Javascript - Coding Behaviors in Websites

JavaScript and HTML

document.write is a method that can be used to output text, HTML , or CSS to the browser.
Other useful input/output statements include alert() and console.log().

```
<!DOCTYPE html>
<body>
<script>
document.write("<h1>Hello World!</h1>");
alert("Welcome to year 2018.");
console.log("Open the Chrome inspector console to see this message");
</script>
</body>
```

JavaScript Comments

Comments in JavaScript begin with double slashes.

```
<!DOCTYPE html>
<body>
<script>
// Write a message to the browser
document.write("<h1>Hello World!</h1>");
document.write("<p>Welcome to the new year.</p>");
</script>
</body>
```

JavaScript Variables

Declare variables using the var statement. **Strings** are created using quotes. **Numbers** have no quotes. **Booleans** are true or false. The value of the variable can be set when declared, or later in the program. Variables can be used in commands. When you refer to variables, do not put quotes around them.
The plus (+) operator **adds** when used with number variables, and **concatenates** when used with strings.

```
<!DOCTYPE html>
<html>
<body>
<script>
var name = "David";
var year;
year = 2019;
// Write a message to the browser
document.write("<h1>Hello " + name + "!</h1>");
alert("Welcome to year " + year + ".");
</script>
</body>
</html>
```

JavaScript Functions

Functions allow programmers to write reusable blocks of code that can be called from a script.

```
function myFunction() {
	... procedural code goes here ...
}
```

After the function is defined it can be called from your main program.

```
myFunction();
```

Functions can accept **parameters**, which are values or variables passed into the function.
They can also pass back a **return** value, which can get assigned into a variable in the main program.

```
function multiply(factor1, factor2) {
	return (factor1 * factor2);
}

var product;
product = multiply(2, 3);
alert(product);
```

Names of variables in the function definition can be different than the names in the function call.

```
function multiply(factor1, factor2) {
	return (factor1 * factor2);
}

var product;
var number1 = 2;
var number2 = 3;
product = multiply(number1, number2);
alert(product);
```

Scope

Scope refers to the locations in a program where the variables exist. ***A variable defined inside a function does not exist outside the function.*** Avoid using the same variable name for different purposes inside and outside functions!

```
function scopeTest(value) {
	alert("Variable value is " + value + ". It exists in the function.");
}

var number = 5;
scopeTest(number);
alert("Variable value is " + typeof value + " outside the function.");
```

Object Oriented Methods

This code demonstrates a **message handler**, or listener, for the HTML element that is selected by the ID "info. " getElementById lets programmers access elements in the document (DOM, Document Object Model) by selecting an id attribute value.

The onclick listener will call the function when a mouse click message is broadcast to the button object.

```
<!DOCTYPE html>
<html>
<body>

<form>
	<input type="button" id="info" value="info">
</form>

<script>
document.getElementById("info").onclick = function() {
	alert("My calculator script");
}
</script>

</body>
</html>
```

JavaScript Arithmetic Operators

Operator	Description	Example	Result
+	Addition	5 + 2	7
-	Subtraction	5 - 2	3
*	Multiplication	5 * 2	10
/	Division	5 / 2	2.5
%	Modulus (division remainder)	5 % 2	1
++	Increment	5++	6
--	Decrement	5--	4

JavaScript Assignment Operators

Operator	Example	Equivalent Statement	
=	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	

Calculator Project

Add form fields, a button, and a function to add numbers together. What error occurs>

```
<!DOCTYPE html>
<html>
<body>

<form>
	<div><input type="text" id="number1"></div>
	<div><input type="text" id="number2"></div>
	<input type="button" id="add" value="+">
	<input type="button" id="info" value="info">
</form>

<script>
document.getElementById("add").onclick = function() {
	var number1,
	    number2,
	    total;
	number1 = document.getElementById("number1").value;
	number2 = document.getElementById("number2").value;
	total = number1 + number2;
	alert ("The answer is " + total);
}
document.getElementById("info").onclick = function() {
	alert("My calculator script");
}
</script>

</body>
</html>
```

Data Types and Casting

Adding string variables concatenates them together, as seen in the first Hello World variables example. Form fields are always treated as strings, even if the user types numbers. To perform addition, the variables storing the form field values must be cast from the string data type to the number data type. The Number() function in JavaScript applies the necessary data cast.

Change the line of code that adds the numbers to the following:

```
	total = Number(number1) + Number(number2);
```

Challenge #1 Add a new button and message handler function for each of the following:

- Subtract
- Multiply
- Divide

Be sure to use a different id for each new button and select the correct button id in each handler!

Arrays

An array is a data structure that stores a list of variables. The values can be any JavaScript variable type.

- Number: 9
- Quoted String: “Text”
- Boolean: true or false

Note the following:

1. Type **brackets** around the list of variables.
2. Separate items in the array using **commas**.
3. Access array list items using bracket notation and the **index** of the item you want to select.
4. **Numeric arrays** like this use numbers for the index values.

Example:

```
var myCars = ["Saab","Volvo","BMW"];

alert(myCars[0]);
alert(myCars[1]);
alert(myCars[2]);
```

If Conditional:

Conditional control structures allow you to create multiple branches which are run only under certain conditions. When the expression evaluates to TRUE, then the following code block in brackets is run.

Pseudocode:

```
IF …
THEN …
ELSE …
ENDIF
```

JavaScript Code:

```
if (expression) {
      code block to be executed
} else {
      code block to be executed
}
```

JavaScript Expressions

Expressions are phrases that evaluate to TRUE or FALSE. For example,

```
(age === 21)
```

JavaScript Comparison Operators for Expressions

Operator	Description	Example	Result
>	Greater Than	5 > 2	TRUE
<	Less Than	5 < 2	FALSE
==	Equality	5 == "5"	TRUE
===	Identity (Equal To, and Same Data Type)	5 === "5"	FALSE
>=	Greater Than or Equal To	5 >= 2 + 1	FALSE
<=	Less Than or Equal To	5 <= 2 + 3	TRUE
!=	Not Equal To	5 != "5"	FALSE
!==	Not Identical To	5 !== "5"	TRUE

JavaScript Boolean Operators for Expressions

Operator	Description	Example	Result
&&	And	TRUE && FALSE	FALSE
||	Or	TRUE || FALSE	TRUE
!	Not	!TRUE	FALSE

Date Object

JavaScript includes built in objects. The date object can retrieve the user's current date and time from their operating system, and perform various date and time calculations.

```
var currentDate = new Date;

var year = currentDate.getFullYear();
var month = currentDate.getMonth();
var day = currentDate.getDate();
var hour = currentDate.getHours();
var minute = currentDate.getMinutes();
var second = currentDate.getSeconds();
```

Challenge #2 Display a sun picture during the day and a moon picture at night.

- Use the JavaScript date object to get the current hour of the day (returns 0-23)
- Write an expression with 2 comparisons and 1 boolean operator
- The result of the expression should be true if the hours are after sunrise AND before sunset.
- Use the if…else conditional to create separate code branches to display each image.

Challenge #2 – solution

```
<!DOCTYPE html>
<html>
<body>
      <script>
      var date = new Date();
      var hours = date.getHours();
      if ( (hours > 8) && (hours < 20) ) {
           document.write("<img SRC='sun.gif'>")
      } else {
           document.write("<img SRC='moon.gif'>");
      }
      </script>
</body>
</html>
```

While Loop:

Looping control structures create a block of code that is repeated until the test expression in parenthesis evaluates to false.

Pseudocode:

```
WHILE …
ENDWHILE
```

JavaScript Code:

```
while (expression) {
      code block to be executed
}
```

Example:

```
var text = "countdown ";
var i = 9;
while (i >= 0) {
    text += String(i);
    i--;
}
alert(text);
```

For Loop:

The for loop has a more condensed syntax where the variable definition, incrementor, and expression text are all written on the same line.

Example:

```
var text = "countdown ";
for (var i=9; i >= 0; i--) {
    text += String(i);
}
alert(text);
```

JSON (JavaScript Object Notation)

Definition

JSON is a human readable text file consisting of data stored in attribute-value pairs.

Originally created for the JavaScript programming language, JSON has become the primary format for sharing data among front end and back end web systems in many programming languages. (Source: www.json.org)

Attribute-Value Pairs (also called name-value, key-value, property-value) are an open-ended data structure consisting of both information and labels for the information.

Example:

<phoneNumber, 555-5555>

The attribute is “phoneNumber,” and the value is “555-5555.”

Syntax

A JSON object starts with an opening curly brace, contains one or more attribute-value pairs, and ends with a closing curly brace.

```
{
      "name": "David Kelleher",
      "phoneNumber": "555-5555"
}
```

Syntax Notes:

1. If there are more than one attribute-value pairs, separate them with a comma. Do not put a comma after the last pair.

2. The following whitespace characters can be used around elements to format the data and make it more readable: space, tab, carriage return, line feed. Use indenting to show where objects and arrays begin and end.

3. Attribute names should be placed in quotes. Format them using camel case – first letter of first word should be lowercase, first letter of additional words should be uppercase, and don’t put spaces between the words.

JSON Values

- **Number**: signed integer or decimal number. Exponent notation (1.6e23) is allowed. Value is not quoted.

```
"age": 26
```

- **String**: characters inside double quotes. Escape double quotes and backslashes with a backslash.

```
"single": "\"One Slip\" \\ \"Terminal Frost / Dogs of War\""
```

- **Boolen**: true or false

```
"isFaculty": true
```

- **Null**: empty value, using the word null

```
"campusAddress": null
```

- **Array**: list of values, which can be of any data type, even nested JSON objects. Open and close with square brackets, and separate list items with commas.

```
[
	"alpha",
	"bravo",
	"charlie"
]
```

- **Object**: a nested JSON object including the opening and closing curly braces

```
{
	"name": "Homer Simpson",
	"address":
	{
		"streetAddress": "742 Evergreen Terrace",
		"city": "Springfield"
	}
}
```

JSON Example

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address":
  {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers":
  [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON-LD (JSON for Linked Data in HTML)

Definition

JSON-LD is a method of publishing structured data so it can interlinked, using the JSON format.

Linked Data is structured data where some values are hyperlinks to records in other databases.

This code goes in your HTML script files, and can get your information into the Google graph database!

Syntax

The syntax follows the rules of JSON. A JSON-LD object includes special attributes, such as:

- @context: link (URL) to a web page describing the allowed vocabulary, or attributes that may be included in the object.

```
"@context": "http://schema.org"
```

- @type: name of a term or data type defined in the selected @context.

```
"@type": "Person"
```

Usage

To include JSON-LD in a web page, wrap it inside script tags, and place it either in the header, or in the body next to the content that is referenced by the structured data.

```
<script type="application/ld+json">
{
    ...
}
</script>
```

schema.org

The schema.org website is the result of a collaborative, community effort to create a standard set of structured data vocabulary for the Internet. It can be used in conjunction with @context. The list of @type data types currently available can be found at: http://schema.org/docs/full.html.

JSON-LD offers multiple solution for linking an object with other structured data. For the schema.org context, the sameAs attribute is typically used:

- sameAs: the attribute defining a link (URL) to a definitive record of the item's identity. Examples include pages on Wikipedia, Freebase, IMDB, or the item's "official" website.

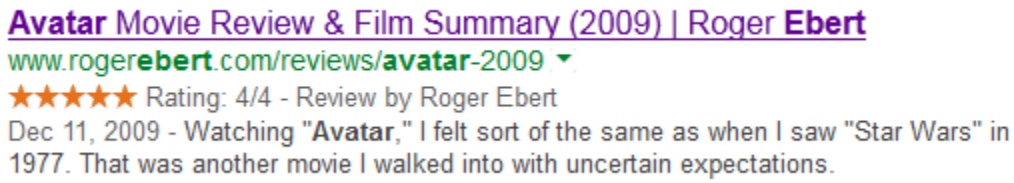
```
"sameAs": "http://www.imdb.com/title/tt0457430/"
```

Google Knowledge Graph

Structured data on a web page is called called **microdata**, since it is used to encode a small amount of key data on a web page, like a company's name and business hours, in a machine readable format.

Programmers have different methods for creating microdata, but JSON-LD is emerging as the standard.

Google reads microdata and stores it in its **knowledge graph**, to answer factual questions like what is the capital of Massachusetts. It is also used to return **rich snippets** for search queries:



Example

```
<script type="application/ld+json">
{
      "@context": "http://schema.org",
      "@type": "LocalBusiness",
      "address":
      {
            "@type": "PostalAddress",
            "addressLocality": "Irvine",
            "addressRegion": "CA",
            "postalCode":"92618",
            "streetAddress": "123 Happy Lane"
      },
      "description": "This is your business description.",
            "name": "Gene's Delicious Donuts",
      "telephone": "555-111-2345",
      "openingHours": "Mo,Tu,We,Th,Fr 09:00-17:00",
      "geo":
      {
            "@type": "GeoCoordinates",
             "latitude": "40.75",
            "longitude": "73.98"
      },
      "sameAs" :
      [
            "http://www.facebook.com/your-profile",
            "http://www.twitter.com/your-profile",
            "http://plus.google.com/your-profile"
      ]
}
</script>
```

Structured Data for a Film Review

```
<script type="application/ld+json">
{
      "@context": "http://schema.org",
      "@type": "Review",
      "author":
      {
            "@type": "Person",
            "name": "Roger Ebert",
            "sameAs": "http://www.rogerebert.com/"
      },
      "datePublished": "2006-12-28",
      "description": "One of the cinema's great fantasies.",
      "inLanguage": "en",
      "itemReviewed":
      {
            "@type": "Movie",
            "name": "Pan’s Labyrinth",
            "sameAs": " http://www.imdb.com/title/tt0457430/"
      },
      "publisher":
      {
            "@type": "Person",
            "name": "Roger Ebert",
            "sameAs": "http://www.rogerebert.com/"
      },
      "reviewRating":
      {
          "@type": "Rating",
          "worstRating": 0,
          "bestRating": 4,
          "ratingValue": 4
      },
      "url": " http://www.rogerebert.com/reviews/pans-labyrinth-2006"
}
</script>
```