CSE 154: Web Programming

Exam “Cheat Sheet”

HTML

Tags Used in the <head> Section

Tag	Description
`<title>` **text** `</title>`	title shown on page tab
`<meta` **attribute="value"** `... />`	page metadata
`<link href="`**url**`" rel="stylesheet" />`	links to a CSS style sheet
`<script src="`**url**`"></script>`	link to JavaScript code
`<!--` **comments** `-->`	comment (can appear in `head` or `body`)

Tags Used in the `<body>` Section

Tag	Display	Description
`<p>`**text** `</p>`	Block	paragraph

`<h1>`**text** `</h1>`
`<h2>`**text** `</h2>`
...
`<h6>`**text** `</h6>`
	Block	(h1 for largest to h6 for smallest)

`<hr />`	Block	horizontal rule (line)
` `	Inline	line break
`<a href="`**url**`">`**text** `</a>`	Inline	anchor (link)
`<img src="`**url**`" alt="`**description**`" />`	Inline-block	image
`<em>`**text**`</em>`	Inline	emphasis (italic)
`<strong>`**text** `</strong>`	Inline	strong emphasis (bold)

`<ol>`
`  <li>`**text** `</li>`
`  <li>`**text** `</li>`
`  <li>`
`  <ul>`
`  <li>`**nested item text**`</li>`
`  <li>`**nested item text**`</li>`
`  </ul>`
`  </li>`
`</ol>`
	Block	ordered (`ol`) and unordered (`ul`) list; list item (`li`)

Tags Used in the <body> Section (Continued)

Tag	Display	Description
<dl> <dt>**term 1** </dt> <dd>**description 1** </dd> <dt>**term 2** </dt> <dd>**description 2** </dd> </dl>	Block	definition list (dl); term (dt), and its description (dd)
<blockquote> **text** ... </blockquote>	Block	block-level quotation
<q>**text** </q>	Inline	inline-level quotation
<code>**text** </code>	Inline	computer code (monospace)
<pre>**text** </pre>	Block	preformatted **text** (preserves whitespace)
<table> <caption>**text** </caption> <tr> <th>**heading 1** </th> <th>**heading 2** </th> </tr> <tr> <td> cell 1 </td> <td> cell 2 </td> </tr> ... </table>	Block	table of data (table) description of table (caption) table row (tr) table heading cell (hr) normal table cell (td)
...	Block	block-level section of a page
...	Inline	inline-level section of a page

HTML5 Semantic Grouping Tags (all block elements)

Tag	Description
<header>	Container for a header of a document
<main>	Specifies the main content of a document. The content inside should be unique to the document and not contain content that is repeated across pages (e.g., sidebars, nav links, search bars, etc.)
<footer>	Container for a footer of a document
<article>	A standalone piece of content (e.g., entire blog post including title, author, etc.)
<section>	A piece of content that is part of another (e.g., a chapter section of a reading)
<aside>	Defines some content aside from the content it is placed in (e.g., a sidebar in an article)
<nav>	Defines content in a navigation bar

HTML Input Tags

Tag	Display	Description

`<button>`
  content
`</button>`
	Inline	clickable button
		type can be `submit, reset, button`

`<input type="`**type**`" name="`**name**`" />`	Inline	form element input tag type can be `text, number, checkbox, radio, file, etc.`

`<textarea rows="`**num**`" cols="`**num**`">`
  initial text
`</textarea>`
	Inline	multi-line **text** input box

`<label>`**text** `</label>`	Inline	clickable **text** label around a form control

`<select`
`  >`
`  <option>`**text** `</option>`
`  <option>`
`  <optgroup label="text">`
`  <option>` **text** `</option>`
`  <option>` **text** `</option>`
`  </optgroup>`
`  </option>`
`  ...`
`</select>`
	Inline	drop-down selection box (`select`);
		each option within the box (`option`);
		a labeled group of option (`optgroup`);

`<fieldset>`
`  <legend>` **text** `</legend>`
  content
`</fieldset>`
	Block	a grouped set of form fields with a legend

HTML Entities Reference

Result	Description	Entity Name
	non-breaking space	` `
<	less than	`<`
@	at symbol	`@`
>	greater than	`>`
&	ampersand	`&`
©	copyright	`©`

CSS

For the following property and value tables, anything *emphasized* represents values that should be replaced with specific units (e.g., *`length`* should be replaced with a `px`, `pt`, `em` or `%` for many properties, and *`color`* should be replaced with a valid color value such as a hex or rgb code).

A use of | refers to separation of possible values (where you cannot provide two of these possible values for one property) and [value value value] refers to a grouping of possible values that can optionally be used together (e.g., *`[h-shadow v-shadow blur spread color]`* for `box-shadow`).

Selector Types

Name	Description	Example(s)
Universal	Any element	`.foo * { font: 10pt Arial; }`
Element	Any element of a given type	`h1 { text-decoration: underline; }`
Grouping	Multiple elements of different types	`h1, h2, h3 { color: purple; }`
Class	Elements with the given class name	`.example { text-decoration: underline; }`
Id	Single element with the given id	`#example { text-decoration: overline; }`
Descendant	Elements that are children at any level of another specified element	`#example h1 { text-decoration: underline; }`
Child	Elements that are direct children of another specified element	`#example > p { font-weight: bold; }`
Attribute	Elements that have the specified attribute	`input[selected]` - inputs that have the `selected` attribute `input[name='test']` - inputs that have a name 'test'

Background Styles

Property	Values
`background-color`	*`color`* | `transparent`
`background-image`	*`url`* | `none`
`background-origin`	`border-box` | `padding-box` | `content-box`
`background-position`	`top left` | `top center` | `top right` | `center left` | `center center` | `center right` | `bottom left` | `bottom center` | `bottom right` [*`x-%`* *`y-%`*] | [*`x-pos`* *`y-pos`*]
`background-size`	*`length`* | `auto` | `cover` | `contain`
`background-repeat`	`repeat` | `repeat-x` | `repeat-y` | `no-repeat`
`background-attachment`	`scroll` | `fixed`

Border Styles

Note: Replace '*' with any side of the border (`top`, `right`, `left`, `bottom`) for the desired effect.

Example style: '`border: 2px solid red`' applies a solid red border with a width of 2px to all four sides of the element, while '`border-left: 2px solid red`' only applies that border to the left border'.

Property	Values
`border`, `border-*` (shorthand)	*width style color*
`border-width`, `border-*-width`	`thin | medium | thick |` *length*
`border-style`, `border-*-style`	`none | hidden | dotted | dashed | solid | double | groove | rigid | inset | outset`
`border-color`, `border-*-color`	*color*
`box-shadow`	`none | inset | [`*h-shadow v-shadow blur spread color* `]`
`border-radius` `border-*-radius`	*length*

Font and Text Styles

Property	Values
`font-style`	`normal | italic | oblique | inherit`
`font-family`	*fontname*
`font-size`	*length*
`font-weight`	`normal | bold | inherit`
`text-align`	`left | right | center | justify`
`text-decoration`	`none | [underline overline line-through blink]`
`text-shadow`	`none | [`*color length* `]`
`text-indent`	*length*
`text-transform`	`none | capitalize | uppercase | lowercase`
`list-style-type`	`none | asterisks | box | check | diamond | disc | hyphen | square | decimal | lower-roman | upper-roman | kower-alpha | upper-alpha | lower-greek | upper-greek | lower-latin | upper-latin | footnotes`

Color Values

Value	Description
`colorname`	Standard name of color, such as red, blue, purple, etc.
`rgb(redvalue, greenvalue, bluevalue)`	Example: red = rgb(255, 0, 0) or red = rgb(100%, 0, 0)
`#RRGGBB`	Example: red = #FF0000

Box Model

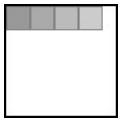
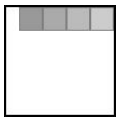
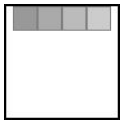
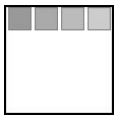
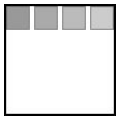
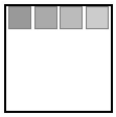
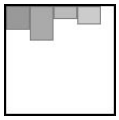
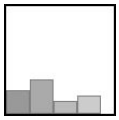
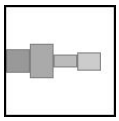
Property	Values
`float`	`left | right | none`
`height`, `width`	`auto |` *`length`*
`min-height`, `max-height` `min-width`, `max-width`	`none |` *`length`*
`margin`, `margin-*`	`auto |` *`length`*
`padding`, `padding-*`	*`length`*
`display`	`none | inline | block | inline-block | flex |` `list-item | compact | table | inline-table`
`overflow`, `overflow-x`, `overflow-y`	`visible | hidden | scroll |` `auto | no-display | no-content`
`clear`	`left | right | both | none`

Flex Box

Property	Values	Element Type	Description
`display`	`flex`	Flex container	Sets all children to become 'flex-items'
`flex-direction`	`row | row-reverse | column | column-reverse`	Flex container	Indicates if the container flows horizontally (`row`) or vertically (`column`)
`flex-wrap`	`nowrap` (default) `| wrap | wrap-reverse`	Flex container	Indicates whether flex items are forced onto one line (`nowrap`) or can `wrap` onto multiple lines (and which direction)

(Flex Box continued on next page)

(Flex Box continued from previous page)

Property	Values	Element Type	Description
`justify-content`		Flex container	Indicates how to position the flex-items in the parent container along the main axis.
	`flex-start`		
	`flex-end`		
	`center`		
	`space-around`		
	`space-between`		
	`space-evenly`		
`align-items`	`stretch` (default)	Flex container	Indicates how to space the items inside the container along the cross axis
	`flex-start`		
	`flex-end`		
	`center`		
	`baseline`		
`order`	*number*	Flex item	Specifies the order in which the element appears in the flex container (by default, flex items are laid out in the source order)
`align-self`	`flex-end` | `flex-start` | `center` | `baseline` | `stretch` (default)	Flex item	Indicates where to place this specific item along the cross axis

JavaScript

`window` Methods and Properties

Method/Property	Description
`getComputedStyle(element)`	Returns an object that reports the values of all CSS properties of an element after applying active stylesheets and resolving any basic computation those values may contain

`localStorage`
`sessionStorage`
	Client side key/value storage options. Each of these have two useful methods:
	
	`setItem(key, value)`:
	- Stores "value" as "key". For example:
	`window.localStorage.setItem("user", "myName");`
	`getItem(key)`:
	- Returns the value stored at "key". For example:
	`let name = window.localStorage.getItem("user");`

`document` Methods and Properties

Method/Property	Description
`getElementById(id)`	Returns a DOM object whose id property matches the specified string. If no matches are found, null is returned.
`getElementsByName(name)`	Returns a collection of all elements which have all of the given name. If no matches are found, null is returned.
`querySelector(sel)`	Returns the first DOM element that matches the specified selector, or group of selectors. If no matches are found, null is returned.
`querySelectorAll(sel)`	Returns a list of the document's elements that match the specified group of selectors. If no matches are found, null is returned.
`createElement(elType)`	Creates and returns an Element node
`createTextNode(data)`	Creates and returns a Text node with the given data

DOM Element Methods and Properties

Method/Property	Description
`el.children`	Returns a collection of the child elements of `el`
`el.parentNode`	Returns the parent node of `el`
`el.appendChild(child)`	Adds a new child node to `el` as the last child node
`el.insertBefore(newNode, refNode)`	Adds `newNode` to parent `el` before `el`'s child `refNode` position
`el.addEventListener(event, fn)`	Attaches an event handler function `fn` to the specified element `el` to listen to `event`
`el.removeEventListener(event, fn)`	Removes the event handler `fn` to the specified `el` listening to `event`
`el.getAttribute(attr)`	Returns the specified attribute value `attr` of `el`
`el.removeChild(child)`	Removes a child node from an element
`el.innerHTML`	Sets or returns the HTML content of an element
`el.textContent`	Sets or returns the text content of the specified node

Other DOM Element Attributes

Recall that if you have an HTML element on your page that has attributes, you can set those attributes through JavaScript as well. For instance if your

```
<img id="dogtag" src="img/doggie.jpg" alt="My Cute Dog" />
```

Your could do the following in your JavaScript code (using the `id` alias for `document.getElementById`):

```
id("dogtag").alt = "My really cute dog";
```

Example DOM Element attributes include are:

Attribute	Description
`id`	The value of the id attribute of an element
`value`	The value attribute of form elements (`input`, `textarea`, `checkbox` `radio`, `select`, etc.)
`name`	The value of the name attribute of a form element
`classList`	Returns the class name(s) of `el`
`className`	Sets or returns the value of the class attribute of `el`
`href`	The href for a `link` or `a` tag
`src`	The value of the src attribute of an image.
`alt`	The value of the alt attribute of an image
`disabled`	Whether or not this DOM element is disabled on the page

DOM Element .classList Methods

Method/Property	Description
`add(class)`	Adds specified class values. These values are ignored if they already exist in the list
`remove(class)`	Removes the specified class value from the `classList`. Does nothing if the class is not in the `classList`.
`toggle(class)`	Toggles the listed class value. If the class exists, then removes it and returns false, if it did not exist in the list add it and return true
`contains(class)`	Returns true if the specified class value exists in the classList for this element

Event Object Methods and Properties

Method/Property	Description
`target`	Returns the element that triggered the event
`type`	Returns the name of the event
`offsetX`	Returns the horizontal coordinate of the mouse pointer, relative to the DOM element clicked
`offsetY`	Returns the vertical coordinate of the mouse pointer, relative to the DOM element clicked

Event Types

`click`	`mousemove`	`keydown`	`change`
`dblclick`	`mouseout`	`error`	`focus`
`mouseenter`	`mouseover`	`success`	`submit`
`mouseleave`	`mouseup`	`load`	`select`
`mousedown`	`keyup`	`unload`	`resize`

JavaScript JSON Methods

Function	Description
`parse(string)`	Returns the given string of JSON data as the equivalent JavaScript object
`stringify(object)`	Returns the given object as a string of JSON data

Other handy JavaScript Methods

Function	Description
`parseInt(string, radix)`	function parses a string argument and returns an integer of the specified radix (the base in mathematical numeral systems). If no radix is passed, returns the integer as base-10.
`console.log(data)`	Writes the `data` to the JavaScript console

JavaScript Array Methods and Properties

Method/Property	Description
`length`	Sets or returns the number of elements in an array
`push(el)`	Adds new elements to the end of an array and returns the new length
`pop()`	Removes and returns the last element of an array
`unshift(el)`	Adds new elements to the beginning of an array and returns the new length
`shift()`	Removes and returns the first element in an array
`sort()`	Sorts the elements of an array
`slice(start, end)`	Returns a new array containing the sequence of elements of the original array from start index (inclusive) to end index (exclusive)
`join()`	Returns a string concatenating all elements of an array (maintaining order)
`concat(list2, ...)`	Joins two or more arrays and returns a copy of the joined arrays
`toString()`	Returns the string representation of an array
`indexOf(el)`	Returns the index of the element in the array, or -1 if not found

JavaScript `string` Methods and Properties

Method/Property	Description
`length`	Returns the length of a string
`charAt(index)`	Returns the character at the specified index
`indexOf(string)`	Returns the position of the first found occurrence of a specified value in a string
`split(delimiter)`	Splits a string into an array of substrings
`substring(start, end)`	Extracts the characters from a string between two specified indices
`trim()`	Removes whitespace from both ends of a string
`toLowerCase()`	Returns a lowercase version of a string
`toUpperCase()`	Returns an uppercase version of a string

JavaScript Timer Functions

Method	Description
`setTimeout(fn, ms)`	Executes a function `fn` after a delay of `ms` milliseconds. Returns a value representing the ID of the timeout being set.
`setInterval(fn, ms)`	Executes a function `fn` at every given time-interval (in milliseconds). Returns a value representing the ID of the interval being set.
`clearTimeout(id)`	Stops the execution of the delay timer specified by id
`clearInterval(id)`	Stops the execution of the interval timer specified by id

JavaScript Math Functions

Method	Description
`Math.random()`	Returns a double between 0 (inclusive) and 1 (exclusive)
`Math.abs(n)`	Returns the absolute value of `n`
`Math.min(a, b, ...)`	Returns the smallest of 0 or more numbers
`Math.max(a, b, ...)`	Returns the largest of 0 or more numbers
`Math.round(n)`	Returns the value of `n` rounded to the nearest integer
`Math.ceil(n)`	Returns the smallest integer greater than or equal to `n`
`Math.floor(n)`	Returns the largest integer less than or equal to `n`
`Math.pow(n, e)`	Returns the base `n` to the exponent `e` power, that is, n^e
`Math.sqrt(n)`	Returns the square root of `n` (`NaN` if `n` is negative)

The Module Pattern

Whenever writing JavaScript, you should use the module pattern, wrapping the content of the code (`window load` event handler and other functions) in an anonymous function. Below is a template for reference:

```
"use strict";
(function() {

  // any module-globals (limit the use of these when possible)
  window.addEventListener("load", init);

  function init() {
    ...
  }

  // other functions
})();
```

Helper Alias Functions

You may use any of the following alias functions in your exam without defining them:

```
function gen(tagName) {
  return document.createElement(tagName);
}

function id(idName) {
  return document.getElementById(idName);
}

function qs(selector) {
  return document.querySelector(selector);
}

function qsa(selector) {
  return document.querySelectorAll(selector);
}
```

Javascript AJAX Fetch Skeleton

```
// you can assume checkStatus is already included in your code
// you do not need to write this on your exam.
 function checkStatus(response) {
    if (!response.ok) {
      throw Error("Error in request: " + response.statusText);
    }
    return response;
  }

// This is an example template for how to make an AJAX fetch request
function makeRequest(){
  let url = ..... // put url string here
  fetch(url) // don't worry about the mode
    .then(checkStatus)
    .then(resp => resp.json())       // line for processing json
    .then(function(resp) {
     //success: do something with the responseJSON
     })
    .catch(function(error) {
      //error: do something with error
     });
}
```

Node.js/Express

This reference summarizes the methods/properties used in CSE 154 for Node.js/Express. It is not an exhaustive reference for everything in Node.js/Express, but provide most functions/properties you will use.

Basic Node.js Project Structure

```
example-node-project/
  .gitignore
  APIDOC.md
  app.js
  node_modules/
    ...
  package.json
  public/
    img/
      ...
    index.html
    index.js
    styles.css
```

Example Express app Template

```
"use strict";

/* File comment */

const express = require("express");
// other modules you use
// program constants

const app = express();

// if handling different POST formats
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(multer().none());

// app.get/app.post endpoints

// helper functions

// if serving front-end files in public/
app.use(express.static("public"));

const PORT = process.env.PORT || 8000;
app.listen(PORT);
```

Useful Core Modules

Module	Description
`fs`	The “file system” module with various functions to process data in the file system.
`util`	Provides various “utility” functions, such as util.promisify.

Other Useful Modules

Module	Description
`express`	A module for simplifying the http-server core module in Node to implement APIs.
`glob`	Allows for quick traversal and filtering of files in a complex directory.
`multer`	Used to support FormData POST requests on the server-side so we can access the req.body parameters.
`mysql2`	Provides functionality for interacting with a database and tables.
`mysql2/promise`	Promisified wrapper over mysql module - each function in the mysql module returns a promise instead of taking a callback as the last argument (recommended).
`cookie-parser`	A module to access cookies with req/res objects in Express.

Express Route Functions

Function	Description

app.get("path", middlewareFn(s));
 	
`app.get("/", (req, res) => {`
`  ...`
`});`
 	
`app.get("/:city", (req, res) => {`
`  let city = req.params.city;`
`  ...`
`});`
 	
`app.get("/cityData", (req, res) => {`
`  let city = req.query.city;`
`  ...`
`});`
 	
`// Example with multiple middleware functions`
`app.get("/", validateInput, (req, res) => {`
`  ...`
`}, handleErr);`
	Defines a server endpoint which accepts a valid GET request. Request and response objects are passed as req and res respectively. Path parameters can be specified in path with ":varname" and accessed via req.params. Query parameters can be accessed via req.query.

app.post("path", middlewareFn(s));
 	
`app.post("/addItem", (req, res) => {`
`  let itemName = req.body.name;`
`  ...`
`}`
	Defines a server endpoint which accepts a valid POST request. Request and response objects are passed as req and res respectively. POST body is accessible via req.body. Requires POST middleware and multer module for FormData POST requests.

Request Object Properties/Functions

Property/Function	Description
req.params	An object containing properties mapped to the named route "parameters". For example, if you have the route /user/:name, then the "name" property is available as req.params.name.
req.query	Captures a dictionary of query parameters, specified in a ?key1=value1&key2=value2& … pattern.
req.body	Holds a dictionary of POST parameters as key/value pairs. Requires multer module for multipart form requests (e.g. FormData) and using middleware functions (see Express template) for other POST request types.
req.cookies	Retrieves all cookies sent in the request. Requires cookie-parser module.

Response Object Properties/Functions

Property/Function	Description

`res.set(headerName, value);`
 	
`res.set("Content-Type", "text/plain");`
`res.set("Content-Type", "application/json");`
	Used to set different response headers - commonly the "Content-type" (though there are others we don't cover).

`res.type("text");` `res.type("json");`	Shorthand for setting the "Content-Type" header.

`res.send(data);`
 	
`res.send("Hello");`
`res.send({ "msg" : "Hello" });`
	Sends the data back to the client, signaling an end to the response (does not terminate your JS program).

`res.end();`	Ends the request/response cycle without any data (does not terminate your JS program).
`res.json(data);`	Shorthand for setting the content type to JSON and sending JSON.
`res.status(statusCode);`	Specifies the HTTP status code of the response to communicate success/failure to a client.

The fs module

Note: Ensure that you are importing the promisified fs functions. While the core `fs` module uses callback functions, `require('fs').promises` wraps these functions to instead return a Promise that resolves with the callback's contents, or rejects if an error occurs. Remember that you should always handle potential fs function errors (try/catch if async/await).

Function	Description
`fs.readFile(filename, "utf8");` `let content = await fs.readFile("file.txt", "utf8");`	Reads the contents of the file located at relative directory **`filename`**. If successful, passes the file contents to the callback as **`contents`** parameter. Otherwise, passes error info to callback as **`error`** parameter.
`fs.writeFile(filename, data, "utf8");` `await fs.writeFile("file.txt", "new contents", "utf8");`	Writes **`data`** string to the file specified by **`filename`**, overwriting its contents if it already exists. If an error occurs, **`error`** is passed to the callback function.
`fs.readdir(path);` `let dirContents = await fs.readdir("dir/path");`	Retrieves all files within a directory located at **`path`**. If successful, passes the array of directory content paths (as strings) to the callback as **`contents`** parameter. Otherwise, passes error info to callback as **`error`** parameter.

The glob module

For the `glob` module, however, you must call `util.promisify`, passing in the `glob` function. This will return a promisified `globPromise` function which you can await in asynchronous function calls.

```
const util = require("util");
const glob = require("glob");
const globPromise = util.promisify(glob);

async function example() {
  try {
    let files = await globPromise("...");
    ...
  } catch(err) {
    ...
  }
}
```

Function	Description

`glob(pattern, callback);`
 	
`glob("img/*", (err, paths) => {`
`  ...`
`});`
 	
`// promisified`
`paths = await globPromise("img/*");`
	Globs all path strings matching a provided **`pattern`**. The pattern will generally follow the structure of a file path, along with any wildcards. If no paths match, the result array will be empty. If successful, passes the array of directory content paths (as strings) to the callback as **`contents`** parameter. Otherwise, passes error info to callback as **`error`** parameter.
	
	Unlike the `fs` module, we must deliberately promisify our `glob` function with util.promisify.
	
	Common selectors:
	***** - A wildcard selector that will contextually match a single filename, suffix, or directory.
	****** - A recursive selector that will search into any subdirectories for the pattern that follows it.

The mysql2/promise module

Function	Description
`mysql.createPool({` **`  host : hostname, // default localhost`** **`  port : port,`** **`  user : username,`** **`  password : pw,`** **`  database : dbname`** **`});`**	Returns a connected database object using config variables. If an error occurs during connection (e.g. the SQL server is not running), does not return a defined database object.
`db.query(qryString);` **`db.query(qryString, [placeholders]);`**	Executes the SQL query. If the query is a SELECT statement, returns a Promise that resolves to an array of RowDataPackets with the records matching the **qryString** passed. If the query is an INSERT statement, the Promise resolves to an OkPacket. Throws an error if something goes wrong during the query. When using variables in your query string, you should use ? placeholders in the string and populate [placeholders] with the variable names to sanitize the input against SQL injection

Regex Reference

`[abc]`	A single character of: a, b, or c	`.`	Any single character	`(...)`	Capture everything enclosed
`[^abc]`	Any single character except: a, b, or c	`\s`	Any whitespace character	`(a|b)`	a or b
`[a-z]`	Any single character in the range a-z	`\S`	Any non-whitespace character	`a?`	Zero or one of a
`[a-zA-Z]`	Any single character in the range a-z or A-Z	`\d`	Any digit	`a*`	Zero or more of a
`^`	Start of line	`\D`	Any non-digit	`a+`	One or more of a
`$`	End of line	`\w`	Any word character (letter, number, underscore)	`a{3}`	Exactly 3 of a
`\A`	Start of string	`\W`	Any non-word character	`a{3,}`	3 or more of a
`\z`	End of string	`\b`	Any word boundary	`a{3,6}`	Between 3 and 6 of a

options: `i` case insensitive `m` make dot match newlines `x` ignore whitespace in regex `o` perform #{...} substitutions only once

Special characters that need to be escaped to match as literals: `[]^$.|?*+(){}\`

SQL

`SELECT`

Description: Used to select data from a database table. If `DISTINCT` is used, no duplicate rows are returned.

Syntax (without `DISTINCT`):

```
SELECT column(s)
FROM table;
```

Syntax (with `DISTINCT`):

```
SELECT DISTINCT column(s)
FROM table;
```

`WHERE`

Description: Used to filter records, returning only those which meet provided conditions.

Syntax:

```
SELECT column(s)
FROM table
WHERE condition(s);
```

Condition types:

- `=, >, >=, <, <=`
- `<>` (not equal)
- `BETWEEN min AND max`
- `LIKE %pattern` (where `%` is a wildcard)
- `LIKE pattern%`
- `LIKE %pattern%`

`ORDER BY`

Description: Used to sort the result set in ascending (default) or descending order.

Syntax:

```
SELECT column(s)
FROM table
ORDER BY column(s) ASC|DESC;
```

`LIMIT`

Description: Used to give the top-n elements of a given category.

Syntax:

```
SELECT column(s)
FROM table
LIMIT n;
```

CREATE TABLE

Description: Used to create a new table.

Syntax:

```
CREATE TABLE tableName(
  column1 datatype,
  column2 datatype,
  ...
  columnN datatype,
  PRIMARY KEY (one or more columns)
);
```

Common Column Data Types:

- `VARCHAR(N)` - strings of up to N characters (e.g., 'Whitaker')
- `INTEGER` - integers (e.g., 10)
- `FLOAT` - floats (e.g., 1.54)
- `DATETIME` - date/time representation (e.g., '2017-05-25 18:20:32')

Alias for tables names

Description: You can use an alias to give a table a different name. You assign a table an alias by using the AS keyword as the following syntax for a partial query:

```
table_name AS table_alias
```

Examples:

```
SELECT g.name, g.platform FROM Games AS g
WHERE g.name LIKE "%Pokemon%"
```

As a shortcut you can also write this same query without the `AS`

```
SELECT g.name, g.platform FROM Games g
WHERE g.name LIKE "%Pokemon%"
```

INSERT INTO

Description: Used to insert a new record (row) into an existing table, where the listed values correspond to the listed columns.

Syntax:

```
INSERT INTO table_name (column1, column2, ..., columnN)
VALUES (value1, value2, ..., valueN);
```

DELETE

Description: Used to remove a record (row) which matches condition(s) from an existing table.

Syntax:

```
DELETE FROM tableName
WHERE condition(s);
```

UPDATE

Description: Used to modify the existing records in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 =
value2, ... WHERE condition(s);
```