HTML Form Input Types	
Button	`<input type="button">`
Checkbox	`<input type="checkbox">`
Color Picker	`<input type="color">`
Date	`<input type="date">`
Local Date	`<input type="datetime-local">`
Email Address	`<input type="email">`
File Upload	`<input type="file">`
Image Upload	`<input type="image">`
Month	`<input type="month">`
Numeric Field	`<input type="number">`
Password Input	`<input type="password">`
Single Choice Select	`<input type="radio">`
Range (Default: 0-100)	`<input type="range">`
Reset Form	`<input type="reset">`
Search Field	`<input type="search">`
Submit Form	`<input type="submit">`
Phone Number	`<input type="tel">`
Text Field	`<input type="text">`
Time	`<input type="time">`
URL Input	`<input type="url">`
Week	`<input type="week">`

HTML SYNTAX

An **HTML tag** is composed of the name of the element, surrounded by angle brackets. An end tag also has a slash after the opening angle bracket, to distinguish it from the start tag.

Void elements do not have an end tag. A void element's behavior is predefined, and it cannot contain any content or other elements.

- example: `<br/> <img/> <link/>`

HTML attributes are specified inside the start tag. **HTML attributes** define desired behavior or indicate additional element properties. Most attributes require a *value*. In HTML, the value can be left unquoted if it does not include spaces (`attribute=value`), or it can be quoted with single or double quotes (`attribute='value' or attribute="value"`).

Global attributes are attributes common to all HTML elements; they can be used on all elements, though they may have no effect on some elements. Examples:

- `class`
- `id`
- `style`
- `title`
- `lang`

Boolean attributes, on the other hand, don't require a value to be specified. An example is the `checked` for checkboxes:

```
<input type=checkbox checked>
```

HTML is used to represent the structure or content of a document, its presentation remains the sole responsibility of **CSS style sheets**.

Behavior (interactivity) is also kept separate from content, and is handled by **scripts**.

- The elements `<style>` and `<script>`, with related HTML attributes, provide style sheets and scripts.

- In the document head, `<style />` and `<script />` may link to shared external documents, or `<style>...</style>` and `<script>...</script>` may contain embedded instructions.

Images are contained in separate graphics files, separate from text, though they can also be considered part of the content of a page.

- External image files are incorporated with the `<img />` or `<object />` elements.

`<meta/>` can be used to specify additional **metadata** (data that provides information about other data) about a document, such as its author, publication date, expiration date, language, page title, page description, keywords, or other information not provided through the other header elements and HTML attributes.

- In general, a meta element conveys hidden information about the document.
- Several meta tags can be used, all of which should be nested in the head element.
- The specific purpose of each <meta /> element is defined by its attributes.
- Multiple Meta elements with different attributes can be used on the same page

Block elements, or block-level elements, have a rectangular structure. By default, these elements will span the entire width of its parent element, and will thus not allow any other element to occupy the same horizontal space as it is placed on.

Web Storage API

When to use Web Storage API?

- can store more data than cookies
- ideal for storing multiple key-value pairs
- data can only be saved as a string
- with web storage, data stored on users machine
    - only accessible client side
- cookies can be read both server and client side

Common Uses for Web Storage

- shopping cart
- input data on forms
- user preferences

Common Cookie Uses

- user's identification info
- users session ID after login

Local Storage (`.localStorage`)

- stores data with no expiration date
- deleted when clearing the browser cache
- has max storage limit in browser
    - setting items with local storage

    localStorage.setItem(key, value)

Session Storage (`.sessionStorage`)

- stores data for a session (until browser window/tab is closed)
- never transfers data to server
- storage limit of 5MB (much bigger than cookie)

JSON

- JSON is just a string
- looks a lot like JS syntax
- invented by avid JS developer (Douglas Crockford)
- commonly used to replace XML
  - used to be the common format for data sent between computers
  - XML looks a lot like HTML
  - not as user friendly as JSON
- **serialization** is the process of converting data into a string (or some value like “binary”)
  - `JSON.stringify(data)`
- **deserialization** is the process of converting text into data
  - `JSON.parse(string)`

```
let array = [1, 'hello, "world"', 3.14, { id: 17 }];
let jsonArray = JSON.stringify(array);
console.log(jsonArray); // [1, "hello, \"world\"", 3.14, {"id":17}]

console.log(JSON.parse(jsonArray))// [1, 'hello, "world"', 3.14, { id: 17 }]
```

<u>JSON representations of JS literals</u>

- turns JS literal into a string representation
- `true` => `"true"`
- `null` => `"null"`
- `12.34` => `"12.34"`

String literals in JSON

- JSON always uses double quotes for strings
- interior quotation marks are escaped with `\`
- JSON requires strings to be on one line
    - represent new line with `\n`

JS		JSON
`'this is "text"'`	=>	`"this is \"text\""`

Array literals in JSON

- same as JS but wrapped in quotes
- `` `[1,2,3]` => `"[1,2,3]"` ``

Object literals in JSON

- keys in JSON must be surrounded by quotes and escaped
    - Javascript:

```
`{ person: true, name: "Alissa" }`
```

    - JSON:

```
`"{ \"person\": true, \"name\": \"Alissa\" }"`
```

kjfdls