CERTIFICATION EXAM STUDY GUIDE V2

W3Schools - JavaScript

How to utilize the guide: This is a compilation of links and subjects to become comfortable with prior to taking the exam. If you prefer to look over notes and an organized list of links all in one place this may help you study better. Completed by Mason (ﾉ*ΦωΦ)ﾉ

LINK: JavaScript Where To

- ☐ **What is the tag for JS code**

  The `<script>` tag

```
<script>
function myFunction() {
      console.log('Hello World!');
}
</script>
```

- ☐ **Where can JavaScript be placed on a HTML document (index.html file)**

  Inside the `<head>` or `<body>` tag

```
<head>
      <script>
      function myFunction() {
            console.log('Hello World!');
      }
      </script>
</head>
```

```
<body>
     <script>
     function myFunction() {
          console.log('Goodbye World!');
     }
     </script>
</body>
```

- ☐ **What is the file extension for a JS external file (script.js file)**

  .js extension: JS main.js

- ☐ **How to reference external script (JS code external to the HTML document)**

```
<script src="myScript.js"></script>
```

LINK: https://www.w3schools.com/js/js_**statements**.asp

- ☐ **What separates JS statements**

  Semicolons ;

  Add a semicolon ';' at the end of each executable statement

```
let a, b, c; // Declare 3 variables
a = 5; // Assign the value 5 to the variable 'a'
b = 6; // Assign the value 6 to the variable 'b'
c = a + b // Assign the sum of 'a' and 'b' to variable 'c'
```

- ☐ **What is a keyword**

  A word to help identify the JS action to be performed at the start of a statement

```
var a = 1;
```

```
let b = 2;
const c = 3;
if (a > 0) ...//do something
switch (fruits) ...//do something
for (let i = 0; i < array.length; i++) ...//do something
function (myFunction) ...//do something
return result; ...//return result
try {thisCode()} catch (error) {thisError()} //try code
```

LINK: https://www.w3schools.com/js/js_**syntax**.asp

- ☐ **What are the 2 types of values in JS**

  Fixed values: literals

  Variable values: variables

```
// literals
10.50
1001
"Hello World"
'Hello World'
// variables
var x;
let y;
const z;
```

- ☐ **What are the 2 JS literals and its syntax**

  Numbers and Strings

```
// number literals
1337
// string literals
"You can do this!"
```

- [ ] **What is a JS expression**

  A combination of values, variables and operators which computes to a value

```
5 * 10 // equals 50
x * 10 // guess we'll never know what it equals to
"Yee" + " " + "haw" // equals "Yee haw"
```

- [ ] **Is JS case sensitive**

  YES

```
myvariable !== myVariable
```

- [ ] **What type of case does JS use**

  camelCase

```
let myVeryLongVariableWillLookLikeThisKinda = 1
```

- [ ] **What character set does JS use**

  Unicode character set. Essentially covers almost all characters, punctuations, and symbols in the world

LINK: https://www.w3schools.com/js/js_**comments**.asp

- [ ] **What is syntax for JS single line and multi-line comments (non-executed statements ignored by browser)**

  // for single line comments

  /* for multiple lines of comments */

```
// this is a comment
/*
why
you do
this
to me?? (Please make sure your comments are actually meaningful
*/
```

LINK: https://www.w3schools.com/js/js_**variables**.asp

- ☐ **What are the three ways to declare a variable**

  `var`: stands for variable

  `let`: I guess this one is self explanatory…

  `const`: stand for constant

  `!!BONUS!!` use nothing / no keyword

```
var x = 1; /* var was the original keyword to declare variables. I do not recommend using it unless you must. General good practice is using let as your keyword */
let y = 2; /* introduced in ES6, is now the preferred method and is a better version of var. Allows variables to be updated but not re-declared */
const z = 3; /* introduced in ES6, great for making variables where you want to make sure the value is neither updated or re-declared by accident */
a = 4; /* creates a global variable 'a' that equals 4, I do not recommend using this at all as well. General good practice is using let or const before declaring */
```

- ☐ **What is a variable**

  Containers for storing data / data values

- ☐ **What are unique JS identifiers and their general rules**

Unique names are used to identify variables in JS. Can be short or long it's up to you (. ❛ ᴗ ❛.)

- Can contain letters, digits, underscores, and dolla signs
- Must start with a letter or $ and _ depending on the case
- Are CaSe SeNSiTiVe
- Reserved JS words cannot be used (ex. JS keywords)

```
let x; // correct
let myVariable; // correct
let 4; // wrong
let let; // wrong
```

NOTE "$" and "_" are not commonly used in JS, but supposedly pros like to use them. $ denotes the main function in a library. "$" is also found in jQuery. "_" is used for "private(hidden)" variables. "_" commonly found in class constructors.
From Codecademy:

> "Notice, we also prepended our property names with underscores (`_name` and `_behavior`), which indicate these properties should not be accessed directly"

- [ ] **What is the assignment operator and the "equal" sign**

Operator to assign your literal to a variable, which is represented by the "=" sign. In other words you are declaring a variable and setting a literal to it

```
let x = 1; /* 'x' is your variable, '=' is the assignment
operator, '1' is the literal */
```

- [ ] **What is the difference between = vs. == vs. ===**

= : Assignment operator (think about algebra)

== : Equality operator

=== : Strict Equality operator

```
let y = 10; /* another way to word this is as if you're storing the value 10 to the variable y */
return a == 20; /* this will return true whether a was declared as 20 (integer) or "20" (string) */
if (b === 100) ... /* the statement will only run if b exactly equals 100 (same data type and data value) */
```

- [ ] **How to create a variable**

ezpz lemon squeezy

```
var newVariable;
// or
let newVariable;
// or
const newVariable;
```

- [ ] **How to declare many variables in one statement**

```
let someGuy = "Joe Mama", someCar = "Honda trashbox", somePrice = 420;
```

- [ ] **What is an undefined variable**

A variable without a value. This is useful when you want to create a variable but keep it empty until you calculate something or assign it something later down in your code.

```
let emptyStomach;
```

- [ ] **How to re-declare a variable**

Initiating the same variable will reset its value to whatever you set it to. This ONLY WORKS with var

```
var indecisivePerson = "I want a new MacBook"
var indecisivePerson = "Wait...maybe I want a Windows laptop..."
console.log(indecisivePerson) // will log "Wait...maybe I want a Windows laptop..."
```

- ☐ **Read about the dollar sign ($) and underscore (_) and how to begin an JS identifier**

```
let $myMoney = 0;
let _topSecret = "You will pass!"
```

-Refer to the third question's note in this section-

- ☐ **Complete the Test Yourself Exercise**

LINK: JavaScript Let

- ☐ **Can a let variable be redeclared or reassigned?**

No, it cannot be redeclared within the same block but you can use the same variable name in separate blocks to assign different values. You can also still modify the value of the variable through other means

```
function one() {
      let x = 9;
};
function two() {
      let x = 10;
};
```

```
let x = 21;
x = 12; // this works and x will equal 12
/* x will stay as 9 within the first function and x will stay as 21
within the second function and x will stay as 12 globally or outside*/
```

LINK: https://www.w3schools.com/js/js_**const**.asp

- ☐ **Can a const variable be redeclared or reassigned?**

  Nope you cannot at all. Zero. Zip. Nada.

```
const x = 21;
x = 12; // this will return an error
```

LINK: https://www.w3schools.com/js/js_**operators**.asp

- ☐ **What are the arithmetic operators**

  Operators used to perform arithmetic a.k.a math

```
9 + 10 // add 9 and 10
10 - 5 // subtract 10 by 5
2 * 4 // multiply 2 and 4
2**3 // 2 to power of 3
15 / 5 // divide 15 by 5
4 % 2 // divide 4 by 2 but return the remainder value
count++ // increment variable count by 1
count-- // decrement variable count by 1
```

- ☐ **What are the assignment operators**

  Operators used to assign values to variables

```
x = 5 // assign variable x to have a value of 5
x += y // x = x + y
x -= y // x = x - y
x *= y // x = x * y
x /= y // x = x / y
x %/ y // x = x % y
x **= y // x = x ** y
```

- [ ] **How to add strings and numbers**

```
let x = "5" + 5 /* this will return "55" and will not add them together mathematically
but physically. Will always return as a string type */
```

- [ ] **What are the comparison operators?**

```
== // equal to
=== // strictly equal to aka equal value + equal type
!= // not equal
!== // strictly not equal to aka not equal value + not equal type
> // greater than
< // less than
>= // greater than or equal to
<= // less than or equal to
? // ternary operator, can think of it as an inline conditional
```

- [ ] **What are the type operators ?**

Operators that check the data type of whatever you're checking

```
console.log(typeof 21) // will return number
console.log(object instanceof constructor) /* will return true or false depending if
the object you are testing appears within the constructor you are testing against */
```

- [ ] **Complete the Test Yourself Exercise**

LINK: https://www.w3schools.com/js/js_**arithmetic**.asp

- ☐ **Complete the Test Yourself Exercise**

LINK: https://www.w3schools.com/js/js_**assignment**.asp

- ☐ **Complete the Test Yourself Exercise**

LINK: https://www.w3schools.com/js/js_**datatypes**.asp

- ☐ **What are the 5 data types**

  numbers, strings, booleans, arrays, and objects

```
5 // a number
"Hello" // a string
true // a boolean
[1, 2, 3] // an array
{name: "Jeff", age: 21, isUndercover: false, favoriteAnimals: ["cat", "dog"]} // an
object
```

- ☐ **Complete the Test Yourself Exercise**

LINK: https://www.w3schools.com/js/js_**functions**.asp

- ☐ **What is a function**

  A block of code designed to perform a particular task

```
function myFunction(param1, param2) {
      // do something
};
```

- ☐ **What are parameters and arguments**

Parameters are variables passed into a function. Arguments are the values assigned to those variables (these are optional and only required if the function depends on the outside variables)

```
function sumOfTwo(a, b) {
      return a + b;
};
console.log(sumOfTwo(2, 3)); /* the function needs two parameters in order to return any result. passing in 2 as the first parameter and 3 as the second, the function can then calculate the sum of them which will log 5 */
```

- [ ] **What is function invocation?**

When you "invoke" or call a function, the code inside the function will execute. You can either execute a function by either:

- Firing an event / an event occurs (ex. when a button on the page is clicked)
- Calling it within the JS code (the normal way)
- Automatically (a.k.a self invoking functions)

```
// Run this function whenever the button with a matching id of "example" is clicked
const container = document.getElementById("example");
container.addEventListener('click', e => {
      // do something
)};
// Run function by calling it normally
function sendHelp() {
      console.log("Help me!!!");
};
sendHelp();
```

```
// Runs immediately one time through self-invoking
(function () {
      console.log("I'm here to save you!");
})();
```

- ☐ **What is a local variable?**

  Variables declared within a function become local variables

```
function myFunction() {
      let local = "This variable only works inside this function";
      // random code...go!
};
// code here will not be able to use the "local" variable
```

- ☐ **Complete the Test Yourself Exercise**

LINK: https://www.w3schools.com/js/js_**objects**.asp

- ☐ **What do objects have**

  Objects share the same properties and methods. Properties hold specific values pertaining to that object. Methods are actions that pertain to that object.

```
const car = {
      brand: "Toyota",
      make: "Supra",
      year: "2020",
      quote: function() {
            return "The " + this.year + " " + this.brand + " " + this.make + "
unfortunately only comes in automatic...";
```

```
};
console.log(car.quote());
/* when the function stored in the property “quote” is run for this object, it will
return the string “The 2020 Toyota Supra unfortunately only comes in automatic...” */
```

- **What are object methods**

  Actions that can be performed on the object and are stored as function definitions.

  There are also predefined methods for objects such as object.create() or object.values()

```
const randomStuff = {
     a: "mom's spaghetti",
     b: 21,
     c: true
};
console.log(Object.values(randomStuff));
// .values() will return the value of each property as an array ["mom's
spaghetti", 21, true]
```

- **While accessing the object, what is the difference between object.property and object.property( )**

  object.property will return the value associated with the property named ‘property’ within the object

object.property() will run the function associated with the method named ‘property’ within the object

```
const anotherOne = {
      person: "John Wick"
      nickname: function () {
            console.log("Babayaga");
      }
};
console.log(anotherOne.person); // will print "John Wick"
anotherOne.nickname(); /* will run the function which logs "Babayaga" */
```

- ☐ **What does the “this” keyword refer to?**

this refers to an object and depends on where you are calling it

- in an object method, this will refer to the object
- alone, this will refer to the global object
- in a function, this will refer to the global object
- in a function utilizing strict mode, this will be undefined
- in an event, this refers to the element receiving the event

```
const lonelyObject = {
      theOnlyProperty: "Sadness",
      theOnlyMethod: function() {
            return this.theOnlyProperty;
      }
};
console.log(lonelyObject.theOnlyMethod()); // will console.log "Sadness"
```

- ☐ **What data types cannot be declared as objects ?**

  Strings, Numbers, and Booleans. This is a no-no

```
x = new String(); // declares x as a string object
y = new Number(); // declares y as a number object
z = new Boolean(); // declares z as a boolean object
```

- ☐ **What happens when a JS variable is declared with the "new" keyword ?**

  It creates the variable as an object. This will complicate your code and slow it down as if you're still using Internet Explorer

- ☐ **Complete the Test Yourself Exercise**

LINK: https://www.w3schools.com/js/js_**date_methods**.asp

- ☐ **What method returns the time?**

  The getTime() method

```
const time = new Date();
time.getTime();
```

- ☐ **In what measurement is the time returned (since January 1, 1970)**

  Milliseconds

```
const time = new Date();
console.log(time.getTime()); /* will return 1667010626801 at the time I am writing
this...like bruh */
```

- ☐ **What method returns the day of the date**

  The getDate() method

```
const time = new Date();
console.log(time.getDate()); /* will return 28 because it is the 28th of this month as
```

```
I am writing this */
```

- [ ] **What method adds days to a date?**

  The `setDate()` method

```
const time = new Date();
console.log(time.setDate()); /* will return Sat Oct 15 2022 22:38:05 GMT-0400 (Eastern Daylight Time) ya boi from NYC I guess */
```

LINK: https://www.w3schools.com/js/js_**events**.asp

- [ ] **Which file are most event handlers added (.html, .css, or .js)?**

  I guess HTML, but I personally don't think it really matters realistically…

```
<!-- HTML Example -->
<button onclick="document.getElementById('example').innerHTML = Date()">What time is it?</button>
<!-- the element matching id "example" will add the current time to the DOM. You're free to use this if the code is short or else it will look too cluttered -->
```

```
// JS Example
const button = document.getElementById('button');
button.addEventListener('click', e=> {
      console.log("You clicked me!");
}
/* the <button> element with matching id "button" will fire this event on click from the JS file. Definitely better when executing longer code blocks */
```

- [ ] **Check out the following events:**

All events can be done in either HTML or JS

- [ ] **Onchange**

  Execute JS code when a user changes an element

```
<!-- HTML -->
<span>Enter name: <input type="text" id="name" onchange="myFunction()"></span>
<script>
function myFunction() {
      const input = document.getElementById('name');
      input.value = input.valuetoUpperCase();
}
</script> <!-- adding text to the input field will uppercase the word when user
leaves the input field -->
```

- [ ] **Onclick**

  Execute JS code when a user clicks an element

```
<!-- HTML -->
<button type="button" id="button" onclick="myFunction()">Click me!</button>
<script>
function myFunction() {
      const button = document.getElementById('button');
      alert("Uh oh that tickles");
}
</script> <!-- when the button with matching id "button" is clicked, an alert
window will pop up with the message "Uh oh that tickles" -->
```

- [ ] **Onmouseover**

  Execute JS code when a user moves their mouse onto a specific element (Look below for combined example)

- [ ] **Onmouseout**

Execute JS code when a user moves their mouse away from a specific element after hovering over it

```
<!-- HTML example for both onmouseover() and onmouseout()-->
<p id="example" onmouseover="mouseOver()" onmouseout="mouseOut()">Come here
mouse</p>
<script>
function mouseOver() {
      document.getElementById('example').style.color = "red";
}
function mouseOut() {
      document.getElementById('example').style.color = "black";
</script> <!-- when the mouse hovers over the paragraph with matching id
'example', the color of the text will turn red. When the mouse leaves, it will
turn black -->
```

- [ ] **Onkeydown**

Execute JS code when a user presses a key on their keeb

BONUS (keydown refers to the downstroke while keyup refers to the upstroke, and keypress is similar to keydown but doesn't trigger when pressing modifier keys like ctrl, alt, delete, etc)

```
<!-- HTML -->
<span>Enter name: <input type="text" id="name" onkeydown="myFunction()"></span>
<script>
function myFunction() {
      document.getElementById('name').style.color = "red";
}
</script> <!-- adding text to the input field will make red letters as you type-->
```

- **Onload**

  Execute JS code when the page is loaded

```
<!-- HTML -->
<img src="example.jpg" onload="myFunction()"/>
<p id="example"></p>
<script>
function myFunction() {
      document.getElementById('example').innerHTML = "Mr. Stark, your image has
loaded.";
}
</script> <!-- when the image loads, a paragraph will show the string "Mr. Stark,
your image has loaded." -->
```

LINK: https://www.w3schools.com/js/js_**popup**.asp

- **What type of popup boxes are there?**

  Alert box, Confirm Box, and Prompt Box

- **How many buttons are included in each type of popup boxes?**

  Alert: 1, Confirm: 2, Prompt: 2

```
/* will show a popup with an alert saying "Uh oh spaghetti meatball" when ran using
whatever method */
function myAlert() {
      alert("Uh oh spaghetti meatball");
};
/* will show a popup with the question "Red pill or Blue pill?" and add the message to
an element with an id "example" corresponding to the button clicked. OK will equal red
and Cancel will equal blue */
```

```
function myConfirm() {
      let msg;
      if (confirm("Red pill or Blue pill?")) {
            msg = "OOOHHH YEAHHHHH";
      } else {
            msg = "I'm blue da bu de da bu da";
      document.getElementById('example').innerHTML = msg;
};
/* will show a popup with a prompt "Your name:" with a placeholder of "Jeff". If the
user cancels or submits with an empty field, an element with an id "example2" will have
a message "Disrespectful". Filling in the prompt and submitting will show the message
"Welcome Deadpool, to Wakanda.." */
function myPrompt() {
      let text;
      let person = prompt("Your name:", "Jeff");
      if (person == null || person == "") {
            text = "Disrespectful.";
      } else {
            text = "Welcome " + person + ", to Wakanda..";
      }
      document.getElementById("example2").innerHTML = text;
};
```

LINK: https://www.w3schools.com/js/js_**window_navigator**.asp

- ☐ **What kind of information does the window.navigator object contain?**
  Contains information about the user's browser
- ☐ **How do you discover the application name of the browser**
  `navigator.appName`

WARNING this property has been deprecated and will return Metaverse…, I mean Netscape as the appCodeName for most major browsers

```
<!-- HTML -->
<p id="example"></p>
<script>
document.getElementById('example').innerHTML = navigator.appName;
</script> <!-- the p element will read Netscape -->
```

- [ ] **How do you discover the application code of the browser**

navigator.appCodeName

WARNING this property has been deprecated and will return Godzilla…, I mean Mozilla as the appCodeName for most major browsers

```
<!-- HTML -->
<p id="example"></p>
<script>
document.getElementById('example').innerHTML = navigator.appCodeName;
</script> <!-- the p element will read Netscape -->
```

- [ ] **How do you discover the product name of the browser engine**

navigator.product

WARNING this property has been deprecated and will return Lizard…, I mean Gecko as the browser engine for most major browsers

```
<!-- HTML -->
<p id="example"></p>
<script>
document.getElementById('example').innerHTML = navigator.product;
</script> <!-- the p element will read Netscape -->
```

LINK: https://www.w3schools.com/js/js_**cookies**.asp

- ☐ **What are cookies?**

  Data stored in small text files on your computer that allow web pages to remember info about the uninvited visitor

- ☐ **How do you create, read, and delete cookies?**

  Create cookie with document.cookie property

```
document.cookie = "username=Jeff; expires=Sat, 29 Oct 2022 12:00:00 UTC";
```

  Read cookie by assigning document.cookie to a variable

```
let cookieMonster = document.cookie;
```

  Delete a cookie by passing an expiration parameter with a date that has already passed

```
document.cookie = "username=Jeff; expires=Fri, 29 Oct 2010 12:00:00 UTC";
```

LINK: https://www.w3schools.com/js/js_ex_browser.asp

- ☐ **BONUS I guess, these are the main browser objects:**

  Window Object, Screen Object, Location Object, History Object, Navigator Object, Popup Boxes, Timing