Webtechnologies

WALKING THE DOM

https://unsplash.com/photos/person-walking-across-a-crosswalk-with-luggage-w172j9FO-gA?utm_source=unsplash&utm_medium=referral&utm_content=creditShareLink

FUNCTIONS — EXERCISE 14

- Solve little function exercises
- Download files from: https://short.jonasscheiwiller.ch/ex14

```
// ============================================================
// EXERCISE: Functions & Parameters
// ============================================================
//
// Open the browser console to see your results.
// (Right-click the page → Inspect → Console)
//
// ============================================================

// Formats a price with a currency symbol
function formatPrice(amount, currency) {
  console.log(`${currency} ${amount.toFixed(2)}`);
}

formatPrice(9.9, "CHF"); // → CHF 9.90
formatPrice(4.5, "EUR"); // → EUR 4.50

// ============================================================

// TODO 1: call formatPrice() with a price of your choice

// TODO 2: define a function called discount
//         it takes two parameters: price and percent
//         it should log the reduced price
//         EXAMPLE: discount(100, 20) → "Price after discount: CHF 80.00"

// TODO 3: call discount() with two different examples

```

FUNCTIONS → PARAMETERS

- Problem with lots of params?

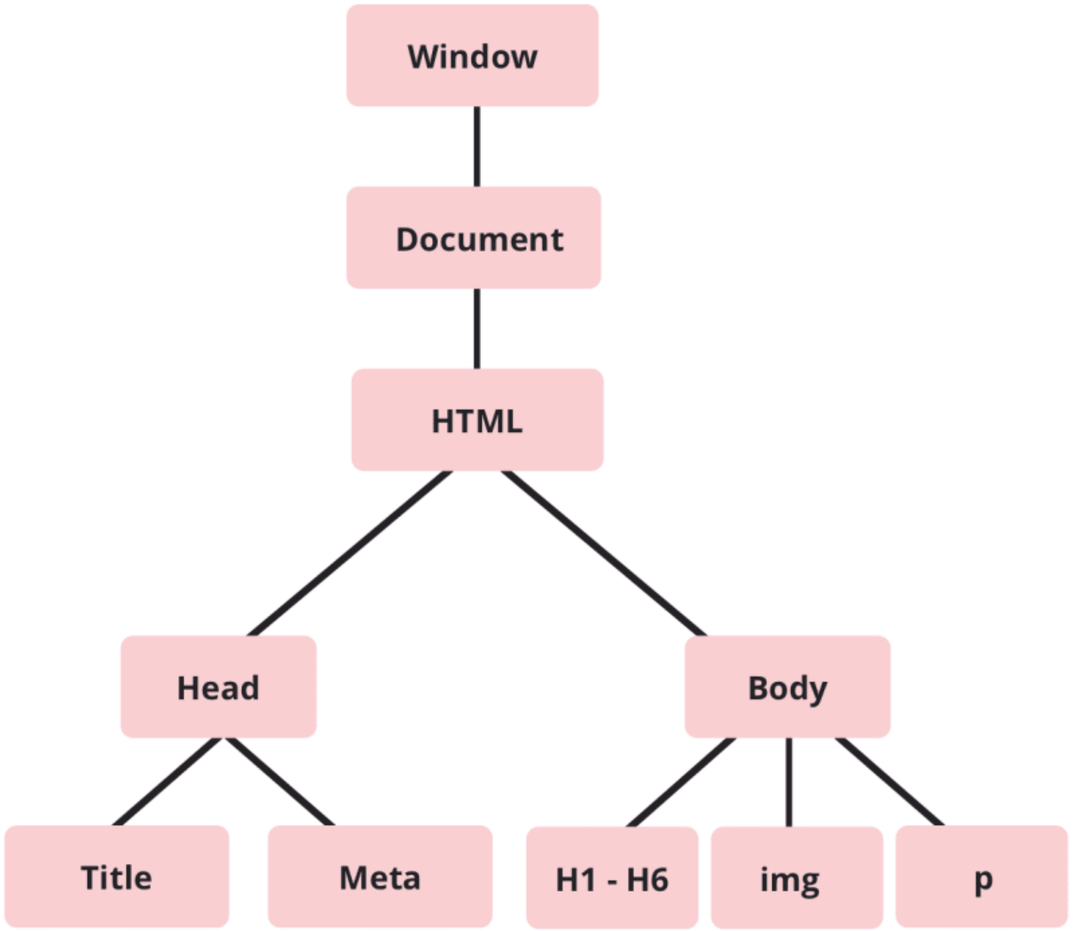
```
// Define a function with multiple parameters
function createCity(inhabitants, foundingYear, hoursOfSunperYear, name, location, country, continent, annualBudget) {
  // Perform some complex operations using the parameters
  console.log(`Inhabitants: ${inhabitants}`);
  console.log(`Founding Year: ${foundingYear}`);
  console.log(`Hours of Light per Year: ${hoursOfSunperYear}`);
  console.log(`Name: ${name}`);
  console.log(`Location: ${location}`);
  console.log(`Country: ${country}`);
  console.log(`Continent: ${continent}`);
  console.log(`Annual Budget: ${annualBudget}`);
}

// Call the function with arguments
createCity(15461681, 1850, 2000, "Springfield", { latitude: 37.220165, longitude: -95.699059}, "United States", "North America", 5000000000);

```

FUNCTIONS → PARAMETERS

- Better to declare with objects if you have lots of params
- Order does not matter anymore
- Params are «named»

```javascript
// Define a function with an object parameter
function printPerson(person) {
  console.log(`Name: ${person.name}`);
  console.log(`Age: ${person.age}`);
  console.log(`City: ${person.city}`);
}

// Call the function with an object argument
printPerson({
  name: "John",
  age: 25,
  city: "New York"
});
```

THE DOM



Document Object Model (DOM) Tree

JAVASCRIPT & THE DOM

```html
<div id="elem">
  <div id="elem-content">Element</div>
</div>

<script>
  // get the element
  let elem = document.getElementById('elem');

  // make its background red
  elem.style.background = 'red';
</script>
```

JAVASCRIPT & THE DOM

querySelectorAll

By far, the most versatile method, `elem.querySelectorAll(css)` returns all elements inside `elem` matching the given CSS selector.

Here we look for all `<li>` elements that are last children:

```
<ul>
  <li>The</li>
  <li>test</li>
</ul>
<ul>
  <li>has</li>
  <li>passed</li>
</ul>
<script>
  let elements = document.querySelectorAll('ul > li:last-child');

  for (let elem of elements) {
    alert(elem.innerHTML); // "test", "passed"
  }
</script>
```

querySelector

The call to `elem.querySelector(css)` returns the first element for the given CSS selector.

In other words, the result is the same as `elem.querySelectorAll(css)[0]`, but the latter is looking for *all* elements and picking one, while `elem.querySelector` just looks for one. So it's faster and also shorter to write.

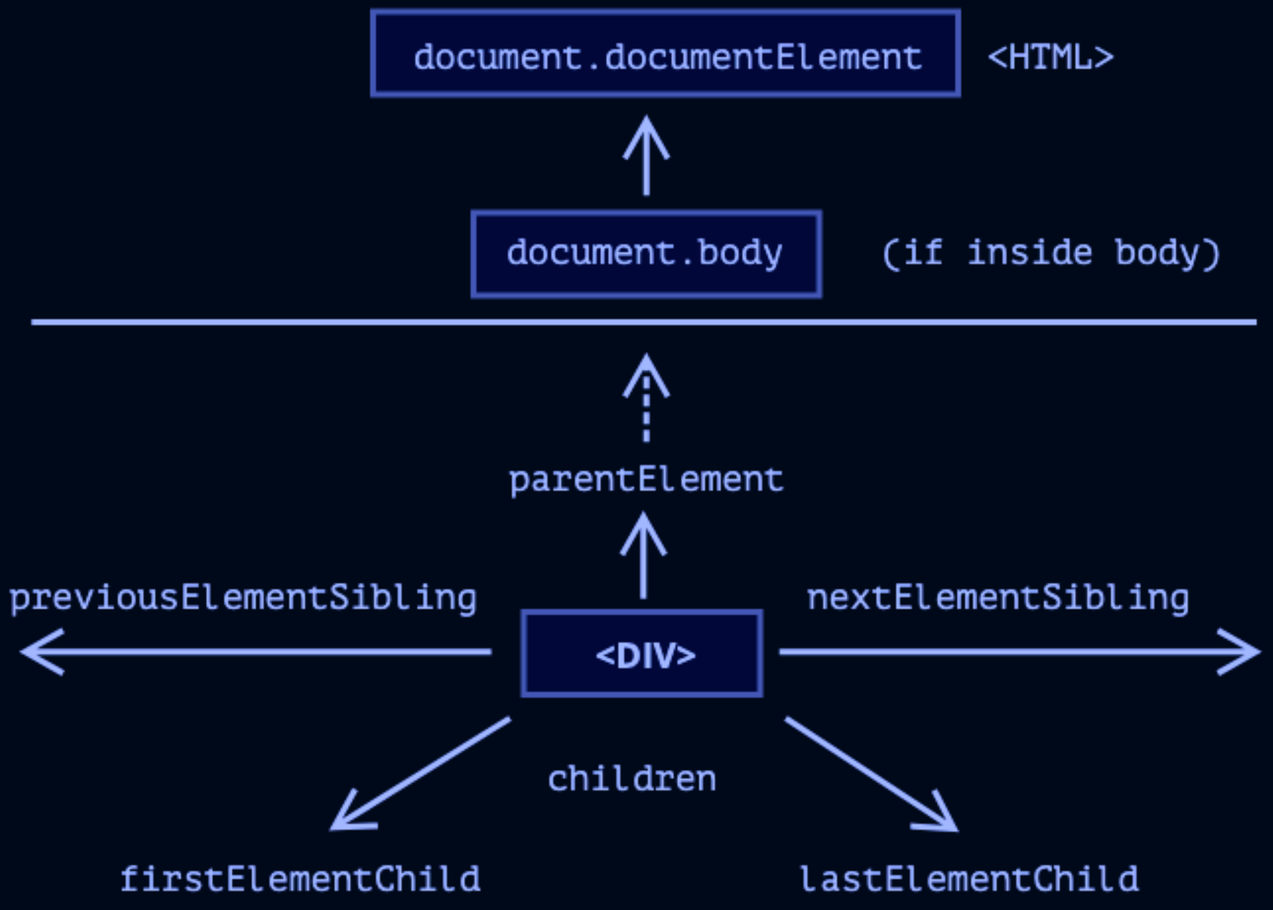
SELECTING — EXERCISE 15

- Solve little exercises
- Download files from: https://short.jonasscheiwiller.ch/ex15

```
// ============================================================
// EXERCISE: Selecting Elements in the DOM
// ============================================================
//
// Open the browser console to see your results.
// (Right-click the page → Inspect → Console)
//
// ============================================================


// TODO 1: select the <h1> by its id and log it
const title = /* ??? */;
console.log(title);


// TODO 2: select the first .recipe div using querySelector and log it
const firstRecipe = /* ??? */;
console.log(firstRecipe);


// TODO 3: select ALL .recipe divs using querySelectorAll and log them
const allRecipes = /* ??? */;
console.log(allRecipes);


// TODO 4: log how many recipes are on the page
// HINT: querySelectorAll returns a NodeList — it also has .length
console.log(/* ??? */);


// TODO 5: loop over allRecipes and log the textContent of each .recipe-title
// HINT: inside the loop, use querySelector on the current element
// to find the .recipe-title inside it


// TODO 6: select the recipe with data-category="dessert" and log it
//
//         WHY data-attributes instead of classes?
//         ---------------------------------------
//         Classes are for styling — they tell CSS how something looks.
//         If you rename or remove a class for visual reasons, your JS breaks.
//         data-attributes are purely for JavaScript and carry no visual meaning,
//         so CSS and JS stay independent and don't step on each other.
//         This is called separation of concerns.
//
//         HINT: querySelector can target data-attributes like CSS:
//         document.querySelector('[data-category="..."]')
const dessert = /* ??? */;
console.log(dessert);
```

“WALKING THE DOM”

- Parents
- Siblings
- Children



https://javascript.info/dom-navigation

“WALKING THE DOM”: PARENTS

- Parents
- Siblings
- Children

```
1  const element = document.querySelector('[data-js="element"]');
2  const parent = element.parentNode;
3  console.log("parent: ", parent);
```

“WALKING THE DOM”: SIBLINGS

- ⚠️ nextElementSibling vs. nextSibling
- nextSibling selects also text nodes or even comment nodes

```
1  const element = document.querySelector('[data-js="box"]');
2  const sibling = element.nextElementSibling;
3  console.log("sibling: ", sibling);
```

“WALKING THE DOM”: CHILDREN

- firstElementChild & lastElementChild
- children

```
1  const element = document.querySelector('[data-js="box"]');
2  const children = element.children;
3  console.log("children: ", children);
```

https://javascript.info/dom-navigation

WALKING THE DOM — EXERCISE 16

- Solve little exercises
- Download files from: https://short.jonasscheiwiller.ch/ex16

- Help: https://javascript.info/dom-navigation (found in comment)

```
// ============================================================
// EXERCISE: Walking the DOM
// ============================================================
//
// Open the browser console to see your results.
// (Right-click the page → Inspect → Console)
//
// The HTML looks like this:
//
//   <ul data-nav="menu">      ← parent
//     <li data-item="first">  ← first child
//     <li data-item="second"> ← middle child
//     <li data-item="third">  ← last child
//   </ul>
//
// ============================================================

const menu = document.querySelector('[data-nav="menu"]');
const second = document.querySelector('[data-item="second"]');

// HINT: https://javascript.info/dom-navigation#element-only-navigation

// TODO 1: log all direct children of the menu
console.log("all children:" /* ??? */);

// TODO 2: log just the first child element of the menu
console.log("first child:" /* ??? */);

// TODO 3: log just the last child element of the menu
console.log("last child:" /* ??? */);

// TODO 4: starting from the second item, log its next sibling
console.log("next sibling:" /* ??? */);

// TODO 5: starting from the second item, log its previous sibling
console.log("previous sibling:" /* ??? */);

// TODO 6: starting from the second item, log its parent element
console.log("parent:" /* ??? */);

// TODO 7: log how many children the menu has
console.log("number of children:" /* ??? */);

```