

## BASICS

Link to external JavaScript	<code>&lt;script type="text/javascript" src="script.js"&gt;&lt;/script&gt;</code>
Script loads independently	<code>&lt;script <b>async</b> src="script.js"&gt;&lt;/script&gt;</code>
Load scripts in order	<code>&lt;script <b>defer</b> src="script.js"&gt;&lt;/script&gt;</code>
Create variable	<code>let <i>varName</i> = <i>value</i>;        // scope-only access</code> <code>var <i>varName</i> = <i>value</i>;        // accessed anywhere</code>
Create constant	<code>const <i>constName</i> = <i>value</i>;   // scope-only access</code>
Getting user input	<code>prompt("Enter your name ");</code>
Save input to variable	<code>var <i>varName</i> = prompt("Enter your name ");</code>

### DATATYPES & CONVERSIONS

Get data type of variable	<code>typeof <i>varName</i>;</code>
Convert to number or NaN	<code>Number(<i>var</i>);</code>
Convert to String	<code>String(<i>var</i>);</code>
Convert to Boolean	<code>Boolean(<i>var</i>);</code>
Check if not numeric	<code>isNaN(<i>var</i>);</code>
Parse string for float value	<code>parseFloat(<i>var</i>);</code>
Parse string for int value	<code>parseInt(<i>var</i>);</code>
String literal	<code>"my string"</code> <code>'my string'</code>
Template literal Example: Prints 'Total: 50'	<code>`my string`</code> <code>`Total: \${100 / 2}`</code>

### operators

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

## SELECTION & ITERATION

if/else selection	<pre>if (condition) {   //code to be run; } else if (condition2) {   //code to be run; } else {   //code to be run; }</pre>
while loop	<pre>while (condition) {   // code to be run while condition = true; }</pre>
do-while loop	<pre>do {   // code run once, then while condition = true; } while (condition);</pre>
for loop	<pre>for (initialization; condition; inc/dec) {   // code run while condition is met; }</pre>
for-in loop: access indices	<pre>for (let index in arrayName) { ... }</pre>
for-of loop: access values	<pre>for (let value of arrayName) { ... }</pre>
Cause loop to end	<pre>break;</pre>
Return to loop condition	<pre>continue;</pre>

## ARRAYS & MAP

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

## FUNCTIONS & METHODS

### array methods

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

### string methods

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

## functions

Define a function	<pre>const functionName = function(parameter) {     statements; };</pre>
Declaration notation	<pre>function functionName(parameter) {     statements; }</pre>
Arrow functions	<pre>const functionName = (parameters) =&gt; {     statements; };</pre>
Single parameter	<pre>const functionName = parameter =&gt; { return ... }</pre>
Single statement	<pre>const functionName = parameter =&gt; expression;</pre>
No parameter	<pre>const functionName = () =&gt; statement;</pre>
Anonymous function:	<pre>parameter =&gt; statement;</pre>
Calling a function	<pre>functionName(arguments);</pre>
<b>Rest parameter:</b> variable length arguments	<pre>function functionName([a,b,...final parameter]) { ... } // only valid for last parameter</pre>
<b>Spread operator:</b> Call function with array values as arguments	<pre>functionName(...arrName); // arrName[0] passed as argument1, arrName[1] as 2, ...</pre>
Setting default values	<pre>function functionName(parameter = defaultValue) { ... }</pre>
Named parameters	<pre>function functionName([par1, par2, par3]) { ... }</pre>
Self-invoking function	<pre>(function() {     statements; })();</pre>
Example:	<pre>(function(n1, n2) {     return n1 * n2; })(2, 5);</pre>
Returning a function in a function	<pre>function func1() {     return func2; }</pre>
Example:	<pre>function multiplier(factor) {     return (number) =&gt; number * factor; }</pre>
Alternate method:	<pre>function multiplier(factor) {     return function(number) {         return number * factor;     }; }</pre>
Calling the function:	<pre>let twice = multiplier(2); twice(5); // returns 10</pre>
Alternate method:	<pre>multiplier(2)(5);</pre>

## LIBRARIES

### Date

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

### Math

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

### Timeout

<code>setTimeout(functionRef, delay)</code>	Executes functionRef once delay (milliseconds) passes
<code>setTimeout(() =&gt; {   console.log("Delayed"); }, 1000);</code>	Runs function which calls console.log("Delayed") after 1 second (1000 milliseconds)
<code>clearTimeout(<i>timeoutObj</i>)</code>	Stops a timer created with setTimeout saved to <i>timeoutObj</i>

## OBJECTS & CLASSES

Create an instance of an object	<pre>let <i>objectName</i> = new Object(); let <i>objectName</i> = {};</pre>
Creating an object with properties	<pre>var <i>objectName</i> = {   key: value,   key: value, ... };</pre>
Example	<pre>let semester = {   version: "Winter-20",   courses: ["COM100", "ENG340", "MAT301", "SCI100"]   grades: [90, 87, 93, 79] };</pre>
Remove a property from an object	<pre>delete <i>objectName</i>.<i>propertyName</i>;</pre>
Check if object has given property	<pre>"<i>propertyName</i>" in <i>objectName</i></pre>
Return all property names (keys)	<pre>Object.keys(<i>objectName</i>)</pre>
Copy properties from source to target	<pre>Object.assign(<i>target</i>, <i>source</i>)</pre>
Mutability: object1 and object2 refer to the same memory; updates to object1 change object2 & vice versa	<pre>let object1 = {value: 10}; let object2 = object1; object1.value = 20; // object2.value = 20</pre>
Create object with method	<pre>let obj = {}; obj.<i>methodName</i> = function() { ... };</pre>
Create class (blueprint for object) Add constructor Add class property  Create class method	<pre>class <i>ClassName</i> {   constructor(<i>parameter</i>) {     this.<i>property</i> = <i>parameter</i>; ...   }   <i>methodName</i>(<i>parameter</i>) {...} }</pre>
Create an object from a class	<pre>let objName = new <i>ClassName</i>(<i>parameter</i>);</pre>
Inheritance	<pre>class <i>SubClass</i> extends <i>BaseClass</i> {</pre>
Polymorphism	<pre>super.<i>superClassFunc</i>()</pre>
Example	<pre>class MainWindow {   constructor(size) {     this.size = size;   } } class Popup extends MainWindow {   constructor(size, textContent) {     super(size);     this.textContent = textContent;   } }</pre>

## 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"	<code>document.querySelector("selector")</code> <code>document.querySelectorAll("selector")</code> <code>document.querySelectorAll(".x")</code>
Get array of tag name elements	<code>document.getElementsByTagName("tag")</code>
Get array of elements with className	<code>document.getElementsByClassName("className")</code>
Returns element with matching id	<code>document.getElementById("idName")</code>
Access attribute of that element Example: Get href of first <a> tag	<code>document.getElement[s]By.attribute</code> <code>document.getElementsByTagName("a")[0].href</code>
Access custom attribute	<code>document.getElementById("id").getAttribute("attr")</code>
Access elements in body Access elements in head	<code>document.body.getElement[s]By...</code> <code>document.head.getElement[s]By...</code>
Save element to variable Access attribute from node	<code>let node = document.getElementById()</code> <code>node.attribute</code>
Access all children of that element Access first child of element Access last child of element Return number of child nodes	<code>node.childNodes</code> <code>node.firstChild</code> <code>node.lastChild</code> <code>node.childElementCount</code>
Access previous sibling of element Access next sibling of an element	<code>node.previousSibling</code> <code>node.nextSibling</code>
Access parent node of an element	<code>node.parentNode</code>
UPDATING HTML IN JAVASCRIPT	
Create new HTML element	<code>let node = document.createElement("tagname")</code>
Add class attribute to element Get or set value using className	<code>node.classList.add('value')</code> <code>node.className [= newvalue]</code>
Add id attribute to element	<code>node.setAttribute("id", "value")</code>
Add child attribute to parent element	<code>parent.appendChild(child)</code>
Remove a child node from parent	<code>parent.removeChild(child);</code>
Replace child node with a new node	<code>parent.replaceChild(newNode, replacedNode)</code>
Insert node1 before node2 in node	<code>node.insertBefore(node1, node2)</code>
Remove node from DOM	<code>node.remove()</code>
Create text node	<code>document.createTextNode("text content")</code>
Change inner html of element	<code>node.innerHTML = "text content";</code>

Example	
Create new div element and add the classname 'event'	<pre>let newdiv = document.createElement("div"); newdiv.classList.add('event');</pre>
Create span element 'spn' and text node with value 'x', append text node to span and append span to div	<pre>var spn = document.createElement("span"); var content = document.createTextNode("x"); spn.appendChild(content); newdiv.appendChild(spn);</pre>
<p>Example HTML:</p> <pre>&lt;form id="months"&gt;   &lt;label&gt;Select Location:&lt;/label&gt;   &lt;select id="selectLoc"&gt;&lt;/select&gt; &lt;/form&gt;</pre> <p>Creates dropdown options for form using location array and indices</p>	<pre>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); });</pre>

## EVENTS

Add event handler syntax	<code>node.addEventListener('event-type', function)</code>
Example: Click event handler on button that calls anonymous function when clicked to log text	<pre>let btn = document.querySelector("button"); btn.addEventListener("click", () =&gt; {   console.log("Button clicked."); });</pre>
Remove event handler	<code>node.removeEventListener("event-type", function);</code>
Event object: Passed to event handler, holds information on event Example: mousedown event holds information on button pressed	<pre>btn.addEventListener("mousedown", event =&gt; {   if (event.button == 0) {     console.log("Left button");   } });</pre>
Pass node to event handler function1, which calls another function2	<pre>node.addEventListener('click', function2(e) {   function2(this); });</pre>
Or using event object <b>target</b> property	<pre>document.body.addEventListener("click", event =&gt; {   if (event.target.nodeName == "BUTTON") {     function(event.target);   } });</pre>
Prevent default action on event	<code>event.preventDefault();</code>
Prevent bubbling (parent element event handlers being called)	<code>event.stopPropagation();</code>



## 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

<b>JSON:</b> javascript object notation for serializing & storing data	
Create JSON-encoded string	<code>const jsonstr = '{"key":"value"}'</code>
Keys must be double-quoted string Value can be string, int, array, object, boolean, but not date or functions	<code>{"age": 30} // works</code> <code>{30: "age"} // not valid JSON</code> <code>{"codes":[001, 010, 011, 100]} // valid</code>
Create object from json	<code>const obj = JSON.parse(jsonstr);</code>
Convert object to json	<code>const myjson = JSON.stringify(obj);</code>
<b>localStorage:</b> client-side limited persistent storage in browser	
Create key-value pair	<code>localStorage.setItem("key", "value");</code>
Get value from key	<code>localStorage.getItem("key");</code>
Remove key-value pair	<code>localStorage.removeItem("key");</code>
<b>sessionStorage:</b> client-side limited session-length storage in browser	
Create key-value pair	<code>sessionStorage.setItem("key", "value");</code>
Get value from key	<code>sessionStorage.getItem("key");</code>
Remove key-value pair	<code>sessionStorage.removeItem("key");</code>
Remove all saved data	<code>sessionStorage.clear();</code>

## PROMISES

<b>callback:</b> a function that is passed as a parameter to another function	<code>function callbackAcceptingFunction(callbackFunc) {   return callback(name); }</code>
<b>promise:</b> object representing the eventual completion or failure of an asynchronous operation	<code>let p = new Promise(function(resolve, reject) {   resolve(); // called when successful   reject(); // called when error occurs });</code>
<b>.then:</b> Returns promise, runs callbacks	<code>promiseObj.then(successCallback, failureCallback);</code>
chaining promises: <b>.then</b> runs when resolve is called, <b>.catch</b> for reject	<code>p   .then(code when when promise successful)   .catch(code run when promise unsuccessful)</code>
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	<code>let p = (name) =&gt; new Promise((resolve, reject) =&gt; {   if (name == 'Anita') resolve('Welcome Anita');   reject('Unauthorized'); }); p('Anita')   .then(result =&gt; console.log(result));   .catch(error =&gt; console.log(error));</code>

async & await	
<p><b>delayed promise:</b> promise returned after setTimeout delay, then result is logged using .then callback function</p> <p>Calling delayedPromise with argument 5 logs result (25) after 3 seconds (3000 milliseconds)</p>	<pre>function delayedPromise(num) {   return new Promise((resolve) =&gt; {     setTimeout(() =&gt; {       resolve(num * num);     }, 3000);   }); } delayedPromise(5).then(result =&gt; console.log(result))</pre>
<p><b>async function:</b> returns a promise resolved with value returned by async function, or rejected error</p>	<pre>async function asyncPromise() {   // code } asyncPromise.then(code run when return successful);</pre>
<p><b>await:</b> suspend async function until results returned</p>	<pre>async function asyncPromise(num) {   const result = await delayedPromise(num); }</pre>
<p>Example: asyncPromise called with argument 10, return value of delayedPromise returned after 3 seconds, then logs result (100)</p>	<pre>async function asyncPromise(num) {   const result = await delayedPromise(num);   return result; } asyncPromise(10)   .then(result =&gt; console.log(result));</pre>
<p>Example: By awaiting results of promises, you can use return values in another promise</p>	<pre>const makeRequest = async () =&gt; {   const value1 = await promise1();   const value2 = await promise2(value1);   return promise3(value1, value2); }</pre>

## 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 <b>exports</b>	<b>exports</b> .myDateTime = function () { // mymodule.js return Date(); };
The module can be accessed in another file using require('./mymodule');	var dt = require('./mymodule'); // another file console.log("The date and time are currently: " + dt.myDateTime());

fs module	
Import fs module	<code>var fs = require('fs');</code>
<b>readFile</b> function: Read file.txt (using utf8) and output content to console	<code>let {readFile} = require("fs"); readFile("file.txt", "utf8", (error, text) =&gt; {   if (error) throw error;   console.log("The file contains:", text); });</code>
<b>writeFile</b> function: Writes "Content" into file.txt, overwriting existing content, and logs if write was successful	<code>const {writeFile} = require("fs"); writeFile("file.txt", "Content", err =&gt; {   if (err) console.log(`Write error: \${err}`);   else console.log("File written."); });</code>
<b>appendFile</b> function: Appends "Content" into file.txt and logs if successful	<code>const {appendFile} = require("fs"); appendFile("file.txt", "Content", (err) =&gt; {   if(err) console.log(err);   else console.log("File appended to."); });</code>
<b>rename</b> file Synchronous rename:	<code>fs.rename(oldPath, newPath, callback) fs.renameSync(file.txt', 'newFile.txt')</code>
<b>delete</b> file Synchronous delete:	<code>fs.unlink(file, callback) fs.unlinkSync(file);</code>
events module	
Load events module	<code>const EventEmitter = require('events');</code>
Create EventEmitter object	<code>const emitter = new EventEmitter();</code>
Register listener ( <i>function</i> called when event is raised)	<code>emitter.on('nameOfEvent', function() {   //code run when emitter.emit raises nameOfEvent });</code>
Raise event	<code>emitter.emit('nameOfEvent')</code>
Example: Emitter iterates over registered listeners and calls synchronously	<code>emitter.on('messageLogged', function() {   console.log('Listener called!'); }); emitter.emit('messageLogged');</code>
http module	
Load http module	<code>var http = require('http');</code>
Example: Create a server object using http.createServer, write a response to the client then close response. The server is on http://localhost:8080	<code>http.createServer(function (req, res) {   res.write('Hello World!');   res.end(); }).listen(8080);</code>

## JQUERY BASICS

Downloaded script	<code>&lt;script src="jquery-3.6.4.min.js"&gt;&lt;/script&gt;</code>
Use CDN	<code>&lt;script src="https://code.jquery.com/jquery-3.7.0.min.js"&gt;&lt;/script&gt;</code>
Put jquery code inside this function in .js file	<code>\$(document).ready(function(){ // runs after page is loaded // jquery goes in here });</code>
Shorthand method:	<code>\$(function(){ // runs after page is loaded // jquery goes in here });</code>
Action syntax	<code>\$("selector").jquery-method()</code>
Event syntax	<code>\$("selector").event(func run when event triggered)</code>
<b>selectors</b>	
Selector types (See CSS-Ref for more selector types)	<code>\$("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</code>
.has function .not function .first function	<code>\$(".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</code>
Access current element	<code>\$(this)</code>
Access all elements	<code>\$("*")</code>
Save to variable	<code>var divs = \$(".div");</code>
<b>events &amp; actions</b>	
jQuery Syntax	<code>\$("<b>selector</b>").<b>event</b>(function() { // <b>element</b> that captures <b>event</b> \$("<b>selector</b>").<b>action</b>() // <b>element</b> that <b>action</b> is }); // performed on</code>
Example: Creates click event on a button that hides all p elements	<code>\$("#button").<b>click</b>(function(){ // <b>click</b> event \$("p").<b>hide</b>(); // <b>hide</b> action });</code>
Event types (See more in <a href="#">events</a> )	<code>.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</code>
Example: hides p tag which was clicked	<code>\$("#p").click(function(){ \$(<b>this</b>).hide(); });</code>
Binding multiple events with different handlers using <b>.on()</b>	<code>\$("#p").on({ "click": function() { console.log("clicked!"); }, "mouseover": function() { console.log("hovered!"); } });</code>
Remove using <b>.off()</b>	<code>\$("#p").off( "click" );</code>

## actions

<code>.hide(speed, callback)</code>	Hide the selected element at speed (fast/slow/milliseconds)
<code>.show(speed, callback)</code>	Show selected element at speed
<code>.toggle(speed, callback)</code>	Toggle element between hidden and showing
<code>.fadeIn(speed, callback)</code>	Fade-in display a hidden element
<code>.fadeOut(speed, callback)</code>	Fade-out hide a visible element
<code>.fadeToggle(speed, callback)</code>	Toggle element between fading in and out
<code>.fadeTo(speed, opacity, callback)</code>	Fade element to specified opacity
<code>.slideDown(speed, callback)</code>	Slide an element down
<code>.slideUp(speed, callback)</code>	Slide an element up
<code>.slideToggle(speed, callback)</code>	Toggles element between sliding up and down
<code>.css(property-name)</code>	Return the value of the element's css property
<code>.css(property-name, value)</code>	Sets element's css property to the specified value
<code>.val([value])</code>	Return the value of a form element, or set to value
<code>.text([value])</code>	Return the text content of an element, or set to value
<code>.html(["html-code"])</code>	Return the html content of an element, or set to html-code
<code>.attr(attribute, [value])</code>	Return value of specified attribute, or set attribute to value
<code>.attr({attr: val, attr: val})</code>	Change multiple attributes
<code>.append(content)</code>	Insert specified content to the end of the element
<code>.prepend(content)</code>	Insert specified content to the beginning of the element
<code>.action().action().action()</code>	<b>Chaining:</b> Multiple actions performed one after another

Examples:

Adds a click event on button elements that causes the background-color of all div elements to change to yellow	<pre>\$(document).ready(function(){     \$("button").click(function(){         \$("div").css("background-color", "yellow");     }); });</pre>
Check if there exists a p element with a class of "test", and if so, changes the font color and font size of all p elements	<pre>\$(document).ready(function() {     if ( \$("p.test").length ) {         \$("p").css({"color": "red", "font-size": "200%"});     } });</pre>
Chaining multiple events onto p element	<pre>\$(document).ready(function(){     \$("p").on({         "mouseover": function() {             \$(this).css("background-color", "pink");         },         "mouseout": function() {             \$(this).css("background-color", "");         }     }); });</pre>
Chaining multiple actions onto an element	<pre>\$("p").css("color", "red")     .slideUp(2000)     .slideDown(2000);</pre>