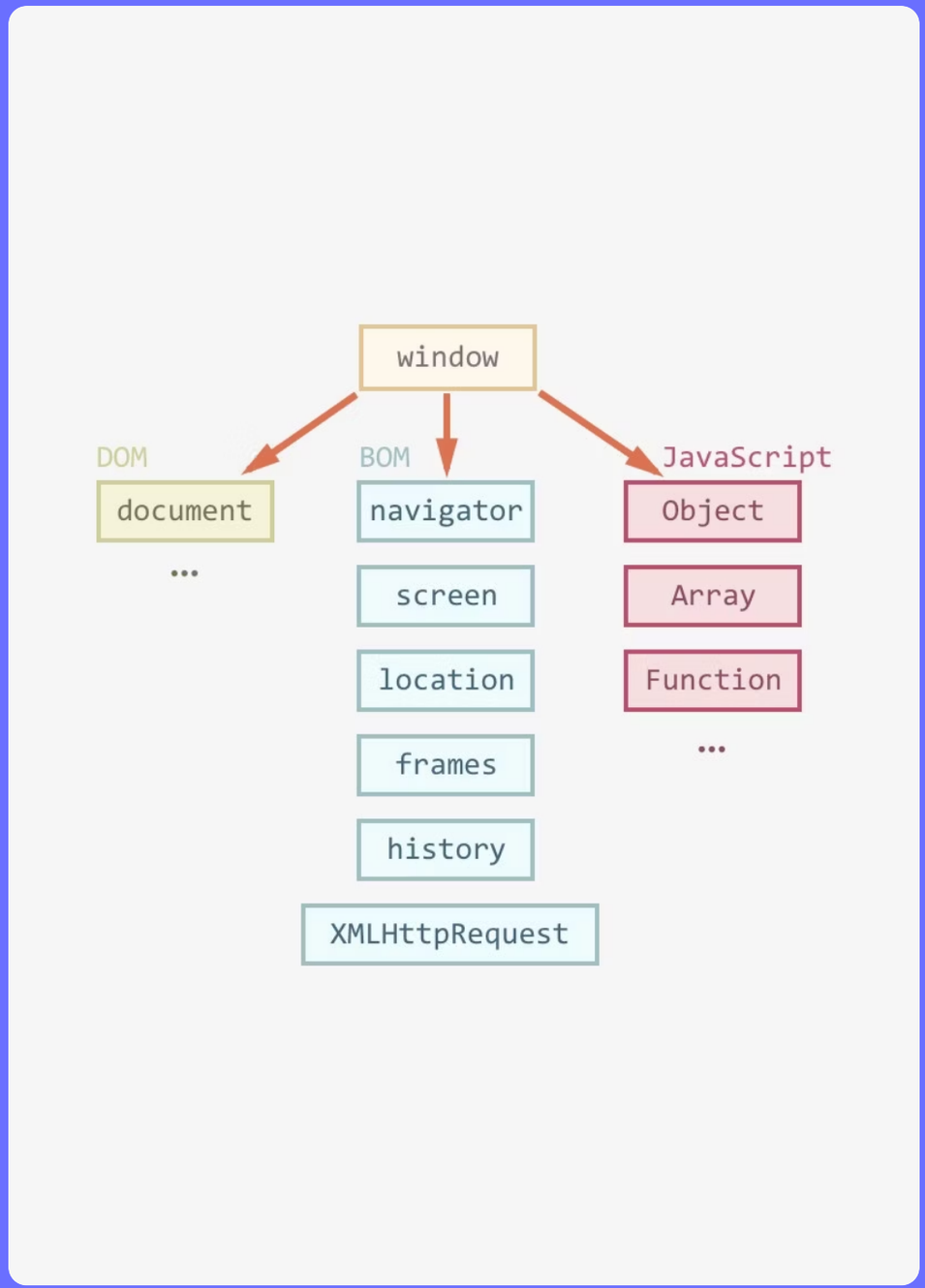
When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The document object represents your web page.

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page



HTML DOM Document

The **document** object is the owner of all other objects in your web page.

Accessing any element in an HTML page, you always start with accessing the document object.

DOM Methods & Properties

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

In the DOM, all HTML elements are defined as **objects**.

Finding HTML Elements

`document.getElementById(` *`id`* `)` - Find an element by element id

`document.getElementsByTagName(` *`name`* `)` - Find elements by tag name

`document.getElementsByClassName(` *`name`* `)` - Find elements by class name

`document.querySelector(CSS_selectors)` - Finding the first HTML Elements which matches with the CSS Selectors

`document.querySelectorAll()` - Finding all HTML Elements by CSS Selectors

Changing HTML Elements

`innerHTML` - The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

`element``.attribute_name =` *`new`* `attribute_name`

`element``.innerHTML =` *`new html content`*

`element``.style.`*`property = new style`* - Change the style of an HTML element

Adding and Deleting Elements

`document.createElement(` *`element`* `)` - Create an HTML element

`element.remove()` - Remove the html element itself

`document.removeChild(` *`element`* `)` - Remove an HTML element

`document.append(` *`element(s)`* `)` - Add HTML element(s)

`document.appendChild(` *`element`* `)` - Add an HTML element

`document.replaceChild(` *`new, old`* `)` - Replace an HTML element

`document.write(` *`text`* `)` - Write into the HTML output stream

Modify Text

`element.``innerText =` *`new html content`*

`element.``textContent =` *`new html content`*

Note: In element nodes, `innerText` evaluates `<br>` elements, while `textContent` evaluates control characters:

Modifying Elements Attributes & Classes

`element``.setAttribute`*`(attribute, value)`* - Sets the value of an attribute on the specified element. If the attribute already exists, the value is updated; otherwise a new attribute is added with the specified name and value

`element.removeAttribute(` *`attribute`* `)` Removes the attribute

`element.getAttribute(` *`attribute`* `)` returns the value of a specified attribute on the element

`element.c``lassList.add(classname)` - adds the class to the element

`element.c``lassList.remove(classname)` - removes the class to the element

`element.c``lassList.contains(classname)` - checking existing of the className in the element

`Traverse the DOM`

`element.parentNode` or `element.parentElement` - returns the parent

Note: Since the `<html>` element (`document.documentElement`) doesn't have a parent that is an element, `parentElement` is `null`

`element.childNodes` - returns child nodes (element nodes, text nodes, and comment nodes).

`element.children` - returns child elements (not text and comment nodes).

`element.firstChild` - returns the node's first child in the tree, or `null` if the node has no children

`element.lastChild` - returns the node's last child in the tree, or `null` if the node has no children

`element.firstElementChild` - returns an element's first child `Element`, or `null` if there are no child elements.

`element.nextSibling` - returns the node immediately following the specified one in their parent's `childNodes` or returns `null` if the specified node is the last child in the parent element.

`element.previousSibling` - returns the node immediately preceding the specified one in its parent's `childNodes` list, or `null` if the specified node is the first in that list.

Events

Here's a list of the most useful DOM events, just to take a look at:

Mouse events:

- `click` – when the mouse clicks on an element (touchscreen devices generate it on a tap).
- `contextmenu` – when the mouse right-clicks on an element.
- `mouseover` / `mouseout` – when the mouse cursor comes over / leaves an element.
- `mousedown` / `mouseup` – when the mouse button is pressed / released over an element.
- `mousemove` – when the mouse is moved.

Keyboard events:

- `keydown` and `keyup` – when a keyboard key is pressed and released.

Form element events:

- `submit` – when the visitor submits a `<form>`.
- `focus` – when the visitor focuses on an element, e.g. on an `<input>`.

Document events:

- `DOMContentLoaded` – when the HTML is loaded and processed, DOM is fully built.

CSS events:

- `transitionend` – when a CSS-animation finishes.

Event handlers

There are 3 ways to assign event handlers:

1. HTML attribute: `onclick="..."`.
2. DOM property: `elem.onclick = function`.
3. Methods: `elem.addEventListener(event, handler[, phase])` to add, `removeEventListener` to remove.

Examples:

`<input value="Click me" id="test" onclick="alert('Click!')" type="button">`

`document.getElementById(id).onclick = function(){code}` - Adding event handler code to an onclick event

`document.getElementById(id).addeventlistener("click",function(){code})` - Adding event handler code to an onclick event

To remove the handler, use `removeEventListener`:

```
element.removeEventListener(event, handler, [options]);
```

Event object

To properly handle an event we'd want to know more about what's happened. Not just a "click" or a "keydown", but what were the pointer coordinates? Which key was pressed? And so on.

When an event happens, the browser creates an *event object*, puts details into it and passes it as an argument to the handler.

Some properties of `event` object:

`event.type` Event type, here it's `"click"`.

`event.currentTarget` Element that handled the event. That's exactly the same as `this`, unless the handler is an arrow function, or its `this` is bound to something else, then we can get the element from `event.currentTarget`.

`event.clientX` / `event.clientY` - window-relative coordinates of the cursor, for pointer events.