**JAVASCRIPT**

**CONTENTS**

|  |  |
| --- | --- |
| Syntax & grammar |  |
| Functions |  |
| Objects |  |
| Built-in objects |  |
| Decisions & loops |  |
| Truthy and falsy values |  |
| Loops |  |
| DOM |  |
| Events |  |

**Syntax & Grammar**

|  |  |
| --- | --- |
| **Syntax** | **Define** |
| Statement | An individual instruction/step |
| Comments | A comment to explain what the code does |
| Variable | Stores data |
| Declare the variable | Naming the variable |
| camelCase | The structure of how to write code with more than one word |
| Data types | Numbers, strings, Booleans, arrays, objects, undefined & null |
| Array | A specific type of variable that can store a list of values E.g. var list = [‘one’,’two]; |
| Index | A number given to an item in an array |
| Arithmetic operators | +, -, /, \*, ++, --, % |
| String operators | + (used to join together) |
| Expression | Results in a single value. There are two types of expressions: 1) Expressions that just assign a value to a variable (var color = brown). 2) Those that use 2+ values to return a single value (var number = 2\*3). |
| Calling a function | When you ask the function to run |
| Parameters | Variables inside the functions parenthesis E.g. function test (**length, height**){code block}; |
| Arguments | The value of the parameters when you run a function. E.g. test (**20, 15**); |
| Property | The name of a variable inside an object |
| Method | The name of a method inside an object |
| Key | The name of properties and methods in an object |
| Instance | A variable created using an object template |
| DOM query | Methods that find elements in the DOM tree |

**Functions**

Functions consist of a series of statements that have been grouped together to perform a specific task.

Getting multiples values out of a function:

Function getSize (width, height, depth){

Var area = width\*height;

Var volume = width\*height\*depth;

Var sizes = [area, volume];

Return sizes;

}

Var areaOne = getSize(3,2,3)[0];

|  |  |
| --- | --- |
| **Variable scope** | **Define** |
| Local variable | Variables inside a function. These take up less memory than global variables |
| Global variable | Outside a function and known by all code |

**Objects**

Objects group together a set of variables and functions to create a model of something you would recognise from the real world.

Object - literal notation

Var hotel = {

Name : ’quay’,

Rooms : 40,

Booked : 25,

roomTypes : [‘twin’,’double’,’suite’],

checkAvailability: function(){

return this.rooms – this.booked;

}

};

How to access an object:

Var hotelName = hotel.Name;

Var roomsFree = hotel.checkAvailability();

How to update an object:

hotel.Name = ‘park’;

Object – constructor notation

Used when you want several objects to represent similar things.

Function hotel (name, rooms, booked){

This.name=name;

This.rooms = rooms;

This.booked = booked;

This.checkAvailability = function(){

Return this.rooms – this.booked;

}

};

Var quayHotel = new hotel (‘quay’,’40’,’25’);

**OR**

Var hotel = new Object ();

Hotel.name=’quay’;

Hotel.rooms = 40;

Hotel.booked = 25;

Hotel.checkAvailability = function(){

Return this.rooms – this.booked;

};

**Built-in objects**

Arrays are technically objects. The index number in an array is similar to an objects property.

Arrays and objects can be combined. Arrays can be used in objects and vice versa.

3 groups of built-in objects:

1. Browser Object Model
2. Document Object Model
3. Global Javascript Objects

Browser Object Model (BOM)

The BOM creates a model of the browser tab or window. E.g. window.print(); - causes the browsers print dialog box to be shown.

The window object is the topmost object in the browser model

Window. – contains other objects that tell you about the browser

Alert(); is a window method (window.alert();). Window. – is set as default which is why ‘window. ‘ does not need to be specified.

Document Object Model (DOM)

The DOM creates a model of the current page. E.g. document.getElementById(‘txt’);

The topmost object in the DOM is the document. Object.

Document. – represent the web page loaded in to the current browser

The DOM is vital to accessing and amending the contents of the current web page

E.g. Document.write();

Document.getElementById();

Document.createElement();

Global Javascript Objects

Global Javascript Objects do not form a single model. They are a group of individual objects that relate to different parts of javascript. E.g. Math.pi;

String objects – has own properties and methods.

E.g. var saying = ‘random text’;

Saying.length = (returns the number of characters of the variable ‘saying’)

Saying.toUppercase();

Number objects - also has own properties and methods like strings.

E.g. var number = 10.2564;

Var decimalPlaces = number.toFixed(3);

Var totalNumbers = number.toPrecision(5);

Math Objects – has properties and methods for math constants and functions.

E.g. var randomNumber = Math.floor((Math.random()\*10)+1);

Date Objects – in order to work with dates, you create an instance of the date object. You can then specify the time and date that you want it to represent

E.g. var today = new Date();

Var year = today.getFullYear();

Other methods = getDate(); getDay(); getHours(); getMinutes();…

**Decisions and loops**

Comparison operators:

|  |  |
| --- | --- |
| == | Equal to |
| != | Is not equal to |
| > | Greater than |
| < | Lesser than |
| === | Strictly equal to |
| !== | Strictly not equal to |
| >= | Greater than or equal to |
| <= | Lesser than or equal to |

Logical operators:

|  |  |
| --- | --- |
| && | Logical and |
| || | Logical or |
| ! | Logical not (does opposite e.g. if was ‘false’ then ‘!’ would return ‘true’) |

IF statements

If (score>=50){

Code block if true

} else {

Code block if false

}

If = keyword

Score>=50 = condition

Also can use ‘if else’ if more options: E.g. if (score>=50){codeblock} elseif(score>=20){codeblock} else {codeblock for all other scenarios};

Switch statements

A switch statement starts with a variable called the ‘switch’ value. Each case indicates a possible value for this variable and the code that should run if the variable matches that value.

Switch statements perform more quickly than if statements as code breaks off after ‘break;’. The default option acts as the ‘else’ statement. Switch statements are better to use than nested if statements.

E.g. var firstName = ‘Ali’;

Switch(firstName){

Case ‘Ali’:

Codeblock;

Break;

Case ‘Lauren’;

Codeblock;

Break;

Default:

Codeblock;

Break;

}

**Truthy and Falsy values**

|  |  |
| --- | --- |
| **Truthy** | **Falsy** |
| Var text = true; | Var text = false; |
| Var text = 1 | Var text = 0; |
| Var text = ‘text’ | Var text = ‘’; |
| Var text = 10/5 | Var text = 10/’score’; |
|  | Var text; |

Because of the above, the strict equality operators === and !== result in fewer unexpected values than == and !=.

**Loops**

Loops check a condition, and if it is true, a code block will run. Then the condition will be checked again, and the code will run again until the condition is false.

3 types:

1. For (most common)
2. While
3. Do while

For loop

use this if you need to run a code a specific number of times. For loops are often used to loop through the items in an array.

E.g. var scores = [25,32,15];

Var msg = ‘’;

Var i;

for (i = 0; i<scores.length; i++){

msg += ‘test ’+ (i+1) + ‘= ‘+scores[i]+’<br />’;

}

Document.write(msg);

While

use this is you are unsure of the specific number of times.

E.g. var i = 1;

Var msg = ‘’;

While (i<10){

Msg+= i+’ x 5 = ’+(i\*5)+’<br />’;

i++;

}

Document.write(msg);

Do while

similar to while but will run the code at least once even if false.

E.g. var i = 1;

Do {

Codeblock;

i++;

}

While (i<1);

Document.write(codeblock);

**DOM**

**Part 1**

The DOM specifies how browsers should create a model of an HTML page and how javascript can access and update the contents of a web page while it is in the browser window:

1. Makes a model of the HTML page in memory
2. Access and change the HTML page

DOM tree

A DOM tree is a model of a web page, it consists of 4 nodes:

1. Document
2. Element
3. Attribute
4. Text

E.g.

Working with the DOM tree

Steps:

1. Locate the node that represents the element you want to work with
2. Use its content, child elements and attributes

Step 1:

*Select individual element* – getElementById(); querySelector();

*Select multiple elements* – getElementsByClassName(); getElementsByTagName(); querySelectorAll();

*Traversing between Element Nodes* – parentNode, previousSibling / nextSibling, firstChild/lastChild

Step 2:

*Access and update text nodes –* select the element, then traverse between elements to find the text node, nodeValue returns contents

*Working with HTML content –* innerHTML, textContent. CreateElement(); createTextNode(); appendChild(); / removeChild();

*Access or update attribute values –* has Attribute(); getAttribute(); setAttribute(); removeAttribute();

Caching DOM queries

Methods that find elements in the DOM tree are called DOM queries. When you need to work with an element more than once, you should use a variable to store the result of this query.

E.g. var itemOne = document.getElementById(‘one’);

**Part 2**

Node lists

Node lists are DOM queries that return more than one element.

Selecting an element from a node list (two ways):

1. The item() method. E.g.

var elements = document.getElementsByClassName(‘hot’);

If (elements.length>=1){

Var firstItem = elements.item(0);

}

1. Array syntax (preferred method)

If (elements.length >=1){

Var firstItem = elements[0];

}

Repeating actions for an entire nodelist:

e.g. var hotItems = document.querySelectorAll(‘li’);

for (var i=0; i<hotItems.length; i++){

hotItems[i].className=’fun’;

}

Traversing the DOM

When you have an element node, you can select another element in relation to is using the following 5 properties: parentNode, previousSibling, nextSibling, firstChild, lastChild.

Access and update text with textContent

the textContent property allows you to collect or update just the text that is in the containing element (and its children).

E.g. document.getElementById(‘one’).textContent = ‘random text here’;

How to get/update element content

Once you have navigated from an element to its text node, one common property you will use is nodeValue.

In order to use nodeValue, you must be on a text node, not the element that contains it.

E.g.

(HTML) <div id=’one’><p>text one</p>text two </div>

(JS) var textTwo = document.getElementById(‘one’).firstChild.nextSibling.nodeValue; (this would be ‘text two’).

*Changing a text node:*

textTwo = textTwo.replace(‘text two’, ‘other text’);

document.getElementById(‘one’).firstChild.nextSibling.nodeValue = textTwo;

Accessing or removing HTML content

1) Inner HTML – create a variable holding the mark up, select element to update, update content.

E.g. var item = ‘<b>hello world</b>’;

Var el = document.getElementById(‘one’);

El.innerHTML = item;

2) DOM manipulation – can be safer than innerHTML but require more code.

Steps:

1. Create new text node
2. Create new element node
3. Add text node to element node
4. Select element you want to add the new fragment to
5. Append the new fragment to the selected element

E.g.

Var newText = document.createTextNode(‘hello world’);

Var newel = document.createElement(‘li’);

newEl.appendChild(newText);

var position = document.getElementsByTagName(‘ul)[0];

position.appendChild(newEl);

Removing elements via DOM manipulation

Steps:

1. Store the element to be removed in a variable
2. Store the parent of that element in a variable
3. Remove the element from its containing element

E.g.

Var removeEl = document.getElementsByTagName(‘li’)[3];

Var containerEl = removeEl.parentNode;

containerEl.removeChild(removeEl);

**Part 3**

Comparing techniques: updating HTML content

|  |  |  |
| --- | --- | --- |
| **Technique** | **Advantages** | **Disadvantages** |
| Document.write | Good for showing beginners | DO NOT USE |
| InnerHTML | Can be faster than DOM manipulation.  Simple to remove all content from an element (by assigning blank string) | Can pose security risk so should not be used to add content from a user.  Difficult to isolate single elements in a large DOM fragment  Event handlers may not work as intended |
| DOM manipulation | Does not affect event handlers  Suited to changing one element from a DOM fragment where there are many siblings | If you want to make a lot of changes to the page, it is slower than innerHTML  You need to write more code to achieve the same thing as innerHTML |

XSS (cross site scripting)

You can safely use innerHTML to add mark up to a page if you have written a code, but content from any untrusted sources should be escaped and added as text (not mark up), using properties like textContent.

**Need more info on this**

Attribute nodes

Once you have an element node, you can use other properties and methods on that element node to access and change its attributes.

*Methods:*

getAttribute();

has Attribute();

setAttribute();

removeAttribute();

*properties:*

className

id

It is good practice to check whether an attribute exists.

E.g.

Var firstItem = document.getElementById(‘one’);

If (firstItem.hasAttribute(‘class’)){

Var attr = firstItem.getAttribute(‘class’);

}

Var el = document.getElementById(‘scriptResults’);

El.innerHTML = ‘<p>the first item has a class name of: ‘ + attr + ’</p>’

Creating attributes and changing their values

Two ways to do this are:

1. className = ‘class name’;
2. setAttribute(‘class’, ‘className’);

E.g. var firstItem = document.getElementById(‘one’);

1) firstItem.className = ‘fun’;

2) firstItem.setAttribute(‘class’, ‘fun’);

Removing attributes

E.g. var firstItem = document.getElementById(‘one’);

If (firstItem.hasAttribute(‘class’)){

firstItem.removeAttribute(‘class’);

}

Examine the DOM in chrome

*More tools > developer tools*

This can help you understand the structure of the DOM tree and also lets you know what properties are available to their specific nodes.

**Events**

**Part 1**

|  |  |  |
| --- | --- | --- |
| **Event type** | **Event** | **Definition** |
| **UI** | Load | Fires when web page finishes loading |
|  | Unload | Fires when web page is unloading |
|  | Error | Fires when browser encounters an error |
|  | Resize | Fires when browser window has been resized |
|  | Scroll | Fires when user has scrolled up or down |
| **Keyboard** | Keydown | Fires when user presses a key |
|  | Keyup | Fires when user releases a key |
|  | Keypress | Character is being inserted |
| **Mouse** | Click | Fires when user clicks mouse button over element |
|  | Dblclick | Fires when user double clicks over an element |
|  | Mousedown | User presses a mouse button while over an element |
|  | Mouseup | User release a mouse button while over an element |
|  | Mousemove | User moves the mouse |
|  | Mouseover | User moves the mouse over an element |
|  | Mouseout | User moves the mouse off an element |
| **Focus** | Focus | Element gains focus |
|  | Blur | Element loses focus |
| **Form** | Input | Value in any <input> or <textarea> element has changed |
|  | Change | Value in select box, check box or radio button changes |
|  | Submit | User submits a form |
|  | Reset | User clicks on a forms’ reset button |
|  | Cut | User cuts content from a form field |
|  | Copy | User copies content from a form field |
|  | Paste | User pastes content into a form field |
|  | Select | User selects some text in a form field |
| **Mutation** | DOMNodeInserted | Fires when a node is inserted into the DOM tree |
|  | DOMNodeRemoved | Fires when a node is removed from the DOM tree |
|  | DOMSubtreeModified | Fires when the DOM structure changes |
|  | DOMInsertedIntoDocument | Fires when a node is inserted into the DOM tree as a descendant of another node that is already in the document |
|  | DOMNodeRemovedFromDocument | Fires when a node is removed from the DOM tree as a descendant of another node that is already in the document |
| **HTML5** | DOMContentLoaded | Events fire when the DOM tree is formed (images, CSS, and JS might still be loading). This makes the page seem faster to load |
|  | Hashchange | Fires when the URL hash changes (without the entire window refreshing) |
|  | Befireunload | Fires on the window object before the page is unloaded |

When the user interacts with the HTML on a web page there are 3 steps involved in getting it trigger some javascript code. Together these steps are known as event handling.

1. Select the element node(s) you want the script to respond to
2. Indicate which event on the selected node(s) will trigger the response
3. State the code you want to run when the event occurs

E.g. element.addEventListener(‘event’, functionName, [Boolean]);

Var el = document.getElementById(‘id’);

El.addEventListener(‘blur’,checkUsername,false);

(where the function checkUsername() runs to see if name is long enough)

Parameters with event listeners

E.g. el.addEventListener(‘blur’, function(){

checkUsername(5)

}, false);

As Internet explorer 5-8 didn’t support addEventListener(), you can use following code to make code work with older versions of IE:

E.g. if (el.addEventListener){

El.addEventListener(‘blur’,function(){checkUsername(5)}, false);

}else{

El.attachEvent(‘onblur’, function(){checkUsername(5);});

}

The event object

When an event occurs, the event object tells you information about the event, and the element it happened upon.

The event object contains data about the event such as:

* Which element the event happened on
* Which key was pressed for a keypress event
* What part of the viewport the user clicked for a click event

Event listener with no parameter:

Function test(e){

Var target = e.target;

}

El.addEventListener(‘blur’,test,false);

Event listener with parameters:

Function test(e, minLength){

Var target = e.target;

}

El.addEventListener(‘blur’, function (e){

Test(e, 5);}, false);

The event object is available in IE 5-8 as a child of the window event:

Function test(e){

If (!e){

E=window.event;

}

}

*Look at example on p265*

Event flow

HTML elements nest inside other elements. If you hover or click on a link, you will also be hovering or clicking on its parent elements.

Event delegation

Creating event listeners for a lot of elements can slow down a page, but event flow allows you to listen for an event on a parent element.

By attaching an event listener to a containing element, you are only responding to one element (rather than having an event handler for each child element).

E.g. place the event handler on the <ul> as opposed to each <li> item.

Changing default behaviour

The event object has methods that can change:

1. The default behaviour of an element
   1. preventDefault() --- (returnValue=false in IE5-8)
2. How the elements ancestors respond to the event
   1. stopPropagation() --- (cancelBubble = true in IE5-8)

Use event delegation

Revise page 268-269

**Part 2**

Which element did an event occur on

When calling a function, the event objects target property is the best way to determine which element the event occurred on. But you may also see the ‘this’ property used.

this. Keyword = refers to owner of the function

E.g.

Function checkUsername(){

Var elMsg = document.getElementById(‘text’);

If (this.value.length<5){

elMsg.innerHTML = ‘random text’;

} else {

elMsg.innerHTML = ‘other random text’;

}

}

this only works when function has no parameters.

*The event object is the preferred way*

UI events

UI events should be attached to the browser window.

E.g. window.addEventListener(‘load’,function,false);

Mouse events

See example on p277

Where events occur

ScreenX and screenY – refers to the entire screen

pageX and pageY – refers to web page

clientX and clientY – refers to browsers viewport

E.g.

Var sx = document.getElementById(‘textbox’);

Function position(e){

sx.value = e.screenX;

}

Var el = document.getElementById(‘body’);

El.addEventListener(‘mousemove’, position, false);

Keyboard events

The event object has a property called ‘keyCode’, which is used to tell which key was last pressed (returns an ASCII code, not actual letter/symbol). In order to get actual character pressed use – string.fromCharCode(event.keyCode);

E.g. function charCount (e){

Var textEntered, charDisplay, counter, lastKey;

textEntered = document.getElementById(‘message’).value;

charDisplay = document.getElementById(‘charactersLeft);

counter = (180 – (textEntered.length));

charDisplay.textContent = counter;

lastKey = document.getElementById(‘lastKey’);

lastKey.textContent = ‘last key pressed was: ‘ + e.keyCode;

}

Var el = document.getElementById(‘message’);

El.addEventListener(‘keyup’, charCount, false);

Form events

Look at example on p283

Mutation events

Whenever elements are added to or removed from the DOM, its structure changes. This change triggers a mutation event.

New ‘mutation observers’ are going to replace mutation events. These are designed to wait until a script has finished its task before reacting, then report the changes as a batch (rather than one at a time).

E.g. var elList, addLink, newEl, newText, counter, listItems;

elList = document.getElementById('list');

addLink = document.querySelector('a');

counter = document.getElementById('counter');

function addItem(e) {

e.preventDefault();

newEl = document.createElement('li');

newText = document.createTextNode('New list item');

newEl.appendChild(newText);

elList.appendChild(newEl);

}

function updateCount() {

listItems = elList.getElementsByTagName('li').length;

counter.innerHTML = listItems;

}

addLink.addEventListener('click', addItem, false);

elList.addEventListener('DOMNodeInserted', updateCount, false);

HTML 5 events

E.g.

HTML

<form id=’messageForm’>

<textarea id=’message’></textarea>

<input type=’submit’ value=’next’/>

</form>

JS

function setup() {

var textInput;

textInput = document.getElementById('message');

textInput.focus();

}

window.addEventListener('DOMContentLoaded', setup, false);

window.addEventListener('beforeunload', function(event) {

var message = 'You have changes that have not been saved';

(event || window.event).returnValue = message;

return message;

});

**SUMMARY**

General

1. Define all syntax in the table on page 1
2. List all truthy and falsy values

Functions & Objects

1. Write an object – literal notation
2. Write an object – constructor notation
3. Create a function which returns the area and volume of a shape separately
4. Create an object for a video game character and return current health

Built-in objects

1. List the three built in objects and define what each one does
2. What is the most used object in the BOM?
3. What is the most used object in the DOM? And list 3 methods
4. List all 4 global objects and give examples of the properties and methods for each one.
5. Create a page that shows:
   1. current date and time on screen
   2. generates a random number in a text box
   3. print the current page

Decisions & loops

1. list all comparison operators
2. List all logical operators
3. If statements vs switch statements
4. Create an if…else statement
5. Create a switch statement

Loops

1. Why would you run a for loop over a while loop?
2. Write a for loop to show the steps of how to cook an egg
3. Create a while loop to do the same as above
4. Create some code using decisions and loops so dependent on situation, the code will either do addition or multiplication of a certain number

DOM

Part 1

1. Describe the DOM
2. List the 4 nodes in the DOM tree
3. How do you locate nodes in order to ‘work’ with them in the DOM tree?
4. How can you change the content of a text node?
5. List the 4 methods that access, change, update and remove attribute values
6. What is meant by DOM query?
7. What is meant by caching DOM queries?
8. Practical – store a dom query in a variable and access its contents (including HTML) to show up in an alert box

Part 2

1. Select an element via id, tag name and CSS selector and then change its class name to apply a style change via css
2. What is a node list
3. What are the two ways to select an element from a node list
4. Use a for loop to change the css class of all list items
5. Navigate through the dom tree to select a text node and use an alert to show its content
6. Change the content of a text node using the replace() method
7. Change the content of an element using:
   * 1. innerHTML
     2. DOM manipulation
8. Remove an element via DOM manipulation

Part 3

1. List 4 methods associated with attributes and describe what each one does
2. List the 2 properties associated with attributes
3. Check whether an element has an attribute and if it does:
   1. Show name of attribute on the page
   2. Remove said attribute
4. Change the class name of an element using the 2 different options
5. How do you examine the DOM in chrome?
6. Complete example on page 240-241 without looking at the code

Events

Part 1

1. Create an event which checks the length of the username. If username is less than 5 then add code adds a hint informing the user to increase character count
2. In above example, add more code to ensure the code will work on older versions of IE
3. What is the event object?
4. How to use event object on:
   1. Event listener with no parameters
   2. Event listener with parameters
5. What is the IE 5-8 equivalent of the event object?
6. Complete example on p265 without looking
7. What is meant by event flow and how does it help?
8. Define event delegation
9. What are the 2 methods that change default behaviour?
10. Look at the event delegation example on p268-269 and study/re-write without looking

Part 2

1. Create an event listener using the ‘this’ keyword in function
2. Use the load event and focus/blur method to check length of username so:
   1. When the page loads the text box is focused
   2. A helpful tip is shown under the text box
   3. When blur occurs, the username length is checked
3. Create a ‘system maintenance’ block that you can click to dismiss and show original content on page
4. Create 6 text input boxes, each one to show individually the X and Y axis of the screen, page and client
5. Create some code with a text area, so that the user has a max of 180 characters to enter, the number of characters remaining should be shown on the page
6. Look at the form event example on page 283 and study/re-write without looking
7. Look at mutation event example on p285 and study/re-write without looking
8. Look at HTML5 event example on p287 and study/re-write without looking