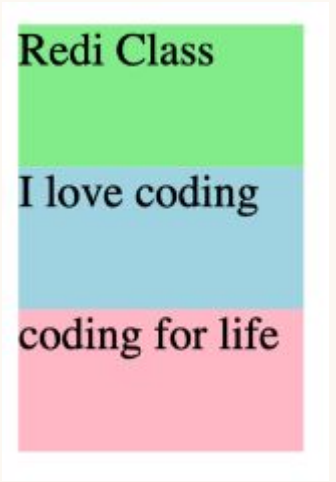
DOM manipulation

Get all elements using CSS Selectors

```
// HTML:
// <ul>
//   <li class="list-item">Item 1</li>
//   <li class="list-item">Item 2</li>
//   <li class="list-item">Item 3</li>
// </ul>
const listItems = document.querySelectorAll('.list-item');
  console.log(listItems.length);
```

Exercise

20 minutes



A. Create 3 html element divs
    a. Add some text to each of them
    b. One div should have an id
    c. One div should have a class attribute
    d. One div has none of the above
    e. Give them color and
B. Try accessing the divs using the methods explained before and log their text content

DOM manipulation

ReDI SCHOOL

Change HTML content & attributes

```
// Modifying text content
heading.textContent = 'New Heading Text';

// Modifying HTML content
paragraph.innerHTML = '<strong>New</strong> paragraph content';

// Modifying element attributes
const btn = document.getElementById('changeTextBtn');
btn.setAttribute('disabled', true);
```

DOM manipulation

Add items to the DOM

```
// Creating a new element
const newParagraph = document.createElement('p');
newParagraph.textContent = 'This is a new paragraph.';

// Appending the new element
heading.appendChild(newParagraph);
```

DOM manipulation

Remove items from the DOM

```
// Removing an element
const oldParagraph = document.querySelector('p');
oldParagraph.remove();
```

DOM manipulation

Change styling from JS: style

In JavaScript, you can access and modify CSS styles of DOM elements using the style property. This property provides access to inline styles (styles defined directly on the element).

```
// HTML:
// <div id="myDiv">Hello, CSS!</div>

  const myDiv = document.getElementById('myDiv');
  myDiv.style.color = 'blue'; // Change text color to blue
  myDiv.style.fontSize = '20px'; // Change font size to 20 pixels
```

DOM manipulation

Change styling from JS: add/remove classes

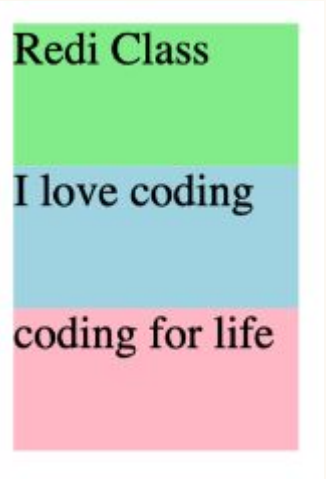
You can add or remove individual classes using classList.add and classList.remove.

```
// HTML:
// <div id="myDiv" class="originalClass">Hello, CSS!</div>
// <button onclick="addClass()">Add Class</button>
// <button onclick="removeClass()">Remove Class</button>


function addClass() {
  const myDiv = document.getElementById('myDiv');
  myDiv.classList.add('additionalClass');
}

function removeClass() {
  const myDiv = document.getElementById('myDiv');
  myDiv.classList.remove('originalClass');
}
```

Exercise



20 minutes

A. Use the HTML content you created in the exercise above
    a. Add a new div to the bottom with black background
    b. Remove the first div
    c. Change the background color of the middle div
    d. Hide one of the divs using a class name and visibility: hidden

Hand-in assignment

https://github.com/ReDISchoolDK/Spring25_Frontend/blob/main/Week-04_Functions-DOM/assignment-01/README.md