W02 Readings

**Modifying the document**

There are two ways to create DOM nodes:

let div = document.createElement('div');

let textNode = document.createTextNode('Text');

**Insertion methods**

* node.append(nodes or strings) – append nodes or strings at the end of node,
* node.prepend(nodes or strings) – insert nodes or strings at the beginning of node,
* node.before(nodes or strings) – insert nodes or strings before node,
* node.after(nodes or strings) – insert nodes or strings after node,
* node.replaceWith(nodes or strings) – replaces node with the given nodes or strings.

**insertAdjacentHTML/Text/Element**

elem.insertAdjacentHTML(where, html)

elem.insertAdjacentText(where, text)

elem.insertAdjacentElement(where, elem)

The “html” inserts html, the “text” inserts text, and the “elem” inserts an element.

Use “insertAdjacentHTML” most of the time because it is shorter to write and more convenient.

There “where” specifies where to insert around or in an element. The “where” must be one of the following:

* "beforebegin" – insert html immediately before the element,
* "afterbegin" – insert html into the element, at the beginning,
* "beforeend" – insert html into the element, at the end,
* "afterend" – insert html immediately after the element.

**LocalStorage, sessionStorage**

**methods and properties:**

* setItem(key, value) – store key/value pair.
* getItem(key) – get the value by key.
* removeItem(key) – remove the key with its value.
* clear() – delete everything.
* key(index) – get the key on a given position.
* length – the number of stored items.

**sessionStorage**

SessionStorage is the same as localStorage but limited.

It will only keep storage onto one tab, and it will remain during page

refresh, but not when you close or open the tab.

**Storage event**

storage event triggers, with properties:

key – the key that was changed (null if .clear() is called).

oldValue – the old value (null if the key is newly added).

newValue – the new value (null if the key is removed).

url – the url of the document where the update happened.

storageArea – either localStorage or sessionStorage object where the update happened.

**Export and Import**

**Export before declarations**

Before declaring anything, you need to export first.

Example: export let number = 25

**Export apart from declarations**

You can also declare something, such as a function, before

exporting. You can also export above functions.

**Import**

You can put a list of what you want to import in curly braces.

Example: import {...}

If you have a lot you want to import, use:

import \* as <object>

**Import “as”**

You can use “as” to import an object with a different name. export has similar syntax.

Example:

import {name as newName}

**Export default**

Use “default” after “export” to export. This is a special syntax to make the “one thing per module” way look better.

Example:

export default class User {…}

Then import it without curly braces. Do this to make the imports look nicer. Import only needs curly braces for named exports, not for the default ones.

Named export:

export class User {…}

import {User} from …

Default export:

export default class User {…}

import User from …

You don’t have to give them names.

**The “default” name**

You can use the default keyword to reference the default export.

Example:

export {myFunctionName as default};

Keep in mind that using default won’t allow you to use the exact right name to import.

Imported variables should always correspond to file names.

Example:

import User from ‘path to file here’;

**Re-export**

The syntax for “Re-export" is: export … from …

Using this, you can quickly import and export things.

**Re-exporting the default export**

If you want to re-export the names and default exports, then use two statements.

export \* from 'file path';

export {default} from 'file path';