BASICS

Link to external JavaScript	`<script type="text/javascript" src="script.js"></script>`
Script loads independently	`<script async src="script.js"></script>`
Load scripts in order	`<script defer src="script.js"></script>`

Create variable	`let varName = value;  // scope-only access`
	`var varName = value;  // accessed anywhere`

Create constant	`const constName = value;  // scope-only access`
Getting user input	`prompt("Enter your name ");`
Save input to variable	`var varName = prompt("Enter your name ");`
DATATYPES & CONVERSIONS	
Get data type of variable	`typeof varName;`
Convert to number or NaN	`Number(var);`
Convert to String	`String(var);`
Convert to Boolean	`Boolean(var);`
Check if not numeric	`isNaN(var);`
Parse string for float value	`parseFloat(var);`
Parse string for int value	`parseInt(var);`
String literal	`"my string"` `'my string'`
Template literal Example: Prints 'Total: 50'	`` `my string` `` `` `Total: ${100 / 2}` ``

operators

Operator	Description	Example
`==`	Equality Operator (Performs type coercion)	`1 == "1"  // True`
`===`	Identity Operator	`1 === "1"  // False`
`!=`	Not equal to	`1 != "1"  // False`
`!==`	Not equal value or type	`1 !== "1"  // True`
`&&`	And	`(1 == 1) && (5 > 10)  // False`
`||`	Or	`(1 == 1) !! (5 > 10)  // True`
`? :`	Ternary operator	`var = True ? this : elseThis`

SELECTION & ITERATION

if/else selection	`if (condition) {`
	`  //code to be run;`
	`} else if (condition2) {`
	`  //code to be run;`
	`} else {`
	`  //code to be run;`
	`}`

while loop	`while (condition) {`
	`  // code to be run while condition = true;`
	`}`

do-while loop	`do {`
	`  // code run once, then while condition = true;`
	`} while (condition);`

for loop	`for (initialization; condition; inc/dec) {`
	`  // code run while condition is met;`
	`}`

for-in loop: access indices	`for (let index in arrayName) { … }`
for-of loop: access values	`for (let value of arrayName) { … }`
Cause loop to end	`break;`
Return to loop condition	`continue;`

ARRAYS & MAP

Create array of size intNum	`new Array([intNum])`
Create array with values	`let arrName = [value1, value2, ...]` `let arrName = new Array(element1, element2, ...)`
Access element	`arrName[#]`
Assign element to index	`arrName[#] = newvalue`
Return number of elements	`arrName.length`
Array destructuring	`[a, b, ...rest] = [10, 20, 30, 40, 50];` `const [cat, , bird] = ["Oliver", "Fido", "Peep"];`
Create map (key-value array)	`const m = new Map()`
Create map with values	`const m = {key1: value, key2: value, …}`
Create key/value pair	`m.set('key', 'value')`
Get value at key	`m.get('key')`
Return true if key exists	`m.has('key')`
Get number of key/value pairs	`m.size`
Delete value at key	`m.delete('key')`

FUNCTIONS & METHODS

array methods

arr.pop()	Removes the last element of an array, and returns that element
arr.push(*item*)	Adds new elements to the end of an array, and returns new length
arr.reverse()	Reverses the order of the elements in an array
arr.shift()	Removes the first element of an array and returns that element
arr.unshift(*item*)	Adds new elements to beginning of array, returns new length
arr.sort()	Sorts the elements of an array (Alphabetical/low-to-high)
arr.join()	Joins elements of array into a string
arr.valueOf()	Returns the primitive value of an array
arr.concat(*arr2*)	Concatenates arr2 to the end of arr, and returns concatenated array
arr.slice(start, end)	Returns elements as a new array from start (inclusive) to end (exclusive)
arr.splice(1)	Remove all items after index 1
arr.splice(1, 2)	Starting at index 1, remove next 2
arr.splice(1, 2, 3)	Remove 2 items at index 1, and add 3
arr.join(*del*)	Changes array into a string separated by delimiter (also .toString())
arr.indexOf(*item*)	Return first index of item if it is within the array
arr.lastIndexOf*(item)*	Return last index of item if it is within the array
arr.forEach*(func)*	Calls function *func* on each element in *arr* (does not change *arr*)
arr.map(*func*)	Returns a new array with *func* applied to elements (does not change *arr*)
arr.reduce(*func, init*)	Call *func* on each element, starting with index *init*, and accumulating value
arr.filter(*func*)	Returns a new array with the elements of which *func* evaluates to true
arr.some(*func*)	Returns true if any element of *arr* exists for which *func* evaluates to true
arr.with(index, new)	Replaces value at *index* with value of *new*

string methods

str.slice(*start, end*)	Return splice of string from start (inclusive) to end (exclusive)
str.trim()	Remove whitespace & newlines from string
str.padStart(*len, char*)	Add char to str until str reaches len
str.repeat(*n*)	Repeat string n times
str.indexOf(*item*)	Return first index of item in string
str.lastIndexOf()	Return last index of item in string
str.split(*del*)	Split string into an array separated by delimiter
str.replace(x, y)	Replace all instances of x in the string with y
str.toLowerCase()	Return str with all characters in lowercase
str.toUpperCase()	Return str with all characters in uppercase

functions

Define a function	`const functionName = function(parameter) {`
	`  statements;`
	`};`

Declaration notation	`function functionName(parameter) {`
	`  statements;`
	`}`

Arrow functions	`const functionName = (parameters) => {`
	`  statements;`
	`};`

Single parameter
Single statement
No parameter
Anonymous function:
	`const functionName = parameter => { return … }`
	`const functionName = parameter => expression;`
	`const functionName = () => statement;`
	`parameter => statement;`

Calling a function	`functionName(arguments);`
Rest parameter: variable length arguments	`function functionName([a,b,]...final parameter) { … }` `// only valid for last parameter`
Spread operator: Call function with array values as arguments	`functionName(...arrName);` `// arrName[0] passed as argument1, arrName[1] as 2, …`
Setting default values	`function functionName(parameter = defaultValue) { … }`
Named parameters	`function functionName([par1, par2, par3]) { … }`

Self-invoking function
 	
Example:
	`(function() {`
	`  statements;`
	`})();`
	
	`(function(n1, n2) {`
	`  return n1 * n2;`
	`})(2, 5);`

Returning a function in a function
 	
Example:
 	
Alternate method:
 	
Calling the function:
 	
Alternate method:
	`function func1() {`
	`  return func2;`
	`}`
	
	`function multiplier(factor) {`
	`  return (number) => number * factor;`
	`}`
	
	`function multiplier(factor) {`
	`  return function(number) {`
	`  return number * factor;`
	`  };`
	`}`
	
	`let twice = multiplier(2);`
	`twice(5); // returns 10`
	
	`multiplier(2)(5);`

LIBRARIES

Date

`Date()`	Get current date
`d = new Date()`	Example: 'Thu Aug 17 2023 15:54:04 GMT-0400 (Eastern Daylight Time)'
`Date(1995, 11, 17)`	Create date object and set Year, Month and Day (Dec 17 1995)
`Date("1995-12-17")`	Create date object using string (Month is not 0-indexed)
`d.getFullYear()`	Return year (Example: 2023)
`d.getMonth()`	Return numeric month from 0 – 11 (Example: 7)
`d.getDate()`	Return day of month from 1 – 31 (Example: 17)
`d.getHours()`	Return hour of day from 0 – 24 (Example: 15)
`d.getMinutes()`	Get minute of the hour from 0 – 59 (Example: 54)
`d.getSeconds()`	Get seconds of the minute from 0 – 59 (Example: 04)
`d.getMilliseconds()`	Get milliseconds (Example: 884)
`d.getDay()`	Return numeric day of the week Sunday – Saturday: 0 – 6 (Example: 4)
`d.setFullYear(`*`year`*`)`	Change date object's year value to *year*
`d.setMonth(`*`month`*`)`	Change date object's month value to numeric *month*
`d.setDate(`*`date`*`)`	Change date object's date value to numeric *date*
`d.setHours(`*`hours`*`)`	Change date object's hours value to *hours*
`d.setMinutes(`*`mins`*`)`	Change date object's minutes value to *mins*

Math

`Math.round(n)`	Returns the value of the number n rounded to the nearest integer
`Math.floor(n)`	Returns the largest integer less than or equal to n
`Math.ceil(n)`	Returns the smallest integer greater than or equal to n
`Math.trunc(n)`	Returns the integer portion of n, removing any fractional digits
`Math.max(n1, n2, …)`	Returns the largest of zero or more numbers
`Math.min(n1, n2, …)`	Returns the smallest of zero or more numbers
`Math.pow(x, y)`	Returns base x to the exponent power y (x^y)
`Math.sign(n)`	Returns the sign of n, indicating whether n is positive, negative, or zero
`Math.random()`	Return a random number between 0-1
`Math.PI`	Ratio of a circle's circumference to its diameter; approximately 3.14159
`Math.E`	Euler's number and the base of natural logarithms; approximately 2.718

Timeout

`setTimeout(functionRef, delay)`	Executes functionRef once delay (milliseconds) passes

`setTimeout(() => {`
`  console.log("Delayed");`
`}, 1000);`
	Runs function which calls console.log("Delayed") after 1 second (1000 milliseconds)

`clearTimeout(`*`timeoutObj`*`)`	Stops a timer created with setTimeout saved to *timeoutObj*

OBJECTS & CLASSES

Create an instance of an object	`let objectName = new Object();` `let objectName = {};`

Creating an object with properties	`var objectName = {`
	`  key: value,`
	`  key: value, …`
	`};`

Example	`let semester = {`
	`  version: "Winter-20",`
	`  courses: ["COM100", "ENG340", "MAT301", "SCI100"]`
	`  grades: [90, 87, 93, 79]`
	`};`

Remove a property from an object	`delete objectName.propertyName;`
Check if object has given property	`"propertyName" in objectName`
Return all property names (keys)	`Object.keys(objectName)`
Copy properties from source to target	`Object.assign(target, source)`

Mutability: object1 and object2 refer to the same memory; updates to object1 change object2 & vice versa	`let object1 = {value: 10};`
	`let object2 = object1;`
	`object1.value = 20;  // object2.value = 20`

Create object with method	`let obj = {};` `obj.methodName = function() { … };`

Create class (blueprint for object)
Add constructor
Add class property
 	
Create class method
	`class ClassName {`
	`  constructor(parameter) {`
	`  this.property = parameter; ...`
	`  }`
	`  methodName(parameter) {...}`
	`}`

Create an object from a class	`let objName = new ClassName(parameter);`
Inheritance	`class SubClass extends BaseClass {`
Polymorphism	`super.superClassFunc()`

Example	`class MainWindow {`
	`  constructor(size) {`
	`  this.size = size;`
	`  }`
	`}`
	`class Popup extends MainWindow {`
	`  constructor(size, textContent) {`
	`  super(size);`
	`  this.textContent = textContent;`
	`  }`
	`}`

DOCUMENT OBJECT MODEL

ACCESSING HTML IN JAVASCRIPT	

Returns the first matching selector
Return array of all matching selectors
Example: Get all elements of class “x”
	`document.querySelector("selector")`
	`document.querySelectorAll("selector")`
	`document.querySelectorAll(".x")`

Get array of tag name elements	`document.getElementsByTagName("tag")`
Get array of elements with className	`document.getElementsByClassName("className")`
Returns element with matching id	`document.getElementById("idName")`
Access attribute of that element Example: Get href of first <a> tag	`document.getElement[s]By.`*`attribute`* `document.getElementsByTagName("a")[0].href`
Access custom attribute	`document.getElementById("id").getAttribute("attr")`
Access elements in body Access elements in head	`document.body.getElement[s]By…` `document.head.getElement[s]By…`
Save element to variable Access attribute from node	`let node = document.getElementById()` `node.`*`attribute`*

Access all children of that element
Access first child of element
Access last child of element
Return number of child nodes
	`node.childNodes`
	`node.firstChild`
	`node.lastChild`
	`node.childElementCount`

Access previous sibling of element Access next sibling of an element	`node.previousSibling` `node.nextSibling`
Access parent node of an element	`node.parentNode`
UPDATING HTML IN JAVASCRIPT	
Create new HTML element	`let node = document.createElement("tagname")`
Add class attribute to element Get or set value using className	`node.classList.add('value')` `node.className [= `*`newvalue`*`]`
Add id attribute to element	`node.setAttribute("id", "value")`
Add child attribute to parent element	`parent.appendChild(child)`
Remove a child node from parent	`parent.removeChild(child);`
Replace child node with a new node	`parent.replaceChild(newNode, replacedNode)`
Insert node1 before node2 in node	`node.insertBefore(node1, node2)`
Remove node from DOM	`node.remove()`
Create text node	`document.createTextNode("text content")`
Change inner html of element	`node.innerHTML = "text content";`

Example	

Create new div element and add the classname 'event'
 	
Create span element 'spn' and text node with value 'x', append text node to span and append span to div
	`let newdiv = document.createElement("div");`
	`newdiv.classList.add('event');`
	
	`var spn = document.createElement("span");`
	`var content = document.createTextNode("x");`
	`spn.appendChild(content);`
	`newdiv.appendChild(spn);`

Example HTML:
 	
`<form id="months">`
`  <label>Select Location:</label>`
`  <select id="selectLoc"></select>`
`</form>`
 	
Creates dropdown options for form using location array and indices
	`const locations = ["Toronto", "New York", "Paris"];`
	`var select = document.getElementById("selectLoc");`
	`locations.forEach(function (item, index) {`
	`  var opt = document.createElement("option");`
	`  opt.textContent = item;`
	`  opt.value = index;`
	`  select.appendChild(opt);`
	`});`

EVENTS

Add event handler syntax	*node*.addEventListener('*event-type*', *function*)

Example: Click event handler on button that calls anonymous function when clicked to log text	`let btn = document.querySelector("button");`
	`btn.addEventListener("click", () => {`
	`  console.log("Button clicked.");`
	`});`

Remove event handler	*node*.removeEventListener("*event-type*", *function*);

Event object: Passed to event handler, holds information on event
Example: mousedown event holds information on button pressed
	`btn.addEventListener("mousedown", event => {`
	`  if (event.button == 0) {`
	`  console.log("Left button");`
	`  }`
	`}`

Pass node to event handler function1, which calls another function2
 	
Or using event object **target** property
	`node.addEventListener('click', function2(e) {`
	`  function2(this);`
	`});`
	
	`document.body.addEventListener("click", event => {`
	`  if (event.target.nodeName == "BUTTON") {`
	`  function(event.target);`
	`  }`
	`});`

Prevent default action on event	`event.preventDefault();`
Prevent bubbling (parent element event handlers being called)	`event.stopPropagation();`

event types

`click`	An element is clicked on
`dblclick`	An element is double-clicked on
`scroll`	A scrollbar is being scrolled
`focus`	An element gets focus
`blur`	An element loses focus
`mousedown`	The mouse button is pressed over an element
`mouseup`	The mouse button is released over an element
`mouseover`	The pointer is moved onto an element
`mouseout`	The pointer is moved out of an element
`keydown`	A key is down
`keypress`	A key is pressed
`keyup`	A key is released
`change`	The content of a form element has changed
`submit`	A form is submitted
`reset`	A form is reset

event properties

PROPERTY	TYPE	DESCRIPTION
`defaultPrevented`	Event	Returns whether or not the preventDefault() method was called for event
`cancelable`	Event	Returns whether or not an event can have its default action prevented
`bubbles`	Event	Returns whether or not a specific event is a bubbling event
`currentTarget`	Event	Returns the element whose event listeners triggered the event
`target`	Event	Returns the element that triggered the event
`type`	Event	Returns the name of the event
`timeStamp`	Event	Returns number of milliseconds from document loading to event
`button`	Mouse	Returns which mouse button was pressed when event was triggered
`x`	Mouse	Returns horizontal coordinate of mouse pointer when event was triggered
`y`	Mouse	Returns vertical coordinate of mouse pointer when event was triggered
`inputType`	Input	Returns the type of the change (i.e "inserting" or "deleting")
`data`	Input	Returns the inserted characters
`repeat`	Keyboard	Returns whether a key is being hold down repeatedly or not
`key`	Keyboard	Returns the key value of the key represented by the event
`deltaX`	Wheel	Returns the horizontal scroll amount of a mouse wheel (x-axis)
`deltaY`	Wheel	Returns the vertical scroll amount of a mouse wheel (y-axis)
`deltaZ`	Wheel	Returns the scroll amount of a mouse wheel for the z-axis

STORAGE

JSON: javascript object notation for serializing & storing data	
Create JSON-encoded string	`const jsonstr = '{"key":"value"}'`

Keys must be double-quoted string
Value can be string, int, array, object, boolean, but not date or functions
	`{"age": 30}  // works`
	`{30: "age"}  // not valid JSON`
	`{"codes":[001, 010, 011, 100]}  // valid`

Create object from json	`const obj = JSON.parse(jsonstr);`
Convert object to json	`const myjson = JSON.stringify(obj);`
localStorage: client-side limited persistent storage in browser	
Create key-value pair	`localStorage.setItem("key", "value");`
Get value from key	`localStorage.getItem("key"));`
Remove key-value pair	`localStorage.removeItem("key");`
sessionStorage: client-side limited session-length storage in browser	
Create key-value pair	`sessionStorage.setItem("key", "value");`
Get value from key	`sessionStorage.getItem("key");`
Remove key-value pair	`sessionStorage.removeItem("key");`
Remove all saved data	`sessionStorage.clear();`

PROMISES

callback: a function that is passed as a parameter to another function	`function callbackAcceptingFunction(callbackFunc) {`
	`  return callback(name);`
	`}`

promise: object representing the eventual completion or failure of an asynchronous operation	`let p = new Promise(function(resolve, reject) {`
	`  resolve(); // called when successful`
	`  reject();  // called when error occurs`
	`});`

.then: Returns promise, runs callbacks	`promiseObj.then(successCallback, failureCallback);`

chaining promises: **.then** runs when resolve is called, **.catch** for reject	`p`
	`  .then(code when when promise successful)`
	`  .catch(code run when promise unsuccessful)`

Example: Function p holds a promise object given one parameter (name)
 	
p is called with an argument and given 2 callbacks for resolve & reject which log the argument passed
	`let p = (name) => new Promise((resolve, reject) => {`
	`  if (name == 'Anita') resolve('Welcome Anita');`
	`  reject('Unauthorized');`
	`});`
	`p('Anita')`
	`  .then(result => console.log(result));`
	`  .catch(error => console.log(error));`

async & await	

delayed promise: promise returned after setTimeout delay, then result is logged using .then callback function
 	
Calling delayedPromise with argument 5 logs result (25) after 3 seconds (3000 milliseconds)
	`function delayedPromise(num) {`
	`  return new Promise((resolve) => {`
	`  setTimeout(() => {`
	`  resolve(num * num);`
	`  }, 3000);`
	`  });`
	`}`
	`delayedPromise(5).then(result => console.log(result))`

async function: returns a promise resolved with value returned by async function, or rejected error	`async function asyncPromise() {`
	`  // code`
	`}`
	`asyncPromise.then(code run when return successful);`

await: suspend async function until results returned	`async function asyncPromise(num) {`
	`  const result = await delayedPromise(num);`
	`}`

Example: asyncPromise called with argument 10, return value of delayedPromise returned after 3 seconds, then logs result (100)	`async function asyncPromise(num) {`
	`  const result = await delayedPromise(num);`
	`  return result;`
	`}`
	`asyncPromise(10)`
	`  .then(result => console.log(result));`

Example: By awaiting results of promises, you can use return values in another promise	`const makeRequest = async () => {`
	`  const value1 = await promise1();`
	`  const value2 = await promise2(value1);`
	`  return promise3(value1, value2);`

NODE BASICS

Run js file locally in console	`$ node file.js`
Run interactive JS interpreter Exit interpreter with status code 0	`$ node` `> process.exit(0)`

Access command line arguments as array inside javascript file
Access first argument using [2]:
	`process.argv`
	`→ ["node", "/path/to/file.js", supplied_arguments]`
	`process.argv[2]`
	`→ supplied_arguments`

Import modules from filename.js	`require('filename');`

Create your own module in a file mymodule.js using **exports**
 	
The module can be accessed in another file using `require('./mymodule');`
	`exports.myDateTime = function () {  // mymodule.js`
	`  return Date();`
	`};`
	
	`var dt = require('./mymodule');  // another file`
	`console.log("The date and time are currently: " +`
	`dt.myDateTime());`

fs module	
Import fs module	`var fs = require('fs');`

readFile function: Read file.txt (using utf8) and output content to console	`let {readFile} = require("fs");`
	`readFile("file.txt", "utf8", (error, text) => {`
	`  if (error) throw error;`
	`  console.log("The file contains:", text);`
	`});`

writeFile function: Writes "Content" into file.txt, overwriting existing content, and logs if write was successful	`const {writeFile} = require("fs");`
	`writeFile("file.txt", "Content", err => {`
	`  if (err) console.log(`Write error: ${err}`);`
	`  else console.log("File written.");`
	`});`

appendFile function: Appends "Content" into file.txt and logs if successful	`const {appendFile} = require("fs");`
	`appendFile("file.txt", "Content", (err) => {`
	`  if(err) console.log(err);`
	`  else console.log("File appended to.");`
	`});`

rename file Synchronous rename:	`fs.rename(oldPath, newPath, callback)` `fs.renameSync(file.txt', 'newFile.txt')`
delete file Synchronous delete:	`fs.unlink(file, callback)` `fs.unlinkSync(file);`
events module	
Load events module	`const EventEmitter = require('events');`
Create EventEmitter object	`const emitter = new EventEmitter();`

Register listener (*`function`* called when event is raised)	`emitter.on('nameOfEvent', function() {`
	` //code run when emitter.emit raises nameOfEvent`
	`});`

Raise event	`emitter.emit('nameOfEvent')`

Example: Emitter iterates over registered listeners and calls synchronously	`emitter.on('messageLogged', function() {`
	`  console.log('Listener called!');`
	`});`
	`emitter.emit('messageLogged');`

http module	
Load http module	`var http = require('http');`

Example: Create a server object using http.createServer, write a response to the client then close response. The server is on http://localhost:8080	`http.createServer(function (req, res) {`
	`  res.write('Hello World!');`
	`  res.end();`
	`}).listen(8080);`

JQUERY BASICS

Downloaded script	`<script src="jquery-3.6.4.min.js"></script>`
Use CDN	`<script src="https://code.jquery.com/jquery-3.7.0.min.js"></script>`

Put jquery code inside this function in .js file	`$(document).ready(function(){ // runs after page is loaded`
	`  // jquery goes in here`
	`});`

Shorthand method:	`$(function(){ // runs after page is loaded`
	`  // jquery goes in here`
	`});`

jQuery Syntax	`$(selector).action()`
Example	`$("p").hide()`
selectors	

Selector types	`$("p") // all p elements`
	`$("#test") // all elements with id=test`
	`$(".test") // all elements with class=test`
	`$("p, a" ) // multiple selectors in comma-separated list`
	`$("div.test") // div with class=test`
	`$("tr:odd") // pseudo-selector for odd table rows`

.has function
.not function
.first function
	`$("div").has("p") // all div elements that contain p tags`
	`$("p").not(".test") // all p tags that do not have class=test`
	`$("ul li").first() // only targets the first unordered list item`

Access current element	`$(this)`
Save to variable	`var divs = $("div");`
Check if element exists	`if ( $("div.test").length )`
events	
Event syntax	`$("selector").event(func run when event triggered)`

Event types
(See more in events)
	`.click(callback) // callback run when element is clicked`
	`.submit(callback) // callback run when form is submitted`
	`.scroll(callback) // callback run when user scrolls`
	`.hover(callback) // callback run when mouse hovers over element`

Example: Creates click event on a button that hides all p elements	`$("button").click(function(){ // click event`
	`  $("p").hide(); // hide action`
	`});`

Example: hides p tag which was clicked	`$("p").click(function(){`
	`  $(this).hide();`
	`});`

Binding multiple events with different handlers using **.on()**	`$("p").on({`
	`  "click": function() { console.log( "clicked!" ); },`
	`  "mouseover": function() { console.log( "hovered!" ); }`
	`});`

Remove using **.off()**	`$("p").off( "click" );`

actions

`.hide(speed, callback)`	Hide the selected element at speed (fast/slow/milliseconds)
`.show(speed, callback)`	Show selected element at speed
`.toggle(speed, callback)`	Toggle element between hidden and showing
`.fadeIn(speed, callback)`	Fade-in display a hidden element
`.fadeOut(speed, callback)`	Fade-out hide a visible element
`.fadeToggle(speed, callback)`	Toggle element between fading in and out
`.fadeTo(speed, opacity, callback)`	Fade element to specified opacity
`.slideDown(speed, callback)`	Slide an element down
`.slideUp(speed, callback)`	Slide an element up
`.slideToggle(speed, callback)`	Toggles element between sliding up and down